Change actions

From incremental computation to discrete derivatives

Mario Alvarez-Picallo

Wolfson College

University of Oxford

A thesis submitted for the degree of

Doctor of Philosophy

Michaelmas Term 2019/2020

Acknowledgements

This thesis represents the fruition of many long years of work and, as such, I feel a profound personal attachment to it. But it is also true that, like most scientific endeavours, it would have been a barren effort were it not for the support and collaboration of many people along the way.

First and foremost, I would like to thank my doctoral supervisors Samson Abramsky and, especially, Luke Ong, without whose support and guidance the very idea for this thesis would have never sparked, let alone flourished. I would also like to acknowledge the valuable suggestions and remarks of my examiners throughout my DPhil: Hongseok Yang and Sam Staton for overseeing my transfer of status, Sam Staton and Andrzej Murawski for overseeing my confirmation, and finally Neel Krishnaswami and once again Sam Staton, who kindly agreed to review this thesis in its entirety.

I am also grateful to the UK Engineering and Physical Sciences Research Council, as well as the Department of Computer Science, for providing the necessary funding for this research, as well as Semmle Ltd. and IOHK, two companies in which I had the privilege of working as an intern. My experiences through these industrial escapades greatly expanded my perspectives on research.

During my work as a DPhil student, I have been lucky to collaborate with some excellent researchers. I must give particular credit to Michael Peyton Jones, whom I had the pleasure to meet while at Semmle, and with whom I would later work again during my internship at IOHK. I would also like to thank Jean-Simon Lemay for some very productive discussions culminating in a joint publication that has informed my work ever since.

Although it is impossible to name them all, I am indebted to many researchers around me. My good friend Alex Kavvos cannot go without mention as, despite my best efforts, his patience with the most outlandish of my ideas has been inexhaustible so far. I owe a good deal to my conversations with Marie Kerjean and her contagious enthusiasm. I should also like to mention in passing all the merry fellows who toiled with me in Office 347, and offer my encouragement to those who remained there after me.

Finally, on a more personal note, I wish to sincerely thank Iris Tomé, my closest friend through these years and, of course, to my mother, Elena Picallo, who not once failed to support me.

Abstract

It seems to be a piece of folkloric knowledge among the incremental computation community that incremental programs behave, in some way, like derivatives. Indeed, they track the effect of a function on finite differences in the input, much like derivatives in calculus track the effect of a function on infinitesimal differences. This idea has recently come to the forefront when Kelly, Pearlmutter and Siskind proposed reinterpreting Cai's incremental lambda-calculus as a basis to understand automatic differentiation.

On the other hand, the differential lambda-calculus, an extension of the lambda-calculus equipped with a differential operator that can differentiate arbitrary higher-order terms, has been shown to constitute a model for differentiation in the traditional sense – that is, there is a model of the differential lambda-calculus where function spaces correspond to spaces of smooth functions, and the term-level derivative operator corresponds to the usual notion of derivative of a multivariate function.

The goal of this thesis is threefold: first, to provide a general semantic setting for reasoning about incremental computation. Second, to establish and clarify the connection between derivatives in the incremental sense and derivatives in the analytic sense, that is to say, to provide a common definition of derivative of which the previous two are particular instances. Third, to give a theoretically sound calculus for this general setting.

To this end we define and explore the notions of change actions and differential maps between change actions and show how these notions relate to incremental computation through the concrete example of the semi-naive evaluation of Datalog queries. We also introduce the notion of a change action model as a setting for higher-order differentiation, and exhibit some interesting examples. Finally, we show how Cartesian difference categories, a family of particularly well-behaved change action models, generalise Cartesian differential categories and give rise to a calculus in the spirit of Ehrhard and Regnier's differential lambda-calculus.

Contents

List of Figures

Chapter 1

Introduction

Consider a piece of input data on which we wish to perform some expensive computation. Accomplishing this is simple enough – it suffices to insert a call to the relevant function in our code [1], pass it our input data and wait until the calculation ends. But what if our input data isn't quite as static as we would like it to be? For example, if we are calculating the clustering coefficient of the graph of friendships on a social network we might expect this graph to change over time – should we, then, restart the whole computation from scratch every time two teenagers bicker? Or should we search for a cleverer algorithm [67]?

Even if the input to our program does not change under our noses of its own volition, many algorithms work by iteratively updating the same piece of data with new information (usually until a fixed point is reached). The famous PageRank algorithm is such an example: even if one assumes that the graph of connections between pages remains static, a naive implementation would require updating the weights of every edge in each iteration. A more cunning algorithm [76] would only recompute the weight of edges that depend on other edges whose weight changed in the last iteration, thus avoiding redundant effort.

Both of these examples are but particular cases of the broader topic of *incremental computation*. At its most general, incremental computation is a family of techniques for updating the output of a program as its input changes (either over time or from an iteration to the next). By its nature, approaches to incrementalising programs tend to be ad-hoc and domain-specific, but general patterns arise: one such common theme is that the incrementalised version of a function is usually referred to as a "derivative" obtained by a process of "differentiation" [82, 12, 13, 23].

[1] Or submit a job to a computing centre, or spin up an AWS instance, depending on your choice of scale.

It seems to be a piece of folkloric knowledge among the incremental computation community that incremental programs behave, in some way, like derivatives. Indeed, they track the effect of a function on *finite differences* in the input, much like derivatives in calculus track the effect of a function on *infinitesimal differences*. This idea has recently come to the forefront when Kelly, Pearlmutter and Siskind proposed reinterpreting Cai's incremental λ-calculus as a basis to understand automatic differentiation [57].

On the other hand, the differential λ-calculus, an extension of the λ-calculus equipped with a differential operator that can differentiate arbitrary higher-order terms, has been shown to constitute a model for differentiation in the traditional sense – that is, there is a model of the differential λ-calculus where function spaces correspond to spaces of smooth functions, and the term-level derivative operator corresponds to the usual notion of derivative of a multivariate function [14, 58].

In full generality, the notion of derivative induced by the differential λ-calculus is captured by Cartesian differential categories, an abstract categorical model for differentiation in calculus and differential geometry [16, 17, 30].

The goal of this thesis is threefold: first, to provide a more general version of Cai and Giarrusso's change structures, and investigate its properties. Second, to establish and clarify the connection between derivatives in the incremental sense and derivatives in the analytic sense, that is to say, to provide a common definition of derivative of which the previous two are particular instances. Third, to give a theoretically sound calculus for this general setting.

To this effect, this thesis is structured as follows: in Chapter 2 we give a more formal account of incremental computation and change structures, the differential λ-calculus and its history and models, together with an in-depth description of Cartesian differential categories. In Chapter 3 we set the basic definitions of a *change action* and a *differential map* between change actions – these being the main conceptual contributions of this work. In Chapter 4 we develop the theory of change actions for the practical application of incrementalising the evaluation of Datalog queries. In Chapter 5 we introduce the notion of a *change action model* as a setting for higher-order differentiation, and exhibit some interesting examples – notably, following our intuition of changes being differences, we show how the calculus of finite differences appears naturally in the context of change actions. In Chapter 6 we define Cartesian difference categories, a generalisation of Cartesian differential categories and a particularly well-behaved class of change action models, for which we develop a higher-order calculus in Chapter 7.

Chapter 2

Preliminaries

This thesis deals with the convergence of two separate notions of derivative: the differentiation of higher-order functions that appears in the differential λ-calculus and its models, and the incrementalisation of programs by a transformation akin to syntactic differentiation. To contextualise the research that is to follow, we offer here a bird's eye view of the corresponding notions. In Section 2.1 we give the historical progression from linear to differential logic, leading up to the development of the differential λ-calculus, of which we give a brief overview in Section 2.2, followed by a discussion of Cartesian differential categories in Section 2.3.

On the other hand, Section 2.4 presents some trends in incremental computation, emphasising Cai and Giarrusso's principled approach to incrementalisation as formal differentiation, from which much of our work is derived. Finally, Section 2.5 closes with a brief introduction to automatic differentiation, and a discussion on how both incremental and differential calculi are connected through it.

2.1 Geometric Models of Linear Logic

While work on the differential λ-calculus would not start until later, the seeds of its development could already be found in earlier works on the semantics of proofs in linear logic [44, 45]. These started as an attempt to formalize the connection between linearity in logic and linearity in the sense of linear algebra, and would culminate in Ehrhard's work on Köthe space semantics [35] and finiteness space semantics [36], which provided the semantic motivation for the development of the differential λ-calculus.

An early glimpse at the connection between linear logic and vector spaces was already present in the foundational work by Girard [44, 45] on the semantics of linear logic, where a denotational semantics is given for proof terms for linear logic which interprets propositions as *coherence spaces*, or reflexive graphs, and proofs as cliques in those graphs. The category of coherence spaces with stable linear maps, i.e. maps that send cliques to cliques and preserve sums (disjoint unions) of cliques can then be endowed with enough structure to interpret every operator in linear logic, including exponentials. As Ehrhard [35] observes, these spaces bear a strong similarity to vector spaces, with the web of a coherence space playing the role of a basis and stable linear maps corresponding to linear (in the algebraic sense) transformations.

This connection was made clear by Blute, Panangaden and Seely [15] when they showed how to model exponential linear logic on a category of vector spaces by using the so called *Fock space construction* [42, pp. 114-119] from quantum physics to model exponentials. Fock spaces are construced as infinite sums of iterated symmetric (or antisymmetric) tensor products $\otimes_s$:

$$\mathcal{F}_s(A) \coloneqq 1 \oplus A \oplus (A \otimes_s A) \oplus \ldots \oplus (\otimes_s^n A) \oplus \ldots$$

where $\otimes_s$ is the coequalizer of the maps $\mathbf{id}, \tau : A \otimes A \to A \otimes A$. Elements of these vector spaces can be thought of as polynomials where variables are guaranteed to commute. The authors then show that the category of Banach spaces [75] together with contractive maps acts as a model of linear logic, with exponentials $!A$ given precisely by the (antisymmetric) Fock spaces $\mathcal{F}_A(A^\perp)^\perp$. Furthermore, they remark, in this particular category the Fock space construction on some Banach space B corsponds to a space of holomorphic (non-linear) functions described by power series on B [15], perhaps the first instance of non-linear function spaces arising as a result of modeling exponentials in linear logic.

The idea of using Banach spaces and, more generally, vector spaces, to model linear logic resulted quite successful: indeed, further work by Girard [47] showed that Banach spaces can be extended into *coherent Banach spaces*, consisting of a pair of (complex) Banach spaces $E, E^\perp$ equipped with a bilinear operator $\langle \cdot, \cdot \rangle$ into $\mathbb{C}$ where the following equations hold:

$$|x| \equiv \sup\{|\langle x, y\rangle|; y \in E^\perp, ||y|| \leq 1\}$$
$$|y| \equiv \sup\{|\langle x, y\rangle|; x \in E^\perp, ||x|| \leq 1\}$$

As noted by Girard, this amounts to requiring that each of the $E, E^\perp$ be isomorphic to a subspace of the dual of the other.

These spaces, together with multilinear maps, can be endowed with enough structure to interpret the usual $\otimes, \parr, +, \&, \cdot^\perp, \multimap$ operators accounting for the corresponding operators in linear logic and coherence spaces. As an interesting note, intuitionistic implication $A \Rightarrow B \equiv !A \multimap B$ happens to coincide with the set of non-linear functions from A to B that are holomorphic on some neighbourhood of 0[1], indicating that exponentials, were they to exist, would allow for non-linearity in the analytic, as well as the logical, sense.

Other geometric models with more structure have been developed, notably Köthe space semantics, due to Thomas Ehrhard. This semantics is based on *Köthe spaces* [61, 34, 35], pairs $X = (|X|, E_X)$ of a countable set $|X|$ of indices and a set E_X of $|X|$-indexed sequences which is *self-dual*. That is to say, E_X satisfies $(E_X^{\perp})^{\perp} = E_X$, where by $(E_X)^{\perp}$ we mean the set F_X of $|X|$-indexed sequences y such that for all $x \in E_X$ the following sum:

$$\sum_{i \in |X|} |x_i y_i|$$

converges absolutely. The E_X that satisfy this condition can be endowed with the structure of a vector space and again in this setting the linear implication $X \multimap Y$ is given by a space of linear functions from E_X to E_Y. Intuitionistic implication, however, corresponds to non-linear, entire (i.e. holomorphic everywhere) functions (as opposed to functions that are holomorphic on some open set). This has the important consequence of compositionality, since the composition of entire functions is guaranteed to be an entire function.

The niceties of the Köthe space semantics already allow a differential structure to arise. In [35, Section 4.1], Ehrhard remarks that given any entire function f between Köthe spaces X, Y, a corresponding arrow $\mathsf{D}f \ : !X \otimes X \to Y$ can be found which is a multilinear map from the Köthe spaces $E_X, E_{!X}$ into E_Y that corresponds to the standard notion of derivative, thus driving the intuitions behind the development of the differential λ-calculus.

Shortly after the advent of the differential λ-calculus, this model would be adapted into the more "discrete" notion of *finiteness spaces* [36]. A finiteness space can be thought of as a generalisation of a coherence space and is given by a pair $X \equiv (I, F)$ where $I = |X|$ is some set and $F(X) = F \subseteq \mathcal{P}(I)$ a family of subsets of $|X|$ called the "finitary" sets (although note that they need not be finite) with the property that $F^{\perp\perp} = F$ (where $F^{\perp}$ denotes the family of subsets of $|X|$ that intersect each element of $F(X)$ in at most a finite number of points).

Morphisms between X and Y in the corresponding category of finiteness spaces are given by relations $f \subseteq X \times Y$ relating finitary sets in X to finitary sets in Y, and

[1]We remind the reader that a complex-valued function is holomorphic on some open set U whenever it admits a complex derivative at every point of U which in particular implies that it is (locally) infinitely differentiable and coincides (locally) with its own Taylor expansion.

finitary sets in $Y^\perp$ into finitary sets in $X^\perp$. Again, under the adequate definitions, this category is a model for multiplicative additive linear logic with exponentials and, similarly to the construction of exponentials in coherence semantics, the exponential $!(X, F)$ is given by $(\mathcal{M}_{fin}(X), F(!X)$, where

$$\begin{aligned} F(!X) &\coloneqq \{u^! \mid u \in F(X)\}^{\perp\perp} \\ u^! &\coloneqq \{\mu \in |!X| \mid |\mu| \subseteq \mu\} \end{aligned}$$

Finiteness spaces also admit a geometric interpretation (namely, given a finiteness space X and a ring R, a module can be constructed $R\langle X\rangle$ of $|X|$-indexed sequences of elements of R whose support is a finitary set). Note the similarity to Köthe spaces, which are given by a set X and some space of X-indexed sequences of complex numbers.

2.2 The Differential λ-Calculus

Motivated by the discovery of this series of strongly "analytically-flavoured" models, Thomas Ehrhard and Laurent Régnier develop the differential λ-calculus [37, 37] to give a computational, syntactic account of these continuous models that also reflects the syntactic notion of linearity as seen in e.g. [32], wherein a *linear substitution* is a substitution that affects only the leftmost occurrence of a given variable. It will be in this sense that the differential operator will produce linear functions.

The linear structure is thus baked into the syntax of the differential λ-calculus: terms are closed under linear combinations with coefficients on some given commutative and unital semiring $\mathbb{K}$. This can be regarded as a generalisation of the λ-calculus with multiplicities [19]. A differential application operator $\mathrm{D}s \cdot t$ is also introduced, representing the derivative of the function s at t (or the application of function s to exactly one copy of t). Terms are divided into classes Λ^s, Λ^d of *simple* and *differential* terms (or simply terms), defined inductively as follows:

$$\begin{array}{llll} \Lambda^s: & s, t & \coloneqq & x \mid \lambda x.s \mid (s\ T) \mid \mathrm{D}(s) \cdot t \\ \Lambda^d: & S, T & \coloneqq & \sum_{i=1}^n a_i s_i \quad \text{where } a_i \in \mathbb{K} \end{array}$$

While the original formulation [37] did not define separate syntactic categories Λ^s, Λ^d and instead relied on a notion of congruence to ensure linearity of the relevant syntactic constructs, later presentations [71, 70] enforce linearity directly on the syntax by defining some constructions as syntactic sugar, e.g.

$$\left(\sum_i a_i s_i\right) T := \sum_i (a_i s_i T)$$
$$\lambda x. \left(\sum_i a_i s_i\right) := \sum_i a_i (\lambda x. s_i)$$
$$\mathrm{D}\left(\sum_i a_i s_i\right) \cdot \left(\sum_j b_j t_j\right) := \sum_{k,j} a_i b_j \mathrm{D} s_i \cdot t_j$$

Importantly, we note that application is linear in the function being applied, but *not* in the argument, as it might be used more than once, whereas differential application is linear in both.

Reduction for the differential λ-calculus is based around a new notion of substitution: the *partial derivative of s with respect to x along u*, denoted by the familiar notation from calculus $\frac{\partial s}{\partial x}(u)$, given by the rules in Figure 2.1. Operationally, the

$$\begin{array}{lcll}
\frac{\partial x}{\partial x}(u) & := & u & \\
\frac{\partial y}{\partial x}(u) & := & 0 & \text{if } x \neq y \\
\frac{\partial (\lambda y.t)}{\partial x}(u) & := & \lambda y. \left(\frac{\partial t}{\partial x}(u)\right) & \text{if } y \notin \mathrm{FV}(t) \\
\frac{\partial (t\ e)}{\partial x}(u) & := & \left[\mathrm{D}(t) \cdot \left(\frac{\partial e}{\partial x}(u)\right)\ e\right] + \left[\frac{\partial t}{\partial x}(u)\ (e)\right] & \\
\frac{\partial (\mathrm{D}(t)\cdot e)}{\partial x}(u) & := & \mathrm{D}(t) \cdot \left(\frac{\partial e}{\partial x}(u)\right) + \mathrm{D}\left(\frac{\partial t}{\partial x}(u)\right) \cdot (e) & \\
\frac{\partial (t+e)}{\partial x}(u) & := & \left(\frac{\partial t}{\partial x}(u)\right) + \left(\frac{\partial e}{\partial x}(u)\right) & \\
\frac{\partial 0}{\partial x}(u) & := & 0 &
\end{array}$$

Figure 2.1: Differential substitution in the differential λ-calculus

differential substitution $\frac{\partial s}{\partial x}(u)$ can be understood as substituting one occurrence of x in s, chosen non-deterministically, by u.

The one-step reduction relation for the differential λ-calculus is then induced by the rules in Figure 2.2 below.

$$\begin{array}{rcl}
(\lambda x.t)\ s & \rightsquigarrow_\beta & t\,[s/x] \\
\mathrm{D}(\lambda x.t) \cdot s & \rightsquigarrow_\partial & \lambda x. \left(\frac{\partial t}{\partial x}(s)\right)
\end{array}$$

Figure 2.2: One-step reduction rules for the differential λ-calculus

In the same work, the authors also study a typed version of the calculus [37, Section 4], by using a straightforward extension of simple types for the λ-calculus. In order to recover some meta-theoretical properties, however, further constraints are required on the underlying semiring R: to prove subject reduction [37, Lemma 22], R must be *positive*, i.e. it must lack any negative elements, otherwise for any type A and (not necessarily well-typed) term t, the sequent $\vdash 0 : A$ is derivable and $0 \to_\beta (t - t)$ but $\vdash (t - t) : A$ may not be provable.

Strong normalization cannot be proven in all generality, either, and Ehrhard and Régnier [37, Theorem 35] prove it only for the case of the natural numbers. Although it is noted that the results can be extended to positive rings without zero divisors as long as every element admits only a finite number of decompositions, Vaux [107] would later show stronger conditions are necessary.

One notable result about the differential λ-calculus is that the usual Taylor expansion from calculus does hold:

$$s\ u = \sum_{n=0}^{\infty} \frac{1}{n!}(\mathrm{D}^n(s) \cdot u^n)0$$

whenever the term $s\ u$ reduces to some distinguished variable $\star$ [37, Theorem 39]. This theorem was later generalised in [39], where it is used to encode the full λ-calculus into a simplified subset of the differential λ-calculus featuring differential application but not ordinary application. While this simplified calculus is not Turing-complete (as every function is linear), adding infinite sums to the language allows one to encode the full λ-calculus in a way which, while considerably more involved, is reminiscing of how arbitrary (analytic) functions can be represented by a power series. We believe these results show that infinite sums of terms may play a pivotal role in interpreting fixpoints.

Further work has been carried out on the differential λ-calculus. On one hand, the theory of proof nets and, more generally, interaction nets [64, 46], graphical formalisms for studying proofs and proof reduction in linear logic, have been extended to the differential case by enriching the structure of usual interaction nets with addition [38] and, more recently, with exponential boxes [104].

Moreover, some work has gone into extending the differential λ-calculus with additional primitives, most notably Lionel Vaux's differential $\lambda\mu$-calculus [107]. This calculus subsumes both the differential λ-calculus and Parigot's μ-calculus [83], an extension of the polymorphic λ-calculus that corresponds to natural deduction for classical (as opposed to intuitionistic) logic. This extension is mostly straightforward, with the exception of the interaction between differential application and μ-abstraction $\mathrm{D}(\mu\alpha.\nu) \cdot t$ which, intuitively, feeds the t linearly to every subterm of ν which is labelled by α.

Vaux's work also revises and extends the results in [37]: some extra notational convenience is provided and some of Ehrhard and Régnier's claims are revisited and expanded upon. Notably, Vaux shows that stronger conditions are necessary for strong normalization [37, Theorem 35] to hold than was believed originally.

2.2.1 Algebraic Calculi

Extensions of the λ-calculus endowed with some form of vector space structure on terms have been studied independently of the differential λ-calculus , particularly, particularly for applications in quantum computation. Despite these calculi being different from the differential λ-calculus , however, a significant amount of useful literature has been written about them from which many insights can be directly applied to the differential λ-calculus, particularly concerning more operational aspects such as termination and confluence.

The first development in this direction was due to Pablo Arrighi and Gilles Dowek [9], who studied the theory of $\mathcal{K}$-vector spaces as rewrite systems and tackled the problem of providing a sound, terminating and confluent rewrite theory, given some reasonable rewrite system for the underlying field $\mathcal{K}$. The main result of this work is [9, Proposition 2.14], which tells us that a certain algebraic structure is a vector space if and only if it is a model of the rewrite system constructed by the authors. Furthermore, this reduction system is terminating ([9, Proposition 2.4]) and confluent in semi-open terms ([9, Proposition 2.10]), that is, terms that do not contain any free variables of scalar sort. This is particularly interesting in a differential context, since the differential λ-calculus forbids syntactically the presence of variables in a scalar context.

Building on this algorithm, Arrighi and Dowek [10] developed an extension of the λ-calculus equipped with a vectorial structure with the intention of providing a model for quantum computation. While this language does not feature any differential constructs, it is far more powerful than the differential λ-calculus when it comes to expressing vectorial operations, including primitives for computing tensor products and dot products, as well as a form of pattern maching. The vectorial aspects of this calculus are, however, limited to boolean values and one cannot take linear combinations of arbitrary terms.

Simpler calculi have later been constructed by Arrighi and Dowek [8] and Vaux [105, 106, 108] that are significantly closer to the differential λ-calculus - indeed, they can be seen as a restriction of the former to terms without differential application. The study of these systems was partly motivated by the reduction issues that arise in the differential λ-calculus if the underlying scalar structure is not "nice" enough. In Ehrhard's original work [37] it is already noted that strong normalization in the differential λ-calculus would fail to hold if coefficients are taken to be the (positive) rational numbers, since the following reduction sequence would be possible (whenever $t \to t'$):

$$t = \frac{1}{2}t + \frac{1}{2}t \to \frac{1}{2}t + \frac{1}{2}t' = \frac{1}{4}t + \frac{1}{4}t + \frac{1}{2}t' \to \frac{1}{4}t + \frac{1}{4}t' + \frac{1}{2}t' = \frac{1}{4}t + \frac{3}{4}t' \ldots$$

Indeed Vaux [107, 105, 106, 108] shows that strict restrictions on the semiring of scalars have to be imposed for the resulting calculus to be strongly normalizing. Particularly, the semiring R of scalars must be *finitely splitting* [106, Theorem 3], i.e. every element $a \in R$ can be expressed as a sum $b_1 + \ldots + b_n$ in only a finite number of ways, although for weak normalization to hold it is only required that R be positive, whereas confluence is shown without any assumptions on the semiring R [106, Theorem 1].

Another relevant issue that has been studied in the context of the algebraic λ-calculus is that of fixpoints. Indeed, it is well known [37, 105] that if the semiring R contains negative elements the reduction system collapses entirely in the untyped case. To see why, consider some fixpoint combinator $\mathbf{Y}$ such that $\mathbf{Y}(\mathbf{f}) = f(\mathbf{Y}(\mathbf{F}))$ and, given some term $\mathbf{s}$, define the following auxiliary combinator:

$$\mathbf{Y_s} \coloneqq \mathbf{Y}(\lambda \mathbf{x}.\mathbf{s} + \mathbf{x})$$

Then $\mathbf{Y_s} \to^* s + \mathbf{Y_s}$ and therefore $\mathbf{s} \equiv \mathbf{s} + \mathbf{Y_s} - \mathbf{Y_s} + \mathbf{Y_t} - \mathbf{Y_t} \to^+ \mathbf{s} - \mathbf{s} + \mathbf{t} = \mathbf{t}$. As this problem only depends on the existence of a fixpoint combinator, it will also arise in any typed extension of the differential λ-calculus equipped with such. This behavior corresponds to the old adage that $\infty - \infty$ does not have a determinate value.

While most work avoids this problem by restricting R to be positive, i.e. $a + b = 0$ implies $a = 0$ and $b = 0$ for any $a, b \in R$, Arrighi and Dowek [8] tackle the issue in a completely different way, by considering a system where algebraic equations such as $at + bt = (a + b)t$ are seen not as an equivalence on terms but rather as rewrite rules in the same style as [9]. Then a series of side conditions are added to the resulting rewrite system so as to eliminate the pathological cases: for example, the term t is required to be in normal form for the rule $at + bt \to (a + b)t$ to apply, which renders the previous pathologic reduction sequence impossible. It is an open question, however, whether a denotational semantics can be obtained for such a calculus and what its models are.

2.3 Cartesian Differential Categories

Since the discovery of the correspondence between the simply-typed λ-calculus and Cartesian closed categories due to Lambek [65], an important area of research within type theory has been the search for categorical models for any new calculi that appear, and the differential λ-calculus would be no different.

Indeed, not long after Ehrhard and Régnier's unveiling of their new calculus, Blute, Cockett and Seely put forth the idea of differential categories [16], i.e. monoidal categories enriched over some commutative monoid (modelling addition of terms) equipped with a coalgebra modality that plays a similar role to the familiar bang modality from linear logic, plus a differential operator D mapping arrows $f : !A \longrightarrow B$ into arrows $D[f] : A \otimes !A \longrightarrow B$. In particular, this means that the hom-sets of a differential category $\mathbf{C}$ are equipped with a commutative operation $+ : \mathbf{C}(A, B) \rightarrow \mathbf{C}(A, B) \rightarrow \mathbf{C}(A, B)$ which admits a neutral element $0 : \mathbf{C}(A, B)$.

This differential operator $\mathsf{D}[\cdot]$ must, of course, satisfy some coherence conditions: it must be linear and natural, but most importantly it must "behave like a derivative", in the sense that the differential satisfies a categorical analogue of the product and chain rules, sends linear maps to themselves and constant maps to the 0 arrow (see [16, Equations D.1-D.4] for details). These conditions are also stated in a graphical setting as equations between proof nets, where the differential is represented as a kind of "box".

In [16], Cockett and Seely also identify various differential categories. As an example, $\mathbf{Vec}_K^{op}$, the dual of the category of K-vector spaces, is a differential category, with the coalgebra modality $!V$ being given by the ring of polynomials $K[X]$ (where X denotes a basis of V). What is more, they vastly generalise this construction: given any commutative semiring R, it is shown that every **polynomial theory** (i.e. a Lawvere theory $\mathbf{T}$ whose constants $\mathbf{T}(0, 1)$ are exactly the elements of R and that includes functions $+, \cdot : \mathbf{T}(2, 1)$ satisfying the equations for a commutative R-algebra, see [16, Section 3.1] for details) induces a coalgebra modality $S_{\mathbf{T}}(V)$ in $\mathbf{Mod}_R^{op}$ given by the space of functions h mapping linear forms $u : V \multimap R$ into scalars in R, such that $h(u)$ corresponds to the action of some arrow $\alpha : \mathbf{T}(n, 1)$ on some finite set of vectors $v_1, \ldots, v_n \in V$. This can be thought of as a more general version of the Fock construction previously used by Blute, Panangaden and Seely to model exponentials [15] (which is obtained when $\mathbf{T}$ is taken to be the theory of polynomials).

Furthermore, if the theory $\mathbf{T}$ admits the existence of some terms behaving like partial derivatives (in a sense that is made precise in [16, Equations pd.1-pd.5]), plus some extra technical conditions (that roughly amount to requiring that independent vectors in each R-module V can be separated by a linear functional), then $\mathbf{Mod}_K^{op}$ with the previously defined coalgebra $S_\mathbf{T}$ is indeed a differential category. This is quite a powerful result, as it allows one to construct differential categories "á la carte".

The notion of differential category was, however, still tied to a monoidal category of linear maps, from which differential operators arose from the interaction between the linear structure and a coalgebra modality, with smooth maps being coKleisli arrows. To give an account of smooth maps directly, Blute, Cockett and Seely introduced the notion of **Cartesian differential categories** [17], or CDCs, which replace the monoidal tensor with Cartesian product.

In a CDC, hom-sets are taken to be formed of smooth, rather than linear, functions. They are also enriched with a monoidal structure, but Cartesian differential categories are *not* additive (which would entail that every arrow is "linear"), but only left-additive.

Definition 2.3.1. A **left-additive category** [17, Definition 1.1.1.] is a category $\mathbf{C}$ together with a choice of a commutative monoid structure $(\mathbf{C}(A, B), +, 0)$ on each hom-set $\mathbf{C}(A, B)$, subject to the constraint that pre-composition (but *not* post-composition) respects the additive structure. That is to say, for all f, g, h:

$$\begin{aligned}(f + g) \circ h &= (f \circ h) + (g \circ h)\\ 0 \circ h &= 0\end{aligned}$$

A certain map $f : A \to B$ in a left additive category is **additive** whenever post-composition by f preserves the additive structure. That is to say, for all g:

$$\begin{aligned}f \circ (g + h) &= (f \circ g) + (f \circ h)\\ f \circ 0 &= 0\end{aligned}$$

A Cartesian differential category is also equipped with a differential operator, which mimics the behaviour of the differential of a smooth function in multi-variate calculus.

Definition 2.3.2. A **Cartesian differential category** [17, Definition 2.1.1.] is a Cartesian left-additive category $\mathbf{C}$ equipped with a **differential combinator** D of the form[2]:

$$\frac{f : A \to B}{\mathsf{D}[f] : A \times A \to B}$$

verifying the following coherence conditions:

[**CDC.1**] $\mathsf{D}[f + g] = \mathsf{D}[f] + \mathsf{D}[g]$ and $\mathsf{D}[0] = 0$

[**CDC.2**] $\mathsf{D}[f] \circ \langle x, u + v\rangle = \mathsf{D}[f] \circ \langle x, u\rangle + \mathsf{D}[f] \circ \langle x, v\rangle$ and $\mathsf{D}[f] \circ \langle x, 0\rangle = 0$

[**CDC.3**] $\mathsf{D}[\mathbf{id}_A] = \pi_2$ and $\mathsf{D}[\pi_1] = \pi_1 \circ \pi_2$ and $\mathsf{D}[\pi_2] = \pi_2 \circ \pi_2$

[**CDC.4**] $\mathsf{D}[\langle f, g\rangle] = \langle \mathsf{D}[f], \mathsf{D}[g]\rangle$ and $\mathsf{D}[!_A] = !_{A\times A}$

[**CDC.5**] $\mathsf{D}[g \circ f] = \mathsf{D}[g] \circ \langle f \circ \pi_1, \mathsf{D}[f]\rangle$

[**CDC.6**] $\mathsf{D}\left[\mathsf{D}[f]\right] \circ \langle\langle x, u\rangle, \langle 0, v\rangle\rangle = \mathsf{D}[f] \circ \langle x, v\rangle$

[**CDC.7**] $\mathsf{D}\left[\mathsf{D}[f]\right] \circ \langle\langle x, u\rangle, \langle v, 0\rangle\rangle = \mathsf{D}\left[\mathsf{D}[f]\right] \circ \langle\langle x, v\rangle, \langle u, 0\rangle\rangle$

We highlight that by [25, Proposition 4.2], axioms [**CDC.6**] and [**CDC.7**] have an equivalent alternative expression.

Lemma 2.3.1. In the presence of the other axioms, [**CDC.6**] and [**CDC.7**] are equivalent to:

[**CDC.6(a)**] $\mathsf{D}\left[\mathsf{D}[f]\right] \circ \langle\langle x, 0\rangle, \langle 0, w\rangle\rangle = \mathsf{D}[f]\,\langle x, w\rangle$

[**CDC.7(a)**] $\mathsf{D}\left[\mathsf{D}[f]\right] \circ \langle\langle x, u\rangle, \langle v, w\rangle\rangle\rangle = \mathsf{D}\left[\mathsf{D}[f]\right] \circ \langle\langle x, u\rangle, \langle v, w\rangle\rangle$

The above axioms are somewhat abstruse and so we give an informal explanation. [**CDC.1**] states the sum rule for the operator D. [**CDC.2**] states that every derivative is additive in its second argument. [**CDC.3**] states that the identity function and projections are "linear", that is, their derivative is equal to the function itself. [**CDC.4**] states that the derivative of a vector-valued map is the vector of the derivatives of each component, and [**CDC.5**] is simply the chain rule. Perhaps the least intuitive of the lot, [**CDC.6**] can be understood as saying that every derivative is "linear" in its second argument, not in the sense that it commutes with the sum (as

[2]We might expect the type of $\mathsf{D}[f]$ to have the form $A \times \mathsf{T}_0 A \to B$. The theory of differential categories assumes that every space is "Euclidean", in the sense that it is equal to its tangent space at any point. Note also that we flip the convention found in [17], so that the linear argument is in the second argument, as usual in calculus, rather than in the first argument.

[CDC.2] states), but in the sense that taking the partial derivative with respect to its second argument is the function itself. Finally, **[CDC.7]** is an abstract formulation of Schwarz's theorem, which states that the matrix of second derivatives of any smooth function is symmetric.

We refer the reader to [17, Section 4] for further explanations and an exposition of a term calculus which may help better understand the axioms of a Cartesian differential category.

Intuitively it might seem like a CDC is much like a differential category where we forget about the difference between linear and non-linear functions. Indeed this intuition can be made precise: it turns out that the coKleisli category of a differential storage category [16] is a Cartesian differential category [17, Proposition 2.3.1], thus providing us with a simple way of constructing CDCs.

The canonical example of a Cartesian differential category is the category of real smooth functions, which we will discuss in more detail in Section 6.2.1. Other interesting examples can be found throughout the literature such as the subcategory of differential objects of a tangent category [25].

The notion of a Cartesian differential category has more recently been generalised by Cruttwell [30], based on the observation that, while a differential operator as usually defined maps arrows $f : A \to B$ to arrows $\mathsf{D}[f] : A \times A \to B$, the two intstances of A in the codomain of $\mathsf{D}[f]$ play radically different roles, as one usually talks about the derivative of a function at a point and along a certain vector. It so happens that in many common cases, as in functions from $\mathbf{R}^m$ to $\mathbf{R}^n$, both points and vectors are elements of the same vector space (as the usual Euclidean spaces $\mathbf{R}^m$ are their own tangent spaces), but this need not be the case in general.

For this reason, Cruttwell proposes to relax the definition of Cartesian differential category: rather than requiring it to be a left-additive category, the monoidal structure is restricted only to certain "spaces of vectors". A **generalised Cartesian differential category**, or GCDC, is then a Cartesian category $\mathbf{C}$ equipped with a commutative monoid $L(X)$ for each X (intuitively corresponding to the tangent space of X) which is involutive (i.e. $L(L(X)) = L(X)$) and Cartesian ($L(A \times B) = L(A) \times L(B)$). In addition, a generalised Cartesian differential category is equipped with a differential operator D satisfying obvious analogues to **[CDC.1-7]**, with the difference that, if a map f has type $f : A \to B$ then its derivative has type $\mathsf{D}[f] : L(A) \times A \to B$.

Definition 2.3.3. A **generalised Cartesian differential category** [30, Definition 2.1.] is a Cartesian category $\mathbf{C}$ such that:

i. For each object A of $\mathbf{C}$, a choice of a commutative monoid $L(A) = (L_0(A), +_A, 0_A)$ internal to $\mathbf{C}$, satisfying

$$L(L_0(A)) = L(A) \qquad L(A \times B) = L(A) \times L(B)$$

ii. For each map $f : A \to B$ in $\mathbf{C}$, a map $\mathsf{D}[f] : A \times L_0(A) \to L_0(B)$ satisfying axioms [**CDC.1-7**].

An immediate consequence of the previous definition is that any CDC is trivially a GCDC, with $L(X) = X$. Not only that, but important categories that are not CDCs can be seen as generalised Cartesian differential categories (an example being the category formed by finitely-dimensional differentiable manifolds and smooth functions, which is not Cartesian differential since an arbitrary differentiable manifold is not its own tangent space). A term calculus for generalised Cartesian differential categories is not yet known, and we believe it to be an important open problem and a first step in addressing some technical problems in certain calculi [73].

While Blute, Cockett and Seely already provided a term calculus which is sound and complete with respect to Cartesian differential categories [17, Section 4.1], the full power of Ehrhard's original differential λ-calculus requires more structure: in particular, exponentials need to be added to the Cartesian differential structure if one is to model differential λ-terms. To this end, Bucciarelli, Ehrhard and Manzonetto developed **differential λ-categories** [22, 71], which are essentially Cartesian closed differential categories where the exponential is well-behaved with respect to the differential structure.

Definition 2.3.4. A Cartesian left-additive category $\mathbf{C}$ is **Cartesian closed left-additive** whenever it is Cartesian closed and verifies the following conditions:

i. $\Lambda(f + g) = \Lambda(f) + \Lambda(g)$

ii. $\Lambda(0) = 0$

A **differential λ-category** [22, Definition 4.4.] is a Cartesian closed left-additive category that verifies the following additional axiom:

[**DλC.1**] $\mathsf{D}[\Lambda(f)] = \Lambda(\mathsf{D}[f] \circ \langle \pi_1 \times \mathbf{id}, \pi_2 \times 0 \rangle)$

The above amounts to saying that taking the partial derivative of an arrow with respect to its first argument is equivalent to currying the arrow, then taking the derivative with respect to its only argument. It can then be shown that differential λ-categories are sound with respect to the differential λ-calculus [22]. Lately, a completeness result has been proven by Mak [70].

More recent work by Manzyuk [74] and Cockett and Cruttwell [25] has also unearthed a deep connection between Cartesian differential categories and the so-called categories with tangent structure, initially developed by Rosický [87] in order to give an abstract characterization of the tangent bundle construction familiar from differential geometry. Essentially, a **category with tangent structure** is a category $\mathbf{C}$ equipped with some endofunctor $\mathrm{T} : \mathbf{C} \to \mathbf{C}$ equipped with a projection $p_A : \mathrm{T}A \to A$ and an additive structure, which, roughly speaking, amounts to natural transformations:

$$+_A : \mathrm{T}A \times_A \mathrm{T}A \Rightarrow \mathrm{T}A$$
$$0_A : A \Rightarrow \mathrm{T}A$$

where $\mathrm{T}A \times_A \mathrm{T}A$ denotes the pullback of the projection p_A along itself (we use the pullback along p_A and not the product because vectors of the tangent space at some point x can be added together, but they cannot be added to vectors of the tangent space of some other point y). Some extra conditions must hold, but we omit them here, instead referring the reader to [25, Definition 2.3]. A **Cartesian tangent category** is precisely a Cartesian category with tangent structure which preserves products, i.e. $T(A \times B) \cong T(A) \times T(B)$ and the corresponding natural isomorphism α preserves the additive structure of T.

Most importantly, one can also show that every Cartesian differential category induces a Cartesian tangent category via the **tangent bundle** functor $TA \coloneqq A \times A$ acting on arrows by $Tf \coloneqq \langle \mathrm{D}f, f \circ \pi_1 \rangle$ [25, Proposition 4.7]. Furthermore, given a Cartesian tangent category $\mathbf{C}$, the full subcategory of differential objects (objects A whose tangent structure is given by the product $A \times A$, see [25, Definition 4.8]) is in fact a Cartesian differential category. These transformations define a pair of adjoint functors between the category of Cartesian differential categories and the category of Cartesian tangent categories, thus making precise the connection between differential categories and categories with tangent structure.

This connection is further explored in Cockett and Cruttwell's recent work [26], via the more general notion of **differential bundle**. It is then shown that a Cartesian tangent category is itself Cartesian differential if and only if one can choose a differential bundle for each object in a coherent way [26, Lemma 3.13, Theorem 3.14].

An area where we believe further work is neccesary is generalising all these helpful results on Cartesian differential categories to GCDCs or to differential λ-categories.

2.3.1 Convenient Models

The theory of differential categories provides a solid categorical foundation for the semantics of differential linear logic, with the differential λ-calculus : terms are interpreted as arrows in some differential λ category $\mathbf{C}$, with differential application $\mathrm{D}t \cdot u$ corresponding to the differential operator in $\mathbf{C}$. The question may arise, however, as to whether there are differential λ-categories whose objects are some family of vector spaces and whose arrows are smooth functions in some geometric sense, especially since the examples presented when the concept was first introduced [22] (corresponding to the relational and finiteness space semantics) are strongly combinatorial in nature.

This question was answered affirmatively by Blute, Ehrhard and Tasson [14], when they showed that so-called **convenient vector spaces** [41, 62], together with linear maps form a differential category which **Con** which is also symmetric monoidal closed. Furthermore, an exponential modality ! can be given in **Con** where $!E$ is given as the Mackey closure (a generalisation of the notion of Cauchy completion) of the linear span of the Dirac delta distributions $\delta(v)$ for $v \in E$ (where $\delta(v)$ sends every smooth curve $f : E \to \mathbb{R}$ to $f(v)$). The coKleisli category for the comonad $!E$ then happens to be precisely $\mathcal{C}^\infty$, the category of convenient vector spaces and smooth functions, and thus, by [17, Proposition 2.3.1], it follows that $\mathcal{C}^\infty$ is, in fact, a Cartesian differential category.

This approach has been later applied by Kerjean and Tasson [58] to **Mackey-complete vector spaces** [62], which slightly extend convenient vector spaces by not requiring a bornological structure as convenient vector spaces do. These vector spaces together with linear maps form a symmetric monoidal closed category **Lin** which is also Cartesian (but not Cartesian closed). The authors then show that Mackey-complete vector spaces with power series form a λ-category [22] **Series** which arises from an exponential modality on **Lin** constructed just as in [14]. The category **Series** constitutes, thus, a reasonably "geometric" concrete model of the differential λ-calculus.

2.4 Incremental Computation With Derivatives

Computational applications are rarely one-shot. As much as functional programmers praise immutable data structres, change is the only constant in the real world, and so it is often the case that one needs to repeat some calculation on an input that changes over time, be it because the corresponding data has actually mutated or simply because the output of our algorithm prompts us to re-evaluate the computation.

We have already mentioned in the introduction the usefulness of applying incremental computation to graph algorithms such as PageRank. A similar case, and the one we will focus on, is the evaluation of Datalog queries. In this field, it has long been known that some degree of incrementalisation results in asymptotic performance gains. This procedure is so widespread that it is usually known as semi-naive evaluation (the implication being that such a trick is so basic and essential that it is only one step above "naive" evaluation). In Chapter 4 we will give a more detailed explanation of semi-naive evaluation, but we refer the interested reader to one of the standard textbooks [1, Chapter 13.1].

While incremental computing is pervasive [86], there is very little material on an underlying, unifying theory, with most of the application-agnostic work focusing on general-purpose implementations like [24] or [3], instead of its mathematical underpinnings.

It is partly to address this lack that recent work by Cai and Giarrusso [23, 43] introduced the notion of *change structures* and the *incremental λ-calculus*. While novel, these concepts have already found applications to the incrementalisation of Datalog [6] and Datafun [7] programs.

Definition 2.4.1. A **change structure** [23, Definition 2.1] is a tuple $\widehat{V} = (V, \Delta, \oplus, \ominus)$ such that:

i. V is a set.

ii. $\Delta : V \to \mathsf{Set}$ is a V-indexed family of sets. We write Δ_v for the set associated to the element $v \in V$.

iii. $\oplus, \ominus$ are dependently-typed functions:

$$\oplus : \forall (v : V),\ \Delta_v \to V$$
$$\ominus : V \to \forall (v : V),\ \Delta_v$$

iv. For all u, v, we have $u \oplus (v \ominus u) = u$.

Intuitively, the set Δ_v is to be understood as a space of possible "changes" or "updates" that can be applied to some value v, with $v \oplus \delta_v$ denoting the result of updating the value v with the change δ_v. A crucial assumption of Cai and Giarrusso's model is that, given any two values u, v, there will necessarily be a change $\delta_u \in \Delta_u$ satisfying $u \oplus \delta_u = v$. Or, more concisely, every value can be updated into any other value. The map $\ominus$ is responsible for selecting such a change (although in general there may be more than one change with the desired property).

The challenge of incremental computation can then be framed in the language of change structures as the problem of propagating changes from the input to the output of a function.

Cai and Giarrusso go on to give concrete change structures on some sets. Notably, they give an explicit construction for a change structure on the function space $A \Rightarrow B$, given change structures on A and B respectively. More importantly, they give an algorithm for incrementalising functions between change structures by "syntactic differentiation". Although they do not frame it in such terms, this algorithm amounts to an iterated application of the chain rule from standard calculus (see the the definition of *Derive* in [23, Figure 4])

2.5 Automatic Differentiation

An interesting area where the differential λ-calculus has tentatively been applied is automatic differentiation. Automatic differentiation, or AD for short, is a family of methods that allow for the efficient computation of the derivative of arbitrary programs operating on numeric values. Given some program $f : \mathbb{R} \to \mathbb{R}$ and some point $x_0 \in \mathbb{R}$, an AD algorithm will compute the value of the derivative $\frac{\partial}{\partial x} f$ evaluated at point x_0.

While an in-depth exposition of these techniques is outside the scope of this work (we refer to [49] for a more complete treatment of the topic), the fundamental intuition driving them is that such a program f must necessarily be a composition of sums, products and some primitive functions (**sin**, **cos**, **log**, ...) whose derivatives are already known. The derivative of f can then be computed by a repeated application of the chain rule, which states that:

$$\frac{\partial (g \circ f)(x)}{\partial x}\bigg|_{x=x_0} = \frac{\partial g(y)}{\partial y}\bigg|_{y=f(x_0)} \times \frac{\partial f(x)}{\partial x}\bigg|_{x_0}$$

The chain rule is applied iteratively until the derivative of f is expressed in terms of the derivatives of elementary functions, which are already known and can be easily evaluated. This process is quite efficient and can be performed in linear time with respect to the complexity of the input program.

Automatic differentiation is significantly more precise than numeric differentiation, as it allows for completely accurate computations (up to the precision of the underlying numeric types). It is also more efficient than approaches based on symbolic differentiation as the derivatives computed by symbolic differentiation are, in general, exponentially larger than the input.

One may observe that the expansion of the expression $\frac{\partial}{\partial x} f$ into partial derivatives can be computed in more than one order. For example:

$$\begin{aligned}
\frac{\partial y}{\partial x} &= \frac{\partial y}{\partial w_1}\frac{\partial w_1}{\partial x} = \frac{\partial y}{\partial w_1}\left(\frac{\partial w_1}{\partial w_2}\frac{\partial w_2}{\partial w_3}\right) \\
\frac{\partial y}{\partial x} &= \frac{\partial y}{\partial w_1}\frac{\partial w_1}{\partial x} = \left(\frac{\partial y}{\partial w_1}\frac{\partial w_1}{\partial w_2}\right)\frac{\partial w_2}{\partial w_3}
\end{aligned}$$

Likewise, automatic differentiation can be carried in a number of semantically equivalent (but operationally different) ways. The first evaluation order leads to what is known as forward-mode automatic differentiation, which works by computing the derivatives of successively larger subexpressions w_i in terms of the input variable x. The second evaluation order, known as reverse-mode automatic differentiation, proceeds backwards by finding the derivative of the output y in terms of derivatives of successively smaller subexpressions w_j. The difference between these methods lies in their complexity: if $f : \mathbb{R}^n \to \mathbb{R}^m$, forward-mode AD will take time proportional to n whereas reverse-mode AD will take time proportional to m.

A common formulation of forward-mode AD is in terms of **dual numbers**. The dual numbers are essentially an extension of the real numbers with an element ε with the property that $\varepsilon \times \varepsilon = 0$. The derivative of the polynomial function $f(x) = x^2 + 3x + 1$ at point 2 can then be computed by evaluating $f(x)$ at $2 + \varepsilon$:

$$\begin{aligned}
f(1+\varepsilon) &= (2+\varepsilon)(2+\varepsilon) + 3(2+\varepsilon) + 1 \\
&= 4 + 4\varepsilon + \varepsilon^2 + 3\varepsilon + 7 \\
&= 11 + 7\varepsilon
\end{aligned}$$

Note that $11 = f(2)$ and $7 = f'(2)$. In general, if $f(x)$ is a polynomial, its derivative at x_0 is given by $\mathcal{E}(f(x_0 + \varepsilon))$, where $\mathcal{E}(x + y\varepsilon) \coloneqq y$ (the term ε is usually called a *perturbation* and understood as some kind of infinitesimal). This approach can be extended to more functions by defining the action of each extra primitive on dual numbers.

While it seems quite natural to talk about AD using a functional language (after all, the derivative is a textbook example of a higher-order function), this approach runs into some subtle issues as soon as we consider the derivative $\mathcal{D}(f)(x) \coloneqq \mathcal{E}(x+\varepsilon)$ as a first-class operator, which have been discussed by Siskind and Pearlmutter [98]. The main problem deals with the confusion of different perturbations corresponding to different (but nested) calls of a differential operator: if $\mathcal{D}(f)(x) \coloneqq \mathcal{E}(f(x+\varepsilon))$, then $\mathcal{D}(\lambda x.x \times (\mathcal{D}(\lambda y.x)2))1 = 1$, whereas we would expect this value to be 0.

These issues are solved by Siskind and Pearlmutter in their system Stalin∇ [96], which efficiently compiles a dialect of Scheme code with a derivative operator $\mathcal{D}$ into performant code. The machinery involved is, however, rather complicated and requires a significant amount of introspection in the shape of the `map-closure` operator introduced by the authors in [97]. Furthermore, as noted by Manzyuk [73], they present no proof that Stalin∇ correctly handles derivatives of higher-order functions.

It is in an attempt to provide a solid basis for the theory of higher-order automatic differentiation that Oleksandr Manzyuk defines the perturbative λ-calculus [73], a language closely resembling the differential λ-calculus but based on categories with tangent structure, rather than differential λ-categories. In the perturbative λ-calculus, the derivative of a function is computed by a *pushforward* operator that, given some function $f : A \to B$, produces a function $Tf : A \times A \to B \times B$ which is precisely the functorial action of the tangent bundle functor on some differential λ-category.

While Manzyuk's calculus is sound with respect to differential λ-categories, and thus theoretically sound, it suffers from the flaw of being non-confluent [72]. We believe Manzyuk's general idea to be solid, but the flaws in his calculus must be addressed before any further work can take place. If this can be done, however, two interesting directions for further research await: extending the calculus with general recursion to give a more practical perspective, and providing a similar foundation for reverse-mode automatic differentiaiton.

This last development would be particularly exciting: reverse-mode AD subsumes backpropagation [84], which is the most common method of training a neural network [88]. A higher-order calculus featuring AD could then constitute a basis for the efficient, modular development of neural networks, as it has been postulated by researchers in the field [31, 81].

On the other hand, Cai and Giarrusso's theory of change structures has recently been suggested by Kelly, Pearlmutter and Siskind [57] as a candidate for both a semantic setting in which to reason about higher-order automatic differentiation and an executable calculus that can be of practical use in end-to-end differentiable programming. Kelly et al. contend that it should be possible to understand the sets of changes Δ_v as power series on some distinguished variable ε, which is then truncated into an infinitesimal perturbation by letting $\varepsilon^2 = 0$.

From a geometric perspective, when interpreted in this way a change structure becomes a representation for a geometric space, with Δ_v being an analogue for the tangent space at a point v, and the syntactic *Derive* transformation from [23, Figure 4] is the chain rule in meaning as well as in form. The operator $\ominus$ is the only obstacle to this reading of change structures – given two points u, v in a manifold[3], it may well be the case that there is no infinitesimal change that can transform u into v, and so there is no candidate for $v \ominus u$.

[3]A manifold in a setting that admits infinitesimal elements, such as synthetic differential geometry[59, 60].

Chapter 3

Change actions

In this chapter we introduce the central notion of this thesis: that of a change action. This definition was originally proposed as an attempt to formalise incremental computation of higher order programs based on Yufei Cai and Paolo Giarrusso's notion of a *change structure*, while addressing some of the issues with their original formulation.

This chapter is divided in three parts. In Section 3.1 we introduce the basic definitions of change action and derivative of a function for the case when the underlying category is **Set**. We propose two different categories of change actions, one where morphisms are maps for which a derivative exist and another where maps are pairs of a function together with a derivative for it and we show ways to construct change actions that give rise to a series of adjunctions between the category of sets and the category of change actions.

In Section 3.2 we generalise the previous definition of a category of change actions to a functor that maps an arbitrary Cartesian category to a corresponding category of change actions internal to it. We show that this category has products and coproducts and we discuss some important subcategories.

Finally, Section 3.3 shows that change actions can be understood entirely as a particularly well-behaved sort of internal categories. Differential maps correspond precisely to functors, thus justifying and motivating our definition.

3.1 Change Actions in Set

In the simplest possible terms, a change action is merely a monoid acting on a set. This is a rather unimpressive definition - indeed, as we shall see, the main interest of change actions is not the change actions themselves but the associated idea of differentiable maps.

Definition 3.1.1. A **change action** is a tuple $\overline{A} = (A, \Delta A, \oplus_A, +_A, 0_A)$ where A and ΔA are sets, $(\Delta A, +_A, 0_A)$ is a monoid, and $\oplus_A : A \times \Delta A \to A$ is an action of the monoid on A.

We omit the subscript from $\oplus_A, +_A$ and 0_A whenever it can be deduced from the context. We will often refer to the set ΔA as the **change set** of $\overline{A}$, and to the set A as its **base set** or **underlying set**.

Remark 3.1.1. Change actions are closely related to the notion of *change structures* introduced in [23] but differ from the latter in neither being dependently typed nor assuming the existence of an $\ominus$ operator. The importance of this will become apparent later in Chapter 5 when we introduce important families of change actions where no such operators exist (for example, this is true of change actions derived from differential categories and Kleene algebras, see Sections 5.2.1 and 5.2.3).

On the other hand, change actions require the change set ΔA to have a monoid structure compatible with the map $\oplus$, hence neither notion is strictly a generalisation of the other. Whenever one has a change structure, however, one can easily obtain a change action by taking ΔA to be the free monoid generated by the set of changes of the original change structure.

Even if it is straightforward, some intuition for the above definition is in order. One should understand the set ΔA as a set of *changes* that can be applied to objects of type A. For example, the base space could be the set of all text files, with elements of ΔA being *diffs*. Or A could be the space of all database configurations, with elements of ΔA representing possible updates to the database. In Chapter 4 we will work with a change action where elements of A are sets of facts and changes $\delta \in \Delta A$ are sets of new facts to be added and old facts to be discarded.

The connection with incremental computation is evident: the problem of incrementalising some function $f : A \to B$, assuming change actions $\overline{A}$ and $\overline{B}$, we can formalise the problem of incrementalising f by assuming an input x_i that results from the application of successive changes $\delta x_i \in \Delta A$ to an initial value x_0. An incremental version of f would then somehow transform these into changes to the output $\delta y_i \in \Delta B$ in such a way that $f(x_{i+1})$ can be obtained by updating $f(x_i)$ with δy_i, as shown in Figure 3.1.

A change action can also be understood in a more geometric way, by thinking of a change δa as a directed path (or, more properly, a family of paths, one for every base point) in A, with the point $a \oplus_A \delta a$ corresponding to the end-point of the path of δa starting at point a. Representing elements of a as points and changes as arrows,

$$\begin{tikzcd}
x_i \arrow[r, "f"] \arrow[dd, dashed, bend right] & y_i \arrow[dd, dashed, bend left] \\
\delta x_i \arrow[r, Rightarrow, "\partial f"] & \delta y_i \\
x_{i+1} \arrow[r, "f"] & y_{i+1}
\end{tikzcd}$$

Figure 3.1: Incremental computation in the abstract

we can picture the condition that $\oplus_A$ be an action by the "commutative diagram" below[1]:

$$\begin{tikzcd}
a \arrow[r, bend left, "\delta_1"] \arrow[dr, bend right, "\delta_1 +_A \delta_2"'] & a \oplus_A \delta_1 \arrow[d, bend left, "\delta_2"] \\
& a \oplus_A \delta_1 \oplus_A \delta_2
\end{tikzcd}$$

Figure 3.2: The action property of $\oplus_A$

Between any two change actions, there is an obvious notion of "structure-preserving" map which is just that of a homomorphism of the corresponding multi-sorted algebras. That is to say, a homomorphism from $\overline{A}$ to $\overline{B}$ would be a function $f : A \to B$ and a monoid homomorphism $\Delta f : \Delta A \to \Delta B$ with the property that $f(x \oplus \delta x) = f(x) \oplus \Delta f(\delta x)$.

It is a natural question whether such a Δf gives us the right notion for an "incremental" version of f – but it turns out to be much too stringent, as it requires that the "incremental" version of f, Δf, does not depend on the current "state" x_i but merely on the incoming update δx_i. In practice this does not seem to be the case (see Chapter 4 for a relevant counterexample).

Instead of working with homomorphisms of change actions, we will work with a much weaker notion, that of a *derivative*, and the associated notion of a *differentiable* function.

[1]Diagrams of this sort are of immense help when reasoning about change actions – as we shall see, their use is justified by the double nature of change actions as categories.

Definition 3.1.2. Let $\overline{A}$ and $\overline{B}$ be change actions. A function $f : A \to B$ is **differentiable** if there is a function $\partial f : A \times \Delta A \to \Delta B$ satisfying $f(a \oplus_A \delta a) = f(a) \oplus_B \partial f(a, \delta a)$, for all $a \in A, \delta a \in \Delta A$. We call ∂f a **derivative** for f, and write $f : \overline{A} \to \overline{B}$ whenever f is differentiable.

The nomenclature may seem overly ambitious, but it is nothing new: the Datalog literature already discusses derivatives of programs for semi-naive evaluation [13, 12], a kind of incremental computation which we will explore in depth in Chapter 4, and this was also the name chosen by Cai and Giarrusso in their work on change structures [23]. We will later provide more instances which show that the name "derivative" is warranted.

The most immediate (and most prominent) parallel with derivatives as they are defined in calculus or differential geometry, and a vital result about change actions, is that derivatives compose via the familiar chain rule.

Theorem 3.1.1 (Chain rule)**.** Given differentiable functions $f : \overline{A} \to \overline{B}$ and $g : \overline{B} \to \overline{C}$ with derivatives ∂f and ∂g respectively, the function $\partial(g \circ f) : A \times \Delta A \to \Delta C$ defined by $\partial(g \circ f)(a, \delta a) := \partial g(f(a), \partial f(a, \delta a))$ is a derivative for $g \circ f : A \to C$.

Proof. Unpacking the definition, we have

$$\begin{aligned}
(g \circ f)(a) \oplus_C \partial(g \circ f)(a, \delta a) &= g(f(a)) \oplus_C \partial g(f(a), \partial f(a, \delta a)) \\
&= g(f(a) \oplus_B \partial f(a, \delta a)) \\
&= g(f(a \oplus_A \delta a))
\end{aligned}$$ □

Example 3.1.1. Whenever $(A, +, 0)$ is a monoid, the action of this monoid on itself defines a change action $(A, A, +, +, 0)$ which we call the **monoidal** action.

Of particular importance is the case when the underlying monoid is the set of natural numbers with addition. We denote this change action by $\overline{\mathbb{N}} := (\mathbb{N}, \mathbb{N}, +, +, 0)$. A function $f : \mathbb{N} \to \mathbb{N}$ is differentiable according to $\overline{\mathbb{N}}$ if and only if it is monotone.

When the monoid $(A, +, 0)$ is in fact a group, every function $f : B \to A$ is differentiable, with exactly one derivative given by $\partial f(x, u) := -f(x) + f(x \oplus_B u)$.

Example 3.1.2. For any set A, there is a **discrete** change action $\overline{A}_\star := (A, \{\star\}, \pi_1, \pi_1, \star)$ whose change set ΔA is the one-element set $\{\star\}$ and whose action "does nothing".

Given any other change action $\overline{B}$, every function $f : A \to B$ is differentiable as a function from $\overline{A}_\star$ into $\overline{B}$ with derivative $\partial f(a, \star) := 0_B$, which trivially satisfies the derivative condition.

Example 3.1.3. Let $A \Rightarrow B$ denote the set of functions from A from B, and $\mathbf{ev}_{A,B} : A \times (A \Rightarrow B) \to B$ be the usual evaluation map. Then $\overline{A}_{\Rightarrow} := (A, A \Rightarrow A, \mathbf{ev}_{A,A}, \circ, \mathbf{id}_A)$ is a change action.

More generally, whenever a family of functions $U \subseteq (A \Rightarrow A)$ contains the identity and is closed under composition, $(A, U, \mathbf{ev}_{A,A} \upharpoonright_{A \times U}, \circ \upharpoonright_{U \times U}, \mathbf{id}_U)$ is a change action.

The derivative condition can also be represented graphically: a differentiable function is precisely one that, fixing a point of origin a, preserves "paths" (i.e. changes) that start at a.

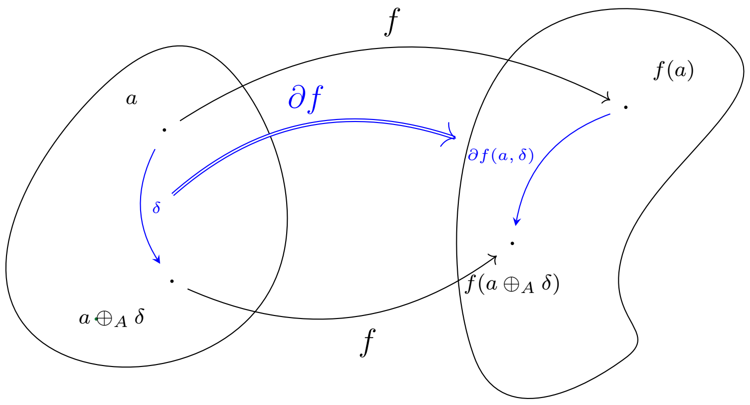


Figure 3.3: A derivative acting on changes

3.1.1 Regular derivatives

The preceding definitions neither assume nor guarantee a derivative to be additive (i.e. they may not satisfy $\partial f(x, \Delta a + \Delta b) = \partial f(x, \Delta a) + \partial f(x, \Delta b)$), as they are in standard differential calculus. A strictly weaker condition that we will now require is *regularity*: if a derivative is additive in its second argument then it is regular, but not vice versa. As we shall show later, the converse is true under certain conditions.

Definition 3.1.3. Given a differentiable map $f : \overline{A} \to \overline{B}$, a derivative ∂f for f is **regular** if, for all $a \in A$ and $\delta a, \delta b \in \Delta A$, we have:

$$\partial f(a, 0) = 0 \tag{3.1}$$

$$\partial f(a, \delta a + \delta b) = \partial f(a, \delta a) + \partial f(a \oplus \delta a, \delta b) \tag{3.2}$$

While we will often just require the regularity conditions to hold, one can show that they follow from moderately weak premises. Indeed, it should be remarked that it is always possible to "pretend" that a derivative is regular. That is to say:

$$\begin{aligned} f(a) \oplus [\partial f(a,\delta_1)a + \partial f(a \oplus \delta_1, \delta_2)] &= f(a) \oplus \partial f(a,\delta_1) \oplus \partial f(a \oplus \delta_1, \delta_2) \\ &= f(a \oplus \delta_1) \oplus \partial f(a, \oplus\delta_1, \delta_2) \\ &= f(a \oplus \delta_1 \oplus \delta_2) \\ &= f(a \oplus (\delta_1 + \delta_2)) \\ &= f(a) \oplus \partial f(a, \delta_1 + \delta_2) \end{aligned}$$

That is to say, $\partial f(a,\delta_1)+\partial f(a\oplus\delta_1,\delta_2)$ has the same effect on $f(a)$ as $\partial f(a,\delta_1+\delta_2)$. In any context where derivatives are unique, then, it is provable that derivatives are necessarily regular. It turns out there is a simple criterion to determine when this is the case.

Proposition 3.1.1. If $f : \overline{A} \to \overline{B}$ is differentiable and has a unique derivative ∂f then this derivative is regular.

Proof. When A is empty, the property follows trivially. Otherwise, if $a \in A$ and $\delta a, \delta b \in \Delta A$, we have:

$$\begin{aligned} f(a \oplus (\delta a + \delta b)) &= f(a \oplus \delta a \oplus \delta b) \\ &= f(a \oplus \delta a) \oplus \partial f(a \oplus \delta a, \delta b) \\ &= \big(f(a) \oplus \partial f(a, \delta a)\big) \oplus \partial f(a \oplus \delta a, \delta b) \\ &= f(a) \oplus \big(\partial f(a,\delta a) + \partial f(a \oplus \delta a, \delta b)\big) \end{aligned}$$

Thus we can define the following derivative for f

$$\partial f_a(x, \delta x) := \begin{cases} \partial f(a,\delta a) + \partial f(a \oplus \delta a, \delta b) & \text{when } x = a, \delta x = \delta a + \delta b \\ \partial f(x, \delta x) & \text{otherwise} \end{cases}$$

Since the derivative is unique, it must be the case that $\partial f = \partial f_a$ and therefore $\partial f(a, \delta a + \delta b) = \partial f(a, \delta a) + \partial f(a \oplus \delta a, \delta b)$. By a similar argument, $\partial f(a, 0) = 0$ and thus ∂f is regular. ☐

Remark 3.1.2. One may wonder whether every differentiable function admits a regular derivative; the answer is no. Consider the change actions:

$$\overline{A_1} = (\mathbb{Z}_2, \mathbb{Z}_2, +, +, 0) \qquad \overline{A_2} = (\mathbb{Z}_2, \mathbb{N}, [+], +, 0)$$

where $[m]\,[+]\,n = [m+n]$. The identity function $\mathbf{id} : \overline{A_1} \to \overline{A_2}$ admits infinitely many derivatives, none of which are regular. The characterisation of differentiable functions admitting a regular derivative is an open problem.

Proposition 3.1.2. Given $f : \overline{A} \to \overline{B}$ and $g : \overline{B} \to \overline{C}$ with regular derivatives ∂f and ∂g respectively, the derivative $\partial(g \circ f) := \partial g \circ \langle f \circ \pi_1, \partial f \rangle$ is regular.

Proof. Suppose ∂f, ∂g are regular derivatives. It is straightforward to check that $\partial(g \circ f)$ preserves the zero change:

$$(\partial g \circ \langle f \circ \pi_1, \partial f \rangle)(a, 0) = \partial g(f(a), \partial f(a, 0)) = \partial g(f(a), 0) = 0$$

Now consider arbitrary changes $\delta_1, \delta_2 \in \Delta A$:

$$\begin{array}{rl} & \partial(g \circ f)(a, \delta_1 + \delta_2) \\ = & (\partial g \circ \langle f \circ \pi_1, \partial f \rangle)(a, \delta_1 + \delta_2) \\ = & \partial g(f(a), \partial f(a, \delta_1 + \delta_2)) \\ = & \partial g(f(a), \partial f(a, \delta_1) + \partial f(a \oplus \delta_1, \delta_2)) \\ = & \partial g(f(a), \partial f(a, \delta_1)) + \partial g(f(a) \oplus \partial f(a, \delta_1), \partial f(a \oplus \delta_1, \delta_2)) \\ = & (\partial g \circ \langle f \circ \pi_1, \partial f \rangle)(a, \delta_1) + (\partial f \circ \langle f \circ \pi_1, \partial f \rangle)(a \oplus \delta_1, \delta_2) \\ = & \partial(g \circ f)(a, \delta_1) + \partial(g \circ f)(a \oplus \delta_1, \delta_2) \end{array}$$ □

Remark 3.1.3. If one thinks of changes (i.e. elements of ΔA) as morphisms between elements of A, then the condition of regularity remarkably resembles functoriality. This intuition is explored in Section 3.3, where we show that categories of change actions organise themselves into 2-categories.

3.1.2 Two categories of change actions

The study of change actions can be undertaken in two ways. First, one can consider functions that are differentiable, without choosing a particular derivative. Alternatively, the derivative itself can be considered part of the morphism. The first option leads to the category **CAct**$^-$, whose objects are change actions and morphisms are all the differentiable maps[2].

The category **CAct**$^-$ was the category we originally proposed in [6]. It is well-behaved, possessing limits, colimits, and exponentials, which is a trivial corollary of Theorem 3.1.2 below.

Theorem 3.1.2. The category **CAct**$^-$ of change actions and differentiable morphisms is equivalent to **PreOrd**, the category of preorders and monotone maps.

To prove this, we introduce the notion of the reachability order generated by a change action $\overline{A}$.

Definition 3.1.4. For $a, b \in A$, we define the **reachability preorder** $a \sqsubseteq_\oplus b$ iff there is a $\delta a \in \Delta A$ such that $a \oplus \delta a = b$. Then $\sqsubseteq_\oplus$ defines a preorder on A.

[2]We have found the study of **CAct**$^-$ to be laborious and not particularly enlightening, so it appears here only as a historical note. The reader in a rush is encouraged to skip to Definition 3.1.6.

The intuitive significance of the reachability preorder induced by $\overline{A}$ is that it contains all the information about differentiability of functions from or into A. This is made precise in the following result:

Lemma 3.1.1. A function $f : A \to B$ is differentiable as a function from $\overline{A}$ to $\overline{B}$ iff it is monotone with respect to the reachability preorders on $\overline{A}$ and $\overline{B}$.

Proof. Let f be a differentiable function, with ∂f an arbitrary derivative, and suppose $a \sqsubseteq_{\oplus} a'$. Hence there is some δa such that $a \oplus \delta a = a'$. Then, by the derivative condition, we have $f(a') = f(a \oplus \delta a) = f(a) \oplus \partial f(a, \delta a)$, hence $f(a) \sqsubseteq_{\oplus} f(a')$.

Conversely, suppose f is monotone, and pick arbitrary $a \in A, \delta a \in \Delta A$. Since $a \sqsubseteq_{\oplus} a \oplus \delta a$ and f is monotone, we have $f(a) \sqsubseteq_{\oplus} f(a \oplus \delta a)$ and therefore there exists a $\delta b \in \Delta B$ such that $f(a) \oplus \delta b = f(a \oplus \delta a)$. We define $\partial f(a, \Delta a)$ to be precisely such a change δb (note that the process of arbitrarily picking a δb for every pair $a, \delta a$ makes use of the Axiom of Choice). $\square$

The correspondence between a change action and its reachability preorder gives rise to a full and faithful functor Reach : $\mathbf{CAct}^{-} \to \mathbf{PreOrd}$ that acts as the identity on morphisms.

Conversely, any preorder $\leq$ on some set A induces an equivalent change action: indeed, let $(A \Rightarrow_{\leq} A)$ denote the set of all $\leq$-monotone functions from A into itself. Since monotone functions are closed under composition, we define the change action $\overline{A_{\leq}}$ as in Example 3.1.3 by setting

$$\overline{A_{\leq}} \coloneqq (A, A \Rightarrow_{\leq} A, \mathbf{ev}_{A,A}, \circ, \mathbf{id}_A)$$

It is immediately apparent that the reachability order in $\overline{A_{\leq}}$ is precisely the preorder $\leq$ (since b is reachable from a if and only if there is some monotone function satisfying $f(a) = b$). Thus this correspondence defines another full and faithful functor Act : $\mathbf{PreOrd} \to \mathbf{CAct}^{-}$ that is the identity on morphisms.

It remains to check that there exists a pair of natural isomorphisms $\mathcal{U}$: Act $\circ$ Reach $\to \mathbf{id}_{\mathbf{CAct}^{-}}$ and $\mathcal{V}$: Reach $\circ$ Act $\to \mathbf{id}_{\mathbf{PreOrd}}$. But these are trivial: it suffices to set $\mathcal{U}_A = \mathbf{id}_A$ and $\mathcal{V}_{(A,\leq)} = \mathbf{id}_{(A,\leq)}$. Hence Act, Reach establish an equivalence of categories between $\mathbf{PreOrd}$ and $\mathbf{CAct}^{-}$.

The explicit structure of the limits and colimits in $\mathbf{CAct}^{-}$ is, however, not so satisfactory. One can, for example, obtain the limit of a certain diagram in $\mathbf{CAct}^{-}$ by taking its limit *qua* diagram in $\mathbf{PreOrd}$ and turning it into a change action, but the corresponding change set is not, in general, easily expressible in terms of those for

$\overline{A}$ and $\overline{B}$. Even then, some limits constructed in this fashion do admit a "sensible" presentation. For example, the previous construction suggests that the product $\overline{A} \times \overline{B}$ should be given by $(A \times B, A \times B \Rightarrow_{\leq} A \times B, \mathbf{ev}, \circ, \mathbf{id}_{A \times B})$, but the space of monotone maps $A \times B \Rightarrow_{\leq} A \times B$ can be thought of as (a quotient of) the set of arbitrary maps $A \times B \Rightarrow \Delta A \times \Delta B$, by encoding a monotone map by the change that it induces at each point.

Despite the above, derivatives of structure morphisms in $\mathbf{CAct}^{-}$ can still be hard to obtain, as exhibiting e.g. $\langle f, g \rangle$ as a morphism in $\mathbf{CAct}^{-}$ merely proves it is differentiable but gives no clue as to how a derivative might be constructed. One way to understand this problem is that the change set ΔA is a set of "witnesses" for the reachability relation in that each δ witnesses the property that $a \sqsubseteq_{\oplus} a \oplus \delta$. Differentiable maps, however, are monotone but do not provide a witness for this property; a choice of witness for the monotonicity of a map would form precisely a derivative for it.

A more constructive approach is to take a morphism to be a function together with a choice of a (regular) particular derivative for it. This has the advantage that a choice of a morphism automatically determines a choice of derivative.

Definition 3.1.5. Given change actions $\overline{A}$ and $\overline{B}$, a **differential map** $\overline{f} : \overline{A} \to \overline{B}$ is a pair $(f, \partial f)$ where $f : A \to B$ is an arbitrary function (which we refer to as the **underlying map** of $\overline{f}$) and $\partial f : A \times \Delta A \to \Delta B$ is a *regular* derivative for f.

Definition 3.1.6. The category **CAct** has change actions as objects and differential maps as morphisms. The identity morphisms are defined by $\overline{\mathbf{id}_A} \coloneqq (\mathbf{id}_A, \pi_1)$. Given morphisms $\overline{f} : \overline{A} \to \overline{B}$ and $\overline{g} : \overline{B} \to \overline{C}$, their composite $\overline{g} \circ \overline{f}$ is the differential map whose underlying map is $g \circ f$ along with the derivative $\partial g \circ \langle f \circ \pi_1, \partial f \rangle$. That is:

$$\overline{g} \circ \overline{f} \coloneqq (g \circ f, \partial g \circ \langle f \circ \pi_1, \partial f \rangle)$$

It is trivial to show that pre- and post-composition with the maps $\overline{\mathbf{id}_A}$ is indeed the identity, but the fact that composition in **CAct** is associative might not be so clear. This is a consequence of the fact that the chain rule itself is associative, which will be proven in more generality in the following section (see Proposition 3.2.1).

Finite products and coproducts exist in **CAct**: see Theorems 3.2.4 and 3.2.6 for a more general statement. However, unlike in $\mathbf{CAct}^{-}$, it is not clear whether general limits and colimits exist.

3.1.3 Adjunctions With Set

There is an obvious forgetful functor $\mathrm{U} : \mathbf{CAct} \to \mathbf{Set}$ that maps every change action $\overline{A}$ to its underlying set A and every differential map $\overline{f} : \overline{A} \to \overline{B}$ to the function on the underlying sets $f : A \to B$.

Recall from Section 3.1.2 that a change action $\overline{A}$ induces a preorder $\sqsubseteq_{\oplus}$ on A. Then we can define a quotient functor $\mathrm{Q} : \mathbf{CAct} \to \mathbf{Set}$ that maps the change action $\overline{A}$ to the set $A/\sim$, where $\sim$ is the transitive and symmetric closure of $\sqsubseteq_{\oplus}$. The action on morphisms $\overline{f} : \overline{A} \to \overline{B}$ is defined by $\mathrm{Q}(\overline{f})([a]) = [f(a)]$, where $[a]$ is the $\sim$-equivalence class of a.

Finally, there is a functor $\mathrm{D} : \mathbf{Set} \to \mathbf{CAct}$ that maps every set A to the discrete change action $\overline{A}_{\star} \equiv (A, \{\star\}, \pi_1, \pi_1, \star)$ (where $\{\star\}$ denotes the set with a single element $\star$). This functor D sends every function f to the differential map $(f, !)$. Notice that both compositions $\mathrm{Q} \circ \mathrm{D}$ and $\mathrm{U} \circ \mathrm{D}$ are in fact equal to the identity endofunctor $\mathrm{Id}_{\mathbf{Set}}$.

In what follows we will make use of the fact that $\mathrm{U} \circ \mathrm{D} = \mathrm{Q} \circ \mathrm{D} = \mathrm{Id}_{\mathbf{Set}}$.

Lemma 3.1.2. The forgetful functor U is right-adjoint to the functor D

Proof. Define the unit and counit as:

$$\begin{aligned} \varepsilon &: \mathrm{D} \circ \mathrm{U} \Rightarrow \mathrm{Id}_{\mathbf{CAct}} \\ \varepsilon_{\overline{A}} &\coloneqq (\mathbf{id}_A, 0) \\ \eta &: \mathrm{Id}_{\mathbf{Set}} \Rightarrow \mathrm{U} \circ \mathrm{D} \cong \mathrm{Id}_{\mathbf{Set}} \\ \eta_A &\coloneqq \overline{\mathbf{id}_A} \end{aligned}$$

We can check that the above definitions satisfy the zig-zag equations by simple calculation:

$$\begin{gathered} \varepsilon_{\mathrm{D}} \circ \mathrm{D}(\eta) = (\mathbf{id}, 0) \circ (\eta, !) = (\mathbf{id} \circ \eta, 0) = (\mathbf{id} \circ \mathbf{id}, 0) = \mathbf{id} \\ \mathrm{U}(\varepsilon) \circ \eta_{\mathrm{U}} = \mathrm{U}(\mathbf{id}, 0) \circ \mathbf{id} = \mathbf{id} \circ \mathbf{id} = \mathbf{id} \end{gathered}$$ □

Lemma 3.1.3. The functor D is right adjoint to the quotient functor Q.

Proof. Define the unit and counit as:

$$\begin{aligned} \varepsilon &: \mathrm{Id}_{\mathbf{Set}} \cong \mathrm{Q} \circ \mathrm{D} \to \mathrm{Id}_{\mathbf{Set}} \\ \varepsilon_A &\coloneqq \mathbf{Id}_A \\ \eta &: \mathrm{Id}_{\mathbf{CAct}} \to \mathrm{D} \circ \mathrm{Q} \\ \eta_{\overline{A}} &\coloneqq ([\mathbf{Id}_{\overline{A}}], !) \end{aligned}$$

where $[\mathbf{Id}_{\overline{A}}]$ is the map that sends an element a in A to the equivalence class $[a]$ of a modulo $\sim_{\leq}$. Note that η is well-defined since whenever $a \oplus \delta a = b$ it is the case that $[a] = [b]$. The zig-zag equations follow by similar calculations as Lemma 3.1.2. $\square$

For the next result, we assume the Axiom of Choice – or rather, we assume its equivalent formulation that there is a map G that sends each non-empty set A to some group G_A whose underlying set is A.

This map can be extended to a functor $\mathrm{G} : \mathbf{Set} \to \mathbf{CAct}$ which sends every (non-empty) set to the monoidal change action $(\mathrm{G}_A, \mathrm{G}_A, +, +, 0)$, the empty set to the trivial change action $(\emptyset, \{\star\}, !, \pi_1, \star)$, and every function $f : A \to B$ to the differential map $(f, \partial^1 f)$, with $\partial^1 f(x, \delta x) = -f(x) + f(x + \delta x)$. It is easy to see that $\mathrm{U} \circ \mathrm{G} = \mathrm{Id}_{\mathbf{Set}}$, and that G is full and faithful.

Lemma 3.1.4. The functor G is a right adjoint to the forgetful functor U.

Proof. Using the hom-set isomorphism definition of adjunction, the result is an immediate consequence from the fact that every function into a change action of the form $\mathrm{G}(A)$ has one and only one derivative. Thus there is a one-to-one correspondence between differential maps $\overline{f} : \overline{A} \to \mathrm{G}(B)$ and (arbitrary) functions $f : \mathrm{U}(\overline{A}) \to B$. $\square$

A version of this functor G which acts directly on groups (rather than sets) will become the key to interpreting the calculus of finite differences in the setting of change actions, see Section 5.2.2.

3.2 Change actions in arbitrary categories

The definition of change actions makes no use of any properties of **Set** beyond the existence of products. Indeed, change actions can be characterised as a multi-sorted algebra, which is definable in any category with products (although we emphasise again that differential maps are *not* the same as homomorphisms of change actions *qua* algebras).

Definition 3.2.1. Given an ambient category $\mathbf{C}$ equipped with all finite products, a **C-change action** is a tuple $\overline{A} = (A, \Delta A, \oplus_A, +_A, 0_A)$ where

- A is an object in $\mathbf{C}$.
- $(\Delta A, +_A, 0_A)$ is a monoid internal to $\mathbf{C}$.
- $\oplus_A : A \times \Delta A \to A$ is an action of ΔA on A, i.e. it is a $\mathbf{C}$-morphism such that the following diagrams commute:

$$\begin{tikzcd}
A \arrow[r, "\langle \mathbf{id}, 0_A \rangle"] \arrow[dr, "\mathbf{id}"'] & A \times \Delta A \arrow[d, "\oplus_A"] \\
 & A
\end{tikzcd}
\qquad
\begin{tikzcd}
A \times \Delta A \times \Delta A \arrow[r, "\langle \oplus_A, \mathbf{id} \rangle"] \arrow[d, "\langle \mathbf{id}, +_A \rangle"'] & A \times \Delta A \arrow[d, "\oplus_A"] \\
A \times \Delta A \arrow[r, "\oplus_A"'] & A
\end{tikzcd}$$

Notation. For the sake of clarity, when composing with one of the "operator" maps $\oplus_A, +_A$ we will write $f +_A g$ for $+_A \circ \langle f, g \rangle$ and $f \oplus_A g$ for $\oplus_A \circ \langle f, g \rangle$. Both of these operators bind less tightly than function composition, that is, $f \circ g +_A h$ should be read as $(f \circ g) +_A h$. When there is no risk of confusion, we will abuse the notation and write $0_A : U \to \Delta A$ instead of $0_A \circ !$.

Definition 3.2.2. Given two $\mathbf{C}$-change actions $\overline{A}, \overline{B}$ and a $\mathbf{C}$-map $f : A \to B$, a **derivative** for f is a $\mathbf{C}$-map $\partial f : A \times \Delta A \to \Delta B$ such that the following diagrams commute:

$$\begin{tikzcd}
A \times \Delta A \arrow[r, "\langle f \circ \pi_1, \partial f \rangle"] \arrow[d, "\oplus_A"'] & B \times \Delta B \arrow[d, "\oplus_B"] \\
A \arrow[r, "f"'] & B
\end{tikzcd}
\qquad
\begin{tikzcd}
A \arrow[r, "!"] \arrow[d, "\langle \mathbf{id}, 0_A \circ ! \rangle"'] & 1 \arrow[d, "0_B"] \\
A \times \Delta A \arrow[r, "\partial f"'] & \Delta B
\end{tikzcd}$$

$$\begin{tikzcd}
A \times (\Delta A \times \Delta A) \arrow[r, "\langle \langle \pi_1, \pi_1 \circ \pi_2 \rangle, \mathbf{a} \rangle"] \arrow[d, "\partial f \circ (\mathbf{id} \times +_A)"'] & (A \times \Delta A) \times ((A \times \Delta A) \times \Delta A) \arrow[d, "\partial f \times (\partial f \circ (\oplus_A \times \mathbf{id}))"] \\
\Delta B & \Delta B \times \Delta B \arrow[l, "+_B"]
\end{tikzcd}$$

The first diagram states the derivative condition, while the other two diagrams amount to a diagrammatic version of the regularity of ∂f. We will often refer to these properties independently (that is, we will say that a certain morphism "satisfies the derivative condition" or "satisfies the regularity conditions").

Notation. In many definitions and proofs we make statements about equalities that may involve some significant "shuffling around" of products. As these details are inconsequential and can significantly obscure the underlying ideas, we will prefer to leave these implicit and work in the internal language of Cartesian categories instead.

For example, and using the more convenient infix notation, the regularity condition for derivatives can be rephrased as stating that the equation

$$\partial f \circ \langle a, \delta_1 + \delta_2 \rangle = (\partial f \circ \langle a, \delta_1 \rangle) +_B (\partial f \circ \langle a \oplus_A \delta_1, \delta_2 \rangle)$$

holds for any choice of $a : U \to A, \delta_1, \delta_2 : U \to \Delta A$, instead of a much more unwieldy presentation using projections.

Lemma 3.2.1 (Chain rule)**.** Given differential maps $(f, \partial f) : \overline{A} \to \overline{B}, (g, \partial g) : \overline{B} \to \overline{C}$, the pair $(g \circ f, \partial g \circ \langle f \circ \pi_1, \partial g\rangle)$ is a differential map from $\overline{A}$ into $\overline{C}$.

The chain rule can be elegantly summarised by pasting together two instances of the diagram in Definition 3.2.2:

$$\begin{tikzcd}
A \times \Delta A \arrow[r, "\langle f\circ\pi_1, \partial f\rangle"] \arrow[d, "\oplus_A"'] \arrow[rr, bend left, "\langle (g\circ f)\circ\pi_1, \partial g\circ\langle f\circ\pi_1, \partial f\rangle\rangle"] & B \times \Delta B \arrow[r, "\langle g\circ\pi_1, \partial g\rangle"] \arrow[d, "\oplus_B"] & C \times \Delta C \arrow[d, "\oplus_C"] \\
A \arrow[r, "f"] \arrow[rr, bend right, "g\circ f"'] & B \arrow[r, "g"] & C
\end{tikzcd}$$

Definition 3.2.3. A **differential map** from $\overline{A}$ to $\overline{B}$ is a pair $\overline{f} \equiv (f, \partial f)$ where:

- $f : A \to B$ is a **C**-map, which we call the **underlying** map of $\overline{f}$.
- $\partial f : A \times \Delta A \to \Delta B$ is a derivative for f.

Given differential maps $\overline{f} : \overline{A} \to \overline{B}, \overline{g} : \overline{B} \to \overline{C}$, we define their composition $\overline{g} \circ \overline{f}$ to be the differential map whose underlying map is $g \circ f$ and whose derivative is the derivative obtained by applying the chain rule to $\partial g, \partial f$. That is:

$$\overline{g} \circ \overline{f} := (g \circ f, \partial g \circ \langle f \circ \pi_1, \partial f\rangle)$$

Proposition 3.2.1. Composition of differential maps is associative. That is, given $\overline{f}, \overline{g}, \overline{h}$, we have $(\overline{h} \circ \overline{g}) \circ \overline{f} = \overline{h} \circ (\overline{g} \circ \overline{f})$. Furthermore, the identity differential maps $\overline{\mathbf{id}}_{\overline{A}} := (\mathbf{id}_A, \pi_2)$ are identities for the composition operation.

Proof. The second property is trivial. We prove associativity; for this, it suffices to show that the derivative of both composites is the same:

$$\begin{aligned}
\partial((\overline{h} \circ \overline{g}) \circ \overline{f}) &= \partial(\overline{h} \circ \overline{g}) \circ \langle f \circ \pi_1, \partial f\rangle \\
&= \partial\overline{h} \circ \langle g \circ \pi_1, \partial g\rangle \circ \langle f \circ \pi_1, \partial f\rangle \\
&= \partial\overline{h} \circ \langle g \circ f \circ \pi_1, \partial g \circ \langle f \circ \pi_1, \partial f\rangle\rangle \\
&= \partial\overline{h} \circ \langle g \circ f \circ \pi_1, \partial g \circ \langle f \circ \pi_1, \partial f\rangle\rangle \\
&= \partial\overline{h} \circ \big\langle (g \circ f) \circ \pi_1, \partial(\overline{g} \circ \overline{f})\big\rangle \\
&= \partial(\overline{h} \circ (\overline{g} \circ \overline{f}))
\end{aligned}$$
□

3.2.1 The Category $\mathrm{CAct}(\mathbf{C})$

Given any object $\mathbf{C}$ in $\mathbf{Cat}_\times$, the category of (small) Cartesian categories and strong monoidal functors, it is straightforward enough to define a category $\mathrm{CAct}(\mathbf{C})$ of change actions and differential maps internal to $\mathbf{C}$, in much the same way that we defined the category **CAct** in the previous section.

This construction also allows one to lift any (strong monidal) functor $\mathrm{F} : \mathbf{C} \to \mathbf{D}$ into a functor $\mathrm{CAct}(\mathrm{F}) : \mathrm{CAct}(\mathbf{C}) \to \mathrm{CAct}(\mathbf{D})$. Thus we can define a functor $\mathrm{CAct} : \mathbf{Cat}_\times \to \mathbf{Cat}$ (as we shall later see, the category $\mathrm{CAct}(\mathbf{C})$ always has products and $\mathrm{CAct}(-)$ is in fact an endofunctor on $\mathbf{Cat}_\times$).

Definition 3.2.4. The objects of $\mathrm{CAct}(\mathbf{C})$ are all the change actions $\overline{A}$ internal to $\mathbf{C}$.

Given objects $\overline{A}, \overline{B}$ in $\mathrm{CAct}(\mathbf{C})$, the morphisms of $\mathrm{CAct}(\overline{A}, \overline{B})$ are differential maps $\overline{f} : \overline{A} \to \overline{B}$. Given a functor $\mathrm{F} : \mathbf{C} \to \mathbf{D}$ and isomorphisms $\varepsilon : \mathbf{1_D} \to \mathrm{F}(\mathbf{1_C}), \mu_{A,B} : F(A) \times F(B) \to F(A \times B)$, the functor $\mathrm{CAct}(\mathrm{F}) : \mathrm{CAct}(\mathbf{C}) \to \mathrm{CAct}(\mathbf{D})$ is defined by:

$$\begin{aligned}\mathrm{CAct}(\mathrm{F})(\overline{A}) &\coloneqq (\mathrm{F}(A), \mathrm{F}(\Delta A), \mathrm{F}(\oplus_A) \circ \mu^{-1}_{A,\Delta A}, \mathrm{F}(+_A) \circ \mu^{-1}_{\Delta A,\Delta A}, \mathrm{F}(0_A) \circ \varepsilon^{-1}) \\ \mathrm{CAct}(\mathrm{F})(\overline{f}) &\coloneqq (\mathrm{F}(f), \mathrm{F}(\partial f) \circ \mu^{-1}_{A,\Delta A})\end{aligned}$$

As is the case for **Set**, the category $\mathrm{CAct}(\mathbf{C})$ has enough information to “encode” the original objects and maps of $\mathbf{C}$, which embed into CAct via the trivial change action.

Theorem 3.2.1. The category $\mathbf{C}$ embeds fully and faithfully into the category $\mathrm{CAct}(\mathbf{C})$ via the (strong monoidal) functor $(-)_\star$ which sends an object A of $\mathbf{C}$ to the trivial change action $\overline{A}_\star = (A, \mathbf{1}, \pi_1, !, !)$ and every morphism $f : A \to B$ of $\mathbf{C}$ to the differential map $(f, !)$.

Furthermore, the family of (strong monoidal) functors $(-)_\star : \mathbf{C} \to \mathrm{CAct}(\mathbf{C})$ defines a natural transformation from the identity endofunctor $\mathrm{Id}_{\mathbf{Cat}_\times}$ into the functor CAct.

Additionally, there is an obvious forgetful functor $\varepsilon_\mathbf{C} : \mathrm{CAct}(\mathbf{C}) \to \mathbf{C}$, which defines the components of a natural transformation ε from the functor CAct to the identity endofunctor **id**.

Theorem 3.2.2. The trivial change action embedding $(-)_\star : \mathbf{C} \to \mathrm{CAct}(\mathbf{C})$ is left-adjoint to the forgetful functor $\varepsilon : \mathrm{CAct}(\mathbf{C}) \to \mathbf{C}$.

Given $\mathbf{C}$, we write $\xi_{\mathbf{C}}$ for the functor $\mathrm{CAct}(\varepsilon_{\mathbf{C}}) : \mathrm{CAct}(\mathrm{CAct}(\mathbf{C})) \to \mathrm{CAct}(\mathbf{C})$[3]. Explicitly, this functor maps an object $(\overline{A}, \overline{B}, \overline{\oplus}, \overline{+}, \overline{0})$ in $\mathrm{CAct}(\mathrm{CAct}(\mathbf{C}))$ to the object $(A, B, \oplus, +, 0)$. Intuitively, $\varepsilon_{\mathrm{CAct}(\mathbf{C})}$ prefers the "original" structure on objects, whereas $\xi_{\mathbf{C}}$ prefers the "higher" structure. The equaliser of these two functors is precisely the category of change actions whose higher structure is the original structure.

This forgetful functor ε is evidently not faithful, as a $\mathbf{C}$-map may admit more than one derivative and so be the image of more than one differential map. There is, however, a way to embed the category of change actions faithfully back into $\mathbf{C}$.

Definition 3.2.5. The **tangent bundle** functor $\mathrm{T} : \mathrm{CAct}(\mathbf{C}) \to \mathbf{C}$ maps a change action $\mathbf{A}$ to the $\mathbf{C}$-object $A \times \Delta A$, and a differential map $\overline{f} : \overline{A} \to \overline{B}$ to the $\mathbf{C}$-map $\langle f \circ \pi_1, \partial f \rangle : A \times \Delta A \to B \times \Delta B$

Remark 3.2.1. So far we have not shown that products exist in $\mathrm{CAct}(\mathbf{C})$, hence all of the functors we have defined above should be read as arrows in $\mathbf{Cat}$, not in $\mathbf{Cat}_\times$. As we shall show, $\mathrm{CAct}(\mathbf{C})$ does have finite products, and it will be the case that all the previous functors are indeed product-preserving.

An interesting property of the category of change actions is that, while regularity is a much weaker notion than additivity, the category of monoids internal to $\mathbf{C}$ can nonetheless be embedded fully and faithfully into the category $\mathrm{CAct}(\mathbf{C})$ of change actions - that is, monoid homomorphisms correspond precisely to differentiable maps between carefully chosen change actions.

Theorem 3.2.3. The category $\mathrm{Mon}(\mathbf{C})$ whose objects are monoids internal to $\mathbf{C}$ and whose morphisms are monoid homomorphisms embeds fully and faithfully into the category $\mathrm{CAct}(\mathbf{C})$ via the functor $\mathcal{M} : \mathrm{Mon}(\mathbf{C}) \to \mathrm{CAct}(\mathbf{C})$ defined as follows:

$$\mathcal{M}(A, +_A, 0_A) := (\mathbf{1}, A, !, +_A, 0_A)$$
$$\mathcal{M}(f : A \to B) := (!, f \circ \pi_2)$$

Proof. First, we show the functor $\mathcal{M}$ is well-defined. Clearly it maps monoids to change actions, and $\mathcal{M}(f)$ trivially satisfies the derivative condition: it remains to show that its derivative is regular. This is an immediate consequence of f being a monoid homomorphism:

[3]One might expect CAct to be a comonad with ε as a counit. But if this were the case, we would have $\xi_{\mathbf{C}} = \varepsilon_{\mathrm{CAct}(\mathbf{C})}$, which is, in general, not true.

$$\begin{aligned}
f \circ \pi_2 \circ \langle !, \delta_1 +_A \delta_2 \rangle &= f \circ (\delta_1 +_A \delta_2) \\
&= (f \circ \delta_1) +_B (f \circ \delta_2) \\
&= (f \circ \pi_2 \circ \langle !, \delta_1 \rangle) +_B (f \circ \pi_2 \circ \langle !, \delta_2 \rangle)
\end{aligned}$$

$\mathcal{M}(\mathbf{id}_A) = (\mathbf{id}_\mathbf{1}, \mathbf{id}_A \circ \pi_2) = (\mathbf{id}_\mathbf{1}, \pi_2) = \overline{\mathbf{id}}_{\mathcal{M}A}$, therefore $\mathcal{M}$ preserves the identity. It also preserves composition since:

$$\begin{aligned}
\mathcal{M}(g) \circ \mathcal{M}(f) &= (\mathbf{id}_\mathbf{1}, g \circ \pi_2) \circ (\mathbf{id}_\mathbf{1}, f \circ \pi_2) \\
&= (\mathbf{id}_\mathbf{1}, g \circ \pi_2 \circ \langle \mathbf{id}_\mathbf{1} \circ \pi_1, f \circ \pi_2 \rangle) \\
&= (\mathbf{id}_\mathbf{1}, g \circ f \circ \pi_2) \\
&= \mathcal{M}(g \circ f)
\end{aligned}$$

It is evident that $\mathcal{M}$ is faithful. To see that it is full, consider a differential map $\overline{f} : \mathcal{M}(A, +_A, 0_A) \to \mathcal{M}(B, +_B, 0_B)$. Clearly the underlying map is just the terminal map $f = !$. Similarly since $\mathbf{1} \times A \cong A$ the derivative ∂f has type $\partial f : \mathbf{1} \times A \to B$. It remains to show that $\partial f \circ (\cong)$ is a monoid homomorphism and therefore $(\mathbf{id}_\mathbf{1}, \partial f) = \overline{f} = \mathcal{M}(\partial f \circ (\cong))$, but this follows immediately from regularity of ∂f. □

3.2.2 Products and Coproducts in CAct($\mathbf{C}$)

We have defined CAct as a functor from $\mathbf{Cat}_\times$ into $\mathbf{Cat}$. This can in fact be strengthened, as CAct($\mathbf{C}$) inherits finite products from the underlying category $\mathbf{C}$.

Theorem 3.2.4. Let $\overline{A} = (A, \Delta A, \oplus_A, +_A, 0_A)$ and $\overline{B} = (B, \Delta B, \oplus_B, +_B, 0_B)$ be change actions on $\mathbf{C}$. Then the following change action is the product of $\overline{A}$ and $\overline{B}$ in CAct($\mathbf{C}$)

$$\overline{A} \times \overline{B} := (A \times B, \Delta A \times \Delta B, \oplus_{A \times B}, +_{A \times B}, \langle 0_A, 0_B \rangle)$$

where

$$\begin{aligned}
(a, b) \oplus_{A \times B} (\delta a, \delta b) &:= (a \oplus_A \delta_1, b \oplus_B \delta_2) \\
(\delta a_1, \delta b_1) +_{A \times B} (\delta a_2, \delta b_2) &:= (\delta a_1 +_A \delta a_2, \delta b_1 +_B \delta b_2)
\end{aligned}$$

The projection maps are

$$\begin{aligned}
\overline{\pi_1} &:= (\pi_1, \pi_1 \circ \pi_2) \\
\overline{\pi_2} &:= (\pi_2, \pi_2 \circ \pi_2)
\end{aligned}$$

Proof. Given any pair of differential maps $\overline{f_1} : \overline{C} \to \overline{A}, \overline{f_2} : \overline{C} \to \overline{B}$, define

$$\left\langle \overline{f_1}, \overline{f_2} \right\rangle := (\langle f_1, f_2 \rangle, \langle \partial f_1 \circ \langle \pi_1 \circ \pi_1, \pi_1 \circ \pi_2 \rangle, \partial f_2 \circ \langle \pi_2 \circ \pi_1, \pi_2 \circ \pi_2 \rangle \rangle)$$

Then $\overline{\pi_i} \circ \langle f_1, f_2 \rangle = f_i$. Furthermore, given any map $\overline{h} : \overline{C} \to \overline{A} \times \overline{B}$ whose projections coincide with $\langle \overline{f_1}, \overline{f_2} \rangle$, by applying the universal property of the product in $\mathbf{C}$, we obtain $h = \langle \overline{f_1}, \overline{f_2} \rangle$. $\square$

Theorem 3.2.5. The change action $\overline{\mathbf{1}} = (\mathbf{1}, \mathbf{1}, \pi_1, \pi_1, \mathbf{id_1})$ is the terminal object in CAct($\mathbf{C}$), where $\mathbf{1}$ is the terminal object of $\mathbf{C}$. Furthermore, if $\overline{A}$ is a change action every point $f : \mathbf{1} \to A$ in $\mathbf{C}$ is differentiable, with (unique) derivative 0_A.

Proof. Given a change action $\overline{A}$, there is exactly one differential map $\overline{!} := (!, !) : \overline{A} \to \overline{\mathbf{1}}$. Now given a differential map $(f, \partial f) : \overline{\mathbf{1}} \to \overline{A}$, applying regularity we obtain:

$$\partial f = \partial f \circ \langle \mathbf{id_1}, \mathbf{id_1} \rangle = \partial f \circ \langle \mathbf{id_1}, 0_1 \rangle = 0_A$$ $\square$

It is a well-known result from differential calculus that a multivariate function is (continuously) differentiable if and only if all its partial derivatives exist and are continuous. It is perhaps surprising that a similar result holds for change actions, given an adequate definition of a partial derivative.

Definition 3.2.6. Given a $\mathbf{C}$-map $f : A \times B \to C$ and fixed change actions $\overline{A}, \overline{B}, \overline{C}$, a **partial derivative with respect to** A for f is a map $\partial_1 f : (A \times B) \times \Delta A \to \Delta C$ such that the following three conditions hold:

- $f(a, b) \oplus_C \partial_1 f((a, b), \delta a) = f(a \oplus_A \delta a, b)$
- $\partial_1 f((a, b), 0_A) = 0_C$
- $\partial_1 f((a, b), \delta a_1 +_A \delta a_2) = \partial_1 f((a, b), \delta a_1) +_C \partial_2 f((a \oplus_A \delta a_1, b), \delta a_2)$

In the same manner, we write $\partial_2 f$ for the partial derivative of f with respect to B, with the obvious definition.

In a simple, set-theoretic setting, a partial derivative for f with respect to A is precisely a function mapping each $b \in A$ to a derivative for the partially applied function $f(-, b) : A \to C$.

Lemma 3.2.2. Let $f : A \times B \to C$ be a $\mathbf{C}$-map and consider fixed change actions $\overline{A}, \overline{B}, \overline{C}$, with $+_A$ commutative. then f admits a derivative if and only if f admits partial derivatives with respect to its first and second arguments.

Proof. Suppose ∂f is a derivative for f. Then we define:

$$\partial_1 f \coloneqq \partial f \circ \langle \pi_1, \langle \pi_2, 0_B \rangle \rangle : (A \times B) \times \Delta A \to \Delta C$$

Clearly $\partial_1 f$ is a first partial derivative for f. We explicitly check the first condition:

$$f(a,b) \oplus_C (\partial_1 f((a,b), \delta a)) = f(a,b) \oplus_C \partial f((a,b), (\delta a, 0_B)) = f(a \oplus_A \delta a, b)$$

Conversely, suppose $\partial_1 f, \partial_2 f$ are partial derivatives for f. Then the following ∂f is a derivative for f:

$$\partial f((a,b), (\delta a, \delta b)) \coloneqq \partial_1 f((a,b), \delta a) +_C \partial_2 f((a \oplus_A \delta a, b), \delta b)$$

We can easily check that the above verifies the derivative condition:

$$\begin{aligned} f(a,b) \oplus_C \partial f((a,b), (\delta a, \delta b)) &= f(a,b) \oplus_C [\partial_1 f((a,b), \delta a) +_C \partial_2 f((a \oplus_A \delta a, b), \delta b)] \\ &= f(a \oplus \delta a, b) \oplus_C \partial_2 f((a \oplus \delta a, b), \delta b) \\ &= f(a \oplus \delta a, b \oplus \delta b) \end{aligned}$$

Regularity follows from straightforward calculation and commutativity of $+_C$. □

Furthermore, as a direct consequence of ∂f being regular, this process is invertible. That is, whenever ∂f is a derivative for f, the derivative obtained by "adding together" the partial derivatives $\partial_1 f, \partial_2 f$ induced by ∂f is precisely itself (and vice versa).

Corollary 3.2.1. Let $\overline{f} : \overline{A} \times \overline{B} \to \overline{C}$ be a differential map, and let $\partial_1 f, \partial_2 f$ denote the partial derivatives for f computed as in the above result. Then $\partial f((a,b), (\delta a, \delta b)) = (\partial_1 f((a,b), \delta a)) +_C (\partial_2 f((a \oplus_A \delta a, b), \delta b))$

When applying derivatives to incremental computation, we will sometimes want to incrementalise a multivariate function only with respect to one of its arguments (for example, because the other remains fixed, as in Section 4.5.2). The previous results guarantee that this sort of differentiation with respect to one variable only is nothing more and nothing less than our usual notion of differentiation.

Whenever $\mathbf{C}$ has distributive coproducts, then these are also inherited by $\mathrm{CAct}(\mathbf{C})$, although in this case the structure is not as obvious: one might expect that the change space of $\overline{A} \coprod \overline{B}$ would be the coproduct $\Delta A \coprod \Delta B$. This is, however, not the case: given the definition of a change action, a change $\delta : \Delta(\overline{A} \coprod \overline{B})$ needs to be applicable to elements of the form $\mathbf{i}_1 a$ as well as elements of the form $\mathbf{i}_2 b$. The natural choice for such a δ is in fact a pair $(\delta a, \delta b) : \Delta A \times \Delta B$, which acts as δa on objects of the form $\mathbf{i}_1 a$ and as δb on objects of the form $\mathbf{i}_2 b$.

The category $\mathrm{CAct}(\mathbf{C})$ also inherits coproducts and the initial element from the base category: whenever $\mathbf{C}$ has (distributive) coproducts, then so does $\mathrm{CAct}(\mathbf{C})$.

Theorem 3.2.6. If $\mathbf{C}$ is distributive, with distributive law $\delta_{A,B,C} : (A \coprod B) \times C \to (A \times C) \coprod (B \times C)$, the following change action is the coproduct of A and B in $\mathrm{CAct}(\mathbf{C})$

$$A \coprod B \coloneqq (A \coprod B, \Delta A \times \Delta B, \oplus_{A \coprod B}, +_{A \coprod B}, \langle 0_A, 0_B \rangle)$$

where

$$\begin{aligned}
\oplus_{A \coprod B} &\coloneqq [\oplus_A \circ (\mathbf{id}_A \times \pi_1), \oplus_B \circ (\mathbf{id}_B \times \pi_2)] \circ \delta_{A,B,C} \\
+_{A \coprod B} &\coloneqq \langle +_A \circ (\pi_1 \times \pi_1), +_B \circ (\pi_2 \times \pi_2) \rangle
\end{aligned}$$

The injections are $\overline{\mathbf{i}_1} = (\mathbf{i}_1, \langle \pi_2, 0_B \rangle)$ and $\overline{\mathbf{i}_2} = (\mathbf{i}_2, \langle 0_A, \pi_2 \rangle)$.

Proof. Given any pair of differential maps $f_1 : A \to C, f_2 : B \to C$ define:

$$\begin{aligned}
\left[\overline{f_1}, \overline{f_2}\right] &\coloneqq ([f_1, f_2], \partial h) \\
\partial h &\coloneqq [\partial f_1 \circ (\times \pi_1), \partial f_2 \circ (\mathbf{id}_B \times \pi_2)] \circ \delta_{A,B,C}
\end{aligned}$$

where $\delta_{A,B,C} : (A \coprod B) \times C \to (A \times C) \coprod (B \times C)$ is the distributive law of $\mathbf{C}$.

We check that, indeed, the relevant diagram commutes since:

$$\begin{aligned}
& [f_1, f_2] \circ \overline{\mathbf{i}_1} \\
= \; & ([f_1, f_2], \partial h) \circ (\mathbf{i}_1, \langle \pi_2, 0_B \rangle) \\
= \; & ([f_1, f_2] \circ \mathbf{i}_1, \partial h \circ \langle \mathbf{i}_1 \circ \pi_1, \langle \pi_2, 0_B \rangle \rangle) \\
= \; & (f_1, [\partial f_1 \circ (\times \pi_1), \partial f_2 \circ (\mathbf{id}_B \times \pi_2)] \circ \delta_{A,B,C} \circ \langle \mathbf{i}_1 \circ \pi_1, \langle \pi_2, 0_B \rangle \rangle) \\
= \; & (f_1, \partial f_1 \circ (\times \pi_1) \circ \langle \pi_1, \langle \pi_2, 0_B \rangle \rangle) \\
= \; & (f_1, \partial f_1 \circ \langle \pi_1, \pi_2 \rangle) \\
= \; & (f_1, \partial f_1) \\
= \; & f_1
\end{aligned}$$

The universal property of the coproduct in $\mathbf{C}$ entails that if $\overline{h} = (h, \partial h)$ is such that $\overline{h} \circ \overline{\mathbf{i}_i} = f_i$, then $h = [f_1, f_2]$. Furthermore, since

$$(A \coprod B) \times \Delta A \times \Delta B \;\cong\; (A \times \Delta A \times \Delta B) \coprod (B \times \Delta A \times \Delta B),$$

the universal property of the coproduct also shows that necessarily $\partial h = \partial f$. □

Proposition 3.2.2. Whenever $\mathbf{C}$ has an initial object $\mathbf{0}$, then the change action $\overline{\mathbf{0}} \coloneqq (\mathbf{0}, \mathbf{1}, \mathbf{id}_{\mathbf{0}}, !, !)$ is an initial object in $\mathrm{CAct}(\mathbf{C})$.

It follows from the results of this section that the category $\mathrm{CAct}(\mathbf{C})$ has all finite products and coproducts (provided that $\mathbf{C}$ has distributive finite coproducts). The existence of more general limits and colimits is an important open question that we have left for future work.

3.3 Change actions as categories

Thus far we have developed the theory of change actions without providing much motivation for our definitions. For example, Definition 3.2.2 may seem like an ad-hoc artifice that does not correspond to any natural notion of “homomorphism of change actions”. Even if this were the case, we would argue its the theoretical interest and practical applications suffice to warrant its study.

No such excuses are necessary, however: as it turns out, our definition of differential map arises elegantly from the fact that change actions are nothing but a very special kind of categories, with differential maps corresponding exactly to functors!

For the sake of convenience, and to fix the notation, we present here the notions of internal category and internal functor which will appear in this section. For a more detailed introduction to internal category theory, we refer the reader to [55, §7], whose definitions we are using mostly verbatim.

Definition 3.3.1. Given an ambient category $\mathbf{C}$ equipped with all finite products, a **category internal to C** is a tuple $\mathbb{C} = (C_0, C_1, s, t, i, c)$ where:

- C_0 and C_1 are objects of $\mathbf{C}$ (the ‘object of objects’ and the ‘object of morphisms’ respectively).
- ‘Source’ and ‘target’ morphisms $s, t : C_1 \to C_0$ such that the following pullbacks exist:

$$\begin{tikzcd} C_2 \coloneqq C_1 \times_{C_0} C_1 \arrow[r, "\pi_2"] \arrow[d, "\pi_1"'] & C_1 \arrow[d, "s"] \\ C_1 \arrow[r, "t"'] & C_0 \end{tikzcd} \qquad \begin{tikzcd} C_3 \coloneqq C_1 \times_{C_0} C_1 \times_{C_0} C_1 \arrow[r] \arrow[d] & C_1 \arrow[d, "s"] \\ C_2 \arrow[r, "t \circ \pi_2"'] & C_0 \end{tikzcd}$$

- An ‘identity-assigning’ morphism $i : C_0 \to C_1$ such that the following diagram commutes:

$$\begin{tikzcd} & C_0 \arrow[dl, equal] \arrow[d, "i"] \arrow[dr, equal] & \\ C_0 & C_1 \arrow[l, "s"] \arrow[r, "t"'] & C_0 \end{tikzcd}$$

- A ‘composition’ morphism $c : C_1 \times_{C_0} C_1 \to C_1$ making the following diagrams commute:

$$\begin{tikzcd}
C_1 \arrow[d, "s"'] & C_2 \arrow[l, "\pi_1"'] \arrow[r, "\pi_2"] \arrow[d, "c"] & C_1 \arrow[d, "t"] \\
C_0 & C_1 \arrow[l, "s"] \arrow[r, "t"'] & C_0
\end{tikzcd}
\quad
\begin{tikzcd}
C_3 \arrow[r, "\mathbf{id}\times c"] \arrow[d, "c\times\mathbf{id}"'] & C_2 \arrow[d, "c"] \\
C_2 \arrow[r, "c"'] & C_1
\end{tikzcd}
\quad
\begin{tikzcd}
C_1 \arrow[r, "\langle i\circ s{,}\mathbf{id}\rangle"] \arrow[dr] & C_2 \arrow[d, "c"] & C_1 \arrow[l, "\langle\mathbf{id}{,}i\circ t\rangle"'] \arrow[dl] \\
& C_1 &
\end{tikzcd}$$

The aforementioned diagrams encode a point-free version of the usual axioms of categories. As with change actions, they could also be given in the internal language of **C**. However, this is not as much help as it is in the case of change actions, since the reliance on pullbacks complicates the expressions significantly.

Definition 3.3.2. Given internal categories $\mathbb{C}$ and $\mathbb{D}$ in **C**, an **internal functor** from $\mathbb{C}$ to $\mathbb{D}$ is a pair of **C**-morphisms, $f_0 : C_0 \to D_0$ and $f_1 : C_1 \to D_1$, such that all of the following diagrams commute:

$$\begin{tikzcd}
C_1 \arrow[r, "f_1"] \arrow[d, "s"'] & D_1 \arrow[d, "s'"] \\
C_0 \arrow[r, "f_0"'] & D_0
\end{tikzcd}
\quad
\begin{tikzcd}
C_1 \arrow[r, "f_1"] \arrow[d, "t"'] & D_1 \arrow[d, "t'"] \\
C_0 \arrow[r, "f_0"'] & D_0
\end{tikzcd}
\quad
\begin{tikzcd}
C_0 \arrow[r, "f_0"] \arrow[d, "i"'] & D_0 \arrow[d, "i'"] \\
C_1 \arrow[r, "f_1"'] & D_1
\end{tikzcd}
\quad
\begin{tikzcd}
C_1 \times_{C_0} C_1 \arrow[r, "f_1 \times_{C_0} f_1"] \arrow[d, "m"'] & D_1 \times_{D_0} D_1 \arrow[d, "m'"] \\
C_1 \arrow[r, "f_1"'] & D_1
\end{tikzcd}$$

These diagrams simply state the usual definition of a functor internally in **C**. It is easy to show that internal functors compose in a strictly associative manner, and that internal categories and internal functors form a category, which we denote by Cat(**C**)

Remark 3.3.1. In order to define the category Cat(**C**) above, we have not required that all (or indeed any) pullbacks exist in the category **C** (in fact the category Cat(**C**) can be defined even when **C** lacks products). For example, if **C** only has trivial pullbacks (i.e. pullbacks of the identity along itself), the resulting Cat(**C**) will likewise be made up of trivial internal categories, that is, those for which $C_0 = C_1$ and $s, t, e = \mathbf{id}_{C_0}$.

The question is, then: what is the relation between Cat(**C**) and CAct(**C**)? As we have remarked before, the changes in ΔA can be thought of as "directed paths", with the peculiarity that they are *free-floating*, that is to say, they don't start at a particular point, but can be freely transported around. Since changes can be applied to any point, a change δ represents both a path from a to $a \oplus \delta$ and a path from b to $b \oplus \delta$ (perhaps a more accurate description of changes would be to say that a change associates a path to every point).

Now given a **C**-change action $\overline{A} = (A, \Delta A, \oplus_A, +_A, 0_A)$ we can obtain a **C**-category by choosing A as the object of objects, with the object of arrows being given by the product $A \times \Delta A$. This might seem odd but, as changes are "free-floating" arrows, one can understand this product as the space of changes with a fixed base. The pair $(a, \delta) : A \times \Delta A$ is then to be interpreted as an arrow from the "object" a into the "object"$a \oplus_A \delta$.

Theorem 3.3.1. Whenever $\mathbf{C}$ is equipped with a choice of finite products, there is a full and faithful functor

$$\kappa_{\mathbf{C}} : \mathrm{CAct}(\mathbf{C}) \to \mathrm{Cat}(\mathbf{C})$$

from the category of $\mathbf{C}$-change actions and differential maps into the category of $\mathbf{C}$-categories and $\mathbf{C}$-functors.

Furthermore, whenever product by a constant is injective (that is to say, whenever the functor $X \mapsto A \times X$ is injective for all objects in $\mathbf{C}$) the functor $\kappa_{\mathbf{C}}$ is in fact an embedding, i.e. it is injective on objects.

Proof. Let $\overline{A} = (A, \Delta A, \oplus_A, +_A, 0)$ be a change action. Define

$$\begin{aligned} C_0 &\coloneqq A & C_1 &\coloneqq A \times \Delta A \\ s &\coloneqq \pi_1 : A \times \Delta A \to A & t &\coloneqq \oplus : A \times \Delta A \to A \\ i &\coloneqq \langle \mathbf{id}_A, 0_A \rangle : A \to A \times \Delta A \end{aligned}$$

To define composition, notice that

$$\begin{tikzcd} A \times \Delta A \times \Delta A \arrow[r] \arrow[d, "\oplus_A \times \mathbf{id}_{\Delta A}"'] & A \times \Delta A \arrow[d, "t = \oplus_A"] \\ A \times \Delta A \arrow[r, "s = \pi_1"'] & A \end{tikzcd}$$

is a pullback, which is to say that $A \times \Delta A \times \Delta A$ is the object of 'composable tuples.' We thus define composition by

$$c \coloneqq A \times \Delta A \times \Delta A \xrightarrow{\mathbf{id}_A \times +_A} A \times \Delta A$$

It is easy to check that the diagrams in Definition 3.3.1 commute (notably, the property that $\oplus_A$ is an action guarantees that composition is associative), and hence $\kappa_C(\overline{A}) \coloneqq (C_0, C_1, s, t, i, c)$ is a category internal to $\mathbf{C}$.

Given a differential map $\overline{f} = (f, \partial f) : \overline{A} \to \overline{B}$, we claim that the pair $\kappa_{\mathbf{C}}(f, \partial f) \coloneqq (f, \mathrm{T}\overline{f})$ is an internal functor. Evidently each component has the right type. The source and target maps commute with $\mathrm{T}f$, since π_1 and $\oplus$ are natural transformations

$\mathrm{T} \Rightarrow U$. Finally, the regularity conditions ensure that $\kappa_{\mathbf{C}}(f, \partial f)$ preserves identities (zeros) and composition (addition).

It is immediate that $\kappa_{\mathbf{C}}$ is injective on morphisms. To prove that it is surjective is not much harder: consider an internal functor $(f_0, f_1) : \kappa_{\mathbf{C}}\overline{A} \rightarrow \kappa_{\mathbf{C}}\overline{B}$. Then $\pi_2 \circ f_1 : A \times \Delta A \rightarrow \Delta B$ is a derivative for f_0, hence $(f_0, \pi_2 \circ f_1)$ is a differential map, and $T(f_0, \pi_2 \circ f_1) = (f_0, f_1)$. $\square$

Remark 3.3.2. When working in **Set**, the category $\kappa\overline{A}$ boils down to a particular instance of the Grothendieck construction: indeed, when considering the monoid ΔA as a single-object category, the data in the change action $\overline{A}$ can be encoded as a functor $\hat{\oplus}_A : \Delta A \rightarrow \mathbf{Set}$, with $A = \hat{\oplus}_A(\star)$ and $a \oplus_A \delta = \hat{\oplus}_A(\delta)(a)$.

One can then construct the category of elements $\mathrm{El}(\oplus_A)$ whose objects are pairs $(\star, a)$, with a an element of $\hat{\oplus}_A(\star) \equiv A$ and where the arrows from $(\star, a)$ to $(\star, b)$ are arrows δ in the one-object category ΔA satisfying $\hat{\oplus}_A(\delta)(a) = b$. That is to say, the arrows from $(\star, a)$ to $(\star, b)$ are precisely changes δ satisfying $a \oplus \delta = b$.

Chapter 4

Incremental computation with change actions

Equipped with the basic machinery of change actions, we set out to show how it can be applied to incremental computation, as we originally intended. For this, our driving example will be the semi-naive (that is to say, incremental) evaluation of Datalog queries, an area where incremental evaluation has immense benefits and has been broadly applied. We will use the theory of change actions and derivatives to derive, in a principled way, an incremental evaluation strategy for Datalog. While this incremental evaluation procedure is not novel, the contribution of this chapter lies in providing a solid mathematical foundation for it, which can easily be generalised or extended with minimal effort.

This chapter proceeds as follows: in Section 4.1 we give a brief overview of the evaluation mechanism for recursive Datalog queries, which will motivate the need for incremental computation. Section 4.2 introduces a family of convenient change actions on Boolean algebras which Section 4.3 uses to build explicit derivatives for the incremental evaluation of the formula semantics of Datalog. Section 4.4 explores the possibility and difficulties of building change actions on function spaces. Finally, Section 4.5 uses these "functional" change actions to incrementalise the computation of fixed points and even the least fixed point operator itself.

Many of the results in this chapter are the result of a collaboration with Semmle Ltd. and have appeare in print in [6]. Some passages and figures have been reproduced verbatim with permission.

4.1 Incremental Evaluation of Datalog Programs

Consider the following classic Datalog program [1], computing the transitive closure of an edge relation e:

$$tc(x, y) \leftarrow e(x, y)$$
$$tc(x, y) \leftarrow e(x, z) \wedge tc(z, y)$$

This pattern of recursion appears very often in Datalog programming to compute e.g. the list of nodes reachable from the root of a tree from the edge relation, or the set of ancestors of a person when $e(x, y)$ denotes the relation "x is the parent of y".

The above Datalog program can be understood as a procedure mapping denotations of $[\![e]\!]$ and $[\![tc]\!]$ for e and tc (given in extensional form as finite sets of tuples) to a new denotation for tc. The semantics of the relation tc is then the least fixed point of this procedure which, in accordance with Kleene's fixed point theorem, can be obtained by repeated iteration starting from the empty relation. Formally, $[\![tc]\!]$ is the least fixed point of the iterative procedure in Figure 4.1.

$$[\![tc]\!]_0 \leftarrow \emptyset$$
$$[\![tc]\!]_{i+1} \leftarrow [\![e]\!] \cup \left([\![e]\!] \circ [\![tc]\!]_i \right)$$

Figure 4.1: Iterative computation of tc

For example, suppose that $e = \{(1, 2), (2, 3), (3, 4)\}$. Performing the above iteration naively yields the following trace:

Iteration	Newly deduced facts	Accumulated data in $[\![tc]\!]$
0	$\emptyset$	$\emptyset$
1	$\{(1, 2), (2, 3), (3, 4)\}$	$\{(1, 2), (2, 3), (3, 4)\}$
2	$\{(1, 2), (2, 3), (3, 4), (1, 3), (2, 4)\}$	$\{(1, 2), (2, 3), (3, 4), (1, 3), (2, 4)\}$
3	$\{(1, 2), (2, 3), (3, 4),$	$\{(1, 2), (2, 3), (3, 4),$
	$(1, 3), (2, 4), (1, 4), (1, 4)\}$	$(1, 3), (2, 4), (1, 4)\}$
4	(as above)	(as above)

Figure 4.2: Execution of a recursive Datalog program

The process ends after the fourth iteration, when no new information has been added to tc and therefore a fixed point has been reached.

[1]The syntax and informal semantics of Datalog programs are outside the scope of this thesis – we refer the reader to [1, part D] for an introduction to Datalog.

However, this approach is quite wasteful: for example, the fact $tc(1,2)$ was deduced at every single iteration, despite it being known since iteration 1. More generally, if the relation e has n edges, the last iteration in the corresponding fixed point calculation will deduce $O(n^2)$ redundant facts.

The standard improvement to this evaluation strategy is known as "semi-naive" evaluation (see [1, section 13.1]), wherein the above program is transformed into an *incrementalised* version, consisting of two steps:

- A *delta* step that computes only the *new* facts at each iteration.
- An *accumulator* step that adds together the increments computed by the delta rula to obtain the final result.

In the transitive closure case the delta step is simple: the only new edges at iteration $n+1$ are the ones that can be deduced from the transitive edges that were newly deduced at iteration n. So, writing $\Delta \llbracket tc \rrbracket_i$ for the facts newly deduced at iteration i, we can rewrite the procedure in Figure 4.1:

$$\begin{aligned}
\Delta \llbracket tc \rrbracket_0 &\leftarrow \llbracket e \rrbracket \\
\Delta \llbracket tc \rrbracket_{i+1}(x,y) &\leftarrow \llbracket e \rrbracket \circ \Delta \llbracket tc \rrbracket_i \\
\llbracket tc \rrbracket_0 &\leftarrow \Delta \llbracket tc \rrbracket_0 \\
\llbracket tc \rrbracket_{i+1} &\leftarrow \llbracket tc \rrbracket_i \cup \Delta \llbracket tc \rrbracket_{i+1}
\end{aligned}$$

Figure 4.3: An incrementalised transitive closure program

Note in particular that computing $\Delta \llbracket tc \rrbracket_{i+1}$ no longer requires calculating the union of all the previously derived facts.

Iteration	$\Delta \llbracket tc \rrbracket_i$	$\llbracket tc \rrbracket_i$
0	$\{(1,2),(2,3),(3,4)\}$	$\{(1,2),(2,3),(3,4)\}$
1	$\{(1,3),(2,4)\}$	$\{(1,2),(2,3),(3,4),$ $(1,3),(2,4)\}$
2	$\{(1,4)\}$	$\{(1,2),(2,3),(3,4),$ $(1,3),(2,4),(1,4)\}$
3	$\{\}$	(as above)

Figure 4.4: The incrementalised transitive closure in action

The performance of this procedure is much better – every iteration only adds $O(n)$ facts, as we have pruned out the redundant ones. The delta transformation is a specific case of incremental computation: at each stage we only compute the change of the rule tc given the previous changes to its inputs.

But this delta translation works only for traditional Datalog. It is common to liberalise the formula syntax with additional features, such as disjunction, existential quantification, negation, and aggregation[2]. This allows one to write programs like the following, which computes whether all the nodes in a sub-tree (given by the relation *child*) have some property p:

$$treeP(x) \leftarrow p(x) \land \neg\exists y.(child(x, y) \land \neg treeP(y))$$

The body of this predicate amounts to recursion through a universal quantifier (encoded as $\neg\exists\neg$). One would like to be able to use the same semi-naive strategy for this rule too, but the standard definition of semi-naive transformation is not well defined for the extended program syntax, and it is unclear how to extend it (and its correctness proof) to handle such cases, since they hinge on the semantics of a recursive formula to be a monotone procedure.

It is possible, however, to write a delta program for *treeP* by hand; indeed, here is a definition for the delta predicate (the accumulator is as before):[3]

$$\begin{aligned}\Delta_{i+1}treeP(x) \leftarrow & p(x) \\ & \land \exists y.(child(x, y) \land \Delta_i treeP(y)) \\ & \land \neg\exists y.(child(x, y) \land \neg treeP_i(y))\end{aligned}$$

The above is a correct delta program: using it to iteratively compute $[\![treeP]\!]$ yields the same answer as the non-incremental version. It is not precise, in that it derives some facts repeatedly – there is still some degree of redundancy. In practice, there is often a trade-off between overly-precise incremental rules (which derive fewer redundant facts but might be harder to compute) and excessively redundant ones.

Handling extended Datalog is of more than theoretical interest – part of the research in this work was undertaken in collaboration with Semmle, a program analysis company which makes heavy use of a commercial Datalog implementation to implement large-scale static program analysis [94, 11, 95, 92]. Semmle's implementation includes parity-stratified negation[4], recursive aggregates [78], and other non-standard features which are not amenable to the traditional semi-naive approach.

[2]See, for example, LogiQL [68, 52], Datomic [33], Souffle [99, 93], and DES [91], which support all of the aforementioned features and more. We do not here explore supporting extensions to the syntax of rule heads, although as long as they allow for a denotational semantics in a similar style our techniques should be applicable.

[3]This rule should be read as: we can newly deduce that x is in *treeP* if x satisfies the predicate, and we have newly deduced that one of its children is in *treeP*, and we currently believe that all of its children are in *treeP*.

[4]Parity-stratified negation means that recursive calls must appear under an even number of negations. This ensures that the overall rules remain monotone, so the least fixed point still exists.

A similar scenario happens when considering *maintenance* of Datalog programs, that is to say, updating the semantics of a Datalog program when one of the involved relations changes; for example, updating the value of tc given a change to e. Again, this is a natural fit for incremental computation, and there are known solutions for traditional Datalog [50], but these break down when the language is extended.

There is a piece of folkloric knowledge in the Datalog community that hints at a solution: the semi-naive translation of a rule corresponds to the *derivative* of that rule[13, 12, section 3.2.2]. This suggests that the incremental version of a Datalog program should correspond to its derivative, given a suitable change action on its semantics.

4.2 Change Actions for Datalog

For this case study, we aim to flesh out the folkloric idea that incremental computation behaves like a derivative. We approach this goal in stages: first, we establish some basic properties of change actions which will be of use later (paying particular attention to change actions in partial orders and Boolean algebras). Then, we apply use these constructions to build a change action on a domain of relations where Datalog can be interpreted and show how to differentiate (possibly non-monotone) Datalog formulae. Finally, we show how these derivatives can be used to incrementalise the computation of fixed points.

4.2.1 Transitive Change Actions and Difference Operators

A particularly convenient class of change actions are those which are *transitive*, that is to say, where every element can be transformed into any other element via some change (in the language of monoid actions, a change action is transitive whenever it has a single *orbit*).

Definition 4.2.1. A change action $\overline{A}$ is **transitive** if for any $a, b \in A$, there exists a change $\delta \in \Delta A$ such that $a \oplus \delta = b$.

Example 4.2.1. Whenever $(A, +, 0)$ is a group, then the monoidal change action $(A, A, +, +, 0)$ is transitive: given points a_1, a_2 in A, the change $-a_1 + a_2$ sends a_1 to a_2.

Example 4.2.2. For any set A, the change action $\overline{A}_{\Rightarrow} := (A, A \Rightarrow A, \mathbf{ev}_{A,A}, \circ, \mathbf{id}_A)$ is transitive: given points a_1, a_2, the constant function sending every point to a_2 in particular sends a_1 to a_2 (as do many other changes).

An equivalent, and more useful, characterisation of transitive change actions is that they are actions that allow for finding the "difference" $a \ominus b$ between two points.[5]

Definition 4.2.2. A *difference operator* for $\overline{A}$ is a function $\ominus : A \times A \to \Delta A$ such that $a \oplus (b \ominus a) = b$ for all $a, b \in A$.

Proposition 4.2.1. Given a minus operator $\ominus$, and a function f, let

$$\partial f_{\ominus}(a, \delta) \coloneqq f(a \oplus \delta) \ominus f(a)$$

Then $\partial f_{\ominus}$ is a derivative for f.

Proposition 4.2.2. Let $\overline{A}$ be a change action. Then the following are equivalent:

i. $\overline{A}$ is transitive.

ii. There is a minus operator on $\overline{A}$.

iii. For any change action $\overline{B}$ all functions $f : B \to A$ are differentiable.

Proof.

- ii. $\Rightarrow$ iii. is a straightforward consequence of Proposition 4.2.1.
- iii. $\Rightarrow$ i.: Given a, b in A, we define the path $f_{ab} : \mathbb{N} \to A$ by:

  $$f_{ab}(n) \coloneqq \begin{cases} a & \text{if } n = 0 \\ b & \text{otherwise} \end{cases}$$

  By hypothesis, f_{ab} admits a derivative $\partial[f_{ab}]$, and therefore $b = a \oplus \partial[f_{ab}](0, 1)$, thus $\overline{A}$ is transitive.
- i. $\Rightarrow$ ii.: Let $\overline{A}$ be transitive. We apply the Axiom of Choice to pick a δ_{ab} for every pair a, b of elements of A such that $a \oplus \delta_{ab} = b$. Then clearly $b \ominus a \coloneqq \delta_{ab}$ is a difference operator. $\square$

In practice this last property is of the utmost importance: since we are concerned with the differentiability of functions, working with transitive change actions will guarantee that derivatives always exist.

[5]Cai and Giarrusso's original definition of change structures was in fact given in terms of such operators.

4.2.2 Comparing Change Actions

Much like topological spaces and partial orders, we can compare change actions on the same base set according to coarseness. This is useful since differentiability of functions between change actions is characterised entirely by the coarseness of the actions.

Definition 4.2.3. Let $\overline{A}_1$ and $\overline{A}_2$ be change actions with the same underlying set A. We say that $\overline{A}_1$ is **coarser** than $\overline{A}_2$ (or that $\overline{A}_2$ is **finer** than $\overline{A}_1$) whenever for every $a \in A$ and change $\delta_1 \in \Delta A_1$, there is a change $\delta_2 \in \Delta A_2$ such that $a \oplus_{A_1} \delta_1 = a \oplus_{A_2} \delta_2$.

We will write $\overline{A}_1 \leq \overline{A}_2$ whenever $\overline{A}_1$ is coarser than $\overline{A}_2$. If $\overline{A}_1$ is both finer and coarser than $\overline{A}_2$, we will say that $\overline{A}_1$ and $\overline{A}_2$ are equivalent.

The relation $\leq$ defines a preorder (but not a partial order) on the set of all change actions over a fixed set A. Least and greatest elements exist (up to equivalence), and correspond respectively to the discrete change action $\overline{A}_\top$ and any transitive change action, such as the ones in Example 4.2.1 or Example 4.2.2.

Proposition 4.2.3. Let $\overline{A}_2 \leq \overline{A}_1$, $\overline{B}_1 \leq \overline{B}_2$ be change actions, and suppose the function $f : A \to B$ is differentiable as a function from $\overline{A}_1$ into $\overline{B}_1$. Then f is differentiable as a function from $\overline{A}_2$ into $\overline{B}_2$.

A consequence of this fact is that whenever two change actions are equivalent they can be used interchangeably without affecting which functions are differentiable. One last parallel with topology is the following result, which establishes a simple criterion for when a change action is coarser than another:

Proposition 4.2.4. Let $\overline{A}_1, \overline{A}_2$ be change actions on A. Then $\overline{A}_1$ is coarser than $\overline{A}_2$ if and only if the identity function $\mathbf{id}_A : A \to A$ is differentiable from $\overline{A}_1$ to $\overline{A}_2$.

The semantic domain of Datalog is a complete Boolean algebra, and so our next step is to construct a suitable change action for Boolean algebras. Along the way, we consider change actions over posets, which give us the ability to entirely characterise the set of derivatives of a given function in terms of the poset structure. This turns out to be desirable in practice, as it gives one a broad range of derivatives from which an "optimal" one might be selected.

4.2.3 Change Actions on Posets

Ordered sets give us a constrained class of functions: monotone functions. We can define *ordered* change actions, which are those that are well-behaved with respect to the order on the underlying set. – that is to say, an ordered change action is precisely a change action internal to the category of posets and monotone functions.

Definition 4.2.4. A change action $\overline{A}$ is **ordered** if

- A and ΔA are posets.
- $\oplus$ is monotone as a map from $A \times \Delta A \to A$
- $+$ is monotone as a map from $\Delta A \times \Delta A \to \Delta A$

In practice, instead of assuming a partial order on the change set ΔA and restricting ourselves to ordered change actions, we will often start with a change action $\overline{A}$ over some poset A and lift the partial order on A to one on ΔA which is compatible with the change action structure.

Definition 4.2.5. Given a change action $\overline{A}$ and a partial order $\leq$ on A, we define a partial order $\leq_\Delta$ on ΔA by

$$\delta_1 \leq_\Delta \delta_2 \quad \Leftrightarrow \quad \forall a \in A.\ a \oplus \delta_1 \leq a \oplus \delta_2$$

Proposition 4.2.5. Let $\overline{A}$ be a change action on a set A equipped with a partial order $\leq$ such that $\oplus$ is monotone in its first argument. Then $\overline{A}$ is an ordered change action when ΔA is equipped with the partial order $\leq_\Delta$.

Proof. Monotonicity of $\oplus$ is immediate. It remains to show that $+$ is monotone. Suppose $\delta_1 \leq \delta_1'$ and $\delta_2 \leq \delta_2'$. Then, for any $a \in A$ we have:

$$\begin{aligned} a \oplus (\delta_1 + \delta_2) &= (a \oplus \delta_1) \oplus \delta_2 \\ &\leq (a \oplus \delta_1') \oplus \delta_2 \\ &\leq (a \oplus \delta_1') \oplus \delta_2' \\ &= a \oplus (\delta_1' + \delta_2') \end{aligned}$$ □

In what follows, we will extend the partial order $\leq_\Delta$ on some change set ΔB pointwise to sets of functions (not necessarily differentiable) from some A into ΔB. This pointwise order implies that the space of valid derivatives of a function is convex, that is to say, all functions between any two derivatives for f are themselves derivatives for f.

Theorem 4.2.1 (Sandwich lemma)**.** Let $\overline{A}, \overline{B}$ be change actions, with $\overline{B}$ ordered, and let $f : A \to B$ admit derivatives $\partial_{\downarrow} f, \partial_{\uparrow} f$. Then any $g : A \times \Delta A \to \Delta B$ satisfying

$$\partial_{\downarrow} f \leq_{\Delta} g \leq_{\Delta} \partial_{\uparrow} f$$

is a derivative for f.

Proof. Take arbitrary $a \in A, \delta \in \Delta A$. By hypothesis we have:

$$\partial_{\downarrow} f(a, \delta) \leq_{\Delta} g(a, \delta) \leq_{\Delta} \partial_{\uparrow} f(a, \delta)$$

Then the first inequality gives:

$$f(a \oplus \delta) = f(a) \oplus \partial_{\downarrow} f(a, \delta) \leq f(a) \oplus g(a, \delta)$$

Applying the second inequality we obtain:

$$f(a) \oplus g(a, \delta) \leq f(a) \oplus \partial_{\uparrow} f(a, \delta) = f(a \oplus \delta)$$

Hence $f(a) \oplus g(a, \delta) = f(a \oplus \delta)$ so g is a derivative for f. $\square$

Whenever unique minimal and maximal derivatives exist, the above theorem suffices to completely characterise all the derivatives for a function as a closed interval.

Corollary 4.2.1. Let $\overline{A}$ and $\overline{B}$ be change actions, with $\overline{B}$ ordered, and let $f : A \to B$ be a function. If there exist $\partial_{\bot} f$ and $\partial_{\top} f$ which are unique minimal and maximal derivatives of f, respectively, then the derivatives of f are precisely the functions ∂f such that

$$\partial_{\bot} f \leq_{\Delta} \partial f \leq_{\Delta} \partial_{\top} f$$

The importance of this result will become clearer when we describe our choice of a change action for Datalog programs. In that setting, unique minimal and maximal derivatives do indeed exist and have particularly simple descriptions, which we can then use to check whether a given strategy for incremental evaluation is correct.

4.2.4 Boolean Algebras

The semantics of a Datalog predicate is usually taken to be a (finite) set of tuples. Under this interpretation, conjunction and disjunction correspond to intersection and union, and every plain Datalog program is a monotone map.

If we were to restrict ourselves to this setting, it would be reasonable to focus on change actions of the form $\overline{D}_{\cup} \coloneqq (\mathcal{P}(D), \mathcal{P}(D), \cup, \cup, \emptyset)$, with D being some (finite) universe containing all possible tuples. The semantics of a Datalog program with n relations would then be the least fixed point of some monotone map $\phi : \mathcal{P}(D)^n \to \mathcal{P}^n$.

This set-up would allow us to give an account of "classic" semi-naive evaluation, that is, incremental evaluation of Datalog without non-monotone extensions. This is because the above change action $\overline{D_\sqcup}$ has the convenient property that any monotone endomorphism $f : \mathcal{P}(D) \to \mathcal{P}(D)$ is differentiable – in particular, so is every Datalog program.

Such an approach cannot, however, integrate such features as parity-stratified negation or universal quantification: any Datalog program that uses such extensions must remain monotone (otherwise its execution might fail to converge to a fixed point), but it may not be possible to give its semantics as a composition of monotone functions (that is, it may contain non-monotone subterms).

Instead of working with the change action $\overline{D_\sqcup}$, we will show how to endow any Boolean algebra with a much richer change action that allows for the differentiation of arbitrary functions.

Proposition 4.2.6. Let L be a Boolean algebra. Define

$$\overline{L_\bowtie} := (L, L \bowtie L, \oplus_\bowtie, +, (\bot, \bot))$$

where

$$\begin{aligned} L \bowtie L &:= \{(a,b) \in L \times L \mid a \wedge b = \bot\} \\ a \oplus_\bowtie (p,q) &:= (a \vee p) \wedge \neg q \\ (p,q) \bowtie (r,s) &:= ((p \wedge \neg s) \vee r, (q \wedge \neg r) \vee s) \end{aligned}$$

with identity element $(\bot, \bot)$. Then $\overline{L_\bowtie}$ is a complete change action on L.

Proof. First we need to check that the monoid addition $\bowtie$ is well-defined – that is, whenever $(p,q), (r,s) \in L \bowtie L$ then $(p,q) \bowtie (r,s) \in L \bowtie L$. Writing the above in disjunctive normal form and cancelling out trivial terms we obtain:

$$(p,q) \bowtie (r,s) = (p \wedge \neg s \wedge q \wedge \neg r) \vee (r \wedge s)$$

Since $(p,q), (r,s) \in L \bowtie L$, we have that both disjunctive terms in the right-hand side are equal to $\bot$. It remains to prove that $\oplus_\bowtie$ is indeed a monoid action. We show that it is compatible with $\bowtie$:

$$\begin{aligned}
&a \oplus_{\bowtie} [(p, q) \bowtie (r, s)] \\
&= a \oplus_{\bowtie} ((p \wedge \neg s) \vee r, (q \wedge \neg r) \vee s) \\
&= (a \vee ((p \wedge \neg s) \vee r)) \wedge \neg ((q \wedge \neg r) \vee s) \\
&= (((a \vee p) \wedge (a \vee \neg s)) \vee r) \wedge (\neg q \vee r) \wedge \neg s \\
&= (((a \vee p) \wedge (a \vee \neg s) \wedge \neg q) \vee r) \wedge \neg s \\
&= (((a \vee p) \wedge \neg q) \vee r) \wedge \neg s \\
&= a \oplus_{\bowtie} (p, q) \oplus_{\bowtie} (r, s)
\end{aligned}$$ □

We can think of $\overline{L}_{\bowtie}$ as tracking changes as pairs of "additions" and "deletions" to a piece of data, where the monoid action simply applies one after the other. The condition on $\bowtie$ then stipulates that changes never "undo" any of their own work.

Example 4.2.3. In the powerset Boolean algebra $\mathcal{P}(\mathbb{N})$, a change to a set of numbers $\{1, 2\}$ might consist of *adding* $\{3, 4\}$ and *removing* $\{1\}$. In $\overline{\mathbb{N}}_{\bowtie}$ this change corresponds to the element $(\{3, 4\}, \{1\})$. Applying it to the set $\{1, 2, 3\}$ would yield:

$$\{1, 2, 3\} \oplus_{\bowtie} (\{3, 4\}, \{1\}) = \{2, 3, 4\}$$

Boolean algebras are naturally endowed with the structure of a partial order, where $a \leq b$ whenever $a \vee b = b$. Furthermore, we observe that, whenever $a \leq b$, then for any change (p, q) we have $a \oplus_{\bowtie} (p, q) \leq b \oplus_{\bowtie} (p, q)$. Therefore, by Proposition 4.2.5, we know that $\overline{L}_{\bowtie}$ is in fact an ordered change action.

Proposition 4.2.7. The change action $\overline{L}_{\bowtie}$ is ordered, when L is endowed with its natural partial order as a Boolean algebra and $L \bowtie L$ is endowed with the order

$$(p, q) \leq_{\bowtie} (r, s) \Leftrightarrow (p \leq r) \wedge (q \geq s)$$

That is to say, a change (p, q) is smaller than a change (r, s) whenever it *adds fewer elements* $(p \leq r)$ and it *removes more elements* $(q \geq s)$.

Under the above order, functions into a Boolean algebra endowed with this change action in fact have unique maximal and minimal derivatives. Theorem 4.2.1 then gives us bounds for all the derivatives on Boolean algebras.

Proposition 4.2.8. Let L be a Boolean algebra with the $\overline{L}_{\bowtie}$ change action, and $f : A \to L$ be a function. Then, the following are minus operators:

$$\begin{aligned} a \ominus_{\bot} b &= (a \wedge \neg b, \neg a) \\ a \ominus_{\top} b &= (a, b \wedge \neg a) \end{aligned}$$

Additionally, $\partial f_{\ominus_{\bot}}$ and $\partial f_{\ominus_{\top}}$ (defined as in Proposition 4.2.1) are unique least and greatest derivatives for f respectively.

Proof. We show that $\ominus_{\bot}$ is indeed a minus operator:

$$\begin{aligned} b \oplus_{\bowtie} (a \ominus_{\bot} b) &= (b \vee (a \wedge \neg b)) \wedge \neg(\neg a) \\ &= (b \vee (a \wedge \neg b)) \wedge a \\ &= (b \wedge a) \vee (a \wedge \neg b) \\ &= a \end{aligned}$$

Similarly for $\ominus_{\top}$:

$$\begin{aligned} b \oplus_{\bowtie} (a \ominus_{\top} b) &= (b \vee a) \wedge \neg(b \wedge \neg a) \\ &= (b \vee a) \wedge (\neg b \vee a) \\ &= a \end{aligned}$$

It remains to prove that the corresponding derivatives are least and greatest – this is a trivial consequence of the following result:

Lemma 4.2.1. For any $a, b \in L$, the change $a \ominus_{\bot} b$ is the least element of the set of changes $(p, q) \in L \bowtie L$ such that $b \oplus_{\bowtie} (p, q) = a$ (and, conversely, $a \ominus_{\top} b$ is the greatest such change).

Proof. Suppose $b \oplus_{\bowtie} (p, q) = a$. Then $(b \vee p) \wedge \neg q = a$. On one hand, this implies $a \leq \neg q$ and so we obtain $\neg a \geq q$. On the other hand, it also implies $a \leq (b \vee p)$, therefore $a \wedge \neg b \leq p$. Hence $(a \wedge \neg b, \neg a) \leq_{\bowtie} (p, q)$ as desired.

□

As an immediate consequence of this result, we can apply Theorem 4.2.1 in practice, since it provides us with concrete bounds for derivatives into Boolean algebras. We will make use of this property in Section 4.3.2 to characterise all possible derivatives of a Datalog program.

Corollary 4.2.2. Let L be a Boolean algebra with the corresponding change action $\overline{L}_{\bowtie}$, $\overline{A}$ be an arbitrary change action, and $f : A \to L$ be a function. Then the derivatives of f are precisely those functions $\partial f : A \times \Delta A \to L \bowtie L$ such that

$$\partial f_{\ominus\bot} \leq_{\Delta} \partial f \leq_{\Delta} \partial f_{\ominus\top}$$

4.3 Derivatives for Non-Recursive Datalog

We now seek to apply the theory we have developed to the specific case of the semantics of Datalog. Giving a differentiable semantics for Datalog will lead us to a strategy for performing incremental evaluation and maintenance of Datalog programs. To begin with, we will restrict ourselves to the non-recursive fragment of the language – the formulae that make up the right hand sides of Datalog rules. That is to say, we will not incrementalise the computation of an entire fixed point; rather, we will only show how to incrementalise the logical semantics of a single predicate.

We should also like to note that the techniques we are using should work for any language. We have chosen Datalog as a non-trivial case study where the need for incremental computation is real and pressing, but also because the use of "derivatives" for incrementalising Datalog programs is so prevalent in the literature.

4.3.1 Semantics of Datalog Formulae

Datalog is usually given a logical semantics where formulae are interpreted as first-order logic predicates and the semantics of a program is the set of models of its constituent predicates. We will instead give a simple denotational semantics (as is typical when working with fixed points, see e.g. [28]) that treats a Datalog formula as directly denoting a relation[6] with variables ranging over a finite schema.

Definition 4.3.1. A **schema** Γ is a finite set of names. A **named tuple** over Γ is an assignment of a value v_i for each name x_i in Γ. Given disjoint schemata $\Gamma = \{x_1, \ldots, x_n\}$ and $\Sigma = \{y_1, \ldots, y_m\}$, the **selection function** σ_Γ is defined as

$$\sigma_\Gamma(\{x_1 \mapsto v_1, \ldots, x_n \mapsto v_n, y_1 \mapsto w_1, \ldots, y_m \mapsto w_m\}) \coloneqq \{x_1 \mapsto v_1, \ldots, x_n \mapsto v_n\}$$

i.e. σ_Γ restricts a named tuple over $\Gamma \cup \Sigma$ into a tuple over Γ with the same values for the names in Γ.

[6]Throughout this section we use the word "relation" to denote a set of named tuples, as is usual in database theory, as opposed to the more familiar definition of a mathematical relation.

$$\llbracket \top \rrbracket_\Gamma (\mathcal{R}) \coloneqq \mathcal{U}_\Gamma \qquad \llbracket T \wedge U \rrbracket_\Gamma (\mathcal{R}) \coloneqq \llbracket T \rrbracket_\Gamma (\mathcal{R}) \cap \llbracket U \rrbracket_\Gamma (\mathcal{R})$$
$$\llbracket \bot \rrbracket_\Gamma (\mathcal{R}) \coloneqq \emptyset \qquad \llbracket T \vee U \rrbracket_\Gamma (\mathcal{R}) \coloneqq \llbracket T \rrbracket_\Gamma (\mathcal{R}) \cup \llbracket U \rrbracket_\Gamma (\mathcal{R})$$
$$\llbracket \neg T \rrbracket_\Gamma (\mathcal{R}) \coloneqq \neg \llbracket T \rrbracket_\Gamma (\mathcal{R})$$

$$\llbracket R_j(x_1, \ldots, x_n) \rrbracket_\Gamma (\mathcal{R}) \coloneqq \{ u \in \mathcal{U}_\Gamma \mid \rho_{x_1,\ldots,x_n/\Gamma_j}(\sigma_{x_1,\ldots,x_n}(u)) \in \mathcal{R}_j \}$$
$$\llbracket \exists x.T \rrbracket_\Gamma (\mathcal{R}) \coloneqq \sigma_\Gamma(\llbracket T \rrbracket_{\Gamma \cup \{x\}} (\mathcal{R}))$$

Figure 4.5: Formula semantics for Datalog

Given signatures Γ_1, Γ_2 with the same cardinality The **renaming function** ρ_{Γ_1/Γ_2} transforms a Γ_2-tuple into a Γ_1-tuple by permuting the names[7]:

$$\rho_{\{x_i\}/\{y_i\}}(\{y_i \mapsto v_i\}) \coloneqq \{x_i \mapsto v_i\}$$

We also write σ_Γ and ρ_{Γ_1/Γ_2} to denote their respective extension to sets of tuples.

We will adopt the usual closed-world assumption to give a denotation to negation. That is to say, for every schema Γ, we will fix a set $\mathcal{U}_\Gamma$ of named tuples over Γ, the "universal relation", representing the set of all the Γ-tuples that are allowed in our programs.

Definition 4.3.2. We write $\mathbf{Rel}_\Gamma$ to denote the set of relations over Γ, that is, the set of all subsets of $\mathcal{U}_\Gamma$.

Negation on relations can then be defined as

$$\neg R \coloneqq \mathcal{U}_\Gamma \setminus R$$

This makes $\mathbf{Rel}_\Gamma$, the set of all subsets of $\mathcal{U}_\Gamma$, a complete Boolean algebra.

Definition 4.3.3. Given a Datalog formula T whose free term variables are contained in Γ, and which depends on n formula constants R_i, its denotation $\llbracket T \rrbracket_\Gamma$ is a function

$$\llbracket T \rrbracket_\Gamma : \Pi_{i=1}^n \mathbf{Rel}_{\Gamma_i} \to \mathbf{Rel}_\Gamma$$

where Γ_i is the signature for the constant R_i.

If $\mathcal{R} = (\mathcal{R}_1, \ldots, \mathcal{R}_n)$ is a choice of a relation $\mathcal{R}_i$ for each of the variables R_i, $\llbracket T \rrbracket (\mathcal{R})$ is inductively defined according to the rules in 4.5.

Since $\mathbf{Rel}_\Gamma$ is a complete Boolean algebra, and so is $\mathbf{Rel}_{\Gamma_i}$, $\llbracket T \rrbracket_\Gamma$ is a function between complete Boolean algebras. For brevity, we will often omit the schemata whenever they can be inferred from the context.

[7]Strictly speaking, this function ρ is not well-defined, as it assumes that both sets are rather ordered sequences. In practice, the free variables involved in a Datalog formula are implicitly ordered by the position in which they appear in its definition, and so this does not cause any issues.

4.3.2 Differentiability of Datalog Formula Semantics

In order to actually perform our incremental computation, we first need to provide a concrete derivative for the semantics of Datalog formulae. Of course, since $[\![T]\!]_\Gamma$ is a function between the complete Boolean algebras $\Pi\mathbf{Rel}_{\Gamma_i}$ and $\mathbf{Rel}_\Gamma$, and we know that the corresponding change actions $\overline{\Pi\mathbf{Rel}_{\Gamma_i}}_{\bowtie}$ and $\overline{\mathbf{Rel}_\Gamma}_{\bowtie}$ are complete, this guarantees the existence of a derivative for $[\![T]\!]$.

Unfortunately, this does not necessarily provide us with an *efficient* derivative for $[\![T]\!]$. The derivatives that we know how to compute (the least and greatest derivatives defined in Corollary 4.2.2) rely on computing $f(a \oplus \delta a)$ itself, which is precisely the recalculation that incremental computation seeks to avoid. Technically both of these derivatives *do* correspond to strategies for incremental computation: in particular, the "trivial" strategy where one simply throws away the partial result $f(a)$ and computes $f(a \oplus \delta a)$ from scratch.

Of course, given a concrete definition of a derivative we can simplify this expression and hopefully make it easier to compute. But we also know from Corollary 4.2.2 that *any* function bounded by $\partial f_{\ominus_\bot}$ and $\partial f_{\ominus_\top}$ is a valid derivative, and we can therefore optimise anywhere within that range to make a trade-off between ease of computation and precision.[8]

There is also the question of how to effectively compute the derivative – the notion of derivative is a purely *semantic* one and, while we know that derivatives do exist, we also need to construct a concrete Datalog program that computes it. Since the change set for $\overline{\mathbf{Rel}}_{\bowtie}$ is a subset of $\mathbf{Rel} \times \mathbf{Rel}$, it is possible and indeed very natural to compute the two components via a pair of Datalog formulae, which allows us to reuse an existing Datalog formula evaluator. Indeed, if this process is occurring in an optimising compiler, the evaluation of the derivative formulae themselves can be optimised. This is very beneficial in practice, since the initial formulae may be quite complex.

This does give us additional constraints that the derivative formulae must satisfy: for starters, we need to be able to evaluate them; and we may wish to pick formulae that will be easy or cheap for our evaluation engine to compute, even if they compute a less precise derivative.

The upshot of these considerations is that the optimal choice of derivatives is likely to be quite dependent on the precise variant of Datalog being evaluated, and the

[8]The idea of using an approximation to the precise derivative, and a soundness condition, appears in Bancilhon [13].

$$
\begin{aligned}
\Delta(\bot) &\coloneqq \bot & \nabla(\bot) &\coloneqq \bot \\
\Delta(\top) &\coloneqq \bot & \nabla(\top) &\coloneqq \bot \\
\Delta(R_j) &\coloneqq \Delta R_j & \nabla(R_j) &\coloneqq \nabla R_j \\
\Delta(T \vee U) &\coloneqq \Delta(T) \vee \Delta(U) & \nabla(T \vee U) &\coloneqq (\nabla(T) \wedge \neg \mathsf{X}(U)) \\
\Delta(T \wedge U) &\coloneqq (\Delta(T) \wedge \mathsf{X}(U)) & &\quad \vee (\nabla(U) \wedge \neg \mathsf{X}(T)) \\
&\quad \vee (\Delta(U) \wedge \mathsf{X}(T)) & \nabla(T \wedge U) &\coloneqq (\nabla(T) \wedge U) \vee (T \wedge \nabla(U)) \\
\Delta(\neg T) &\coloneqq \nabla(T) & \nabla(\neg T) &\coloneqq \Delta(T) \\
\Delta(\exists x.T) &\coloneqq \exists x.\Delta(T) & \nabla(\exists x.T) &\coloneqq \exists x.\nabla(T) \wedge \neg \exists x.\mathsf{X}(T)
\end{aligned}
$$

$$
\mathsf{X}(R) \coloneqq (R \vee \Delta(R)) \wedge \neg \nabla(R)
$$

Figure 4.6: Upwards and downwards formula derivatives for Datalog

specifics of the evaluation engine. The version that we present here is a simplification of the formula actually in use in the Datalog engine developed by Semmle Ltd..

4.3.2.1 A Concrete Datalog Formula Derivative

Figure 4.6, defines a "symbolic" derivative operator as a pair of mutually recursive functions, Δ and ∇, which turn a Datalog formula T into new formulae that compute the upwards and downwards parts of the derivative, respectively. Our definition uses an auxiliary function, X, which computes the "neXt" value of a term by applying the upwards and downwards derivatives. As is typical for a derivative, the new formulae will have additional free relation variables for the upwards and downwards derivatives of the free relation variables of T, denoted as ΔR_i and ∇R_i respectively. Evaluating the formula as a derivative means evaluating it as a normal Datalog formula with the new relation variables set to the input relation changes.

While the definitions mostly exhibit the dualities we would expect between corresponding operators, there are a few asymmetries to explain.

The asymmetry between the cases for $\Delta(T \vee U)$ and $\nabla(T \wedge U)$ is for operational reasons. The symmetrical version of $\Delta(T \vee U)$ is $(\Delta(T) \wedge \neg U) \vee (\Delta(U) \wedge \neg T)$ (which is also precise). The reason we omit the negated conjuncts is simply that they are costly to compute and not especially helpful to our evaluation engine. On the other hand, while we could have set $\nabla(T \wedge U) \coloneqq \nabla(T) \vee \nabla(U)$ (and this would lead to a valid derivative), we prefer the present form as it results in smaller relations at no cost (since the extensions of T and U are already available at this point).

The asymmetry between the cases for $\exists$ is because our dialect of Datalog does not have a primitive universal quantifier. If we did have one, the cases for $\exists$ would be dual to the corresponding cases for $\forall$.

Theorem 4.3.1. Let Δ, ∇, X : Formula $\to$ Formula be mutually recursive functions defined by structural induction as in Figure 4.6.

Then $\Delta(T)$ and $\nabla(T)$ are disjoint, and for any schema Γ and any Datalog formula T whose free term variables are contained in Γ,

$$\partial[\![T]\!]_\Gamma(\mathcal{R}_1 \ldots, \delta\mathcal{R}_1 \ldots) \coloneqq ([\![\Delta(T)]\!]_\Gamma(\mathcal{R}_i \ldots, \delta\mathcal{R}_i \ldots), [\![\nabla(T)]\!]_\Gamma(\mathcal{R}_i \ldots, \delta\mathcal{R}_i \ldots))$$

is a derivative for $[\![T]\!]_\Gamma$.

Proof. Let T be a Datalog formula with free relation variables $R_1, \ldots, R_n$, a choice of a semantics for the free relation variables $\mathcal{R}_1, \ldots, \mathcal{R}_n \in \mathbf{Rel}$ and differences $\delta\mathcal{R}_1, \ldots, \delta\mathcal{R}_n \in \mathbf{Rel} \bowtie \mathbf{Rel}$. For brevity, we refer to the tuple $(\mathcal{R}_1, \ldots, \mathcal{R}_n)$ as $\mathcal{R}$ and the tuple $(\delta\mathcal{R}_1, \ldots, \delta\mathcal{R}_n)$ as $\delta\mathcal{R}$. We abuse the notation and refer to $(\mathcal{R}_1 \bowtie \delta\mathcal{R}_1, \ldots, \mathcal{R}_n \bowtie \delta\mathcal{R}_n)$ as $\mathcal{R} \bowtie \delta\mathcal{R}$. We will also omit the arguments to $[\![\Delta T]\!]$ and $[\![\nabla T]\!]$ often, as there is never room for ambiguity.

We want to show:

$$[\![T]\!]_\Gamma(\mathcal{R} \bowtie \delta\mathcal{R}) = [\![T]\!]_\Gamma(\mathcal{R}) \bowtie ([\![\Delta T]\!]_\Gamma(\delta\mathcal{R}), [\![\nabla T]\!]_\Gamma(\delta\mathcal{R}))$$

We proceed by structural induction on T. We omit the cases for $\top$, $\bot$ and relational variables for being trivial.

For the conjunction case, we make use of the following elementary set-theoretic identity:

Proposition 4.3.1. For any sets A, B, C such that $C \subseteq B$ and $A \subseteq \neg B \cup C$, we have $A \cap \neg B = A \cap \neg C$.

Proof.

$$\begin{aligned}
A \cap \neg C &= A \cap \neg C \cap (\neg B \cup C) && \text{(since } A \subseteq \neg B \cup C\text{)} \\
&= A \cap ((\neg C \cap \neg B) \cup (\neg C \cap C)) && \text{(distributivity)} \\
&= A \cap (\neg C \cap \neg B) && \text{(cancelling out the empty set)} \\
&= A \cap \neg(C \cup B) && \text{(De Morgan's)} \\
&= A \cap \neg B && \text{(since } C \subseteq B\text{)}
\end{aligned}$$

With the above in mind, the proof of correctness for the conjunctive case follows by the following straightforward, if lengthy, derivation.

$$\begin{aligned}
&[\![T \wedge U]\!]_\Gamma(\mathcal{R} \oplus_{\bowtie} \delta\mathcal{R}) \\
&= [\![T]\!]_\Gamma(\mathcal{R} \oplus_{\bowtie} \delta\mathcal{R}) \cap [\![U]\!]_\Gamma(\mathcal{R} \oplus_{\bowtie} \delta\mathcal{R}) && \text{(semantics of } \wedge\text{)}
\end{aligned}$$

$$
\begin{aligned}
&= [\![T]\!]_\Gamma(\mathcal{R}) \oplus_{\bowtie} ([\![\Delta T]\!]_\Gamma, [\![\nabla T]\!]_\Gamma) && \text{(induction hypothesis)}\\
&\qquad \cap [\![U]\!]_\Gamma(\mathcal{R}) \oplus_{\bowtie} ([\![\Delta U]\!]_\Gamma, [\![\nabla U]\!]_\Gamma)\\
&= ([\![T]\!]_\Gamma(\mathcal{R}) \cup [\![\Delta T]\!]_\Gamma) \cap \neg [\![\nabla T]\!]_\Gamma && \text{(definition of } \oplus_{\bowtie})\\
&\qquad \cap ([\![U]\!]_\Gamma(\mathcal{R}) \cup [\![\Delta U]\!]_\Gamma) \cap \neg [\![\nabla U]\!]_\Gamma\\
&= ([\![T]\!]_\Gamma(\mathcal{R}) \cap \neg [\![\nabla T]\!]_\Gamma \cup [\![\Delta T]\!]_\Gamma) && \text{(disjointness of } [\![\Delta]\!], [\![\nabla]\!])\\
&\qquad \cap ([\![U]\!]_\Gamma(\mathcal{R}) \cap \neg [\![\nabla U]\!]_\Gamma \cup [\![\Delta U]\!]_\Gamma)\\
&= ([\![T]\!]_\Gamma(\mathcal{R}) \cap \neg [\![\nabla T]\!]_\Gamma) && \text{(distributing } [\![\Delta T]\!] \text{ out)}\\
&\qquad \cap ([\![U]\!]_\Gamma(\mathcal{R}) \cap \neg [\![\nabla U]\!]_\Gamma \cup [\![\Delta U]\!]_\Gamma)\\
&\qquad \cup ([\![\Delta T]\!]_\Gamma \cap ([\![U]\!]_\Gamma(\mathcal{R}) \cup [\![\Delta U]\!]_\Gamma) \cap \neg [\![\nabla U]\!]_\Gamma)\\
&= \Big([\![T]\!]_\Gamma(\mathcal{R}) \cap \neg [\![\nabla T]\!]_\Gamma \cap [\![U]\!]_\Gamma(\mathcal{R}) \cap \neg [\![\nabla U]\!]_\Gamma \Big) && \text{(distributing } [\![\Delta U]\!] \text{ out)}\\
&\qquad \cup \Big([\![\Delta T]\!]_\Gamma \cap ([\![U]\!]_\Gamma(\mathcal{R}) \cup [\![\Delta U]\!]_\Gamma) \cap \neg [\![\nabla U]\!]_\Gamma \Big)\\
&\qquad \cup \Big([\![\Delta U]\!]_\Gamma \cap ([\![T]\!]_\Gamma(\mathcal{R}) \cup [\![\Delta T]\!]_\Gamma) \cap \neg [\![\nabla T]\!]_\Gamma \Big)\\
&= \Big[([\![T]\!]_\Gamma(\mathcal{R}) \cap [\![U]\!]_\Gamma(\mathcal{R})) && \text{(disjointness of } [\![\Delta]\!], [\![\nabla]\!])\\
&\qquad \cup \Big([\![\Delta T]\!]_\Gamma \cap ([\![U]\!]_\Gamma(\mathcal{R}) \cup [\![\Delta U]\!]_\Gamma) \cap \neg [\![\nabla U]\!]_\Gamma \Big)\\
&\qquad \cup \Big([\![\Delta U]\!]_\Gamma \cap ([\![T]\!]_\Gamma(\mathcal{R}) \cup [\![\Delta T]\!]_\Gamma) \cap \neg [\![\nabla T]\!]_\Gamma \Big)\Big]\\
&\qquad\quad \cap (\neg [\![\nabla T]\!]_\Gamma \cap \neg [\![\nabla U]\!]_\Gamma)\\
&= \Big[([\![T]\!]_\Gamma(\mathcal{R}) \cap [\![U]\!]_\Gamma(\mathcal{R})) && \text{(De Morgan's)}\\
&\qquad \cup \Big([\![\Delta T]\!]_\Gamma \cap ([\![U]\!]_\Gamma(\mathcal{R}) \cup [\![\Delta U]\!]_\Gamma) \cap \neg [\![\nabla U]\!]_\Gamma \Big)\\
&\qquad \cup \Big([\![\Delta U]\!]_\Gamma \cap ([\![T]\!]_\Gamma(\mathcal{R}) \cup [\![\Delta T]\!]_\Gamma) \cap \neg [\![\nabla T]\!]_\Gamma \Big)\Big]\\
&\qquad\quad \cap \neg([\![\nabla T]\!]_\Gamma \cup [\![\nabla U]\!]_\Gamma)\\
&= \Big[([\![T]\!]_\Gamma(\mathcal{R}) \cap [\![U]\!]_\Gamma(\mathcal{R})) && \text{(definition of } \mathsf{X})\\
&\qquad \cup ([\![\Delta T]\!]_\Gamma \cap [\![\mathsf{X}U]\!]_\Gamma) \cup ([\![\Delta U]\!]_\Gamma \cap [\![\mathsf{X}T]\!]_\Gamma)\Big]\\
&\qquad\quad \cap \neg([\![\nabla T]\!]_\Gamma \cup [\![\nabla U]\!]_\Gamma)\\
&= \Big[([\![T]\!]_\Gamma(\mathcal{R}) \cap [\![U]\!]_\Gamma(\mathcal{R})) \cup [\![\Delta(T \wedge U)]\!]_\Gamma \Big] && \text{(definition of } [\![\Delta]\!])\\
&\qquad \cap \neg [\![\nabla T]\!]_\Gamma \cup [\![\nabla U]\!]_\Gamma\\
&= \Big[([\![T]\!]_\Gamma(\mathcal{R}) \cap [\![U]\!]_\Gamma(\mathcal{R})) \cup [\![\Delta(T \wedge U)]\!]_\Gamma \Big] && \text{(Proposition 4.3.1)}\\
&\qquad \cap \neg \Big[([\![\nabla T]\!]_\Gamma \cap [\![U]\!]_\Gamma(\mathcal{R})) \cup ([\![\nabla U]\!]_\Gamma \cap [\![T]\!]_\Gamma(\mathcal{R})) \Big]
\end{aligned}
$$

$$= \Big[(\llbracket T \rrbracket_\Gamma (\mathcal{R}) \cap \llbracket U \rrbracket_\Gamma (\mathcal{R})) \cup \llbracket \Delta(T \wedge U) \rrbracket_\Gamma \Big] \cap \neg(\llbracket \nabla(T \cap U) \rrbracket) \quad \text{(definition of } \llbracket \nabla \rrbracket)$$

$$= \llbracket T \wedge U \rrbracket_\Gamma (\mathcal{R}) \oplus_{\bowtie} (\llbracket \Delta(T \wedge U) \rrbracket_\Gamma , \llbracket \nabla(T \wedge U) \rrbracket_\Gamma) \qquad \text{(definition of } \oplus_{\bowtie})$$

The disjunctive case is identical to conjunction, minus the application of Proposition 4.3.1. Negation follows by calculation:

$$\begin{aligned}
&\llbracket \neg T \rrbracket_\Gamma (\mathcal{R} \oplus_{\bowtie} \delta\mathcal{R}) \\
&= \neg \llbracket T \rrbracket_\Gamma (\mathcal{R} \oplus_{\bowtie} \delta\mathcal{R}) && \text{(semantics of } \neg) \\
&= \neg((\llbracket T \rrbracket_\Gamma (\mathcal{R}) \cup \llbracket \Delta T \rrbracket_\Gamma) \cap \neg \llbracket \nabla T \rrbracket_\Gamma) && \text{(induction hypothesis, definition of } \oplus_{\bowtie}) \\
&= \neg \llbracket T \rrbracket_\Gamma (\mathcal{R}) \cap \neg \llbracket \Delta T \rrbracket_\Gamma \cup \llbracket \nabla T \rrbracket_\Gamma && \text{(De Morgan's)} \\
&= (\neg \llbracket T \rrbracket_\Gamma (\mathcal{R}) \cup \llbracket \nabla T \rrbracket_\Gamma) \cap \neg \llbracket \Delta T \rrbracket_\Gamma && \text{(disjointness of } \llbracket \Delta \rrbracket , \llbracket \nabla \rrbracket) \\
&= \llbracket \neg T \rrbracket_\Gamma (\mathcal{R}) \oplus_{\bowtie} (\llbracket \nabla T \rrbracket_\Gamma , \llbracket \Delta T \rrbracket_\Gamma) && \text{(definition of } \oplus_{\bowtie}) \\
&= \llbracket \neg T \rrbracket_\Gamma (\mathcal{R}) \oplus_{\bowtie} (\llbracket \Delta \neg T \rrbracket_\Gamma , \llbracket \nabla \neg T \rrbracket_\Gamma) && \text{(definition of } \llbracket \Delta \rrbracket , \llbracket \nabla \rrbracket)
\end{aligned}$$

Before establishing the derivative property for the existential, we state the following straightforward properties of the selection map σ_Γ:

Lemma 4.3.1. For any $A, B \in \mathbf{Rel}_{\Gamma,x}$ we have:

i. $\sigma_\Gamma(A \cup B) = \sigma_\Gamma(A) \cup \sigma_\Gamma(B)$

ii. $\sigma_\Gamma(A \cap \neg B) = \sigma_\Gamma(A) \cap (\neg\sigma_\Gamma(B) \cup \sigma_\Gamma(A \cap \neg B))$

Proof. We prove 4.3.1.ii explicitly, using a double inclusion argument. First:

$$\sigma_\Gamma(A \cap \neg B) \subseteq \sigma_\Gamma(A) \cap \sigma_\Gamma(A \cap \neg B) \subseteq \sigma_\Gamma(A) \cap (\neg\sigma_\Gamma(B) \cup \sigma_\Gamma(A \cap \neg B))$$

For the second inclusion, we note that the right-hand side of the equation can be equivalently written as:

$$\Big[\sigma_\Gamma(A) \cap \neg\sigma_\Gamma(B)\Big] \cup \Big[\sigma_\Gamma(A) \cap \sigma_\Gamma(A \cap \neg B)\Big]$$

It suffices to show that both of the bracketed terms are contained in $\sigma_\Gamma(A \cap \neg B)$. Clearly this is the case for the term on the right and it remains to show that $\sigma_\Gamma(A) \cap \neg\sigma_\Gamma(B) \subseteq \sigma_\Gamma(A \cup \neg B)$.

For this, consider a tuple $\rho \equiv \{\Gamma \mapsto w\}$ over Γ belonging to $\sigma_\Gamma(A) \cap \neg\sigma_\Gamma(B)$. Since $\rho \in \sigma_\Gamma(A)$, there must be some w_x satisfying $\rho_x \equiv \{x \mapsto w_x, \Gamma \mapsto w\} \in A$. But since $\rho \in \neg\sigma_\Gamma(B)$, it follows that ρ_x is in $\neg B$, and hence $\rho_x \in A \cup \neg B$. Therefore $\rho \in \sigma_\Gamma(A \cup \neg B)$. □

Equipped with these, we can proceed with the proof proper:

$$\begin{aligned}
&[\![\exists x.T]\!]_\Gamma\,(\mathcal{R} \oplus_{\bowtie} \delta\mathcal{R}) \\
&= \sigma_\Gamma([\![T]\!]_{\Gamma,x}\,(\mathcal{R} \oplus_{\bowtie} \delta\mathcal{R})) && \text{(semantics of } \exists) \\
&= \sigma_\Gamma([\![T]\!]_{\Gamma,x}\,(\mathcal{R}) \oplus_{\bowtie} ([\![\Delta T]\!]_{\Gamma,x}\,, [\![\nabla T]\!]_{\Gamma,x})) && \text{(induction hypothesis)} \\
&= \sigma_\Gamma(([\![T]\!]_{\Gamma,x}\,(\mathcal{R}) \cup [\![\Delta T]\!]_{\Gamma,x}) \cap \neg\, [\![\nabla T]\!]_{\Gamma,x}) && \text{(definition of } \oplus_{\bowtie}) \\
&= \sigma_\Gamma([\![T]\!]_{\Gamma,x}\,(\mathcal{R}) \cup [\![\Delta T]\!]_{\Gamma,x}) && \text{(Lemma 4.3.1.ii)} \\
&\qquad \cap \Big[\neg\sigma_\Gamma([\![\nabla T]\!]_{\Gamma,x}) \cup \sigma_\Gamma(([\![T]\!]_{\Gamma,x}\,(\mathcal{R}) \cup [\![\Delta T]\!]_{\Gamma,x}) \cap \neg\, [\![\nabla T]\!]_{\Gamma,x})\Big] \\
&= \Big[\sigma_\Gamma([\![T]\!]_{\Gamma,x}\,(\mathcal{R})) \cup \sigma_\Gamma([\![\Delta T]\!]_{\Gamma,x})\Big] && \text{(Lemma 4.3.1.i)} \\
&\qquad \cap \Big[\neg\sigma_\Gamma([\![\nabla T]\!]_{\Gamma,x}) \cup \sigma_\Gamma(([\![T]\!]_{\Gamma,x}\,(\mathcal{R}) \cup [\![\Delta T]\!]_{\Gamma,x}) \cap \neg\, [\![\nabla T]\!]_{\Gamma,x})\Big] \\
&= \Big[\sigma_\Gamma([\![T]\!]_{\Gamma,x}\,(\mathcal{R})) \cup [\![\Delta(\exists x.T)]\!]_\Gamma\Big] && \text{(definition of } [\![\Delta]\!]\,, [\![\mathsf{X}]\!]) \\
&\qquad \cap \Big[\neg\sigma_\Gamma([\![\nabla T]\!]_{\Gamma,x}) \cup \sigma_\Gamma([\![\mathsf{X}T]\!]_{\Gamma,x})\Big] \\
&= \Big[\sigma_\Gamma([\![T]\!]_{\Gamma,x}\,(\mathcal{R})) \cup [\![\Delta(\exists x.T)]\!]_\Gamma\Big] && \text{(De Morgan's)} \\
&\qquad \cap \neg\Big[\sigma_\Gamma([\![\nabla T]\!]_{\Gamma,x}) \cap \neg\sigma_\Gamma([\![\mathsf{X}T]\!]_{\Gamma,x})\Big] \\
&= \Big[\,[\![\exists x.T]\!]_\Gamma\,(\mathcal{R}) \cup [\![\Delta(\exists x.T)]\!]_\Gamma\Big] && \text{(semantics of } \exists) \\
&\qquad \cap \neg\Big[\,[\![\exists x.\nabla T]\!]_\Gamma \cap \neg\, [\![\exists x.\mathsf{X}T]\!]_\Gamma\Big] \\
&= \Big[\,[\![\exists x.T]\!]_\Gamma\,(\mathcal{R}) \cup [\![\Delta(\exists x.T)]\!]_\Gamma\Big] \cap \neg\, [\![\nabla(\exists x.T)]\!]_\Gamma && \text{(definition of } [\![\nabla]\!]) \\
&= [\![\exists x.T]\!]_\Gamma\,(\mathcal{R}) \oplus_{\bowtie} ([\![\Delta(\exists x.T)]\!]_\Gamma\,, [\![\nabla(\exists x.T)]\!]_\Gamma) && \text{(definition of } \oplus_{\bowtie})
\end{aligned}$$

□

We can give a derivative for our *treeP* predicate by mechanically applying the recursive functions defined in Figure 4.6.

$$\begin{aligned}
&\Delta(\mathit{treeP}(x)) \\
&= p(x) \wedge \exists y.(\mathit{child}(x,y) \wedge \Delta(\mathit{treeP}(y))) \wedge \neg\exists y.(\mathit{child}(x,y) \wedge \neg\mathsf{X}(\mathit{treeP}(y))) \\
&\nabla(\mathit{treeP}(x)) \\
&= p(x) \wedge \exists y.(\mathit{child}(x,y) \wedge \nabla(\mathit{treeP}(y)))
\end{aligned}$$

The upwards difference in particular is not especially easy to compute. If we naively compute it, the third conjunct requires us to recompute the whole of the recursive part. However, the first and second conjuncts act as a guard of sorts: if either is empty we then the whole formula will be, so we only need to evaluate the third conjunct if the first and second conjuncts are non-empty, i.e if there is *some* change in the body of the existential.

This shows that our derivatives aren't a panacea: it is simply *hard* to compute downwards differences for $\exists$ (and, equivalently, upwards differences for $\forall$) because we must check that there is no other way of deriving the same facts.[9] However, we can still avoid the re-evaluation in many cases, and the inefficiency is local to this subformula.

4.3.3 Extensions to Datalog

Our formulation of Datalog formula semantics and derivatives is generic and modular, so it is easy to extend the language with new formula constructs: all we need to do is add cases for Δ and ∇.

In fact, because we are using a transitive change action, we can *always* do this by using the maximal or minimal derivative. This justifies our claim that we can support *arbitrary* additional formula constructs: although the maximal and minimal derivatives are likely to be impractical, they can still be used as a fall-back option for constructs that do not support incremental evaluation – even in those cases, the compositional nature of this approach means that, while the problematic subformula will use one of these inefficient derivatives, the rest of the formula can still be incrementalised efficiently.

This power is important in practice: here is a real example from Semmle's variant of Datalog, which includes a kind of aggregate expressions which have well-defined recursive semantics. Aggregates have the form

$$r = \mathrm{agg}(p)(vs \mid T \mid U)$$

where agg refers to an aggregation function (such as "sum" or "min"), vs is a sequence of variables, p and r are variables, T is a formula possibly mentioning vs, and U is a formula possibly mentioning vs and p. The full details can been found in Moor and Baars [78], but for example this allows us to write

$$\begin{aligned} \mathit{height}(n, h) \leftarrow & \neg\exists c.(\mathit{child}(n, c)) \land h = 0 \\ & \lor \exists h'.(h' = \max(p)(c \mid \mathit{child}(n, c) \mid \mathit{height}(c, p)) \land h = h' + 1) \end{aligned}$$

which recursively computes the height of a node in a tree.

[9]The "support" data structures introduced by [50] are an attempt to avoid this issue by tracking the number of derivations of each tuple.

Here is an upwards derivative for an aggregate formula:

$$\Delta(r = \text{agg}(p)(vs \mid T \mid U)) := \exists vs.(T \land \Delta U) \land r = \text{agg}(p)(vs \mid T \mid U)$$

While this isn't a precise derivative, it is still substantially cheaper than re-evaluating the whole subformula, as the first conjunct acts as a guard, allowing us to skip the second conjunct when U has not changed.

4.4 Changes on Functions

So far we have defined change actions for the kinds of objects that typically make up the relations of a Datalog program, but we would also like to have change actions on *functions*. This would allow us to define derivatives for higher-order languages (where functions are first-class); and to give a derivative for important higher-order maps, like fixed point operators $\mathbf{fix} : (A \to A) \to A$. The latter is of special interest to us as evaluation of Datalog queries necessitates finding a fixed point, thus the differentiation of fixed point operators will be the object of Section 4.5.2.

Function spaces, however, differ from products and disjoint unions in that there is no obvious "best" change action on $A \to B$. Therefore instead of trying to define a single choice of change action, we will instead pick out subsets of function spaces which have "well-behaved" change actions.

Definition 4.4.1. Given change actions $\overline{A}$ and $\overline{B}$ and a set $U \subseteq A \to B$, a change action $\overline{U} = (U, \Delta U, \oplus_U, +_U, 0_U)$ is **functional** whenever the evaluation map $\mathbf{ev} : U \times A \to B$ is differentiable, that is to say, whenever there exists a function $\partial\mathbf{ev} : (U \times A) \times (\Delta U \times \Delta A) \to \Delta B$ such that:

$$(f \oplus_U \delta f)(a \oplus_A \delta a) = f(a) \oplus_B \partial\mathbf{ev}((f, a), (\delta f, \delta a))$$

We will write $\overline{U} \subseteq \overline{A} \Rightarrow \overline{B}$ whenever $U \subseteq A \to B$ and $\overline{U}$ is functional.

There are two reasons why the definition of a functional change action consists of a *subset* of $U \subseteq A \to B$. Firstly, it allows one to restrict themselves to spaces of monotone or continuous functions. But more importantly, functional change actions are necessarily made up of differentiable functions, and thus a functional change action may not exist for the entire function space $A \to B$.

Proposition 4.4.1. Let $\overline{U} \subseteq \overline{A} \Rightarrow \overline{B}$ be a functional change action. Then every $f \in U$ is differentiable, with a derivative ∂f given by:

$$\partial f(x, \delta x) = \partial\mathbf{ev}((f, x), (0, \delta x))$$

Conversely, the derivative of the evaluation map can also be used to compute the effect of a given function change δf:

Proposition 4.4.2. Let $\overline{U} \subseteq \overline{A} \Rightarrow \overline{B}$. Then for every $f \in U$ and $\delta f \in \Delta U$ we have:

$$(f \oplus \delta f)(x) = f(x) \oplus \mathbf{ev}'((f, x), (\delta f, \mathbf{0}))$$

Even if we were to restrict ourselves to the differentiable functions between $\overline{A}$ and $\overline{B}$, however, it is hard to find a concrete functional change action for this set. Fortunately, in many important cases there is a simple change action on the set of differentiable functions.

Definition 4.4.2. Let $\overline{A}$ and $\overline{B}$ be change actions. The **pointwise functional change action** $\overline{A} \Rightarrow_{pt} \overline{B}$, when it is defined, is given by

$$\overline{A} \Rightarrow_{pt} \overline{B} \coloneqq (\overline{A} \to \overline{B}, A \to \Delta B, \oplus_{\to}, +_{\to}, 0_{\to})$$

with the monoid structure $(A \to \Delta B, +_{\to}, 0_{\to})$ and the action $\oplus_{\to}$ defined as:

$$\begin{aligned}(f \oplus_{\to} \delta f)(x) &\coloneqq f(x) \oplus_B \delta f(x)\\ (\delta f +_{\to} \delta g)(x) &\coloneqq \delta f(x) +_B \delta g(x)\\ \mathbf{0}_{\to}(x) &\coloneqq \mathbf{0}_B\end{aligned}$$

That is, a change is given pointwise, mapping each point in the domain to a change in the codomain.

The above definition is not always well-typed, since given $f : \overline{A} \to \overline{B}$ and $\delta f : A \to \Delta B$ there is no guarantee that $f \oplus_{\to} \delta f$ is differentiable. We present two sufficient criteria that guarantee this.

Theorem 4.4.1. Let $\overline{A}$ and $\overline{B}$ be change actions, and suppose that $\overline{B}$ satisfies one of the following conditions:

- $\overline{B}$ is a transitive change action.
- The change action $\overline{\Delta B} \coloneqq (\Delta B, \Delta B, +_B, +_B, 0_B)$ is transitive and $\oplus_B : B \times \Delta B \to B$ is differentiable with respect to it.

Then the pointwise functional change action $(\overline{A} \to \overline{B}, A \to \Delta B, \oplus_{\to})$ is well defined[10].

[10]Either of these conditions is enough to guarantee that the pointwise functional change action is well defined, but it can be the case that $\overline{B}$ satisfies neither and yet pointwise change actions into $\overline{B}$ do exist. A precise account of when pointwise functional change actions exist is outside the scope of this thesis.

Proof. We need to show that, given a differentiable function $f : \overline{A} \to \overline{B}$ and a change $\delta f : A \to \Delta B$, the updated function $f \oplus_{\to} \delta f$ is also differentiable.

- When $\overline{B}$ is transitive, every function into it is differentiable and so the above holds trivially.
- In the second case, since $\overline{\Delta B}$ is transitive, every change $\delta f : A \to \Delta B$ is differentiable. In particular, consider a differentiable function f and a change δf, with derivatives $\partial f, \partial \delta f$ respectively. Then:

$$\begin{aligned}
&(f \oplus_{\to} \delta f)(x \oplus_A \delta x)\\
&= (f(x \oplus_A \delta x)) \oplus_B (\delta f(x \oplus_A \delta x))\\
&= (f(x) \oplus_B \partial f(x, \delta x)) \oplus_B (\delta f(x) \oplus_B \partial \delta f(x, \delta x))\\
&= (f(x) \oplus_B \delta f(x)) \oplus_B \partial[\oplus_B](f(x), \delta f(x), \partial f(x, \delta x), \partial \delta f(x, \delta x))\\
&= (f \oplus_{\to} \delta f)(x) \oplus_B \partial[\oplus_B](f(x), \delta f(x), \partial f(x, \delta x), \partial \delta f(x, \delta x))
\end{aligned}$$

  where $\partial[\oplus_B]$ denotes the derivative of the map $\oplus_B$. It follows that the map

$$\partial[f \oplus_{\to} \delta f] := \partial[\oplus_B](f(x), \delta f(x), \partial f(x, \delta x), \partial \delta f(x, \delta x))$$

  is a derivative for $f \oplus_{\to} \delta f$. □

Remark 4.4.1. The above result is interesting as it already hints at the difficulties with functional change actions, as well as their solutions. As we have shown in Proposition 4.4.2, every functional change action is made of differentiable functions. However, it is in general not possible to guarantee that $f \oplus \delta f$ be differentiable, for f differentiable.

In order to solve this problem, the previous theorem required that $\delta f, \oplus$ were themselves differentiable, but for this approach to work one needs to endow the change set itself with the structure of a change action.

As a direct consequence of Theorem 4.4.1, it follows that whenever L is a Boolean algebra (and hence has a complete change action), the pointwise functional change action $\overline{A} \Rightarrow_{pt} \overline{L}_{\bowtie}$ is well-defined.

Pointwise functional change actions are functional in the sense of Definition 4.4.1. Moreover, the derivative of the evaluation map is quite easy to compute.

Proposition 4.4.3. Let $\overline{A}$ and $\overline{B}$ be change actions such that the pointwise functional change action $\overline{A} \Rightarrow_{pt} \overline{B}$ is well defined, and let $f : \overline{A} \to \overline{B}$, $a \in A$, $\delta a \in \Delta A$, $\delta f \in A \to \Delta B$.

Then the following are both derivatives of the evaluation map:

$$\begin{aligned}
\partial \mathbf{ev}_1((f, a), (\delta f, \delta a)) &\coloneqq \partial f(a, \delta a) + \delta f(a \oplus \delta a) \\
\partial \mathbf{ev}_2((f, a), (\delta f, \delta a)) &\coloneqq \delta f(a) + \partial[f \oplus \delta f](a, \delta a)
\end{aligned}$$

Proof. Straightforward calculation:

$$\begin{aligned}
(f \oplus_{\to} \delta f)(a \oplus_A \delta a) &= f(a \oplus_A \delta a) \oplus_B \delta f(a \oplus_A \delta a) \\
&= f(a) \oplus_B \left[\partial f(a, \delta a) + \delta f(a \oplus_A \delta a)\right] \\
&= f(a) \oplus_B \partial \mathbf{ev}_1((f, a), (\delta f, \delta a)) \\
(f \oplus_{\to} \delta f)(a \oplus_A \delta a) &= (f \oplus_{\to} \delta f)(a) \oplus_B \partial[f \oplus_{\to} \delta f](a, \delta a) \\
&= f(a) \oplus_B \left[\delta f(a) + \partial[f \oplus_{\to} \delta f](a, \delta a)\right]
\end{aligned}$$ □

Functional change actions merely require that a derivative of the evaluation map exist – a pointwise change action, on the other hand, actually provides a definition for it. In practice, this means that we will only be able to use the results in Section 4.5.2 (which allow us to differentiate fixed point h) in settings where we have pointwise change actions, or when we have some other way of computing a derivative of the evaluation map.

4.5 Directed-Complete Partial Orders and Fixed Points

As we have seen in Section 4.3, a Datalog program can be understood as a function on Boolean algebras mapping the extension of its free predicate variables to the extension of the entire program. Since predicates may be recursive, the semantics of the entire program is then obtained by jointly taking the least fixed point of the denotations of the predicates that constitute it.

The computation of this fixed point computation is licit, as Datalog predicates are necessarily monotone[11], and their denotation lies in the directed-complete partial order of sets of tuples.

[11]Even in the case where parity-stratified negation is allowed, recursion can only happen through an even number of negations.

We have already obtained a suitable change action on Boolean algebras in Proposition 4.2.6, which can be used to interpret the semantics of a single Datalog predicate; our goal is now to show how to incrementalise the fixed-point operator in this change action. To do this, we will first define what it means for a change action to be compatible with a dcpo structure. Then we will use the functional change actions in Section 4.4 to show how to differentiate iteration and fixed point operators.

4.5.1 Continuous Actions on Dcpos

Before proceeding on to the rest of the section we remind the reader of the basic definitions of which we will make use throughout.

Definition 4.5.1. Given a partial order $(D, \sqsubseteq)$, a (non-empty) **directed set** is a subset $U \subseteq D$ such that, whenever $u, v \in D$, there is some $w \in D$ satisfying $u \sqsubseteq w, v \sqsubseteq w$.

If every directed subset $U \subseteq D$ has a least upper bound, the partial order $(D, \sqsubseteq)$ is said to be **directed-complete**[12]. We will denote such least upper bounds by $\bigsqcup U$ or $\bigsqcup_{i \in I} u_i$ when $U = \{u_i \mid i \in I\}$.

A dcpo is **pointed** whenever it has a least element, which we will denote by $\bot$. In what follows, we will assume that every dcpo we work with is in fact a pointed dcpo.

Definition 4.5.2. Given dcpos $(D, \sqsubseteq_D)$ and $(E, \sqsubseteq_E)$, a function $f : D \to E$ is **continuous** whenever it is monotone and furthermore it preserves least upper bounds, that is, for every directed subset $U \subseteq D$:

$$f(\bigsqcup U) = \bigsqcup f(U)$$

As was the case with partial orders, we can define change actions on dcpos, rather than sets, as change actions whose base and change sets are endowed with a dcpo structure, and where the monoid operation and action are continuous. This is in fact the same thing as a change action internal to the category **DCPO** of dcpos and continuous maps.

Definition 4.5.3. A change action $\overline{A}$ is *continuous* if

- A and ΔA are dcpos.

[12]Some sources refer to directed-complete partial orders as simply “complete partial orders”. We write dcpo instead for clarity and to distinguish them from similar objects such as complete lattices and ω-complete partial orders.

- $\oplus$ is a continuous function from $A \times \Delta A \to A$.
- $+$ is a continuous map from $\Delta A \times \Delta A \to \Delta A$.

Unlike posets, the change order $\leq_\Delta$ does *not*, in general, induce a dcpo on ΔA. As a counterexample, consider the change action $(\mathbb{N}^*, \mathbb{N}, +)$, where $\mathbb{N}^*$ denotes the dcpo of natural numbers extended with positive infinity. In particular, the corresponding change order in $\mathbb{N}$ coincides with the standard order $\leq$, which is not a dcpo.

A key example of a continuous change action is the $\overline{L}_\bowtie$ change action on Boolean algebras - provided the underlying Boolean algebra L is itself complete.

Proposition 4.5.1. Whenever L is a complete Boolean algebra, the change action $\overline{L}_\bowtie$ is continuous.

Proof. L is a complete lattice, thus in particular it is a dcpo. $L \bowtie L$ is a partial order with the order $\leq_\Delta$. Now let $U \equiv \{(p_i, q_i) \mid i \in I\} \subseteq L \bowtie L$ be a $\leq_\Delta$-directed set. We claim that

$$\bigsqcup U = \left(\bigvee\{p_i \mid i \in I\}, \bigwedge\{q_i \mid i \in I\}\right)$$

is a least upper bound for U. To see this, first note that $\bigsqcup U$ is indeed the least upper bound for U in the complete lattice $L \times L^{op}$, thus all that remains to prove is that $\bigsqcup U$ is in fact an element of $L \bowtie L$, i.e. $\bigvee\{p_i\} \wedge \bigwedge\{q_i\} = \bot$. But this is straightforward: since $\wedge, \vee$ distribute over infinite meets and joins, we have:

$$\bigvee \{p_i \mid i \in I\} \wedge \bigwedge \{q_j \mid j \in I\} = \bigvee \left(p_i \wedge \bigwedge \{q_j \mid j \in I\}\right) \sqsubseteq \bigvee (p_i \wedge q_i) = \bot$$

Continuity of $\oplus_\bowtie$ in its second argument is easy to prove:

$$\begin{aligned}
&a \oplus_\bowtie \bigvee_{i \in I} \{(p_i, q_i)\} \\
&= a \oplus_\bowtie \left(\bigvee_{i \in I} p_i, \bigwedge_{j \in I} q_j\right) \\
&= \left(a \vee \bigvee_{i \in I} p_i\right) \wedge \neg \bigwedge_{j \in I} q_j \\
&= \left(a \vee \bigvee_{i \in I} p_i\right) \wedge \bigvee_{j \in I} \neg q_j \\
&= \bigvee_{i \in I} \bigvee_{j \in I} a \oplus_\bowtie (p_i, q_j) \\
&= \bigvee_{i \in I} a \oplus_\bowtie (p_i, q_i)
\end{aligned}$$

Continuity of $\oplus_{\bowtie}$ in its first argument and continuity of $\bowtie$ follow easily from their definitions and the continuity of $\vee$ and $\wedge$. $\square$

For a general overview of results in domain theory and dcpos, we refer the reader to an introductory work such as [2], but we will state here some specific results that we find useful, such as the following, whose proof can be found in [2, Lemma 3.2.6]:

Proposition 4.5.2. A function $f : A \times B \to C$ is continuous iff it is continuous in each variable separately.

It is a well-known result in standard calculus that the limit of an absolutely convergent sequence of differentiable functions $\{f_i\}$ is itself differentiable, and its derivative is equal to the limit of the derivatives of the f_i. A consequence of Proposition 4.5.2 is the following analogous result:

Theorem 4.5.1. Let $\overline{A}$ and $\overline{B}$ be change actions, with $\overline{B}$ continuous and let $\{f_i\}$ and $\{\partial f_i\}$ be I-indexed directed sets of functions in $A \to B$ and $A \times \Delta A \to \Delta B$ respectively.

Then, if ∂f_i is a derivative for f_i for every $i \in I$, the least upper bound of this set $\bigsqcup_{i \in I} \partial f_i$ is a derivative for $\bigsqcup_{i \in I} f_i$.

Proof. It suffices to apply $\oplus$ and Proposition 4.5.2 to the directed families $\{f_i(a)\}$ and $\{\partial f_i(a, \delta a)\}$. $\square$

We also state the following additional fixed point lemma. This is a specialisation of Bekić's Theorem [109, section 10.1], but it has a straightforward direct proof.

Proposition 4.5.3. Let A and B be dcpos, $f : A \to A$ and $g : A \times B \to B$ be continuous functions, and let

$$h(a, b) := (f(a), g(a, b))$$

Then

$$\mathbf{lfp}(h) = (\mathbf{lfp}(f), \mathbf{lfp}(\lambda b.g(\mathbf{lfp}(f), b)))$$

Proof. Let

$$p(b) = (\mathbf{lfp}(f), g(\mathbf{lfp}(f), b))$$

Then, by induction, $h(h^i(\bot)) \leq p(p^i(\bot))$ and, since h is continuous, we have:

$$\mathbf{lfp}(h) = \bigsqcup_{i \in \mathbb{N}} h^i(\bot) \leq \bigsqcup_{i \in \mathbb{N}} p^i(\bot) = \mathbf{lfp}(p)$$

But $h(\mathbf{lfp}(p)) = \mathbf{lfp}(p)$, therefore $\mathbf{lfp}(p)$ is a fixed point for h and so $\mathbf{lfp}(h) \leq \mathbf{lfp}(p)$. Hence $\mathbf{lfp}(h) = \mathbf{lfp}(p) = (\mathbf{lfp}(f), \mathbf{lfp}(\lambda b.g(\mathbf{lfp}(f), b)))$. $\square$

4.5.2 Fixed Points

The usefulness of directed-complete partial orders lies in the ability to apply Kleene's fixed point theorem to compute least fixed points of monotone maps – indeed, this is how we will define the full semantics of a recursive Datalog program.

In order to apply our derivative-based approach to the computation of fixed points, we now focus on obtaining derivatives for iteration and fixed point operators. As we shall see, these provide us with a concrete way of applying the derivative for the (non-recursive) formula semantics of Datalog from Section 4.3 to the problem of incrementalising the fixed point computation involved in the evaluation of an entire Datalog program.

For reference, we state here Kleene's fixed point theorem.

Theorem 4.5.2. Let $(D, \sqsubseteq)$ a directed-complete partial order (with a least element), and $f : D \to D$ a continuous function. Then f has a least fixed point $\mathbf{lfp}(f)$ which is the least upper bound of the ascending chain:

$$\bot \sqsubseteq f(\bot) \sqsubseteq f(f(\bot)) \sqsubseteq \dots$$

Alternatively, this least fixed point can be expressed in terms of the finite iteration operator **iter** defined by:

$$\mathbf{iter} : (A \to A) \times \mathbb{N} \to A$$
$$\mathbf{iter}(f, n) \coloneqq f^n(\bot)$$

Then the least fixed point $\mathbf{lfp}(f)$ can be written as:

$$\mathbf{lfp}(f) = \bigsqcup_{i \in \mathbb{N}} \mathbf{iter}(f, i)$$

The iteration function is the basis for all the results in this section. We will show that it is differentiable; in doing so, we obtain two tools: differentiating it with respect to the second argument (the number of iterations) will provide us with a way to incrementally computing $\mathbf{iter}(f, i + 1)$ from $\mathbf{iter}(f, i)$. On the other hand, differentiating it with respect to the first argument (the function to be iterated) gives us a solution for the *incremental maintenance* problem, that is to say, it allows one to incrementally recompute the entire fixed point computation given a change in the rules of our Datalog program.

The following theorems provide a generalisation of semi-naive evaluation to any differentiable function over a continuous change action. Throughout this section we will assume that we have a continuous change action $\overline{A}$, and any reference to the change action $\overline{\mathbb{N}}$ will refer to the monoidal change action on the naturals defined in Example 3.1.1.

Since we are trying to incrementalise the iterative step, we start by taking the partial derivative of **iter** with respect to n.

Proposition 4.5.4. Let $\overline{A}$ be a transitive change action and let $f : A \to A$ be a function with derivative ∂f (note that such a derivative always exists as $\overline{A}$ is transitive). Then **iter** is differentiable with respect to its second argument, and a partial derivative is given by:

$$\begin{aligned}
&\partial_2\mathbf{iter} : (A \to A) \times \mathbb{N} \times \Delta\mathbb{N} \to \Delta A\\
&\partial_2\mathbf{iter}(f, n, m) := \begin{cases} \mathbf{iter}(f, m) \ominus \mathbf{iter}(f, 0) & \text{if } n = 0\\ \partial f(\mathbf{iter}(f, n-1), \partial_2\mathbf{iter}(f, n-1, m)) & \text{otherwise}\end{cases}
\end{aligned}$$

Proof. The proof proceeds by induction on n. We show the inductive step (the base step is trivial).

$$\begin{aligned}
\mathbf{iter}(f, (n+1) + m) &= f(\mathbf{iter}(f, n+m))\\
&= f(\mathbf{iter}(f, n) \oplus \partial_2\mathbf{iter}(f, n, m))\\
&= \mathbf{iter}(f, n+1) \oplus \partial f(\mathbf{iter}(f, n), \partial_2\mathbf{iter}(f, n, m))
\end{aligned} \qquad \square$$

Looking back at our leading example from Section 4.1, we saw how the semantics of the transitive closure relation can be obtained naively by repeatedly iterating its semantics until a fixed point is reached, or more cleverly by incrementalising the iteration procedure by computing a sequence of increments $\Delta \llbracket tc \rrbracket_i$. Abstractly, this corresponds to computing $\mathbf{iter}(f, i+1)$ by updating the previous iteration with its derivative. This process itself can be elegantly iterated, as the following result shows.

Proposition 4.5.5. Let $\overline{A}$ be a transitive change action and let $f : A \to A$ be a function with derivative ∂f, and define the recurrence relation **recur** as follows:

$$\begin{aligned}
&\mathbf{recur}_f : A \times \Delta A \to A \times \Delta A\\
&\mathbf{recur}_f(\bot, \bot) := (\bot, f(\bot) \ominus \bot)\\
&\mathbf{recur}_f(a, \delta a) := (a \oplus \delta a, \partial f(a, \delta a))
\end{aligned}$$

Then **recur** verifies the equation:

$$\mathbf{recur}_f^n(\bot, \bot) = (\mathbf{iter}(f, n), \partial_2\mathbf{iter}(f, n, 1))$$

As we have remarked, this result gives us a way to compute a fixed point incrementally, by adding successive changes to an accumulator until we reach it. This is exactly how our semi-naive evaluation example works: by computing the increment $\Delta \llbracket tc \rrbracket_i$ in lock-step with the accumulator $\llbracket tc \rrbracket_i$, and adding the delta into the accumulator at each step until converging to the final output.

Theorem 4.5.3. Let $\overline{A}$ be a continuous, transitive change action, with $f : \overline{A} \to \overline{A}$ a continuous function.

Then $\mathbf{lfp}(f) = \bigsqcup_{n \in \mathbb{N}} (\pi_1(\mathbf{recur}_f^n(\bot, \bot)))$.[13]

With this we have shown how to use derivatives to compute fixed points more efficiently, but we also want to consider the derivative of the fixed point operator itself. A typical use case for this is where we have calculated some fixed point

$$F_E \coloneqq \mathbf{fix}(\lambda X.F(E, X))$$

then update the parameter E with some change δE and wish to compute the new value of the fixed point, i.e.

$$F_{E \oplus \delta E} \coloneqq \mathbf{fix}(\lambda X.F(E \oplus \delta E, X))$$

This process can be understood as applying a change to the *function* whose fixed point we are taking. We go from computing the least fixed point of $F(E, \_)$ to computing the least fixed point of $F(E \oplus \delta E, \_)$. If we have a pointwise functional change action then we can express this change as a function giving the change at each point, that is:

$$\lambda X.F(E \oplus \delta E, X) \ominus F(E, X)$$

In Datalog this would allow us to update a recursively defined relation given an update to one of its non-recursive dependencies, or the extensional database. Continuing with the transitive closure example, we might make changes to the edge relation e and expect tc to be recalculated efficiently, without starting from scratch.

However, to compute these examples requires us to provide a derivative for the fixed point operator **fix**, as we want to know how the resulting fixed point changes given a change to its input function.

Definition 4.5.4. Let $\overline{A}$ be a continuous change action, let $\overline{U} \subseteq \overline{A} \Rightarrow \overline{A}$ be a functional change action (not necessarily pointwise) and suppose **lfp** and $\mathbf{lfp}_{\Delta A}$ are (least) fixed point operators on U and ΔA respectively.

[13]Note that we have *not* taken the fixed point of $\mathbf{recur}_f$, since it may not be continuous.

Then we define

$$\begin{aligned}
&\mathbf{adjust} : U \times \Delta U \to (\Delta A \to \Delta A)\\
&\mathbf{adjust}(f, \delta f) \coloneqq \lambda \delta a. \partial\mathbf{ev}((f, \mathbf{fix}_U(f)), (\delta f, \delta a))
\end{aligned}$$

$$\begin{aligned}
&\partial\mathbf{lfp} : U \times \Delta U \to \Delta A\\
&\partial\mathbf{lfp}(f, \delta f) \coloneqq \mathbf{lfp}_{\Delta A}(\mathbf{adjust}(f, \delta f))
\end{aligned}$$

The suggestively named $\partial\mathbf{lfp}_U$ will in fact turn out to be a derivative. The appearance of $\partial\mathbf{ev}$, a derivative for the evaluation map, in the definition of **adjust** is also no coincidence: as evaluating a fixed point consists of many steps of applying the evaluation map, so computing the derivative of a fixed point consists of many steps of applying the derivative of the evaluation map.

Since the least fixed point of f, $\mathbf{lfp}(f)$ is characterised as the limit of a chain of functions, Theorem 4.5.1 suggests a way to compute its derivative. It suffices to find a derivative $\partial\mathbf{iter}_n$ of each iteration map such that the resulting set $\{\partial\mathbf{iter}_n \mid n \in \mathbb{N}\}$ is directed, which will entail that $\bigsqcup_{n\in\mathbb{N}} \partial\mathbf{iter}_n$ is a derivative of **lfp**.

These correspond to the first partial derivative of **iter** – this time with respect to f. As before, we will provide a definition for such derivatives by induction on n.

Proposition 4.5.6. Let $\overline{A}$ be a change action, let $\overline{U} \subseteq \overline{A} \Rightarrow \overline{A}$ be a functional change action and let $\mathbf{iter}|_U$ be the restriction of **iter** to the set U.

Then $\mathbf{iter}|_U$ is differentiable with respect to its first argument, with a partial derivative given by:

$$\begin{aligned}
&\partial_1\mathbf{iter}|_U : U \times \mathbb{N} \times \Delta U \to \Delta A\\
&\partial_1\mathbf{iter}|_U(f, n, \delta f) \coloneqq \begin{cases} 0 & \text{when n} = 0\\ \partial\mathbf{ev}((f, \mathbf{iter}|_U(f, n-1)), (\delta f, \partial_1\mathbf{iter}|_U(f, n-1), \delta f)) & \text{otherwise}\end{cases}
\end{aligned}$$

Proof. The base case is easy to prove. For the inductive step:

$$\begin{aligned}
&\mathbf{iter}|_U(f \oplus \delta f, n+1)\\
&= (f \oplus \delta f)(\mathbf{iter}|_U(f \oplus \delta f, n))\\
&= (f \oplus \delta f)(\mathbf{iter}|_U(f, n) \oplus \partial_1\mathbf{iter}|_U(f, n, \delta f))\\
&= f(\mathbf{iter}|_U(f, n)) \oplus \partial\mathbf{ev}((f, \mathbf{iter}|_U(f, n)), (\delta f, \partial_1\mathbf{iter}|_U(f, n, \delta f)))\\
&= \mathbf{iter}|_U(f, n+1) \oplus \partial_1\mathbf{iter}|_U(f, n, \delta f)
\end{aligned}$$

□

As before, we can now compute $\partial_1\mathbf{iter}$ together with $\mathbf{iter}$ by mutual recursion.[14]

$$\begin{aligned}
&\mathbf{recur}_{f,\delta f} : A \times \Delta A \to A \times \Delta A \\
&\mathbf{recur}_{f,\delta f}(a, \delta a) \coloneqq (f(a), \partial\mathbf{ev}((f, a), (\delta f, \delta a)))
\end{aligned}$$

Which has the property that

$$\mathbf{recur}^n_{f,\delta f}(\bot, \bot) = (\mathbf{iter}(f, n), \partial_1\mathbf{iter}(f, \delta f, n)).$$

This provides us with a sequence of derivatives for $\mathbf{iter}|_U$ whose limit we can take to obtain a derivative for the least fixed point operator, as in Theorem 4.5.1. If we do so we will discover that it is exactly $\partial\mathbf{lfp}$ (defined as in Definition 4.5.4), showing that $\partial\mathbf{lfp}$ is indeed a derivative for $\mathbf{lfp}$.

Theorem 4.5.4. Let

- $\overline{A}$ be a continuous change action
- $\overline{U} \subseteq A \Rightarrow A$ be a functional change action where U is precisely the set of continuous functions $f : A \to A$.
- $f \in U$ be a continuous, differentiable function
- $\delta f \in \Delta U$ be a function change
- $\partial\mathbf{ev}$ be a derivative of the evaluation map which is continuous with respect to a and δa.

Then $\partial\mathbf{lfp}$ is a derivative of $\mathbf{lfp}$.

Proof. $\partial_1\mathbf{iter}|_U$ and $\mathbf{recur}_{f,\delta f}$ are continuous since $\partial\mathbf{ev}$ and f are. Hence the set $\{\partial_1\mathbf{iter}|_U(\cdot, \cdot, n)\}$ is directed, and so $\bigsqcup_{i\in\mathbb{N}}\{\partial_1\mathbf{iter}|_U(\cdot, \cdot, n)\}$ is indeed a derivative for

[14] In fact, the recursion here is not *mutual*: the first component does not depend on the second. However, writing it in this way makes it amenable to computation by fixed point, and we will in fact be able to avoid the re-computation of $\mathbf{iter}_n$ when we show that it is equivalent to $\partial\mathbf{lfp}$.

lfp. We now show that it is equivalent to ∂**lfp**:

$$
\begin{aligned}
& \bigsqcup_{n\in\mathbb{N}} \partial_1 \mathbf{iter}|_U\,(f, \delta f, n) \\
&= \bigsqcup_{n\in\mathbb{N}} \pi_2\left(\mathbf{recur}^n_{f,\delta f}\,(\bot)\right) \\
&= \pi_2\left(\bigsqcup_{n\in\mathbb{N}} \mathbf{recur}^n_{f,\delta f}\,(\bot)\right) && (\pi_2 \text{ is continuous}) \\
&= \pi_2\left(\mathbf{lfp}\left(\mathbf{recur}_{f,\delta f}\right)\right) && (\text{continuity of } \mathbf{recur}_{f,\delta f}, \text{ Kleene's Theorem}) \\
&= \pi_2\left(\left(\mathbf{lfp}\,(f), \mathbf{lfp}\left(\lambda\ \delta a.\partial\mathbf{ev}\left((f, \mathbf{lfp} f), (\delta f, \delta a)\right)\right)\right)\right) \\
& && (\text{by 4.5.3, and the definition of } \mathbf{recur}) \\
&= \pi_2\left(\mathbf{lfp}\,(f), \mathbf{lfp}\left(\mathbf{adjust}\,(f, \delta f)\right)\right) \\
&= \mathbf{lfp}\left(\mathbf{adjust}\,(f, \delta f)\right) \\
&= \partial\mathbf{lfp}\,(f, \delta f) && \square
\end{aligned}
$$

Computing this derivative still requires computing a fixed point (over the poset of changes) but this may still be significantly less expensive than recomputing the full new fixed point.

4.5.3 Derivatives for Recursive Datalog

Given a semantics for the non-recursive fragment of a language, we can extend it to handle recursive definitions using fixed point operators. Section 4.5.2 lets us extend our derivative for the non-recursive semantics to a derivative for the recursive semantics, as well as letting us compute the fixed points themselves incrementally.

We have demonstrated the technique with Datalog, although we believe our approach to be generic enough that it should work in many other settings. First of all, we define the usual "immediate consequence operator" which computes "one step" of our program semantics.

Definition 4.5.5. Given a Datalog program $\mathbb{P} = (P_1, \ldots, P_n)$, where P_i is a predicate, with schema Γ_i, the **immediate consequence operator** $\mathcal{I} : \mathbf{Rel}^n \to \mathbf{Rel}^n$ is defined as follows:

$$\mathcal{I}(\mathcal{R}_1, \ldots, \mathcal{R}_n) = (\llbracket P_1 \rrbracket_{\Gamma_1}(\mathcal{R}_1, \ldots, \mathcal{R}_n), \ldots, \llbracket P_n \rrbracket_{\Gamma_n}(\mathcal{R}_1, \ldots, \mathcal{R}_n))$$

That is, given a value for the program, we pass in all the relations to the denotation of each predicate, to get a new tuple of relations.

Proposition 4.5.7. The immediate consequence operator $\mathcal{I}$ is differentiable.

Proof. Straightforward corollary of 4.3.1 □

Definition 4.5.6. The semantics of a program $\mathbb{P}$ is defined to be

$$[\![\mathbb{P}]\!] \coloneqq \mathbf{lfp}_{\mathbf{Rel}^n}(\mathcal{I})$$

and may be calculated by iterative application of $\mathcal{I}$ to $\perp$ until a fixed point is reached.

Whether or not this program semantics is correct will depend of course on whether the fixed point exists. Typically this is ensured by constraining the program such that $\mathcal{I}$ is monotone (or, in the context of a dcpo, continuous). In the setting of Datalog, all programs are continuous and, since the underlying lattices are finite, the fixed point computation terminates in a finite number of iterations, so we can apply 4.5.4.

We can easily extend a derivative for the formula semantics to a derivative for the immediate consequence operator $\mathcal{I}$. Putting this together with the results from 4.5.2, we now have created *modular* proofs for the two main results, which we hope would allow us to preserve them in the face of changes to the underlying language.

Corollary 4.5.1. Datalog program semantics can be evaluated incrementally.

Proof. Corollary of Theorem 4.5.3 and Proposition 4.5.7. □

Corollary 4.5.2. Datalog program semantics can be incrementally maintained with changes to relations.

Proof. Corollary of Theorem 4.5.4 and Proposition 4.5.7. □

Neither of these results are novel: indeed the first one is merely the very well-known method of semi-naive evaluation, while there is already extensive discussion of incremental maintenance in the literature (see e.g. [50, 51]). Our contribution lies in the principled, compositional nature of our approach: our proofs follow trivially from our derivatives for non-recursive Datalog programs and the derivative for the least fixed point operator.

Chapter 5

Change action models

The category $\mathrm{CAct}(\mathbf{C})$ of change actions offers a natural and simple model for talking about the notion of "discrete" differentiation that arises in the context of incremental computation. One might object, however, that no definition of differentiability is complete without a study of smoothness or higher differentiability of some sort.

Indeed, a differential map $\overline{f} : \overline{A} \to \overline{B}$ associates a derivative ∂f to a $\mathbf{C}$-map f. But what should the derivative of ∂f be? Its type is $\partial f : A \times \Delta A \to \Delta B$ – it seems reasonable, then, that the derivative $\partial^2 f$ of such a ∂f should have type

$$\partial^2 f : (A \times \Delta A) \times \Delta(A \times \Delta A) \to \Delta(\Delta B)$$

But this type does not immediately make sense, as it is not clear what the "second-order change spaces" $\Delta(\Delta A), \Delta(\Delta B)$ should be.

We recall from Remark 4.4.1 that a similar issue arose when we tried to build pointwise functional change actions. The existence of these was predicated upon changes of type $\delta f : A \to \Delta B$ being differentiable, but there was, in general, no clear way to pick a change action on ΔB with respect to which differentiability could be decided.

A change action model is precisely a way of endowing each object of a category $\mathbf{C}$ with the structure of a change action, and associating every $\mathbf{C}$-map to a corresponding differential map in such a way that the chain rule is satisfied (that is, the derivative associated to the map $g \circ f$ is precisely the one that would be obtained by applying the chain rule to the derivatives associated to f and g respectively).

In a change action model, every object A is assigned a fixed change action $\overline{A}$ – and since these change actions are internal to $\mathbf{C}$, the change space ΔA itself carries a change action $\overline{\Delta A}$, thus the higher change spaces above can be given an interpretation.

This chapter has two main sections: first, in Section 5.1, we introduce the formal definition of a change action model and study some basic properties, such as the structure of products and exponentials and the existence of a tangent bundle functor. Section 5.2 then lists a series of concrete examples of change action models of particular interest.

Many of the results in this chapter have appeare in print in [5]. Some passages and figures have been reproduced verbatim with permission.

5.1 Coalgebras for CAct and Their Properties

Recall that a *copointed endofunctor* is a pair (F, σ) where the endofunctor $\mathrm{F} : \mathbf{C} \to \mathbf{C}$ is equipped with a natural transformation $\sigma : \mathrm{F} \dot{\to} \mathrm{Id}$. A *coalgebra for a copointed endofunctor* (F, σ) is an object A of $\mathbf{C}$ equipped with a section α of σ, that is, a morphism $\alpha : A \to \mathrm{F}A$ such that $\sigma_A \circ \alpha = \mathbf{id}_A$.

Definition 5.1.1. A *change action model* on a Cartesian category $\mathbf{C}$ is a coalgebra $\alpha : \mathbf{C} \to \mathrm{CAct}(\mathbf{C})$ for the copointed endofunctor $(\mathrm{CAct}, \varepsilon)$.

Explicitly, a product-preserving functor $\alpha : \mathbf{C} \to \mathrm{CAct}(\mathbf{C})$ is a change action model whenever $\varepsilon \circ \alpha = \mathbf{id}_{\mathbf{C}}$, i.e. whenever α endows every object of $\mathbf{C}$ with the structure of a change action, and associates every map with a choice of derivative for it.

Assumption. Throughout Section 5.1, we fix a change action model $\alpha : \mathbf{C} \to \mathrm{CAct}(\mathbf{C})$.

Given an object A of $\mathbf{C}$, the coalgebra α specifies a (internal) change action $\alpha(A) = (A, \Delta A, \oplus_A, +_A, 0_A)$ in $\mathrm{CAct}(\mathbf{C})$. We abuse the notation and write ΔA for the carrier object of the monoid specified by $\alpha(A)$; similarly for $+_A, \oplus_A$, and 0_A – we will also write $\partial[f]$ for the derivative assigned to f by α[1].

Given a morphism $f : A \to B$ in $\mathbf{C}$, there is an associated differential map $\alpha(f) = (f, \partial f) : \alpha(A) \to \alpha(B)$. Since

$$\partial[f] : A \times \Delta A \to \Delta B$$

is also a $\mathbf{C}$-morphism, there is a corresponding differential map $\alpha(\partial[f]) = (\partial f, \partial^2 f)$ in $\mathrm{CAct}(\mathbf{C})$, where

$$\partial^2[f] : (A \times \Delta A) \times (\Delta A \times \Delta^2 A) \to \Delta^2 B$$

[1]It may seem inconsistent to write ∂f for a derivative and $\partial[f]$ for the derivative that α associates to f. We write $\partial[f]$ to emphasise the role of ∂ as an *operator*, whereas writing ∂f for a derivative of f is simply a convention.

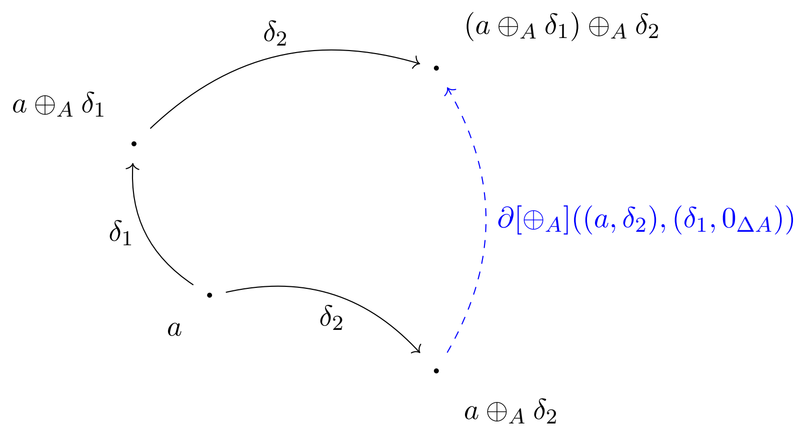

δ_2
$(a \oplus_A \delta_1) \oplus_A \delta_2$
$a \oplus_A \delta_1$
δ_1
$\partial[\oplus_A]((a, \delta_2), (\delta_1, 0_{\Delta A}))$
a
δ_2
$a \oplus_A \delta_2$

Figure 5.1: Square-filling property of $\partial[\oplus_A]$

is a second derivative for f – note that $\Delta(A \times \Delta A) = \Delta A \times \Delta^2 A$, since α is, by definition, product-preserving. Iterating this process, we obtain an n-th derivative $\partial^n[f]$ for every $\mathbf{C}$-morphism f.

An immediate consequence of the definition of a change action model is that the structure maps $\oplus_A, +_A, 0_A$ associated to any object A are themselves differentiable, but we can make no further claims about their derivatives[2] other that they are indeed derivatives (in the sense that they satisfy Definition 3.2.2).

Yet there is some deep geometric intuition to be extracted from the existence of a derivative for $\oplus$. As mentioned before, one can think of elements of A as points in a space, and a change δ as a path from a to $a \oplus_A \delta$. Let us consider, then, a configuration of paths as in Figure 5.1. In a more "classical" setting (e.g. if the δ_i were simply paths in some geometric space), we could always go from the point $a \oplus_A \delta_2$ to the point $(a \oplus_A \delta_1) \oplus_A \delta_2$, by composing δ_2^{-1} with δ_1. In change actions, however, changes may not be invertible and so such a δ_2^{-1} may not exist. Since $\oplus_A$ is differentiable, however, we can always find a path that "fills" the missing edge in the diagram, as shown in blue.

What's more, consider a second-order change δ^2 – this change maps some first-order change δ to the first-order change $\delta \oplus_{\Delta A} \delta^2$. Stretching the metaphor, such a second-order change can perhaps be thought of as a (directed) homotopy from path δ to path $\delta \oplus_{\Delta A} \delta^2$. Again one can think of the derivative of the structure map $\oplus_A$ as allowing us to "fill in" the missing edge in the triangle in Figure 5.2.

[2]This is not entirely true – we shall show later that the derivatives of constant maps are always zero, and so $\partial[0_A] = 0_{\Delta A}$.

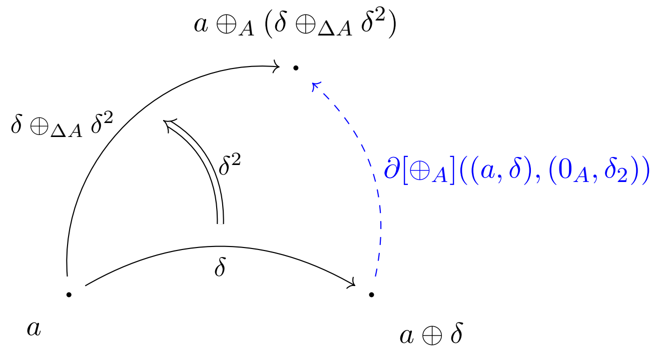

$a \oplus_A (\delta \oplus_{\Delta A} \delta^2)$
$\delta \oplus_{\Delta A} \delta^2$
δ^2
$\partial[\oplus_A]((a, \delta), (0_A, \delta_2))$
δ
a
$a \oplus \delta$

Figure 5.2: Triangle-filling property of $\partial[\oplus_A]$

Composition of changes $+_A$ in a change action model is likewise differentiable, a fact which sheds light on the regularity condition on derivatives. Indeed, take a point a and changes δ_1, δ_2 – derivatives in change actions need not be additive, and so $\partial[f](a, \delta_1 +_A \delta_2) \neq \partial[f](a, \delta_1) +_B \partial[f](a, \delta_2)$. But derivatives do satisfy the (weaker) condition of regularity, which states the following:

$$\partial[f](a, \delta_1 +_A \delta_2) = \partial[f](a, \delta_1) +_A \partial[f](a \oplus_A \delta_1, \delta_2)$$

Observe, then, that in any change action model $\partial[f]$ is itself differentiable, admitting a derivative $\partial^2[f]$. In particular, letting $\omega = \partial^2[f]((a, \delta_2), (\delta_1, 0_{\Delta A}))$, the above equation can be equivalently written as:

$$\begin{aligned}\partial[f](a, \delta_1 +_A \delta_2) &= \partial[f](a, \delta_1) +_A \partial[f](a, \delta_2) \oplus_{\Delta A} \omega \\ &= (\partial[f](a, \delta_1) +_A \partial[f](a, \delta_2)) \oplus_{\Delta A} \partial[+_A]((\partial[f](a, \delta_1), \partial[f](a, \delta_2)), (0_{\Delta A}, \omega))\end{aligned}$$

Independently of the exact value of the second-order term, the above expression can be read as stating that regular derivatives are additive "up to some second-order perturbation". More generally, whenever we have first-order changes δ_1, δ_2 and second-order changes δ_1^2, δ_2^2, differentiability of addition defines something akin to a "horizontal" composition $\delta_1^2 \star \delta_2^2 \equiv \partial[+_A]((\delta_1, \delta_2), (\delta_1^2, \delta_2^2))$, represented by the red arrow in the diagram below.

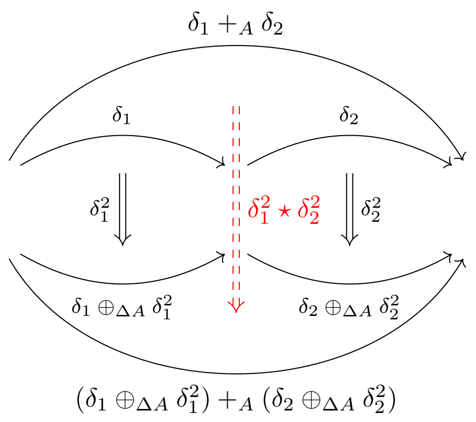

$\delta_1 +_A \delta_2$
δ_1
δ_2
δ_1^2
$\delta_1^2 \star \delta_2^2$
δ_2^2
$\delta_1 \oplus_{\Delta A} \delta_1^2$
$\delta_2 \oplus_{\Delta A} \delta_2^2$
$(\delta_1 \oplus_{\Delta A} \delta_1^2) +_A (\delta_2 \oplus_{\Delta A} \delta_2^2)$

Figure 5.3: Horizontal composition of second-order changes

This horizontal composition is strikingly similar to the horizontal composition of 2-cells in a 2-category – this is because, as we shall show later, it is precisely that!

5.1.1 Tangent Bundles in Change Action Models

The tangent bundle functor, which maps every manifold to its tangent bundle and pairs every function with its differential, is a construction of significant importance in differential geometry, to the extent that one can formalise much of it simply in terms of a "category with tangent structure" [25], that is to say, a Cartesian category equipped with an endofunctor that behaves like the tangent bundle functor in some suitable sense.

It is possible to define an analogue of the tangent bundle functor in the setting of change action models, which exhibits similar properties. In particular, the derivative condition can be stated very elegantly in terms of the tangent bundle functor.

Definition 5.1.2. The **tangent bundle functor** $\mathrm{T} : \mathbf{C} \to \mathbf{C}$ is defined by:

$$\mathrm{T}(A) \coloneqq A \times \Delta A$$
$$\mathrm{T}(f) \coloneqq \langle f \circ \pi_1, \partial[f] \rangle$$

Notation. For clarity and brevity, we will abbreviate $\pi_i \circ \pi_j$ as π_{ij}.

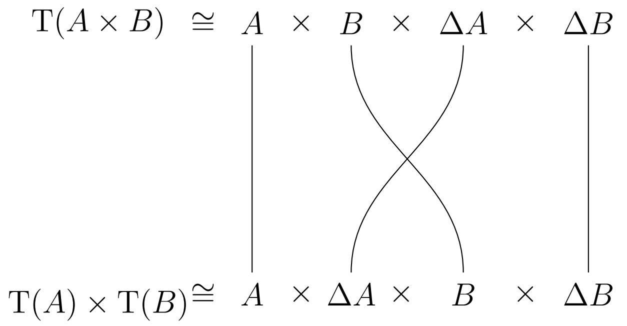
Theorem 5.1.1. The tangent bundle functor T is strong monoidal. In particular, the map

$$\phi_{A,B} : \mathrm{T}(A \times B) \to \mathrm{T}(A) \times \mathrm{T}(B)$$
$$\phi_{A,B} \coloneqq \langle \mathrm{T}(\pi_1), \mathrm{T}(\pi_2) \rangle$$

is an isomorphism. Consequently, given maps $f : A \to B$ and $g : A \to C$, we have

$$\mathrm{T}(\langle f, g\rangle) \circ \phi_{A,B} = \langle \mathrm{T}(f), \mathrm{T}(g)\rangle$$

Proof. Note that $\mathrm{T}(\pi_i) = \langle \pi_{i1}, \pi_{i2}\rangle$, hence $\phi_{A,B} = \langle\langle \pi_{11}, \pi_{12}\rangle, \langle \pi_{21}, \pi_{22}\rangle\rangle$. That it is a natural isomorphism is immediately clear from its representation as a string diagram:


$\mathrm{T}(A \times B) \cong A \times B \times \Delta A \times \Delta B$
$\mathrm{T}(A) \times \mathrm{T}(B) \cong A \times \Delta A \times B \times \Delta B$

□

A consequence of the structure of products in CAct(**C**) is that the map $\oplus_{A\times B}$ is defined pointwise in terms of $\oplus_A, \oplus_B$. This result can be stated in terms of the previous isomorphism.

Lemma 5.1.1. Let $\phi_{A,B} : \mathrm{T}(A \times B) \to \mathrm{T}(A) \times \mathrm{T}(B)$ be the canonical isomorphism described above. Then $\oplus_{A\times B} \circ \phi_{A,B} = \oplus_A \times \oplus_B$.

It will often be convenient to operate directly on the functor T, rather than on the underlying derivatives. In particular, in Chapter 6 we will make use of the natural transformations below.

Lemma 5.1.2. The following families of morphisms are natural transformations:

$$\begin{aligned}
\pi_1, \oplus_A &: \mathrm{T}(A) \to A \\
\eta_A &: A \to \mathrm{T}(A) \\
\eta_A &:= \langle \mathbf{id}_A, 0\rangle \\
\mu_A &: \mathrm{T}^2(A) \to \mathrm{T}(A) \\
\mu_A &:= \langle \pi_{11}, \pi_{12} + (\pi_{21} \oplus \pi_{22})\rangle
\end{aligned}$$

Furthermore, $\mu_A \circ \mathrm{T}(\eta_A) = \mu_A \circ \eta_{\mathrm{T}(A)} = \mathbf{id}_{\mathrm{T}(A)}$. However, $\mu_A \circ \mu_{\mathrm{T}(A)} \neq \mu_A \circ \mathrm{T}(\mu_A)$ and therefore the triple (T, η, μ) is *not* a monad (we establish monadicity of T for the special case of difference categories in Theorem 6.3.1).

Proof. We shall show naturality of η and μ, as we shall make use of it later. For η we have:

$$\mathrm{T}(f) \circ \eta_A = \langle f \circ \pi_1, \partial[f] \rangle \circ \eta_A = \langle f \circ \pi_1 \circ \langle \mathbf{id}_A, 0 \rangle, \partial[f] \circ \langle \mathbf{id}_A, 0 \rangle \rangle = \langle f, 0 \rangle = \eta_B \circ f$$

For the naturality of μ, first, it is easy to check that:

$$\mathrm{T}^2(f) = \big\langle \langle f \circ \pi_{11}, \partial[f] \circ \pi_1 \rangle, \big\langle \partial[f] \circ (\pi_1 \times \pi_1), \partial^2[f] \big\rangle \big\rangle$$

from which we obtain:

$$\begin{aligned} \mu_B \circ \mathrm{T}^2(f) &= \langle \pi_{11}, \pi_{12} + (\pi_{21} \oplus \pi_{22}) \rangle \circ \mathrm{T}^2(f) \\ &= \big\langle f \circ \pi_{11}, \partial[f] \circ (\pi_1 \times \pi_1) + \big((\partial[f] \circ \pi_1) \oplus \partial^2[f] \big) \big\rangle \end{aligned}$$

On the other hand:

$$\begin{aligned} \mathrm{T}(f) \circ \mu_A &= \langle f \circ \pi_1, \partial[f] \rangle \circ \langle \pi_{11}, \pi_{12} + (\pi_{21} \oplus \pi_{22}) \rangle \\ &= \langle f \circ \pi_{11}, \partial[f] \circ \langle \pi_{11}, \pi_{12} + (\pi_{21} \oplus \pi_{22}) \rangle \rangle \\ &= \langle f \circ \pi_{11}, \partial[f] \circ \langle \pi_{11}, \pi_{12} \rangle + \partial[f] \circ \langle \pi_{11} \oplus \pi_{12}, \pi_{21} \oplus \pi_{22} \rangle \rangle \\ &= \big\langle f \circ \pi_{11}, \partial[f] \circ \langle \pi_{11}, \pi_{12} \rangle + (\partial[f] \circ \langle \pi_{11}, \pi_{21} \rangle \oplus (\partial^2[f] \circ \langle \langle \pi_{11}, \pi_{21} \rangle, \langle \pi_{12}, \pi_{22} \rangle \rangle)) \big\rangle \\ &= \big\langle f \circ \pi_{11}, \partial[f] \circ (\pi_1 \times \pi_1) + (\partial[f] \circ \pi_1 \oplus \partial^2[f]) \big\rangle \end{aligned}$$

Now we show $\mu_A \circ \eta_A = \mathbf{id}_{\mathrm{T}(A)}$.

$$\begin{aligned} \mu_A \circ \eta_{\mathrm{T}(A)} &= \langle \pi_{11}, \pi_{12} + (\pi_{21} \oplus \pi_{22}) \rangle \circ \big\langle \mathbf{id}_{T(A)}, 0 \big\rangle \\ &= \langle \pi_1, 0 + (\pi_2 \oplus 0) \rangle \\ &= \langle \pi_1, \pi_2 \rangle \\ &= \mathbf{id}_{\mathrm{T}(A)} \end{aligned}$$

Finally, we check $\mu_A \circ \eta_{\mathrm{T}(A)} = \mathbf{id}_{\mathrm{T}(A)}$.

$$\begin{aligned} \mu_A \circ T(\eta_A) &= \langle \pi_{11}, \pi_{12} + (\pi_{21} \oplus \pi_{22}) \rangle \circ \mathrm{T}(\langle \mathbf{id}_A, 0 \rangle) \\ &= \langle \pi_{11}, \pi_{12} + (\pi_{21} \oplus \pi_{22}) \rangle \circ \langle \langle \pi_1, 0 \rangle, \langle \pi_2, 0 \rangle \rangle \\ &= \langle \pi_1, \pi_2 + (0 \oplus 0) \rangle \\ &= \mathbf{id}_{\mathrm{T}(A)} \end{aligned}$$

The second monad law, however, fails to hold: intuitively, the term $\mu_A \circ \mathrm{T}(\mu_A)$ involves the derivative $\partial[\mu_A]$ which in turn makes use of the derivatives $\partial[\oplus], \partial[+]$ which, however, do not appear in the expansion of $\mu_A \circ \mu_{\mathrm{T}(A)}$. Hence the equation cannot be established without making assumptions about the form of $\partial[\oplus], \partial[+]$. $\square$

Remark 5.1.1. Although we have omitted the corresponding proof for being trivial, it is an interesting observation that naturality of $\oplus$ is equivalent to the derivative condition, in that both propositions amount to the commutativity of the diagram below.

$$\begin{tikzcd}
\mathrm{T}(A) \arrow[r, "\langle f\circ\pi_1, \partial[f]\rangle \equiv \mathrm{T}(f)"] \arrow[d, "\oplus_A"'] & \mathrm{T}(B) \arrow[d, "\oplus_B"] \\
A \arrow[r, "f"] & B
\end{tikzcd}$$

The tangent bundle functor in a change action model behaves similarly to the homonymous notion in Cartesian differential categories – in particular, the above natural transformations π_1, ε, μ also exist in Cartesian differential categories (the calculations being essentially identical). An important property of the tangent bundle in a CDC which does *not* hold for change action models, however, is that ε, μ make T into a monad. In Chapter 6 we will describe a setting of particularly well-behaved change actions where the tangent bundle functor is indeed a monad.

5.1.2 Exponentials in Change Action Models

A particularly interesting class of change action models are those that are also Cartesian closed. Surprisingly, this has as an immediate consequence that the operation of differentiation is itself internal to the category, in the following sense:

Lemma 5.1.3. Whenever $\mathbf{C}$ is Cartesian closed, there is a morphism $\mathbf{d}_{A,B} : (A \Rightarrow B) \to (A \times \Delta A) \Rightarrow \Delta B$ such that, for any morphism $f : \mathbf{1} \times A \to B$, $\mathbf{d}_{A,B} \circ \Lambda f = \Lambda(\partial[f] \circ \langle\langle \pi_1, \pi_{12}\rangle, \langle \pi_1, \pi_{22}\rangle\rangle)$.

Proof. Consider the evaluation map $\mathbf{ev}_{A,B} : (A \Rightarrow B) \times A \to B$ in $\mathbf{C}$. Its derivative $\partial[\mathbf{ev}_{A,B}]$ has type

$$\partial[\mathbf{ev}_{A,B}] : ((A \Rightarrow B) \times A) \times (\Delta(A \Rightarrow B) \times \Delta A) \to \Delta B$$

(note that we make no assumptions about the structure of $\Delta(A \Rightarrow B)$).

Then consider the following composite:

$$\partial[\mathbf{ev}_{A,B}] \circ \langle\langle \pi_1, \pi_{12}\rangle, \langle 0_{A\Rightarrow B}, \pi_{22}\rangle\rangle : (A \Rightarrow B) \times (A \times \Delta A) \to \Delta B$$

By the universal property of the exponential, we have $\mathbf{ev}_{A,B} \circ (\Lambda f \times \mathbf{id}_A) = f$ and therefore $\alpha(\mathbf{ev}_{A,B} \circ (\Lambda f \times \mathbf{id}_A)) = \alpha(f)$. Thus:

$$\begin{aligned}
\partial[f] &= \partial[\mathbf{ev}_{A,B} \circ (\Lambda f \times \mathbf{id}_A)] \\
&= \partial[\mathbf{ev}_{A,B} \circ \langle \Lambda f, \mathbf{id}_A\rangle] \\
&= \partial[\mathbf{ev}_{A,B}] \circ \langle\langle \Lambda f, \mathbf{id}_A\rangle \circ \pi_1, \langle 0_{A\Rightarrow B}, \pi_2\rangle\rangle \\
&= \partial[\mathbf{ev}_{A,B}] \circ \langle\langle \pi_1, \pi_{12}\rangle, \langle 0_{A\Rightarrow B}, \pi_{22}\rangle\rangle \circ \langle \Lambda f, \langle \pi_1, \pi_2\rangle\rangle \\
&= \partial[\mathbf{ev}_{A,B}] \circ \langle\langle \pi_1, \pi_{12}\rangle, \langle 0_{A\Rightarrow B}, \pi_{22}\rangle\rangle \circ \langle \Lambda f, \mathbf{id}_{A\times\Delta A}\rangle
\end{aligned}$$

from which it follows trivially that

$$\mathbf{d}_{A,B} = \Lambda(\partial[\mathbf{ev}_{A,B}]) \circ \langle\langle\pi_1, \pi_{12}\rangle, \langle 0_{A\Rightarrow B}, \pi_{22}\rangle\rangle$$

is the desired morphism. □

We would like the tangent bundle functor to preserve the exponential structure in the same way that it preserves the structure of products. In particular, we would hope the tangent structure of the exponentials to be pointwise, i.e. $\mathrm{T}(A \Rightarrow B) \cong A \Rightarrow \mathrm{T}(B)$, with $\oplus_{A\Rightarrow B}$ being the pointwise lifting of $\oplus_B$. Furthermore we aim for a result of the form $\frac{\partial(\lambda y.t)}{\partial x}(u) = \lambda y. \left(\frac{\partial t}{\partial x}(u)\right)$, which holds in Cartesian differential categories and is used to propagate differentiation through abstractions in the differential λ-calculus. These results can in fact be obtained as a consequence of the tangent bundle functor being representable.

Definition 5.1.3. If $\mathbf{C}$ is Cartesian closed, an *infinitesimal object* [3] U is an object of $\mathbf{C}$ such that the tangent bundle functor T is represented by the internal Hom-functor $U \Rightarrow (\cdot)$, i.e. there is a natural isomorphism $\mathcal{D} : (U \Rightarrow (\cdot)) \cong \mathrm{T}$.

Theorem 5.1.2. Whenever there is an infinitesimal object in $\mathbf{C}$, the tangent bundle $\mathrm{T}(A \Rightarrow B)$ is naturally isomorphic to $A \Rightarrow \mathrm{T}(B)$. Furthermore, this isomorphism respects the structure of T, in the sense that the following diagrams commutes:

$$\begin{tikzcd}
\mathrm{T}(A \Rightarrow B) \arrow[r, "\cong"] \arrow[d, "\oplus_{A\Rightarrow B}"'] & A \Rightarrow \mathrm{T}(B) \arrow[dl, "\mathbf{id}_{A\Rightarrow\oplus_B}"] \\
A \Rightarrow B &
\end{tikzcd}
\qquad
\begin{tikzcd}
\mathrm{T}(A \Rightarrow B) \arrow[r, "\cong"] \arrow[d, "\pi_1"'] & A \Rightarrow \mathrm{T}(B) \arrow[dl, "\mathbf{id}_{A\Rightarrow\pi_1}"] \\
A \Rightarrow B &
\end{tikzcd}$$

$$\begin{tikzcd}
\mathrm{T}^2(A \Rightarrow B) \arrow[r, "\cong"] \arrow[d, "\mu_{A\Rightarrow B}"'] & A \Rightarrow \mathrm{T}^2(B) \arrow[d, "\mathbf{id}_{A\Rightarrow\mu_B}"] \\
\mathrm{T}(A \Rightarrow B) \arrow[r, "\cong"] & A \Rightarrow \mathrm{T}(B)
\end{tikzcd}
\qquad
\begin{tikzcd}
A \Rightarrow B \arrow[r, "\cong"] \arrow[d, "\varepsilon_{A\Rightarrow B}"'] & A \Rightarrow \mathrm{T}(B) \arrow[d, "\mathbf{id}_{A\Rightarrow\varepsilon_B}"] \\
\mathrm{T}(A \Rightarrow B) \arrow[r, "\cong"] & A \Rightarrow \mathrm{T}(B)
\end{tikzcd}$$

Proof. The desired isomorphism can be obtained from the sequence of (natural) isomorphisms:

$$\mathrm{T}(A \Rightarrow B) \cong U \Rightarrow (A \Rightarrow B) \cong A \Rightarrow (U \Rightarrow B) \cong A \Rightarrow \mathrm{T}(B)$$

[3]The concept of an "infinitesimal object" is borrowed from synthetic differential geometry[59]. However, there is nothing intrinsically "infinitesimal" about these objects in our setting – we will later see an example which is distinctly discrete .

By the Yoneda lemma, the natural transformation

$$\oplus_A \circ \mathcal{D} : (U \Rightarrow A) \to A$$

is precisely evaluation at some fixed element $U_\oplus : \mathbf{1} \to U$ (and, conversely, evaluating $\mathcal{D}(t)$ at $U_\oplus$ is precisely $\oplus_A(t)$).

In more detail, the map $\oplus_A \circ \mathcal{D}_A : (U \Rightarrow A) \to A$ is a natural transformation from $(U \Rightarrow (\cdot))$ into Id and, therefore, corresponds to a natural transformation from $\mathbf{C}\,[\mathbf{1}, U \Rightarrow (\cdot)]$ into $\mathbf{C}\,[\mathbf{1}, \cdot]$ or, equivalently, a natural transformation from $\mathbf{C}\,[U, \cdot]$ into $\mathbf{C}\,[\mathbf{1}, \cdot]$. Hence, by the covariant Yoneda lemma, it must correspond to some element $U_\oplus \in \mathbf{C}\,[\mathbf{1}, U]$. By a similar argument, μ corresponds precisely to precomposition with a map $U_\mu : U \to U \times U$, and ε corresponds to sending each $a \in A$ to the constant function mapping every $u \in U$ to a.

Commutativity of the above diagrams then can be shown by equational reasoning in the internal logic of the CCC $\mathbf{C}$ (we show the calculations for the first diagram, the rest being obtained in a similar manner). Whenever $f : \mathrm{T}(A \Rightarrow B)$, we obtain:

$$\begin{aligned}\lambda a. \oplus_B (\mathcal{D}(\lambda u. \mathcal{D}^{-1}(f)(u)(a))) &= \lambda a. \mathcal{D}^{-1}(f)(U_\oplus)(a) \\ &= \mathcal{D}^{-1}(f)(U_\oplus) \\ &= \oplus_{A \Rightarrow B}(f)\end{aligned}$$

□

The above result is quite significant, as it characterises $\mathrm{T}(A \Rightarrow B)$ (almost) entirely in terms of the structure of $\mathrm{T}(B)$, lifted pointwise. This tells us that, in a change action model with an infinitesimal object, the change action structure of the exponential objects is essentially the convenient pointwise structure that we introduced in Chapter 4, Definition 4.4.2. That is to say, a change on $A \Rightarrow B$ is precisely a function mapping an element of A to a change on B, and applying a functional change is done by lifting $\oplus_B$ pointwise.

Whenever the object B is equipped with a point $b : \mathbf{1} \to B$, it is possible to use the previous result to infer that addition and zero on the change space $\Delta(A \Rightarrow B)$ are also pointwise as, for example, $+_B$ can be written in terms of μ as:

$$+_B = \pi_2 \circ \mu_B \circ \langle \langle a \circ !, \pi_1 \rangle, \langle \pi_2, 0_{\Delta B} \rangle \rangle : \Delta B \times \Delta B \to \Delta B$$

We might expect a property similar to **[DλC.1]** to hold in Cartesian closed change action models or, at least, in those models that admit an infinitesimal object. While this is not quite true, a weaker result is available: given a map $f : A \times B \to C$ we cannot, in general, prove that $\partial[\Lambda(f)]$ is equal to $\Lambda(\partial[f] \circ \langle \pi_1 \times \mathbf{id}, \pi_2 \times 0 \rangle)$, but we can prove that both expressions have the same *effect*, i.e. their action *qua* changes in $\Delta(B \Rightarrow C)$ is identical.

Theorem 5.1.3. Given objects A, B, let ∂_1 denote the map

$$\partial_1 \coloneqq \langle \pi_1 \times \mathbf{id}_B, \pi_2 \times 0 \rangle : \mathrm{T}(A) \times B \to \mathrm{T}(A \times B)$$

Then for all $f : A \times B \to C$, we have

$$\oplus \circ \mathrm{T}(\Lambda(f)) = (\Rightarrow \oplus) \circ \Lambda(\mathrm{T}(f) \circ \partial_1)$$

Proof. The map ∂_1 can alternatively written as $\phi \circ (\mathbf{id}_{A \times B} \times \langle \mathbf{id}_{\Delta A}, 0 \rangle)$. We can then apply naturality of $\oplus$ and the fact that $\oplus \circ \phi = \oplus \times \oplus$:

$$\begin{aligned}
(\Rightarrow \oplus) \circ \Lambda(\mathrm{T}(f) \circ \partial_1) &= \Lambda(\oplus \circ \mathrm{T}(f) \circ \partial_1) \\
&= \Lambda(f \circ \oplus \circ \partial_1) \\
&= \Lambda(f \circ \oplus \circ \phi \circ (\mathbf{id} \times \langle \mathbf{id}, 0 \rangle)) \\
&= \Lambda(f \circ ((\oplus \circ \mathbf{id}) \times (\oplus \circ \langle \mathbf{id}, 0 \rangle))) \\
&= \Lambda(f \circ (\oplus \times \mathbf{id})) \\
&= \Lambda(f) \circ \oplus \\
&= \oplus \circ \mathrm{T}(\Lambda(f))
\end{aligned}$$

□

5.2 Examples of Change Action Models

5.2.1 Generalised Cartesian Differential Categories

Cartesian differential categories, as described in Section 2.3, and generalised CDCs, are perhaps the most well-studied categorical formulation of the "classical" notion of derivative from differential calculus.

Example 5.2.1 ([17])**.** The category **Smooth** of smooth maps between Euclidean spaces $\mathbb{R}^n$ is a Cartesian differential category.

Example 5.2.2 ([30])**.** The category $\mathbf{Smooth}_{\subseteq}$ of smooth maps between open subsets of $\mathbb{R}^n$ is a generalised Cartesian differential category.

We show that change action models generalise GCDC in that GCDCs give rise to change action models in two different ways.Throughout this subsection we will take $\mathbf{C}$ to be a GCDC. The reader may find it helpful to have Definition 2.3.3 at hand for reference.

1. The Flat Model. Define the functor $\alpha : \mathbf{C} \to \mathrm{CAct}(\mathbf{C})$ as follows. Let $f : A \to B$ be a $\mathbf{C}$-morphism. Then $\alpha(A) \coloneqq (A, L_0(A), \pi_1, +_A, 0_A)$ and $\alpha(f) \coloneqq (f, \mathsf{D}[f])$.

Theorem 5.2.1. The functor α is a change action model.

Proof. We need to check that α is well-defined and a right-inverse to the forgetful functor.

First, note that $\alpha(f)$ trivially satisfies the derivative property:

$$f \circ \pi_1 = \pi_1 \circ \langle f \circ \pi_1, \mathsf{D}[f] \rangle$$

Furthermore, by the axiom [**CDC.2**] of generalised Cartesian differential categories, we have:

$$\begin{gathered}\mathsf{D}[f] \circ \langle \mathbf{id}, 0_A \circ ! \rangle = 0_B \\ \mathsf{D}[f] \circ \langle a, + \circ \langle u, v \rangle \rangle = + \circ \langle \mathsf{D}[f] \circ \langle a, u \rangle, \mathsf{D}[f] \circ \langle a, v \rangle \rangle\end{gathered}$$

This entails that the map $\alpha(f) = (f, \mathsf{D}[f])$ is regular and, therefore, a differential map. Functoriality of α is a direct consequence of [**CDC.3**] and [**CDC.5**].

Furthermore, α preserves products (up to isomorphism) since, by definition, $L(A \times B) = L(A) \times L(B)$ and by axioms [**CDC.3**] and [**CDC.4**], and is trivially a right-inverse to the forgetful functor. Therefore α is a change action model. $\square$

2. The Kleisli Model. GCDCs admit a tangent bundle functor, defined analogously to the standard notion in differential geometry. Let $f : A \to B$ be a $\mathbf{C}$-morphism. Define its tangent bundle functor $\mathrm{T} : \mathbf{C} \to \mathbf{C}$ as: $\mathrm{T}(A) \coloneqq A \times L_0(A)$, and $\mathrm{T}(f) \coloneqq \langle f \circ \pi_1, \mathsf{D}[f] \rangle$. The functor T is in fact a monad, with unit $\eta = \langle \mathbf{id}, 0_A \rangle : A \to A \times L_0(A)$ and multiplication $\mu : (A \times L_0(A)) \times (L_0(A) \times L_0(A)) \to A \times L_0(A)$ defined by the composite:

$$(A \times L_0(A)) \times (L_0(A) \times L_0(A)) \xrightarrow{\langle \pi_{11}, \langle \pi_{21}, \pi_{12} \rangle \rangle} A \times (L_0(A) \times L_0(A)) \xrightarrow{\mathbf{id} \times +_A} A \times L_0(A)$$

Thus we can work on the Kleisli category of this functor by $\mathbf{C}_{\mathrm{T}}$ which has geometric significance as a category of generalised vector fields.

Lemma 5.2.1. The tuple $(A, L_0(A), \mathbf{id}_A \times \mathbf{id}_{L_0(A)}, \eta \circ +_A, \eta \circ 0_A)$ is a change action in the category $\mathbf{C}_{\mathrm{T}}$.

Theorem 5.2.2. We define the functor $\alpha_{\mathrm{T}} : \mathbf{C}_{\mathrm{T}} \to \mathrm{CAct}(\mathbf{C}_{\mathrm{T}})$ as follows. Given an object A in $\mathbf{C}_{\mathrm{T}}$, we set:

$$\alpha_{\mathrm{T}}(A) \coloneqq (A, L_0(A), \mathbf{id}_A \times \mathbf{id}_{L_0(A)}, \eta \circ +_A, \eta \circ 0_A)$$

Given a $\mathbf{C}_\mathrm{T}$-map $f : A \to_\mathrm{T} B$, we set:

$$\alpha_\mathrm{T}(f) \coloneqq (f, \mathsf{D}[f])$$

Thus defined, the functor α_T is a change action model on $\mathbf{C}$.

Proof. A proof for a much stronger result will be shown in Chapter 6. Although it is written for the setting of Cartesian differential categories, it can be easily adapted to GCDCs. □

Remark 5.2.1. The converse is not true: in general the existence of a change action model on $\mathbf{C}$ does not imply that $\mathbf{C}$ satisfies the axioms of a Cartesian differential category. However, as we shall see in Chapter 6, most of the CDC axioms do hold in many well-behaved classes of change action models (with the exception that derivatives may not be additive).

5.2.2 Finite Differences and Boolean Differential Calculus

Consider the category $\mathbf{Grp}_\star$ whose objects are all the groups (in $\mathbf{Set}$) and where morphisms $f : \mathcal{F} \to \mathcal{G}$ are all the (set-theoretical) functions between the carrier sets F and G. This is a Cartesian closed category which can be endowed with the structure of a $(\mathrm{CAct}, \varepsilon)$-coalgebra α in the following way.

Definition 5.2.1. Given a group $\mathcal{F} = (F, \cdot, 1)$, define the change action $\alpha(\mathcal{G})$ by:

$$\alpha(\mathcal{F}) \coloneqq (F, F, \cdot, \cdot, 1).$$

Given an arbitrary function $f : F \to G$ between the carrier sets of $\mathcal{F}$ and $\mathcal{G}$ respectively, define the differential map $\alpha(f) \coloneqq (f, \partial[f])$ by:

$$\partial[f](x, \delta x) \coloneqq f(x)^{-1} \cdot f(x \cdot \delta x).$$

Notice that the derivative condition is satisfied:

$$f(x) \oplus_\mathcal{G} \partial[f](x, \delta x) = f(x) \cdot (f(x)^{-1} \cdot f(x \cdot \delta x)) = f(x \cdot \delta x) = f(x \oplus_\mathcal{F} \delta x);$$

hence $\partial[f]$ is a (regular) derivative[4] for f, and therefore $\alpha(f)$ is a map in $\mathrm{CAct}(\mathbf{Grp}_\star)$. The following result is then immediate.

[4]Note that $\partial[f]$ need not be additive in its second argument, and so derivatives in $\mathbf{Grp}_\star$ do not satisfy all the axioms of a Cartesian differential category.

Lemma 5.2.2. The mapping α defines a product-preserving functor from $\mathbf{Grp}_\star$ into $\mathrm{CAct}(\mathbf{Grp}_\star)$. Furthermore $\varepsilon_{\mathbf{Grp}_\star} \circ \alpha = \mathrm{Id}_{\mathbf{Grp}_\star}$ and hence α is a coalgebra for the pointed endofunctor CAct[5].

Remark 5.2.2. This category may seem rather strange, as its objects have "more structure" than its arrows do. That is, every object in $\mathbf{Grp}_\star$ is isomorphic to every other object with the same carrier set (in fact, assuming the Axiom of Choice, $\mathbf{Grp}_\star$ is straightforwardly equivalent, but not isomorphic, to the category of non-empty sets). The specifics of the group structure of each object are encoded into the functor α, since addition and subtraction can be recovered from α.

For example, given a group $\mathcal{F}$, and an element $x \in F$, the identity of $\mathcal{F}$ can be obtained by evaluating the derivative $\partial(\alpha\kappa_x)(x, x)$, where κ_x is the constant function sending every element of F to x. Similarly, addition can be recovered by differentiating the map 0_x which maps x to 0 and every other element to itself.

Despite its seeming novelty, this category is actually a generalisation of the well-known calculus of finite differences (see [48] for a quick introduction, or [56, 77] for a more in-depth treatment of the topic). The calculus of finite differences deals mainly with the notion of *discrete derivative* (or *discrete difference operator*) of a function $f : \mathbb{Z} \to \mathbb{Z}$, which is defined as $\delta f(x) \coloneqq f(x+1) - f(x)$. In fact this discrete derivative δf is (an instance of) the derivative of f *qua* morphism in $\mathbf{Grp}_\star$, i.e. $\delta f(x) = \partial[f](x, 1)$.

The calculus of finite differences has found applications in combinatorics and numerical computation. Our formulation via this change action model on $\mathbf{Grp}_\star$ has several advantages. First it justifies the chain rule, which is an important novel result. Secondly, it generalises the calculus to arbitrary groups. To illustrate this, consider the *Boolean differential calculus* [102, 103], a theory that applies methods from calculus to the space $\mathbb{B}^n$ of vectors of elements of some Boolean algebra $\mathbb{B}$, with applications to the analysis and synthesis of combinatorial digital circuits.

Definition 5.2.2. Given a Boolean algebra $\mathbb{B}$ and function $f : \mathbb{B}^n \to \mathbb{B}^m$, the *$i$-th Boolean derivative of f at* $(u_1, \ldots, u_n) \in \mathbb{B}^n$ is the value

$$\frac{\partial[f]}{\partial[x_i]}(u_1, \ldots, u_n) \coloneqq f(u_1, \ldots, u_n) \not\leftrightarrow f(u_1, \ldots, \neg u_i, \ldots, u_n),$$

writing $u \not\leftrightarrow v \coloneqq (u \wedge \neg v) \vee (\neg u \wedge v)$ for exclusive-or.

[5]One can think of the functor α as a "constructive" version of the functor G from Lemma 3.1.4, making the choice of a group structure explicit rather than applying the Axiom of Choice.

Now $\mathbb{B}^n$ is clearly a commutative group, with the group operation given by pointwise exclusive disjunction, and hence it is an object in $\mathbf{Grp}_\star$. Set

$$\top_i := (\bot, \overset{i-1}{\ldots}, \bot, \top, \bot, \overset{n-i}{\ldots}, \bot) \in \mathbb{B}^n$$

Lemma 5.2.3. The Boolean derivative of $f : \mathbb{B}^n \to \mathbb{B}^m$ coincides with its derivative *qua* morphism in $\mathbf{Grp}_\star$: $\frac{\partial[f]}{\partial[x_i]}(u_1, \ldots, u_n) =$
$d[f]((u_1, \ldots, u_n), \top_i)$.

Proof. Note first that in any Boolean algebra $\mathbb{B}$, we have $\neg u = u \nleftrightarrow \top$. Moreover

$$(u_1, \ldots, \neg u_i, \ldots, u_n) = (u_1, \ldots, u_n) \oplus (\bot, \ldots, \top, \ldots, \bot)$$

Furthermore:

$$\begin{array}{rl} & f(u_1, \ldots, u_n) \oplus \dfrac{\partial[f]}{\partial[x_i]}(u_1, \ldots, u_n) \\ = & f(u_1, \ldots, u_n) \oplus (f(u_1, \ldots, u_n) \nleftrightarrow f(u_1, \ldots, \neg u_i, \ldots, u_n)) \\ = & f(u_1, \ldots, u_n) \nleftrightarrow (f(u_1, \ldots, u_n) \nleftrightarrow f(u_1, \ldots, \neg u_i, \ldots, u_n)) \\ = & (f(u_1, \ldots, u_n) \nleftrightarrow f(u_1, \ldots, u_n)) \nleftrightarrow f(u_1, \ldots, \neg u_i, \ldots, u_n) \\ = & \bot \nleftrightarrow f(u_1, \ldots, \neg u_i, \ldots, u_n) \\ = & f(u_1, \ldots, \neg u_i, \ldots, u_n) \\ = & f((u_1, \ldots, u_n) \nleftrightarrow \top_i) \\ = & f((u_1, \ldots, u_n) \oplus \top_i) \end{array}$$

Thus, since derivatives in $\mathbf{Grp}_\star$ are unique, the Boolean derivative

$$\frac{\partial[f]}{\partial[x_i]}(u_1, \ldots, u_n)$$

is precisely the derivative $\partial[f]((u_1, \ldots, u_n), \top_i)$. □

One surprising fact about $\mathbf{Grp}_\star$ is that it is a Cartesian closed category – and what is more, it has remarkable infinitesimal object: the cyclic group $\mathbb{Z}_2$! Although it does not look very "infinitesimal" in any meaningful geometric sense, it is in fact the smallest (non-trivial) group.

Theorem 5.2.3. The category $\mathbf{Grp}_\star$ is Cartesian closed, with the exponential $\mathcal{F} \Rightarrow \mathcal{G}$ being the group of (arbitrary) functions $F \Rightarrow G$, with the group operation being the pointwise lifting of $\cdot_{\mathcal{G}}$.

For any group $\mathcal{G}$, the group $\mathbb{Z}_2 \Rightarrow \mathcal{G}$ is naturally isomorphic to the tangent bundle $\mathrm{T}(\mathcal{G}) \equiv \mathcal{G} \times \mathcal{G}$.

Proof. First we show that the group $\mathcal{F} \Rightarrow \mathcal{G}$ is indeed the exponential – this follows immediately from the fact that we can freely take the evaluation map $\mathbf{ev} : \mathcal{F} \times \mathcal{F} \Rightarrow \mathcal{G} \to \mathcal{G}$ to be the evaluation map in **Set**.

One might expect the isomorphism $\mathcal{U} : \mathbb{Z}_2 \Rightarrow \mathcal{G} \to \mathcal{G} \times \mathcal{G}$ to be defined by $\mathcal{U}(u) = (u(0), u(1))$. As natural as this definition is, it would not be a natural isomorphism, and so a slightly cleverer mapping is necessary.

We instead define the isomorphism $\mathcal{U}$ by:

$$\mathcal{U}(u) \coloneqq (u(0), u(0)^{-1} \cdot u(1))$$

To show that this is indeed natural, consider an arbitrary function $f : \mathcal{G} \to \mathcal{F}$. Then:

$$\begin{aligned}
(\mathrm{T}(f) \circ \mathcal{U}_{\mathcal{G}})(u) &= \mathrm{T}(f)(u(0), u(0)^{-1} \cdot u(1)) \\
&= (f(u(0)), \partial[f](u(0), u(0)^{-1} \cdot u(1))) \\
&= (f(u(0)), f(u(0))^{-1} \cdot f(u(1))) \\
&= \mathcal{U}_{\mathcal{G}}(f \circ u) \\
&= (\mathcal{U}_{\mathcal{G}} \circ (\Rightarrow f))(u)
\end{aligned}$$

□

The existence of a concrete infinitesimal object in $\mathbf{Grp}_\star$ lets us illustrate some of the more abstract results in Section 5.1.2. For example, we can verify that indeed the action $\oplus_{\mathcal{G}}$ in any group $\mathcal{G}$ corresponds precisely to evaluation at a certain point $U_\oplus : \mathbf{0} \to \mathbb{Z}_2$ (in this case the point $U_\oplus$ being the number 1). That is to say:

$$a \oplus_{\mathcal{G}} b = (\mathcal{U}(a, b))$$

5.2.3 Polynomials Over Commutative Kleene Algebras

The case of polynomials over a commutative Kleene algebra is yet another setting where a "non-standard" notion of derivative is useful – in particular Hopkins and Kozen [54] employ it to give a proof of Parikh's theorem, which can be informally summarised as saying that every context-free language is "equivalent" to a regular language if one is unconcerned about the order of the characters. Similar sorts of derivatives have been applied to reconstructing regular expressions from automata [69] or accelerating the computation of fixed points for program analysis [40]

We study here the category of polynomials over a (fixed) commutative Kleene algebra, a remarkable setting where a (non-linear) derivative exists whose characterising property is that an analogue of the first-order Taylor approximation holds up to strict equality.

Definition 5.2.3 (Commutative Kleene algebra)**.** Formally a **Kleene algebra** $\mathbb{K}$ is a tuple $(K, +, \cdot, ^\star, 0, 1)$ such that $(K, +, \cdot, 0, 1)$ is an idempotent semiring under $+$ satisfying, for all $a, b, c \in K$:

$$1 + a\,a^\star = a^\star \quad 1 + a^\star a = a^\star \quad b + a\,c \leq c \to a^\star\,b \leq c \quad b + c\,a \leq c \to b\,a^\star \leq c$$

where $a \leq b \coloneqq a + b = b$. A Kleene algebra is **commutative** whenever $\cdot$ is. Recall that, informally, a Kleene algebra is the algebra of regular expressions over some alphabet [21, 29].

In what follows, we fix a commutative Kleene algebra $\mathbb{K}$. Define its **algebra of polynomials** on variables $x_1, \ldots, x_n$, denoted by $\mathbb{K}[\overline{x}]$, as the free extension of the algebra $\mathbb{K}$ with elements $\overline{x} = x_1, \ldots, x_n$. We write $p(\overline{a})$ for the value of $p(\overline{x})$ evaluated at $\overline{x} \mapsto \overline{a}$. Polynomials, viewed as functions, are closed under composition: when $p \in \mathbb{K}[\overline{x}], q_1, \ldots, q_n \in \mathbb{K}[\overline{y}]$ are polynomials, so is the composite $p(q_1(\overline{y}), \ldots, q_n(\overline{y}))$.

Definition 5.2.4. Given a polynomial $p = p(\overline{x})$, we define its ***i*-th derivative** $\frac{\partial p}{\partial x_i}(\overline{x}) \in \mathbb{K}[\overline{x}]$ by induction on the structure of p according to the following rules:

$$\frac{\partial\, a}{\partial x_i}(\overline{x}) = 0 \qquad \frac{\partial\, p^\star}{\partial x_i}(\overline{x}) = p^\star(\overline{x})\,\frac{\partial\, p}{\partial x_i}(\overline{x}) \qquad \frac{\partial\, x_j}{\partial x_i}(\overline{x}) = \begin{cases} 1 \text{ if } i = j \\ 0 \text{ otherwise} \end{cases}$$

$$\frac{\partial\,(p+q)}{\partial x_i}(\overline{x}) = \frac{\partial\, p}{\partial x_i}(\overline{x}) + \frac{\partial\, q}{\partial x_i}(\overline{x}) \qquad \frac{\partial\,(p\,q)}{\partial x_i}(\overline{x}) = p(\overline{x})\frac{\partial\, q}{\partial x_i}(\overline{x}) + q(\overline{x})\frac{\partial\, p}{\partial x_i}(\overline{x})$$

Most of the previous rules should be familiar from calculus, except for the derivative of the Kleene star – informally, this is justified through unfolding its definition and applying the other rules.

$$\begin{aligned}
\frac{\partial\, p^\star}{\partial x}(\overline{x}) &= \frac{\partial\,(1 + p\,p^\star)}{\partial x}(\overline{x}) \\
&= \frac{\partial\, 1}{\partial x}(\overline{x}) + \frac{\partial\,(p\,p^\star)}{\partial x}(\overline{x}) \\
&= p(\overline{x})\frac{\partial\, p^\star}{\partial x}(\overline{x}) + p^\star(\overline{x})\frac{\partial\, p}{\partial x}(\overline{x}) \\
&= \frac{\partial\, p}{\partial x}(\overline{x}) + p(\overline{x})\left(\frac{\partial\, p^\star}{\partial x}(\overline{x}) + p^\star(\overline{x})\frac{\partial\, p}{\partial x}(\overline{x})\right)
\end{aligned}$$

Solving the above recurrence relation one then obtains $\frac{\partial p^\star}{\partial x}(\overline{x}) = p^\star(\overline{x})\frac{\partial p}{\partial x}(\overline{x})$.

Theorem 5.2.4 (Taylor's formula [54])**.** Let $p(x) \in \mathbb{K}[x]$. For all $a, b \in \mathbb{K}[x]$, we have $p(a + b) = p(a) + b \cdot \frac{\partial p}{\partial x}(a + b)$.

Definition 5.2.5. Fix a commutative Kleene algebra $\mathbb{K}$. Its **category of polynomials**, $\mathbb{K}_\times$ has all the natural numbers as objects. The morphisms $\mathbb{K}_\times[m, n]$ are n-tuples of polynomials $(p_1, \ldots, p_n)$ where $p_1, \ldots, p_n \in \mathbb{K}[x_1, \ldots, x_m]$. Composition of morphisms is the usual composition of polynomials.

Lemma 5.2.4. The category $\mathbb{K}_\times$ is a Cartesian category, endowed with a change action model $\alpha : \mathbb{K}_\times \to \mathrm{CAct}(\mathbb{K}_\times)$ whereby $\alpha(\mathbb{K}) := (\mathbb{K}, \mathbb{K}, +, +, 0)$, $\alpha(\mathbb{K}^i) := \alpha(\mathbb{K})^i$; for $\overline{p} = (p_1(\overline{x}), \ldots, p_n(\overline{x})) : \mathbb{K}^m \to \mathbb{K}^n$, $\alpha(\overline{p}) := (\overline{p}, (p'_1, \ldots, p'_n))$, where $p'_i = p'_i(x_1, \ldots, x_m, y_1, \ldots, y_m) := \sum_{j=1}^{n} y_j \cdot \frac{\partial p_i}{\partial x_j}(x_1 + y_1, \ldots, x_m + y_m)$.

Proof. We consider the essential case of $m = n = 1$; the proof of the lemma is then a straightforward generalisation.

We shall make use of the following properties of *commutative* Kleene algebras.

1. $(a_1 + \cdots + a_m)^n = \sum\{a_1^{i_1} \cdots a_m^{i_m} \mid i_1 + \cdots + i_m = n; i_1, \cdots, i_m \geq 0\}$.

   Since $(a + b)^\star = a^\star\, b^\star$, we have

   $$\big((a_1 + \cdots + a_m)^n\big)^\star = \prod\{(a_1^{i_1} \cdots a_m^{i_m})^\star \mid i_1 + \cdots + i_m = n; i_1, \cdots, i_m \geq 0\}.$$

   For example $((a + b + c)^2)^\star = (a\,a)^\star\,(a\,b)^\star\,(a\,c)^\star\,(b\,b)^\star\,(b\,c)^\star\,(c\,c)^\star$.

2. Pilling's Normal Form Theorem [85, 54]: every (regular) expression is equivalent to a sum $y_1 + \cdots + y_n$ where each y_i is a product of atomic symbols and expressions of the form $(a_1 \cdots a_k)^\star$, where the a_i are atomic symbols. For example $(((a\,b)^\star c)^\star + d)^\star = d^\star + (a\,b)^\star c^\star c\, d^\star$.

Take $p(x) \in \mathbb{K}[x]$, viewed as a function from change action $(\mathbb{K}, \mathbb{K}, +, +, 0)$ to itself. For $a, b \in \mathbb{K}$, we have

$$\partial[p](a, b) := \frac{\partial p}{\partial x}(a + b) \cdot b.$$

That this satisfies the derivative condition with respect to $p(x)$ is an immediate consequence of Theorem 5.2.4.

We need to prove that the derivative is regular. Trivially $\partial[p](a, 0) = 0$. It remains to prove: for $u, a, b \in \mathbb{K}$

$$\frac{\partial p}{\partial x}(u + a + b) \cdot (a + b) = \frac{\partial p}{\partial x}(u + a) \cdot a + \frac{\partial p}{\partial x}(u + a + b) \cdot b \qquad (5.1)$$

which we argue by structural induction, presenting the cases of $p = q^\star$ and $p = q\,r$ explicitly.

Let $p = q^\star$. Thanks to Pilling's Normal Form Theorem, without loss of generality we may assume $q = x^{n+1}\, c$. Now $\dfrac{\partial\, x^{n+1}\, c}{\partial x}(x) = x^n\, c$. Then $\dfrac{\partial\, p}{\partial x}(x) = q^\star(x)\, \dfrac{\partial\, q}{\partial x}(x) = (x^{n+1}\, c)^\star (x^n\, c)$. Clearly RHS(5.1) $\leq$ LHS(5.1). For the opposite containment, it suffices to show

$$\frac{\partial\, p}{\partial x}(u + a + b) \cdot a \;\leq\; \frac{\partial\, p}{\partial x}(u + a) \cdot a + \frac{\partial\, p}{\partial x}(u + a + b) \cdot b$$

I.e.

$$(\theta^{n+1}\, c)^\star\, (\theta^n\, c)\, a \;\leq\; ((u+a)^{n+1}\, c)^\star\, ((u+a)^n\, c)\, a + (\theta^{n+1}\, c)^\star\, (\theta^n\, c)\, b \qquad (5.2)$$

using the shorthand $\theta = u + a + b$.

A typical element that matches the left-hand side of Equation 5.2 has shape

$$\Xi := (u^{i'}\, a^{j'}\, b^{k'}\, c)^l\, (u^i\, a^j\, b^k\, c)\, a$$

satisfying

$$l \geq 0, \quad i' + j' + k' = n + 1, \quad i + j + k = n.$$

It suffices to consider two cases: $l = 0$ and $l = 1$, for if $l > 1$ and $(u^{i'}\, a^{j'}\, b^{k'}\, c)\, (u^i\, a^j\, b^k\, c)\, a$ matches the right-hand side of Equation 5.2 then so does $(u^{i'}\, a^{j'}\, b^{k'}\, c)^l\, (u^i\, a^j\, b^k\, c)\, a$.

- Now suppose $l = 0$. If $k = 0$ then Ξ matches the first summand of the right-hand side of Equation 5.2; otherwise note that $\Xi = (u^i\, a^{j+1}\, b^{k-1}\, c)\, b$ matches the second summand of the right-hand side of Equation 5.2.

- Next suppose $l = 1$. If $k = k' = 0$ then Ξ matches the first summand of the right-hand side of Equation 5.2; otherwise suppose $k' > 0$ then $\Xi = (u^{i'}\, a^{j'+1}\, b^{k'-1}\, c)\, (u^i\, a^j\, b^k\, c)\, b$ matches the second summand of the right-hand side of Equation 5.2.

Let $p(x) = q(x)\, r(x)$. Applying the product rule of derivatives, Equation 5.1 is equivalent to LHS = RHS where

$$\begin{array}{rcl}
\text{LHS} & := & \left[r(\theta)\, \dfrac{\partial\, q}{\partial x}(\theta) + q(\theta)\, \dfrac{\partial\, r}{\partial x}(\theta) \right] \cdot (a + b) \\
\text{RHS} & := & \left[r(u+a)\, \dfrac{\partial\, q}{\partial x}(u+a) + q(u+a)\, \dfrac{\partial\, r}{\partial x}(u+a) \right] \cdot a \\
& & +\; \left[r(\theta)\, \dfrac{\partial\, q}{\partial x}(\theta) + q(\theta)\, \dfrac{\partial\, r}{\partial x}(\theta) \right] \cdot b
\end{array}$$

using the shorthand $\theta = u + a + b$ as before. Similar to the preceding case, clearly RHS $\leq$ LHS. To show LHS $\leq$ RHS, it suffices to show:

$$\begin{aligned} r(\theta)\,\frac{\partial\, q}{\partial x}(\theta)\, a &\leq \text{RHS} \\ q(\theta)\,\frac{\partial\, r}{\partial x}(\theta)\, a &\leq \text{RHS} \end{aligned}$$

We consider the first; the same reasoning applies to the second. As before, thanks to Pilling's Normal Form Theorem, we may assume that $r(x) = (x^{m+1}\, c)^\star$ and $q(x) = (x^{n+1}\, d)^\star$. Then $r(\theta)\,\dfrac{\partial\, q}{\partial x}(\theta)\, a = (\theta^{m+1}\, c)^\star\, (\theta^{n+1}\, d)^\star\, (\theta^n\, d)\, a$. By considering a typical element Ξ that matches the preceding expression, and using the same reasoning as the preceding case, we can then show that Ξ matches R, as desired. $\square$

In this setting, again, derivatives fail to be additive in their second argument. Take $p(x) = x^2$, for example: as $\frac{\partial p}{\partial x} = x + x = x$, we have:

$$\begin{aligned} \partial[p](a, b + c) &= (b + c)(a + b + c) \\ &= b(a + b + c) + c(a + b + c) \\ &= b(a + b) + c(a + c) + bc \\ &= \partial[p](a, b) + \partial[p](a, c) + bc \\ &\neq \partial[p](a, b) + \partial[p](a, c) \end{aligned}$$

It follows that $\mathbb{K}[\overline{x}]$ cannot be modelled by a Cartesian differential category, generalised or otherwise, as axiom **[CDC.2]** requires that every derivative be additive.

Chapter 6

From differentials to differences

As we saw in Section 2.3, one of the most notable and general settings for generalised differentiation in the current literature is that of Cartesian differential categories [17]. Introduced by Blute, Cockett and Seely, these provide an abstract categorical axiomatisation of the directional derivative from differential calculus. The relevance of Cartesian differential categories lies in their ability to model both "classical" differential calculus (with the canonical example being the category of Euclidean spaces and smooth functions between) and the differential λ-calculus (as every categorical model for it gives rise to a Cartesian differential category [71]).

However, while Cartesian differential categories have proven to be an immensely successful formalism, they have, by design, some limitations. Firstly, they cannot account for the "exotic" notions of derivative which change actions can successfully represent, such as the calculus of finite differences (Section 5.2.2) or the derivatives of programs involved in incremental computation (Chapter 4).

This is because the axioms of a Cartesian differential category stipulate that derivatives should be linear in their second argument (in the same way that the directional derivative is), whereas these aforementioned discrete sorts of derivative need not be. Additionally, every Cartesian differential category is equipped with a tangent bundle monad [25, 74] whose Kleisli category can be intuitively understood as a category of generalised vector fields. This Kleisli category comes equipped with a natural choice of differentiation operator, inherited from the underlying category, which comes close to making it a Cartesian differential category, but again fails the requirement of being linear in its second argument.

On the other hand, change action models impose no such restrictions on the derivatives, but they can be excessively general, in that they do not guarantee many useful properties that are commonly satisfied – for example, commutativity of addition, which holds in important cases such as the calculus of finite differences.

In Section 6.1 we formulate the notion of *Cartesian difference categories*, a class of categories that bridge the gap between the generality of change action models and Cartesian differential categories. Some motivating examples are given in Section 6.2. Section 6.3 focuses on the tangent bundle functor in Cartesian difference categories, which has many properties in common with the differential case. Importantly, we show that the tangent bundle functor is a monad, and that the induced Kleisli category is itself a Cartesian difference category, something which is *not* the case of differential categories. Finally, in Section 6.4, we generalize the notion of differential λ-categories to our setting, which will later provide the basis for interpreting a higher-order calculus.

Many of the results in this chapter are the result of a collaboration with Jean-Simon Lemay and will be appearing in print in [4]. Some passages and figures have been reproduced verbatim with permission.

6.1 Cartesian Difference Categories

In this section we introduce *Cartesian difference categories*, which are generalisations of Cartesian differential categories equipped with an operator that turns a map into an "infinitesimal" version of it, in the sense that every map coincides with its Taylor approximation on infinitesimal elements.

What is meant by this? Consider a smooth function $f : \mathbb{R} \to \mathbb{R}$. Taylor's theorem states that its value can be approximated at a neighbourhood of $a \in \mathbb{R}$ by the expression:

$$f(a + b) \approx f(a) + b \cdot f'(a)$$

More precisely, given any $b \in \mathbb{R}$, the value of $f(a + \delta \cdot d)$ tends to $f(a) + \delta \cdot d \cdot f'(a)$ as δ approaches zero. One might then say that, when δ is an infinitesimally small magnitude, the above is a strict equality. Infinitesimal extensions should then be understood as an abstraction of the idea of "multiplying by an infinitesimal δ".

Definition 6.1.1. A Cartesian left-additive category $\mathbf{C}$ (Definition 2.3.1) is said to have an **infinitesimal extension** ε if every hom-set $\mathbf{C}(A, B)$ comes equipped with a monoid morphism $\varepsilon : \mathbf{C}(A, B) \to \mathbf{C}(A, B)$. That is, ε must satisfy the following conditions:

$$\varepsilon(f + g) = \varepsilon(f) + \varepsilon(g)$$
$$\varepsilon(0) = 0$$

Furthermore, we require that ε be compatible with the Cartesian structure, in the sense that the following equations hold:

$$\begin{aligned}\varepsilon(\pi_1) &= \pi_1 \circ \varepsilon(\mathbf{id})\\ \varepsilon(\pi_2) &= \pi_2 \circ \varepsilon(\mathbf{id})\\ \varepsilon(\langle f, g\rangle) &= \langle \varepsilon(f), \varepsilon(g)\rangle\end{aligned}$$

Lemma 6.1.1. In any Cartesian left-additive category with infinitesimal extension ε, the infinitesimal extension of a map $f : A \to B$ is equivalent to post-composition with $\varepsilon_B \coloneqq \varepsilon(\mathbf{id}_B)$. That is to say:

$$\varepsilon(f) = \varepsilon(\mathbf{id}_B) \circ f$$

Lemma 6.1.2. Whenever $f : A \to B$ is additive (as in Definition 2.3.1), then so is $\varepsilon(f)$. As an immediate consequence, ε_A is additive.

As a consequence of the above two results, we could have defined an infinitesimal extension to be a choice of a monoid homomorphism $\varepsilon_A : A \to A$ for every object A of our category, and used this to formulate the infinitesimal extension of a map $f : A \to B$ to be $\varepsilon(f) \coloneqq \varepsilon_B \circ f$. Similarly, additive categories can be defined by postulating the existence of an (internal) addition map $+_A : A \times A \to A$ for every object A (in our formulation this map is given by $\pi_1 + \pi_2$) and defined $f + g$ by $+_B \circ \langle f, g\rangle$. Both the "internal" and "external" approaches are equivalent and we have chosen the second one for notational convenience and consistency with Cartesian differential categories.

Lemma 6.1.3. Let $\mathbf{C}$ be a Cartesian left-additive category with infinitesimal extension ε. For every object A, define the maps $\oplus_A : A \times A \to A$, $+_A : A \times A \to A$, and $0_A : \mathbf{1} \to A$ respectively as follows:

$$\begin{aligned}\oplus_A &= \pi_1 + \varepsilon(\pi_2)\\ +_A &= \pi_1 + \pi_2\\ 0_A &= 0\end{aligned}$$

Then $(A, A, \oplus_A, +_A, 0_A)$ is a change action internal to $\mathbf{C}$.

Proof. By [17, Proposition 1.2.2], $(A, +_A, 0_A)$ is a commutative monoid was shown On the other hand, that $\oplus_A$ is an action follows immediately from the fact that ε is a monoid homomorphism. □

Setting $\overline{A} \equiv (A, A, \oplus_A, +_A, 0_A)$, we note that $f \oplus_{\overline{A}} g = f + \varepsilon(g)$ and $f +_{\overline{A}} g = f + g$, and so in particular $+_{\overline{A}} = +$. Therefore, from now on we will omit the subscripts and simply write $\oplus$ and $+$.

Any Cartesian left-additive category $\mathbf{C}$ can always be endowed with two different infinitesimal extensions, that correspond respectively to stating that the only infinitesimal element is zero, and that every element is infinitesimal.

Lemma 6.1.4. For any Cartesian left additive category $\mathbf{C}$,

1. Setting $\varepsilon(f) = 0$ defines an infinitesimal extension on $\mathbf{C}$ and therefore in this case, $\oplus_A = \pi_1$ and $f \oplus g = f$.

2. Setting $\varepsilon(f) = f$ defines an infinitesimal extension on $\mathbf{C}$ and therefore in this case, $\oplus_A = +_A$ and $f \oplus g = f + g$.

While these examples of infinitesimal extensions may seem trivial, they are both very significant as they will give rise to key examples of Cartesian difference categories.

Definition 6.1.2. A **Cartesian difference category** is a Cartesian left additive category with an infinitesimal extension ε which is equipped with a **difference combinator** ∂ of the form:

$$\frac{f : A \to B}{\partial[f] : A \times A \to B}$$

verifying the following coherence conditions (for clarity, we have highlighted the terms which differ from the corresponding Cartesian differential axioms):

[C∂C.0] $f \circ (x + \varepsilon(u)) = f \circ x + \varepsilon\left(\partial[f] \circ \langle x, u\rangle\right)$

[C∂C.1] $\partial[f + g] = \partial[f] + \partial[g]$, $\partial[0] = 0$, and $\partial[\varepsilon(f)] = \varepsilon(\partial[f])$

[C∂C.2] $\partial[f] \circ \langle x, u + v\rangle = \partial[f] \circ \langle x, u\rangle + \partial[f] \circ \langle x + \varepsilon(u), v\rangle$ and $\partial[f] \circ \langle x, 0\rangle = 0$

[C∂C.3] $\partial[\mathbf{id}_A] = \pi_2$ and $\partial[\pi_1] = \pi_2; \pi_1$ and $\partial[\pi_2] = \pi_2; \pi_1$

[C∂C.4] $\partial[\langle f, g\rangle] = \langle \partial[f], \partial[g]\rangle$ and $\partial[!_A] = !_{A \times A}$

[C∂C.5] $\partial[g \circ f] = \partial[g] \circ \langle f \circ \pi_1, \partial[f]\rangle$

[C∂C.6] $\partial^2[f] \circ \langle\langle x, u\rangle, \langle 0, v\rangle\rangle = \partial[f] \circ \langle x + \varepsilon(u), v\rangle$

[C∂C.7] $\partial^2[f] \circ \langle\langle x, u\rangle, \langle v, 0\rangle\rangle = \partial^2[f] \circ \langle\langle x, v\rangle, \langle u, 0\rangle\rangle$

Before giving some intuition on the axioms **[C∂C.0]** to **[C∂C.7]**, we first remark that one could have used the language of change actions to express **[C∂C.0]**, **[C∂C.2]** and **[C∂C.6]** which could then be written as:

[C∂C.0] $f \circ (x \oplus u) = (f \circ x) \oplus (\partial[f] \circ \langle x, u \rangle)$

[C∂C.2] $\partial[f] \circ \langle x, u + v \rangle = \partial[f] \circ \langle x, u \rangle + \partial[f] \circ \langle x \oplus u, v \rangle$ and $\partial[f] \circ \langle x, 0 \rangle = 0$

[C∂C.6] $\partial\left[\partial[f]\right] \circ \langle \langle x, u \rangle, \langle 0, v \rangle \rangle = \partial[f] \circ \langle x \oplus u, v \rangle$

And also, just like Cartesian differential categories, **[C∂C.6]** and **[C∂C.7]** have alternative equivalent expressions.

Lemma 6.1.5. In the presence of the other axioms, **[C∂C.6]** and **[C∂C.7]** are equivalent to:

[C∂C.6(a)] $\partial\left[\partial[f]\right] \circ \langle \langle x, 0 \rangle, \langle 0, y \rangle \rangle = \partial[f] \circ \langle x, y \rangle$

[C∂C.7(a)] $\partial\left[\partial[f]\right] \circ \langle \langle x, y \rangle, \langle z, w \rangle \rangle = \partial\left[\partial[f]\right] \circ \langle \langle x, z \rangle, \langle y, w \rangle \rangle$

Proof. The proof is essentially the same as [25, Proposition 4.2]. □

The keen eyed reader will notice that the axioms of a Cartesian difference category are very similar to the axioms of a Cartesian differential category. Indeed, **[C∂C.1]**, **[C∂C.3]**, **[C∂C.4]**, **[C∂C.5]**, and **[C∂C.7]** are identical to their counterparts in a Cartesian differential category. Both definitions, however, differ in axioms **[C∂C.2]** and **[C∂C.6]**, where the infinitesimal extension ε now appears, and of course the inclusion of the additional axiom **[C∂C.0]**.

On the other hand, interestingly enough, **[C∂C.6(a)]** is the same as **[CDC.6(a)]** We also point out that, writing out **[C∂C.0]** and **[C∂C.2]** using change action notion, we see that these axioms are precisely an equational version of the conditions in Definition 3.2.2.

In element-like notation, **[C∂C.0]** is written as:

$$f(x + \varepsilon(u)) = f(x) + \varepsilon\left(\partial[f](x, u)\right)$$

This condition can be read as a generalisation of the Kock-Lawvere axiom that characterises the derivative in from synthetic differential geometry [59]. Broadly speaking, the Kock-Lawvere axiom states that, for any map $f : \mathcal{R} \to \mathcal{R}$ and any $x \in \mathcal{R}$ and $d \in \mathcal{D}$, there exists a unique $f'(x) \in \mathcal{R}$ verifying

$$f(x + d) = f(x) + d \cdot f'(x)$$

where $\mathcal{D}$ is the subset of $\mathcal{R}$ consisting of infinitesimal elements. It is by analogy with the Kock-Lawvere axiom that we refer to ε as an "infinitesimal extension" as it can be thought of as embedding the space A into a subspace $\varepsilon(A)$ of infinitesimal elements.

[**C∂C.2**] is the first of the shared axioms that differs between a Cartesian difference category and a Cartesian differential category. In a Cartesian differential category, the differential of a map is assumed to be additive in its second argument. In a Cartesian difference category, just as derivatives for change actions, while the differential is still required to preserve zeros in its second argument, it is only additive "up to a small perturbation", that is:

$$\partial[f](x, u+v) = \partial[f](x, u) + \partial[f](x + \varepsilon(u), v)$$

By taking [**C∂C.0**] into consideration, the above can equivalently be written in a more symmetric form as:

$$\partial[f](x, u+v) = \partial[f](x, u) + \partial[f](x, v) + \varepsilon(\partial[\partial[f]](x, v, u, 0))$$

[**C∂C.6**] and [**C∂C.7**] tell us how to work with second order differentials. [**C∂C.6**] is expressed as follows:

$$\partial\left[\partial[f]\right](x, y, 0, z) = \partial[f](x + \varepsilon(y), z)$$

and has some remarkable and counter-intuitive consequences.

Lemma 6.1.6. Given any map $f : A \to B$ in a Cartesian difference category $\mathbf{C}$, its derivatives satisfy the following equations for any $x, u, v, w : C \to A$:

i. $\partial[f] \circ \langle x, \varepsilon(u)\rangle = \varepsilon(\partial[f]) \circ \langle x, u\rangle$

ii. $\partial[f] \circ \langle x, u+v\rangle = \partial[f] \circ \langle x, u\rangle + \partial[f] \circ \langle x + \varepsilon^2(u), v\rangle$

iii. $\varepsilon(\partial^2[f]) \circ \langle\langle x, u\rangle, \langle v, 0\rangle\rangle = \varepsilon^2(\partial^2[f]) \circ \langle\langle x, u\rangle, \langle v, 0\rangle\rangle$

Proof.

i. $$\partial[f] \circ \langle g, \varepsilon(h)\rangle = \partial[f] \circ \langle g, 0\rangle + \varepsilon(\partial^2[f]) \circ \langle\langle g, 0\rangle, \langle 0, h\rangle\rangle = \varepsilon(\partial[f]) \circ \langle g, h\rangle$$

ii. $$\begin{aligned}
&\partial[f] \circ \langle x, u+v\rangle \\
&= \partial[f] \circ \langle x, u\rangle + \partial[f] \circ \langle x + \varepsilon(u), v\rangle && \text{(by } [\mathbf{C\partial C.2}]) \\
&= \partial[f] \circ \langle x, u\rangle + \partial^2[f] \circ \langle\langle x, u\rangle, \langle 0, v\rangle\rangle && \text{(by } [\mathbf{C\partial C.6}]) \\
&= \partial[f] \circ \langle x, u\rangle + \partial^2[f] \circ \langle\langle x, 0\rangle, \langle u, v\rangle\rangle && \text{(by } [\mathbf{C\partial C.7}]) \\
&= \partial[f] \circ \langle x, u\rangle + \partial^2[f] \circ \langle\langle x, 0\rangle, \langle u, 0\rangle\rangle + \partial^2[f] \circ \langle\langle x, \varepsilon(u)\rangle, \langle 0, v\rangle\rangle && \text{(by } [\mathbf{C\partial C.2}]) \\
&= \partial[f] \circ \langle x, u\rangle + \partial^2[f] \circ \langle\langle x, u\rangle, \langle 0, 0\rangle\rangle + \partial^2[f] \circ \langle\langle x, \varepsilon(u)\rangle, \langle 0, v\rangle\rangle && \text{(by } [\mathbf{C\partial C.7}]) \\
&= \partial[f] \circ \langle x, u\rangle + \partial[f] \circ \left\langle x + \varepsilon^2(u), v\right\rangle && \text{(by } [\mathbf{C\partial C.2}] \text{ and } [\mathbf{C\partial C.6}])
\end{aligned}$$

iii.
$$\begin{aligned}
&\varepsilon(\partial^2[f]) \circ \langle\langle x, u\rangle, \langle v, 0\rangle\rangle \\
&= \partial^2[f] \circ \langle\langle x, \varepsilon(u)\rangle, \langle v, 0\rangle\rangle && \text{(by } \mathbf{[C\partial C.7]} \text{ and 6.1.6.i)} \\
&= \partial^3[f] \circ \langle\langle\langle x, 0\rangle, \langle 0, u\rangle\rangle, \langle\langle 0, 0\rangle, \langle v, 0\rangle\rangle\rangle && \text{(by } \mathbf{[C\partial C.6]}\text{)} \\
&= \partial^3[f] \circ \langle\langle\langle x, 0\rangle, \langle 0, 0\rangle\rangle, \langle\langle 0, u\rangle, \langle v, 0\rangle\rangle\rangle && \text{(by } \mathbf{[C\partial C.7]}\text{)} \\
&= \partial^3[f] \circ \langle\langle\langle x, 0\rangle, \langle 0, 0\rangle\rangle, \langle\langle 0, u\rangle, \langle 0, 0\rangle\rangle\rangle \\
&\qquad + \partial^3[f] \circ \big\langle\big\langle\big\langle x, \varepsilon^2(u)\big\rangle, \langle 0, 0\rangle\big\rangle, \langle\langle 0, 0\rangle, \langle v, 0\rangle\rangle\big\rangle && \text{(by 6.1.6.ii)} \\
&= \partial^3[f] \circ \big\langle\big\langle\big\langle x, \varepsilon^2(u)\big\rangle, \langle 0, 0\rangle\big\rangle, \langle\langle 0, 0\rangle, \langle v, 0\rangle\rangle\big\rangle && \text{(by } \mathbf{[C\partial C.7]} \text{ and } \mathbf{[C\partial C.2]}\text{)} \\
&= \partial^2[f] \circ \big\langle\big\langle x, \varepsilon^2(u)\big\rangle, \langle v, 0\rangle\big\rangle && \text{(by } \mathbf{[C\partial C.6]}\text{)} \\
&= \varepsilon^2(\partial^2[f]) \circ \langle\langle x, u\rangle, \langle v, 0\rangle\rangle && \text{(by 6.1.6.i)}
\end{aligned}$$

□

In particular, the above result lets us work as if the infinitesimal extension was idempotent – but only when it acts on a second derivative carrying a zero as its fourth argument! Indeed, as Section 6.2.3 shows, even though every difference category satisfies this "infinitesimal cancellation" property, the infinitesimal extension ε may not be idempotent. These facts will turn out to be crucial once we set out to define a term calculus (which we shall do in Chapter 7). However, one of their consequences is of immediate interest.

Lemma 6.1.7. Let $\mathbf{C}$ be a Cartesian difference category with a nilpotent infinitesimal extension. That is to say, there is some $k \in \mathbb{N}$ such that $\varepsilon^k(f) = 0$ for any f. Then every derivative $\partial[f]$ is additive in its second argument.

Proof. Fix a map $f : A \to B$, and consider arbitrary $x, u, v : C \to A$. Since $\partial[f]$ is regular, it follows that it is additive "up to a second-order term":

$$\begin{aligned}
&\partial[f](x, u + v) \\
&= \partial[f](x, u) + \partial[f](x + \varepsilon(u), v) && \text{(by } \mathbf{[C\partial C.2]}\text{)} \\
&= \partial[f](x, u) + \partial[f](x, v) + \varepsilon\partial^2[f](x, v, u, 0) && \text{(by } \mathbf{[C\partial C.0]}\text{)} \\
&= \partial[f](x, u) + \partial[f](x, v) + \varepsilon^k\partial^2[f](x, v, u, 0) && \text{(iterating 6.1.6.iii)} \\
&= \partial[f](x, u) + \partial[f](x, v) && \text{(since } \varepsilon \text{ is nilpotent)}
\end{aligned}$$

□

6.1.1 Differentials as Trivial Differences

From a cursory look at the axioms of a Cartesian difference category, one might expect that the notion is neither stronger nor weaker than that of a Cartesian differential category, since neither set of axioms seems to subsume the other.

As we established in Section 5.2, however, we know that every Cartesian differential category gives rise to a change action model. While we have not yet established the connection between change action models and difference categories, it is clear that the two concepts are closely related. It should come as no surprise, then, that Cartesian differential categories are, in fact, a special case of difference categories.

Theorem 6.1.1. Every Cartesian differential category $\mathbf{C}$ with differential combinator D is a Cartesian difference category with an infinitesimal extension defined by $\varepsilon(f) = 0$ and a difference combinator defined to be identical to the differential combinator, that is to say, $\partial[f] = \mathsf{D}[f]$.

Proof. Clearly, the first two parts of the [**C∂C.1**], the second part of [**C∂C.2**], [**C∂C.3**], [**C∂C.4**], [**C∂C.5**], and [**C∂C.7**] are precisely the same as their differential counterparts.

On the other hand, since $\varepsilon(f) = 0$, we have that [**C∂C.0**] and the third part of [**C∂C.1**] both hold trivially, since they merely state that $0 = 0$. The first part of [**C∂C.2**] and [**C∂C.6**] are exactly equivalent to the first part of [**CDC.2**] and [**CDC.6**] respectively. Therefore, the differential combinator satisfies the Cartesian difference axioms. □

Conversely, one can always build a Cartesian differential category from a Cartesian difference category by considering the objects for which the infinitesimal extension is the zero map.

Proposition 6.1.1. For a Cartesian difference category $\mathbf{C}$ with infinitesimal extension ε and difference combinator ∂, define the category $\mathbf{C}_0$ as the full subcategory of objects A such that $\varepsilon(\mathbf{id}_A) = 0$. Then $\mathbf{C}_0$ is a Cartesian differential category with a differential combinator given by $\mathsf{D}[f] = \partial[f]$.

Proof. First note that if $\varepsilon(\mathbf{id}_A) = 0$ and $\varepsilon(\mathbf{id}_B) = 0$, then by definition it also follows that $\varepsilon(\mathbf{id}_{A\times B}) = 0$. For the terminal object we have $\varepsilon(\mathbf{id}_{\mathbf{1}}) = !_{\mathbf{1}} = 0$ since maps into $\mathbf{1}$ are necessarily unique. Thus $\mathbf{C}_0$ is closed under finite products and is therefore a Cartesian left-additive category.

Furthermore, for any map f in $\mathbf{C}_0$, we have $\varepsilon(f) = 0$. This implies that, for all such maps, the Cartesian difference axioms are precisely the Cartesian differential axioms. Therefore, the difference combinator is a differential combinator for this subcategory, and so $\mathbf{C}_0$ is a Cartesian differential category. □

In any Cartesian difference category $\mathbf{C}$, the terminal object $\mathbf{1}$ always satisfies that $\varepsilon(\mathbf{id_1}) = 0$, thus $\mathbf{C}_0$ is never empty. On the other hand, applying Proposition 6.1.1 to a Cartesian differential category results in the entire category.

The above result, however, does not imply that if a difference combinator is a differential combinator then infinitesimal extension must be zero. In Section 6.2.3, we provide such an example of a Cartesian differential category which comes equipped with a non-zero infinitesimal extension such that the differential combinator is a difference combinator with respect to this non-zero infinitesimal extension.

6.1.2 Change Action Models as Difference Categories

In this section we show how every Cartesian difference category is a particularly well-behaved change action model, and conversely how every change action model contains a Cartesian difference category.

Proposition 6.1.2. Let $\mathbf{C}$ be a Cartesian difference category with infinitesimal extension ε and difference combinator ∂. Define the functor $\alpha : \mathbf{C} \to \mathrm{CAct}(\mathbf{C})$ as $\alpha(A) = (A, A, \oplus_A, +_A, 0_A)$ (as defined in Lemma 6.1.3) and $\alpha(f) = (f, \partial[f])$. Then $(\mathbf{C}, \alpha : \mathbf{C} \to \mathrm{CAct}(\mathbf{C}))$ is a change action model.

Proof. By Lemma 6.1.3, $(A, A, \oplus_A, +_A, 0_A)$ is a change action and so α is well-defined on objects. Given a map f, $\partial[f]$ is a derivative of f in the change action sense since **[C∂C.0]** and **[C∂C.2]** are precisely a restatement of Definition 3.2.2, and so α is well-defined on maps. That α preserves identities and composition follows from **[C∂C.3]** and **[C∂C.5]** respectively, and so α is a functor. That α preserves finite products will follow from **[C∂C.3]** and **[C∂C.4]**. Finally, it is evident that α is a section of the forgetful functor, and therefore the pair $(\mathbf{C}, \alpha)$ is a change action model. □

It is clear that not every change action model is a Cartesian difference category, since for example, change action models do not require addition to be commutative. On the other hand, it can be shown that every change action model contains a Cartesian difference category as a full subcategory, formed of well-behaved objects.

Definition 6.1.3. Let $(\mathbf{C}, \alpha : \mathbf{C} \to \mathrm{CAct}(\mathbf{C}))$ be a change action model. An object A of $\mathbf{C}$ is **flat** whenever the following hold:

[F.1] $\Delta A = A$

[F.2] $\alpha(\oplus_A) = (\oplus_A, \oplus_A \circ \pi_2)$

[F.3] $0 \oplus_A (0 \oplus_A f) = 0 \oplus_A f$ for any $f : U \to A$.

[F.4] $\oplus_A$ is right-injective, that is, if $\oplus_A \circ \langle f, g\rangle = \oplus_A \circ \langle f, h\rangle$ then $g = h$.

We define the category $\mathbf{Flat}_\alpha$ as the full subcategory of $\mathbf{C}$ which contains precisely all the flat objects of $\mathbf{C}$.

Example 6.1.1. In the category $\mathbf{Grp}_\star$ of groups and set-theoretic functions, every object satisfies conditions **[F.1]**, **[F.3]** and **[F.4]**. Additionally, $\mathcal{F}$ satisfies **[F.2]** if and only if it is Abelian. Indeed, in $\mathbf{Grp}_\star$:

$$\begin{aligned}\partial[\oplus_A](x, u, v, w) &= -(x \oplus_A u) + ((x \oplus_A) v \oplus_A (u \oplus_A w))\\ &= -(x + u) + (x + v + u + w)\end{aligned}$$

which is equal to $v + w$ if and only if $+$ is commutative.

Since α preserves finite products, it is straightforward to see that $\mathbf{1}$ is flat and if A and B are flat then so is $A \times B$. The sum of maps $f : A \to B$ and $g : A \to B$ in $\mathbf{Flat}_\alpha$ is defined using addition on the change action αB, that is, we define $f + g \coloneqq +_B \circ \langle f, g\rangle$, while the zero map $0 : A \to B$ is likewise defined as $0 \coloneqq 0_B \circ !_A$.

Lemma 6.1.8. $\mathbf{Flat}_\alpha$ is a Cartesian left additive category.

Proof. Most of the Cartesian left-additive structure is immediate. However, since addition is not required to be commutative for arbitrary change actions, we will prove that, in any flat object, addition must necessarily be commutative. Indeed, it suffices to compute. Using set-theoretic notation, we obtain the following identity, for any $\mathbf{Flat}_\alpha$-maps $f, g : A \to B$:

$$\begin{aligned}0 \oplus (f + g) &= (0 \oplus f) \oplus g && \text{(by the action property of } \oplus)\\ &= (0 \oplus g) \oplus \partial[\oplus]((0, g), (f, 0)) && \text{(by the derivative condition)}\\ &= (0 \oplus g) \oplus (f \oplus 0) && \text{(by } \mathbf{[F.2]})\\ &= 0 \oplus (g + f) && \text{(by the action property)}\end{aligned}$$

By **[F.4]**, we can cancel the 0 on both sides to obtain $f + g = g + f$. □

We use the action of the change action structure to define the infinitesimal extension. So for a map $f : A \to B$ in $\mathbf{Flat}_\alpha$, define $\varepsilon(f) : A \to B$ by:

$$\varepsilon(f) \coloneqq \oplus_B \circ \langle 0_B \circ !_A, f\rangle$$

or, using more convenient infix notation, $\varepsilon(f) \coloneqq 0_B \oplus_B f$.

Lemma 6.1.9. ε is an infinitesimal extension for $\mathbf{Flat}_\alpha$.

Proof. We have to show that ε preserves the addition. Following the same idea as in the proof of Lemma 6.1.8, we pick arbitrary $f, g : A \to B$ and obtain the following:

$$\begin{aligned}
0 \oplus \varepsilon(f+g) &= 0 \oplus (0 \oplus (f+g)) && \text{(by definition of } \varepsilon\text{)}\\
&= 0 \oplus (0 \oplus f \oplus g) && \text{(by the action property)}\\
&= \big(0 \oplus (0 \oplus f)\big) \oplus (0 \oplus g) && \text{(by the derivative condition, } \textbf{[F.2]}\text{)}\\
&= \big(0 \oplus (0 \oplus f + 0 \oplus g)\big) && \text{(by the action property)}\\
&= 0 \oplus (\varepsilon(f) + \varepsilon(g)) && \text{(by definition of } \varepsilon\text{)}
\end{aligned}$$

Then, by **[F.4]**, it follows that $\varepsilon(f+g) = \varepsilon(f) + \varepsilon(g)$. The remaining infinitesimal extension axioms can be proven in a similar fashion. □

Lastly, the difference combinator for $\mathbf{Flat}_\alpha$ is defined in the obvious way, that is to say, $\partial[f]$ is simply the second component of $\alpha(f)$.

Theorem 6.1.2. Let $(\mathbf{C}, \alpha : \mathbf{C} \to \mathrm{CAct}(\mathbf{C}))$ be a change action model. Then $\mathbf{Flat}_\alpha$ is a Cartesian difference category.

Proof. [$\mathbf{C\partial C.0}$] and [$\mathbf{C\partial C.2}$] are simply a restatement of the derivative condition. [$\mathbf{C\partial C.3}$] and [$\mathbf{C\partial C.4}$] follow immediately from the fact that α preserves finite products and from the structure of products in $\mathrm{CAct}(\mathbf{C})$ (as per Theorem 3.2.4), while [$\mathbf{C\partial C.5}$] follows from the definition of composition in $\mathrm{CAct}(\mathbf{C})$ (as per Definition 3.2.4).

We now prove [$\mathbf{C\partial C.1}$]. First, by definition of $+$ and applying the chain rule, we have:

$$\partial[f+g] = \partial[+ \circ \langle f, g\rangle] = \partial[+_B] \circ \langle \langle f, g\rangle \circ \pi_1, \langle \partial[f], \partial[g]\rangle\rangle$$

It suffices to show that $\partial[+] = + \circ \pi_2$, from which [$\mathbf{C\partial C.1}$] will follow. Consider arbitrary maps $u, w : A \to B$. Then (again using set-like notation) we have:

$$0 \oplus \big(u \oplus w)\big) = \big(0 \oplus u\big) \oplus (0 \oplus w) = 0 \oplus \big(u + (0 \oplus w)\big)$$

By **[F.4]** we obtain the identity below:

$$u \oplus w = u + \varepsilon(w) \tag{6.1}$$

From this, it follows that, for any $u, v, w, l : A \to B$, the maps $(u \oplus w) + (v \oplus l)$ and $u + v + \varepsilon(w) + \varepsilon(l)$ are identical. But, since ε is an infinitesimal extension, the second map can also be written as $(u+v) + \varepsilon(w+l)$, which is precisely $(u+v) \oplus (w+l)$. So we have

$$(u+v) \oplus (w+l) = (u \oplus w) + (v \oplus l) = (u+v) \oplus \partial[+](u,v,w,l))$$

Applying **[F.4]** again gives $\partial[+] = + \circ \pi_2$ as desired. Axiom **[C∂C.7]** can be established in a similar way and we omit the calculations, proceeding directly to axiom **[C∂C.6(a)]** – which, per Lemma 6.1.5, is equivalent to, and easier to prove than **[C∂C.6]**. As before, we pick arbitrary $x, u : A \to B$ and calculate:

$$\begin{aligned}
&f(x) \oplus \partial^2[f](x,0,0,u) \\
&= f(x) + (0 \oplus \partial^2[f](x,0,0,u)) && \text{(by 6.1)} \\
&= f(x) + (0 \oplus (0 \oplus \partial^2[f](x,0,0,u))) && \text{(by }\textbf{[F.3]}\text{)} \\
&= f(x) + (0 \oplus (\partial[f](x,0) \oplus \partial^2[f](x,0,0,u))) && \text{(by regularity)} \\
&= f(x) + (0 \oplus \partial[f](x, 0 \oplus u)) && \text{(by the derivative condition)} \\
&= f(x) \oplus \partial[f](x, 0 \oplus u) && \text{(by 6.1)} \\
&= f(x \oplus (0 \oplus u)) && \text{(by the derivative condition)} \\
&= f(x + 0 \oplus (0 \oplus u)) && \text{(by 6.1)} \\
&= f(x + 0 \oplus u) && \text{(by }\textbf{[F.3]}\text{)} \\
&= f(x \oplus u) && \text{(by 6.1)} \\
&= f(x) \oplus \partial[f](x,u) && \text{(by the derivative condition)}
\end{aligned}$$

Hence $\partial^2[f](x,0,0,u) = \partial[f](x,u)$ as desired. □

Note that the proof above goes through without appealing to **[F.3]**, except for axiom **[C∂C.6]**, for which only a weaker version is provable. More precisely, without **[F.3]**, one can only show

$$\varepsilon(\partial^2[f](x,0,0,u)) = \partial[f](x, \varepsilon(u))$$

which is strictly weaker than **[C∂C.6]** and, in fact, holds trivially whenever $\varepsilon = 0$.

6.1.3 Linear and ε-Linear Maps

An important subclass of maps in a Cartesian differential category are the so-called *linear maps* [17, Definition 2.2.1]. These generalise the usual notion from linear algebra: a map in a Cartesian differential category is said to be linear whenever it coincides with its derivative. Notably, linear maps are *not* defined as those maps which are additive – additivity of linear maps follows, in fact, as a theorem.

The same definition of linear maps goes through in a Cartesian difference category **C**. Using set-like notation, a map f is linear whenever $\partial[f](x,y) = f(y)$. Linear maps in a Cartesian difference category satisfy many of the same properties found in [17, Lemma 2.2.2] (in particular, they form a Cartesian subcategory of **C**).

Definition 6.1.4. In a Cartesian difference category, a map f is said to be **linear** whenever it satisfies the identity $\partial[f] = f \circ \pi_2$[1].

Lemma 6.1.10. In any Cartesian difference category, the following properties hold.

i. If $f : A \to B$ is linear then $\varepsilon(f) = f \circ \varepsilon(\mathbf{id}_A)$.

ii. If $f : A \to B$ is linear, then f is additive (as per Definition 2.3.1).

iii. Identity maps, projection maps, and zero maps are linear.

iv. The composite, sum, and pairing of linear maps is linear.

v. If $f : A \to B$ and $k : C \to D$ are linear, then for any map $g : B \to C$, the following equality holds:

$$\partial[k \circ g \circ f] = k \circ \partial[g] \circ (f \times f)$$

vi. If an isomorphism is linear, then its inverse is linear.

vii. For any object A, $\oplus_A$ and $+_A$ are linear.

viii. Whenever ε is nilpotent, then every derivative evaluated at zero is linear. That is, every map $\partial[f] \circ \langle 0, \mathbf{id} \rangle$ is linear.

Proof. Most of the above results are either immediate or admit a similar proof to the ones in [17, Lemma2.2.2]. We will prove 6.1.10.i and 6.1.10.viii, as they differ from the differential setting.

For the first, we note that $f \circ 0 = f \circ \pi_2 \circ \langle 0, 0 \rangle = \partial[f] \circ \langle 0, 0 \rangle = 0$, therefore:

$$\begin{aligned}
\varepsilon(f) &= 0 + \varepsilon(f) && \text{(by the additive axioms)}\\
&= (f \circ 0) + (\varepsilon(f \circ \pi_2) \circ \langle 0, \mathbf{id}_A \rangle) && \text{(by the extension axioms)}\\
&= (f \circ 0) + (\varepsilon(\partial[f]) \circ \langle 0, \mathbf{id}_A \rangle) && \text{(by linearity)}\\
&= f \circ (0 + \varepsilon(\mathbf{id}_A)) && \text{(by } \mathbf{[C\partial C.0]}\text{)}\\
&= f \circ \varepsilon(\mathbf{id}_A) && \text{(by the additive axioms)}
\end{aligned}$$

For the second we need to show that $\partial[\partial[f] \circ \langle 0, \mathbf{id} \rangle] = \partial[f] \circ \langle 0, \mathbf{id} \rangle \circ \pi_2$. The left-hand side expands to:

$$\partial[\partial[f] \circ \langle 0, \mathbf{id} \rangle] = \partial^2[f] \circ \big((\langle 0, \mathbf{id} \rangle) \times (\langle 0, \mathbf{id} \rangle)\big)$$

[1]Compare and contrast the notion of *self-maintainable derivatives*[23, Section 4.3], those which do not depend on the base point but only the change, i.e. $\partial[f](x, \delta) = g(\delta)$ for some g. The notion of a linear map is strictly stronger, as we require the map g to coincide with the function itself.

Fixing arbitrary u, w of the appropriate types and using set-like notation, we obtain:

$$\begin{aligned}
\partial^2[f](0,u,0,w) &= \partial[f](\varepsilon(u),w) && \text{(by } [\mathbf{C\partial C.6}]\text{)}\\
&= \partial[f](0,w) + \varepsilon\partial^2[f](0,w,u,0) && \text{(by } [\mathbf{C\partial C.0}]\text{)}\\
&= \partial[f](0,w) + \varepsilon^k\partial^2[f](0,w,u,0) && \text{(by 6.1.6.iii)}\\
&= \partial[f](0,w) && \text{(since } \varepsilon \text{ is nilpotent)}
\end{aligned}$$

□

While all linear maps are additive, the converse is not necessarily true (see [17, Corollary 2.3.4]). However, an immediate consequence of the above lemma is that every Cartesian difference category has a Cartesian subcategory of linear maps.

Another interesting subclass of maps in a Cartesian difference category are the ε-linear maps, which are maps whose infinitesimal extension is linear.

Definition 6.1.5. In a Cartesian difference category, a map f is **ε-linear** if $\varepsilon(f)$ is linear.

Lemma 6.1.11. In a Cartesian difference category,

i. If $f : A \to B$ is ε-linear then $f \circ (x + \varepsilon(y)) = f \circ x + \varepsilon(f) \circ y$.

ii. Every linear map is ε-linear.

iii. The composite, sum, and pairing of ε-linear maps is ε-linear.

iv. If an isomorphism is ε-linear, then its inverse is again ε-linear.

Using element-like notation, the first point of the above lemma says that if f is ε-linear then $f(x + \varepsilon(y)) = f(x) + \varepsilon(f(y))$. This sheds some light on ε-linear maps: these are those which are additive on "infinitesimal" elements (i.e. elements of the form $\varepsilon(y)$).

For a Cartesian differential category considered as a difference category, linear maps in the Cartesian difference category sense are precisely the same as Cartesian differential category sense [17, Definition 2.2.1], while every map is trivially ε-linear since $\varepsilon = 0$.

6.2 Examples of Cartesian Difference Categories

6.2.1 Euclidean Spaces and Smooth Functions

Every Cartesian differential category is a Cartesian difference category where the infinitesimal extension is zero. As a particular example, we consider the category of real smooth functions, which as mentioned above, can be considered to be the canonical (and motivating) example of a Cartesian differential category.

Let $\mathbb{R}$ be the set of real numbers and let **SMOOTH** be the category whose objects are Euclidean spaces $\mathbb{R}^n$ (we take $\mathbb{R}^0$ to be the single-point space $\{*\}$) and whose maps are all the smooth functions $F : \mathbb{R}^n \to \mathbb{R}^m$. **SMOOTH** is a Cartesian left additive category where the product structure is given by the standard Cartesian product of Euclidean spaces and where the additive structure is defined by point-wise addition, $(F + G)(\mathbf{x}) = F(\mathbf{x}) + G(\mathbf{x})$ and $0(\mathbf{x}) = (0, \ldots, 0)$, where $\mathbf{x} \in \mathbb{R}^n$. **SMOOTH** is a Cartesian differential category where the differential combinator is defined by the directional derivative of smooth functions. Explicitly, for a smooth function $F : \mathbb{R}^n \to \mathbb{R}^m$, which is in fact a tuple of smooth functions $F = (f_1, \ldots, f_n)$ where $f_i : \mathbb{R}^n \to \mathbb{R}$, $\mathsf{D}[F] : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}^m$ is defined as follows:

$$\mathsf{D}[F](\mathbf{x}, \mathbf{y}) := \left(\sum_{i=1}^{n} \frac{\partial f_1}{\partial u_i}(\mathbf{x}) y_i, \ldots, \sum_{i=1}^{n} \frac{\partial f_n}{\partial u_i}(\mathbf{x}) y_i \right)$$

where $\mathbf{x} = (x_1, \ldots, x_n), \mathbf{y} = (y_1, \ldots, y_n) \in \mathbb{R}^n$. Alternatively, $\mathsf{D}[F]$ can also be defined in terms of the Jacobian matrix of F. Therefore **SMOOTH** is a Cartesian difference category with infinitesimal extension $\varepsilon = 0$ and with difference combinator D. Since $\varepsilon = 0$, the induced action is simply $\mathbf{x} \oplus_{\mathbb{R}^n} \mathbf{y} = \mathbf{x}$. Also a smooth function is linear in the Cartesian difference category sense precisely if it is $\mathbb{R}$-linear in the classical sense, and every smooth function is ε-linear.

6.2.2 Calculus of Finite Differences

In Section 5.2.2 we discussed the category $\mathbf{Grp}_\star$ of groups and arbitrary functions, and we showed that it is a change action model. Clearly $\mathbf{Grp}_\star$ fails to be a Cartesian difference category, as the induced additive structure need not be commutative. The subcategory of abelian groups, however, is indeed a Cartesian difference category.

Let $\mathbf{Ab}_\star$ be the full subcategory of $\mathbf{Grp}_\star$ whose objects are abelian groups $(G, +, 0)$. As in $\mathbf{Grp}_\star$, a map $f : (F, +, 0) \to (G, +, 0)$ is simply an arbitrary function $f : \mathcal{F} \to \mathcal{G}$, which does not necessarily preserve the group structure. Clearly $\mathbf{Ab}_\star$ is Cartesian left-additive[2], and we could explicitly prove that it has an infinitesimal extension and a difference operator. Instead, we will apply Theorem 6.1.2 to show directly that $\mathbf{Ab}_\star$ is Cartesian differential.

Proposition 6.2.1. The full subcategory $\mathbf{Ab}_\star$ of $\mathbf{Grp}_\star$ consisting of all the abelian groups is precisely the category $\mathbf{Flat}_{\mathbf{Grp}_\star}$ of flat objects in $\mathbf{Grp}_\star$.

Proof. Axioms [**F.1**], [**F.3**], [**F.4**] hold trivially, as they are true of any object in $\mathbf{Grp}_\star$. It remains only to prove [**F.1**]. And since functions in $\mathbf{Ab}_\star$ admit exactly one derivative, it suffices to show that $+ \circ \pi_2$ is a derivative for $+$. But this is an immediate consequence of commutativity of $+$, since:

$$(a \oplus u) \oplus (b \oplus v) = (a + u) + (b + v) = (a + b) + (u + v) = (a \oplus b) \oplus (u \oplus v) \qquad \square$$

The induced infinitesimal extension is simply the identity, and the corresponding difference combinator ∂ in $\mathbf{Ab}_\star$ is given, as in $\mathbf{Grp}_\star$, by:

$$\partial[f](x, u) \coloneqq f(x + u) - f(x)$$

On the other hand, ∂ is not a differential combinator for $\mathbf{Ab}_\star$, since it satisfies neither [**CDC.6**] nor [**CDC.2**]. Thanks to the addition of the infinitesimal extension, ∂ does satisfy [**C∂C.2**] and [**C∂C.6**], as well as [**C∂C.0**]. However, as noted in [20], it is interesting to note that this ∂ does satisfy [**CDC.1**], the second part of [**CDC.2**], [**CDC.3**], [**CDC.4**], [**CDC.5**], [**CDC.7**], and [**CDC.6(a)**]. It is worth noting that in [20], the goal was to drop the addition and develop a "non-additive" version of Cartesian differential categories. Cartesian difference categories can be seen as a stronger form of the "non-linear Cartesian differential categories" presented therein, which simply drop the requirement that derivatives be additive altogether.

As $\mathbf{Ab}_\star$ is the paradigmatic example of a Cartesian difference category (indeed, the axioms of difference categories were engineered precisely to fit the case of finite differences in abelian groups), it is worth using it to build some intuition on the less intuitive quirks of difference categories.

For example, in $\mathbf{Ab}_\star$, axiom [**C∂C.6**] consists of folding a telescoping expression:

$$\begin{aligned}
\partial^2[f](x, u, 0, v) &= \partial[f](x, u + v) - \partial[f](x, u) \\
&= \Big(f(x + u + v) - f(x)\Big) - \Big(f(x + u) - f(x)\Big) \\
&= f(x + u + v) - f(x + u) \\
&= \partial[f](x + u, v)
\end{aligned}$$

[2]It would be more appropriate to say that $\mathbf{Ab}_\star$ can be endowed with the structure of a Cartesian left-additive category, since the choice of a group operation for every object is somewhat arbitrary.

The curious "duplication of infinitesimals" from Lemma 6.1.6.iii holds trivially, since $\varepsilon = \mathbf{id}$ is idempotent.

The linear maps in $\mathbf{Ab}_\star$ (in the sense of Definition 6.1.4) are those which satisfy $\partial[f](x, y) = f(y)$ – that is to say, $f(x + y) - f(x) = f(y)$. So the linear maps are precisely the group homomorphisms!

6.2.3 Module Morphisms

Here we provide a simple example of a Cartesian difference category whose difference combinator is also a differential combinator, but whose infinitesimal extension is neither zero nor the identity.

Let R be a commutative semiring and let $\mathbf{MOD}_R$ be the category of R-modules and R-linear maps between them. $\mathbf{MOD}_R$ has finite biproducts and is therefore a Cartesian left additive category where every map is additive. Every $r \in R$ induces an infinitesimal extension ε^r defined by scalar multiplication, $\varepsilon^r(f)(m) = rf(m)$. Then $\mathbf{MOD}_R$ is a Cartesian difference category with the infinitesimal extension ε^r for any $r \in R$ and difference combinator ∂ defined as:

$$\partial[f](m, n) = f(n)$$

R-linearity of f assures that $[\mathbf{C\partial C.0}]$ holds, while the remaining Cartesian difference axioms hold trivially. In fact, ∂ is also a differential combinator and therefore $\mathbf{MOD}_R$ is also a Cartesian differential category. The induced action is given by $m \oplus_M n = m + rn$. By definition of ∂, every map in $\mathbf{MOD}_R$ is linear and, by 6.1.11.ii, also ε-linear.

6.2.4 Stream Calculus

Here we show how one can extend the calculus of finite differences example to sets of infinite sequences. This is a particularly interesting model, as the methods and language of differential calculus have been often used to describe functions on streams - see [89] or [90] for an introduction to this so-called stream calculus. A particularly important class of stream functions are the so-called *causal* functions, those where the n-th element of the output may only depend on the first n elements of the input. These are known to arise from certain classes of "stream differential equations" [63] and have recently been studied in connection to machine learning and backpropagation [101, 100].

Definition 6.2.1. For a set A, let A^ω denote the set of infinite sequences of elements of A, where we write $[a_i]$ for the infinite sequence $[a_i] = (a_1, a_2, a_3, \ldots)$ and $a_{i:j}$ for the (finite) subsequence $(a_i, a_{i+1}, \ldots, a_j)$. A function $f : A^\omega \to B^\omega$ is **causal** whenever the n-th element $f\left([a_i]\right)_n$ of the output sequence only depends on the first n elements of $[a_i]$, that is, f is causal if and only if whenever $a_{0:n} = b_{0:n}$ then $f\left([a_i]\right)_{0:n} = f\left([b_i]\right)_{0:n}$.

We now consider streams over abelian groups, so let $\mathbf{Ab}_\star{}^\omega$ be the category whose objects are all the abelian groups and where the morphisms between $\mathcal{G}$ and $\mathcal{G}$ are the causal maps from G^ω to H^ω. Every object in $\mathbf{Ab}_\star{}^\omega$ is itself an abelian group (internal to $\mathbf{Ab}_\star{}^\omega$), with addition being defined pointwise, that is to say, $[a_i] + [b_i] = [a_i + b_i]$.

Hence, $\mathbf{Ab}_\star{}^\omega$ is a Cartesian left-additive category, where the product is given by the standard product of abelian groups and where the additive structure is similarly lifted point-wise from the structure of $\mathbf{Ab}_\star$: $(f+g)\left([a_i]\right)_n = f\left([a_i]\right)_n + g\left([a_i]\right)_n$ and $0\left([a_i]\right)_n = 0$. In order to define the infinitesimal extension, we first need to define the truncation operator $\mathbf{z}$.

Definition 6.2.2. Given an abelian group $\mathcal{G}$, we define the *truncation operator* $\mathbf{z}_\mathcal{G} : G^\omega \to G^\omega$ by

$$(\mathbf{z}\,[a_i])_0 = 0$$
$$(\mathbf{z}\,[a_i])_{j+1} = a_{j+1}$$

Lemma 6.2.1. $\mathbf{z}_\mathcal{G}$ is a causal monoid homomorphism according to the pointwise monoid structure on G^ω. Thus all the $\mathbf{z}_\mathcal{G}$ define an infinitesimal extension[3] $\mathbf{z}$ for $\mathbf{Ab}_\star{}^\omega$.

The corresponding action updates every element after the first by adding it to the corresponding element in the applied change, while leaving the first element untouched. Formally:

$$([a_i] \oplus [b_i])_0 = a_0 \qquad\qquad ([a_i] \oplus [b_i])_{n+1} = a_{n+1} + b_{n+1}$$

Thus defined, the category $\mathbf{Ab}_\star{}^\omega$ is a Cartesian difference category whose the infinitesimal extension is given by the truncation operator, $\varepsilon(f)\left([a_i]\right) = \mathbf{z}f\left([a_i]\right)$, and where the difference combinator ∂ is defined as follows:

$$\partial[f]\left([a_i], [b_i]\right)_0 = f\left([a_i] + [b_i]\right)_0 - f\left([a_i]\right)_0$$
$$\partial[f]\left([a_i], [b_i]\right)_{n+1} = f\left([a_i] + \mathbf{z}\,[b_i]\right)_{n+1} - f\left([a_i]\right)_{n+1}$$

[3]It may seem odd that we should choose this truncation operator as our infinitesimal extension, rather than the more natural right shift operator which maps an input stream $(a_1, a_2, a_3, \ldots)$ to $(0, a_1, a_2, a_3, \ldots)$. But one can show that, with this choice of an infinitesimal extension, the derivative of a causal map may itself not be causal, hence the category of causal functions would not admit higher derivatives.

Lemma 6.2.2. For any arbitrary function $f : G^\omega \to H^\omega$, the map $\partial[f]$ is causal.

Proof. Follows immediately from the definition of $\partial[f]$ – but we remark that $\partial[f]$ won't satisfy the derivative condition unless f itself is causal. $\square$

Note the similarities between the difference combinator on $\mathbf{Ab}_\star$ and that on ${\mathbf{Ab}_\star}^\omega$. One might suggest that a simpler difference combinator could be defined by letting $\partial[f]\left([a_i], [b_i]\right)_n = (f([a_i] + \mathbf{z}\,[b_i]) - f([a_i])_n)$. This, however, does not quite work, as it fails to satisfy axiom **[C∂C.3]**. Indeed, the derivative of the identity computes to $\partial[\mathbf{id}]\left([a_i], [b_i]\right) = \mathbf{z}\,[b_i]$, rather than $[b_i]$.

Lemma 6.2.3. The category ${\mathbf{Ab}_\star}^\omega$ is a Cartesian difference category.

Proof. **[C∂C.1-7]** are trivial. **[C∂C.0]** follows immediately from the fact that every map in ${\mathbf{Ab}_\star}^\omega$ is causal (and hence any update to the input which does not change the first element gives rise to an update to the output which does not change the first element). $\square$

A causal map is linear (in the Cartesian difference category sense) if and only if it is a group homomorphism. On the other hand, a causal map f is ε-linear if and only if it is a group homomorphism in every component except the first. $\mathbf{z}f([a_i] + [b_i]) = \mathbf{z}f([a_i]) + \mathbf{z}f([b_i])$.

6.3 Tangent Bundles in Cartesian Difference Categories

As we showed in Section 5.1.1, every change action model is equipped with an endofunctor T that behaves in some ways like well-known *tangent bundle monad* in Cartesian differential categories. [25, 74]. However, change action models do not have enough structure to show that this functor is a monad.

In a Cartesian difference category, however, this issue disappears and we recover the full monad structure. What is more, the Kleisli category of this monad is again a Cartesian difference category! This is an important result, as it is not true in the differential setting: the Kleisli category of the tangent monad in a Cartesian differential category is *not*, itself, Cartesian differential, but it *will* be a Cartesian difference category.

6.3.1 The Tangent Bundle Monad

Let $\mathbf{C}$ be a Cartesian difference category with infinitesimal extension ε and difference combinator ∂. Remember that, in Definition 5.1.2, we defined the functor $\mathrm{T} : \mathbf{C} \to \mathbf{C}$ as follows:

$$\begin{aligned}\mathrm{T}A &\coloneqq A \times \Delta A\\ \mathrm{T}f &\coloneqq \langle f \circ \pi_1, \partial f\rangle\end{aligned}$$

and define the natural transformations $\eta : \mathrm{Id}_{\mathbf{C}} \Rightarrow \mathrm{T}$ and $\mu : \mathrm{T}^2 \Rightarrow \mathrm{T}$ by:

$$\begin{aligned}\eta_A &\coloneqq \langle \mathbf{id}_A, 0\rangle\\ \mu_A &\coloneqq \langle \pi_{11}, \pi_{21} + \pi_{12} + \varepsilon(\pi_{22})\rangle\end{aligned}$$

Theorem 6.3.1. (T, μ, η) defines a monad.

Proof. Let $f : A \to B$ be an arbitrary map. Naturality of η and μ as well as the first of the monad laws ($\mu \circ \eta_{\mathrm{T}} = \mu \circ \mathrm{T}(\eta)$) were established in Theorem 5.1.2.

For the last of the monad laws, we first note that, since μ is linear, it follows that $\mathrm{T}(\mu) = \mu \times \mu$. Then it suffices to compute:

$$\begin{aligned}\mu_A \circ \mathrm{T}(\mu_A) &= \mu_A \circ (\mu_A \times \mu_A)\\ &= \langle \pi_{11}, \pi_{21} + \pi_{12} + \varepsilon(\pi_{22})\rangle \circ (\mu_A \times \mu_A)\\ &= \langle \pi_1 \circ \mu_A \circ \pi_1, \pi_2 \circ \mu_A \circ \pi_1 + \pi_1 \circ \mu_A \circ \pi_2 + \varepsilon(\pi_2 \circ \mu_A \circ \pi_2)\rangle\\ &= \langle \pi_{111}, \pi_{211} + \pi_{121} + \varepsilon(\pi_{221}) + \pi_{112} + \varepsilon\left(\pi_{212} + \pi_{122} + \varepsilon(\pi_{222})\right)\rangle\\ &= \langle \pi_1 \circ \pi_{11}, \pi_2 \circ \pi_{11} + \pi_1 \circ (\pi_{21} + \pi_{12} + \varepsilon(\pi_{22})) + \varepsilon\left(\pi_2 \circ (\pi_{21} + \pi_{12} + \varepsilon(\pi_{22}))\right)\rangle\\ &= \langle \pi_{11}, \pi_{21} + \pi_{12} + \varepsilon(\pi_{22})\rangle \circ \langle \pi_{11}, \pi_{21} + \pi_{12} + \varepsilon(\pi_{22})\rangle\\ &= \mu_A \circ \mu_{\mathrm{T}(A)}\end{aligned}$$

□

When $\mathbf{C}$ is a Cartesian differential category with the difference structure arising from setting $\varepsilon = 0$, this tangent bundle monad coincides with the standard tangent monad corresponding to its tangent category structure [25, 74].

Example 6.3.1. For a Cartesian differential category, since $\varepsilon = 0$, the induced monad is precisely the monad induced by its tangent category structure [25, 74]. For example, in the Cartesian differential category SMOOTH (as defined in Section 6.2.1), one has that $\mathrm{T}(F)(\mathbf{x}, \mathbf{y}) = (F(\mathbf{x}), \mathsf{D}[F](\mathbf{x}, \mathbf{y}))$, $\eta_{\mathbb{R}^n}(\mathbf{x}) = (\mathbf{x}, \mathbf{0})$, and $\mu_{\mathbb{R}^n}((\mathbf{x}, \mathbf{y}), (\mathbf{z}, \mathbf{w})) = (\mathbf{x}, \mathbf{y} + \mathbf{z})$.

Example 6.3.2. In the Cartesian difference category $\overline{\mathsf{Ab}}$ (as defined in Section 6.2.2), the monad is given by $\mathrm{T}(f)(x, y) = (f(x), f(x + y) - f(x))$, $\eta_G(x) = (x, 0)$, and $\mu_G((x, y), (z, w)) = (x, y + z + w)$.

Example 6.3.3. In the Cartesian difference category MOD_R (as defined in Section 6.2.3) with infinitesimal extension ε^r, for $r \in R$, $\mathrm{T}(f)(m,n) = (f(m), f(n))$, $\eta_M(m) = (m, 0)$, and $\mu_M((m,n),(p,q)) = (m, n + p + rq)$.

Proposition 6.3.1. The family of maps $\oplus_A : \mathrm{T}(A) \to A$, as defined in Lemma 6.1.3, constitute a natural transformation $\oplus : \mathrm{T} \to \mathrm{Id}$. Furthermore, every pair $(A, \oplus_A)$ is an algebra for the tangent bundle monad.

Proposition 6.3.2. Every map $f : A \to B$ is a homomorphism of T-algebras from $(A, \oplus_A)$ into $(B, \oplus_B)$

6.3.2 The Kleisli Category of T

Recall that the Kleisli category of the monad (T, μ, η) is defined as the category $\mathbf{C}_{\mathrm{T}}$ whose objects are the objects of $\mathbf{C}$, and where a map $A \to B$ in $\mathbf{C}_{\mathrm{T}}$ is a map $f : A \to \mathrm{T}(B)$ in $\mathbf{C}$, which would be a pair $f = \langle f_0, f_1 \rangle$ where $f_1, f_2 : A \to B$. The identity map in $\mathbf{C}_{\mathrm{T}}$ is the monad unit $\eta_A : A \to \mathrm{T}(A)$, while composition of Kleisli maps $f : A \to \mathrm{T}(B)$ and $g : B \to \mathrm{T}(C)$ is defined as the composite $\mu_C \circ \mathrm{T}(g) \circ f$. To distinguish between composition in $\mathbf{C}$ and $\mathbf{C}_{\mathrm{T}}$, we denote Kleisli composition by $g \circ^{\mathrm{T}} f = \mu_C \circ \mathrm{T}(g) \circ f$. If $f = \langle f_0, f_1 \rangle$ and $g = \langle g_0, g_1 \rangle$, then their Kleisli composition can be explicitly computed out to be:

$$g \circ^{\mathrm{T}} f = \langle g_0, g_1 \rangle \circ^{\mathrm{T}} \langle f_0, f_1 \rangle = \langle g_0 \circ f_0, \partial[g_0] \circ \langle f_0, f_1 \rangle + g_1 \circ (f_0 + \varepsilon(f_1)) \rangle$$

As noted in [25], Kleisli maps can be understood as "generalised" vector fields. Indeed, $\mathrm{T}(A)$ should be thought of as the tangent bundle over A, and therefore the analogue of a vector field would be a map of the form $\langle \mathbf{id}_A, f \rangle : A \to \mathrm{T}(A)$, which is of course a Kleisli map from A to A. For more details on the intuition behind this Kleisli category see [25]. We now turn to the task of showing that the Kleisli category of any Cartesian difference category is again a Cartesian difference category.

We begin by exhibiting the Cartesian left additive structure of the Kleisli category. The product of objects in $\mathbf{C}_{\mathrm{T}}$ is defined as $A \times B$ with projections $\pi_0^{\mathrm{T}} : A \times B \to \mathrm{T}(A)$ and $\pi_1^{\mathrm{T}} : A \times B \to \mathrm{T}(B)$ defined respectively as $\pi_0^{\mathrm{T}} = \langle \pi_1, 0 \rangle$ and $\pi_1^{\mathrm{T}} = \langle \pi_2, 0 \rangle$. The pairing of Kleisli maps $f = \langle f_0, f_1 \rangle$ and $g = \langle g_0, g_1 \rangle$ is defined as $\langle f, g \rangle^{\mathrm{T}} = \langle \langle f_0, g_0 \rangle, \langle f_1, g_1 \rangle \rangle$. The terminal object is again $\mathbf{1}$ and where the unique map to the terminal object is $!_A^{\mathrm{T}} = \langle 0, 0 \rangle$. The sum of Kleisli maps $f = \langle f_0, f_1 \rangle$ and $g = \langle, g_0, g_1 \rangle$ is defined as $f +^{\mathrm{T}} g = f + g = \langle f_0 + g_0, f_1 + g_1 \rangle$, and the zero Kleisli maps is simply $0^{\mathrm{T}} = 0 = \langle 0, 0 \rangle$. Therefore we conclude that the Kleisli category of the tangent monad is a Cartesian left additive category.

Lemma 6.3.1. $\mathbf{C}_{\mathrm{T}}$ is a Cartesian left additive category.

For a Kleisli map $f : \langle f_0, f_1\rangle$, the infinitesimal extension $\varepsilon^{\mathrm{T}}(f)$ is defined as follows:

$$\varepsilon^{\mathrm{T}}(f) = \langle 0, f_0 + \varepsilon(f_1)\rangle$$

Lemma 6.3.2. ε^{T} is an infinitesimal extension on $\mathbf{C}_{\mathrm{T}}$.

It is interesting to point out that for an object A the induced action $\oplus^{\mathrm{T}}_A$ in the Kleisli category can be computed out to be:

$$\oplus^{\mathrm{T}}_A = \pi^{\mathrm{T}}_1 +^{\mathrm{T}} \varepsilon^{\mathrm{T}}(\pi_2) = \langle \pi_1, 0\rangle + \langle 0, \pi_2\rangle = \langle \pi_1, \pi_2\rangle = \mathbf{id}_{\mathrm{T}(A)}$$

which, perhaps surprisingly, turns out to be the identity map on $\mathrm{T}(A)$ in the base category $\mathbf{C}$ (but not the Kleisli identity). In fact, it was the observation that the identity in $\mathbf{C}$ had the right type to be an action in $\mathbf{C}_{\mathrm{T}}$ that initially led to this result.

To define the difference combinator for the Kleisli category, first note that difference combinators by definition do not change the codomain. That is, if $f : A \to \mathrm{T}(B)$ is a Kleisli arrow, then the type of its derivative *qua* Kleisli arrow should be $A \times A \to \mathrm{T}(B) \times \mathrm{T}(B)$, which coincides with the type of its derivative in $\mathbf{C}$. Therefore, the difference combinator ∂^{T} for the Kleisli category can be defined to be the difference combinator of the base category, that is, for a Kleisli map $f = \langle f_0, f_1\rangle$:

$$\partial^{\mathrm{T}}[f] = \partial[f] = \langle \partial[f_0], \partial[f_1]\rangle$$

Theorem 6.3.2. For a Cartesian difference category $\mathbf{C}$, the Kleisli category $\mathbf{C}_{\mathrm{T}}$ is a Cartesian difference category with infinitesimal extension ε^{T} and difference combinator ∂^{T}.

Proof. Let $\phi = \langle\langle \pi_{11}, \pi_{12}\rangle, \langle \pi_{21}, \pi_{22}\rangle\rangle$ be the isomorphism between $\mathrm{T}(A) \times \mathrm{T}(B)$ and $T(A \times B)$. We first note that, for any map f in $\mathbf{C}$, the following equality holds:

$$\mathrm{T}(\partial[f]) = \partial\left[\mathrm{T}(f)\right] \circ \phi$$

Since ϕ is linear, we also have

$$\partial[\phi] = \phi \circ \pi_2$$

This will help simplify many of the calculations to follow, since $\mathrm{T}(\partial[f])$ appears everywhere due to the definition of Kleisli composition.

We will also make use of the following auxiliary results:

Lemma 6.3.3. Whenever $f : A \to B$ is linear in $\mathbf{C}$, then so is $\eta_B \circ f$ in $\mathbf{C}_{\mathrm{T}}$, that is, $\partial^{\mathrm{T}}[\eta_B \circ f] = (\eta_B \circ f) \circ^{\mathrm{T}} \pi^{\mathrm{T}}_2$.

Proof.

$$\begin{aligned}\partial^{\mathrm{T}}[\eta_B \circ f] &= \partial[\langle \mathbf{id}_B, 0\rangle \circ f] \\ &= \langle \pi_2, 0\rangle \circ \langle f \circ \pi_1, f \circ \pi_2\rangle \\ &= \langle f \circ \pi_2, 0\rangle \\ &= \eta \circ f \circ \pi_2 \\ &= \mu \circ \mathrm{T}(\eta_B) \circ \mathrm{T}(f) \circ \eta \circ \pi_2 \\ &= (\eta_B \circ f) \circ^{\mathrm{T}} \pi_2^{\mathrm{T}}\end{aligned}$$

Lemma 6.3.4. Then the following identities hold:

i. $\partial^2[f] \circ \phi = \partial^2[f]$

ii. $\mathrm{T}^2(f) \circ \phi = \phi \circ \mathrm{T}^2(f)$

iii. $\phi \circ \mathrm{T}(\mu) = \mu \circ \mathrm{T}(\phi) \circ \phi$

Proof. The first identity is simply a restatement of **[C∂C.7]**. For the second identity:

$$\begin{aligned}&\mathrm{T}^2(f) \circ \phi \circ \langle\langle x, u\rangle, \langle v, w\rangle\rangle \\ &\quad= \mathrm{T}^2(f) \circ \langle\langle x, v\rangle, \langle u, w\rangle\rangle \\ &\quad= \big\langle\langle f \circ x, \partial[f] \circ \langle x, v\rangle\rangle, \big\langle \partial[f] \circ \langle x, u\rangle, \partial^2[f] \circ \langle\langle x, v\rangle, \langle u, w\rangle\rangle\big\rangle\big\rangle \\ &\quad= \big\langle\langle f \circ x, \partial[f] \circ \langle x, v\rangle\rangle, \big\langle \partial[f] \circ \langle x, u\rangle, \partial^2[f] \circ \langle\langle x, u\rangle, \langle v, w\rangle\rangle\big\rangle\big\rangle \\ &\quad= \phi \circ \big\langle\langle f \circ x, \partial[f] \circ \langle x, u\rangle\rangle, \big\langle \partial[f] \circ \langle x, v\rangle, \partial^2[f] \circ \langle\langle x, u\rangle, \langle v, w\rangle\rangle\big\rangle\big\rangle \\ &\quad= \phi \circ \mathrm{T}^2(f)\end{aligned}$$

Finally, for the last identity:

$$\begin{aligned}&\phi \circ \mathrm{T}(\mu) \circ \big\langle\langle\langle x, u\rangle, \langle v, w\rangle\rangle, \big\langle\langle x', u'\rangle, \langle v', w'\rangle\big\rangle\big\rangle \\ &\quad= \phi \circ \big\langle\langle x, u + v + \varepsilon(w)\rangle, \langle x', u' + v' + \varepsilon(w')\rangle\big\rangle \\ &\quad= \big\langle\langle x, x'\rangle, \langle u + v + \varepsilon(w), u' + v' + \varepsilon(w')\rangle\big\rangle \\ &\quad= \big\langle\langle x, x'\rangle, \langle u, u'\rangle + \langle v, v'\rangle + \varepsilon(\langle w, w'\rangle)\big\rangle \\ &\quad= \mu \circ \big\langle\langle\langle x, x'\rangle, \langle u, u'\rangle\rangle, \langle\langle v, v'\rangle, \langle w, w'\rangle\rangle\big\rangle \\ &\quad= \mu \circ \mathrm{T}(\phi) \circ \big\langle\langle\langle x, u\rangle, \langle x', u'\rangle\rangle, \langle\langle v, w\rangle, \langle v', w'\rangle\rangle\big\rangle \\ &\quad= \mu \circ \mathrm{T}(\phi) \circ \phi \circ \big\langle\langle\langle x, u\rangle, \langle v, w\rangle\rangle, \langle\langle x', u'\rangle, \langle v', w'\rangle\rangle\big\rangle\end{aligned}$$

We can now prove each of the Cartesian difference category axioms.

[C∂C.0] Pick an arbitrary Kleisli map $f = \langle f_1, f_2\rangle : A \to \mathrm{T}(B)$. Then:

$$\begin{aligned}f \circ^{\mathrm{T}} \oplus^{\mathrm{T}} &= \mu \circ \mathrm{T}(f) \circ \mathbf{id}_{\mathrm{T}(A)} \\ &= \langle f_1 \circ \pi_1, f_2 \circ \pi_1 + \partial[f_1] + \varepsilon(\partial[f_2])\rangle\end{aligned}$$

On the other hand:

$$\begin{aligned}
\oplus^{\mathrm{T}} \circ^{\mathrm{T}} \left\langle f \circ^{\mathrm{T}} \pi_1^{\mathrm{T}}, \partial^{\mathrm{T}}[f] \right\rangle^{\mathrm{T}} &= \mu \circ \mathrm{T}(\mathbf{id}) \circ \left\langle f \circ^{\mathrm{T}} \pi_1^{\mathrm{T}}, \partial^{\mathrm{T}}[f] \right\rangle^{\mathrm{T}} \\
&= \mu \circ \left\langle f \circ^{\mathrm{T}} \pi_1^{\mathrm{T}}, \partial^{\mathrm{T}}[f] \right\rangle^{\mathrm{T}} \\
&= \mu \circ \left\langle \mu \circ \mathrm{T}(f) \circ \langle \pi_1, 0 \rangle, \partial^{\mathrm{T}}[f] \right\rangle^{\mathrm{T}} \\
&= \mu \circ \left\langle \langle f_1 \circ \pi_1, f_2 \circ \pi_1 \rangle, \partial^{\mathrm{T}}[f] \right\rangle^{\mathrm{T}} \\
&= \mu \circ \phi \circ \left\langle \langle f_1 \circ \pi_1, f_2 \circ \pi_1 \rangle, \partial^{\mathrm{T}}[f] \right\rangle \\
&= \mu \circ \phi \circ \langle \langle f_1 \circ \pi_1, f_2 \circ \pi_1 \rangle, \langle \partial[f_1], \partial[f_2] \rangle \rangle \\
&= \langle f_1 \circ \pi_1, f_2 \circ \pi_1 + \partial[f_1] + \varepsilon(\partial[f_2]) \rangle
\end{aligned}$$

[C∂C.1] Since $+^{\mathrm{T}} = \eta \circ +$ and $0^{\mathrm{T}} = \eta \circ 0$, and both maps are linear in $\mathbf{C}$, it immediately follows that they are linear in $\mathbf{C}^{\mathrm{T}}$. It remains to show $\partial^{\mathrm{T}}[\varepsilon^{\mathrm{T}}] = \varepsilon^{\mathrm{T}} \circ^{\mathrm{T}} \pi_2^{\mathrm{T}}$

$$\partial^{\mathrm{T}}[\varepsilon^{\mathrm{T}}] = \langle 0, \varepsilon \rangle \circ \pi_2 = \mu \circ \eta \circ \langle 0, \varepsilon \rangle \circ \pi_2 = \mu \circ \mathrm{T}(\langle 0, \varepsilon \rangle) \circ \eta \circ \pi_2 = \varepsilon^{\mathrm{T}} \circ^{\mathrm{T}} \pi_2^{\mathrm{T}}$$

[C∂C.2] We prove the first condition (the second follows by trivial calculation):

$$\begin{aligned}
&\partial^{\mathrm{T}}[f] \circ^{\mathrm{T}} \left\langle x, u +^{\mathrm{T}} v \right\rangle^{\mathrm{T}} \\
&= \mu \circ \mathrm{T}(\partial[f]) \circ \phi \circ \langle x, u + v \rangle \\
&= \mu \circ \left\langle \partial[f] \circ \langle x, u + v \rangle_0, \partial^2[f] \circ \langle x, u + v \rangle \right\rangle \\
&= \mu \circ \left\langle \partial[f] \circ \langle x, u \rangle_0 + \partial[f] \circ \langle x + \varepsilon(u), v \rangle_0, \partial^2[f] \circ \langle x, u \rangle + \partial^2[f] \circ \langle x + \varepsilon(u), v \rangle \right\rangle \\
&= \mu \circ \left\langle \partial[f] \circ \langle x, u \rangle_0 + \partial[f] \circ \langle x + u, v \rangle_0, \partial^2[f] \circ \langle x, u \rangle + \partial^2[f] \circ \langle x + \varepsilon(u), v \rangle \right\rangle \\
&= \mu \circ \Big(\left\langle \partial[f] \circ \langle x, u \rangle_0, \partial^2[f] \circ \langle x, u \rangle \right\rangle \Big) \\
&\qquad + \mu \circ \Big(\left\langle \partial[f] \circ \langle x + \varepsilon(u), u \rangle_0, \partial^2[f] \circ \langle x + \varepsilon(u), v \rangle \right\rangle \Big) \\
&= \partial^{\mathrm{T}}[f] \circ \langle x, u \rangle + \partial^{\mathrm{T}}[f] \circ \langle x + \varepsilon(u), v \rangle
\end{aligned}$$

[C∂C.3] This axiom follows immediately from the fact that $\mathbf{id}_A^{\mathrm{T}} = \eta_A \circ \mathbf{id}_A$ and $\pi_i^{\mathrm{T}} = \eta_{A_i} \circ \pi_i$, with $\mathbf{id}_A, \pi_i$ linear in $\mathbf{C}$.

[C∂C.4] Take arbitrary $f = \langle f_1, f_2 \rangle, g = \langle g_1, g_2 \rangle$. Then:

$$\partial^{\mathrm{T}}[\langle f, g \rangle^{\mathrm{T}}] = \partial[\phi \circ \langle f, g \rangle] = \phi \circ \langle \langle \partial[f_1], \partial[f_2] \rangle, \langle \partial[g_1], \partial[g_2] \rangle \rangle = \left\langle \partial^{\mathrm{T}}[f], \partial^{\mathrm{T}}[g] \right\rangle^{\mathrm{T}}$$

[C∂C.5] It follows from the more general fact that the tangent bundle is functorial in the Kleisli category: define $\mathrm{T}^{\mathrm{T}}(f) = \left\langle f \circ^{\mathrm{T}} \pi_1^{\mathrm{T}}, \partial^{\mathrm{T}}[f] \right\rangle^{\mathrm{T}}$. We show that $\mathrm{T}^{\mathrm{T}}(g \circ^{\mathrm{T}} f) = \mathrm{T}^{\mathrm{T}}(g) \circ \mathrm{T}^{\mathrm{T}}(f)$. First, note that:

$$\begin{aligned}
\mathrm{T}^{\mathrm{T}}(f) &= \left\langle f \circ^{\mathrm{T}} \pi_1^t, \partial^{\mathrm{T}}[f] \right\rangle^{\mathrm{T}} \\
&= \phi \circ \left\langle f \circ^{\mathrm{T}} \pi_1^t, \partial^{\mathrm{T}}[f] \right\rangle \\
&= \phi \circ \langle \mu \circ \mathrm{T}(f) \circ \eta \circ \pi_1, \partial[f] \rangle \\
&= \phi \circ \langle \mu \circ \eta \circ f \circ \pi_1, \partial[f] \rangle \\
&= \phi \circ \langle f \circ \pi_1, \partial[f] \rangle \\
&= \phi \circ \mathrm{T}(f)
\end{aligned}$$

Then:

$$\begin{aligned}
\mathrm{T}^{\mathrm{T}}(g \circ^{\mathrm{T}} f) &= \phi \circ \mathrm{T}(g \circ^{\mathrm{T}} f) \\
&= \phi \circ \mathrm{T}(\mu \circ \mathrm{T}(g) \circ f) \\
&= \phi \circ \mathrm{T}(\mu) \circ \mathrm{T}(\mathrm{T}(g) \circ f) \\
&= \mu \circ \mathrm{T}(\phi) \circ \phi \circ \mathrm{T}(\mathrm{T}(g) \circ f) \\
&= \mu \circ \mathrm{T}(\phi) \circ \mathrm{T}(\mathrm{T}(g)) \circ \phi \circ \mathrm{T}(f) \\
&= \mu \circ \mathrm{T}(\phi \circ \mathrm{T}(g)) \circ \phi \circ \mathrm{T}(f) \\
&= \mathrm{T}^{\mathrm{T}}(g) \circ^{\mathrm{T}} \mathrm{T}^{\mathrm{T}}(f)
\end{aligned}$$

[**C∂C.6**] We prove [**C∂C.6(a)**] instead. For this, the following auxiliary result will be of use:

$$\partial^3[f] \circ (\phi \times \phi) = \partial^3[f]$$

This is a consequence of [**C∂C.7**] and the following calculation:

$$\begin{aligned}
\partial^3[f] \circ (\phi \times \phi) &= \partial[\partial^2[f]] \circ (\phi \times \phi) \\
&= \partial[\partial^2[f] \circ \phi] \circ (\phi \times \phi) \\
&= \partial[\partial^2[f]] \circ (\phi \times \phi) \circ (\phi \times \phi) \\
&= \partial^3[f]
\end{aligned}$$

We can then show [**C∂C.6(a)**]:

$$\begin{aligned}
&\partial^{\mathrm{T}}[\partial^{\mathrm{T}}[f]] \circ^{\mathrm{T}} \langle \langle x, 0 \rangle, \langle 0, w \rangle \rangle^{\mathrm{T}} \\
&= \mu \circ \mathrm{T}(\partial^2[f]) \circ \phi \circ (\phi \times \phi) \circ \langle \langle x, 0 \rangle, \langle 0, w \rangle \rangle \\
&= \mu \circ \partial\left[\mathrm{T}(\partial[f])\right] \circ (\phi \times \phi) \circ \langle \langle x, 0 \rangle, \langle 0, w \rangle \rangle \\
&= \mu \circ \left\langle \partial^2[f] \circ (\pi_1 \times \pi_1), \partial^3[f] \right\rangle \circ (\phi \times \phi) \circ \langle \langle x, 0 \rangle, \langle 0, w \rangle \rangle \\
&= \mu \circ \left\langle \partial^2[f] \circ \langle \langle x_0, 0 \rangle, \langle 0, w_0 \rangle \rangle, \partial^3[f] \circ \langle \langle x, 0 \rangle, \langle 0, w \rangle \rangle \right\rangle \\
&= \mu \circ \left\langle \partial[f] \circ \langle x_0, w_0 \rangle, \partial^2[f] \circ \langle x, w \rangle \right\rangle \\
&= \partial^{\mathrm{T}}[f] \circ^{\mathrm{T}} \langle x, w \rangle
\end{aligned}$$

[**C∂C.7**] Follows by similar calculations to [**C∂C.2**]. □

As a result, the Kleisli category of a Cartesian difference category is again a Cartesian difference category, whose infinitesimal extension is neither the identity or the zero map. This allows one to build numerous examples of interesting and exotic Cartesian difference categories, such as the Kleisli category of the tangent bundle in any Cartesian differential category (one can, of course, iterate this process by taking the Kleisli category of the Kleisli category, *ad infinitum*). We highlight the importance of this construction for the Cartesian differential case as it does not in general result in a Cartesian differential category: even if $\varepsilon = 0$, it is always the case that $\varepsilon^{\mathrm{T}} \neq 0$.

We conclude this section by taking a look at the linear maps and the ε^{T}-linear maps in the Kleisli category. A Kleisli map $f = \langle f_0, f_1 \rangle$ is linear in the Kleisli category if $\partial^{\mathrm{T}}[f] = f \circ^{\mathrm{T}} \pi_1^{\mathrm{T}}$, which amounts to requiring that:

$$\langle \partial[f_0], \partial[f_1] \rangle = \langle f_0 \circ \pi_2, f_1 \circ \pi_2 \rangle$$

Therefore a Kleisli map is linear in the Kleisli category if and only if it is the pairing of maps which are linear in the base category. On the other hand, f is ε^{T}-linear if $\varepsilon^T(f) = \langle 0, f_0 + \varepsilon(f_1) \rangle$ is linear in the Kleisli category, which in this case amounts to requiring that $f_0 + \varepsilon(f_1)$ be linear. Therefore, if f_0 is linear and f_1 is ε-linear, then f is ε^{T}-linear.

6.4 Difference λ-Categories

In order to give a semantics for the differential λ-calculus, it does not suffice to ask for a Cartesian differential category equipped with exponentials – the exponential structure has to play well with both the additive and the differential structure, in the sense of Definition 2.3.4.

The same is true of difference categories: if we hope to interpret any reasonable sort of higher-order calculus, such as the one we will define in the next chapter, we will require an axiom similar to [**DλC.1**], together with a condition requiring higher-order functions to respect the infinitesimal extension. Intuitively, we shall require that $\lambda x.\varepsilon(t)$ be equal to $\varepsilon(\lambda x.t)$.

Definition 6.4.1. We remind the reader that a Cartesian left-additive category is **Cartesian closed left-additive** whenever it is Cartesian closed and the following equations hold:

i. $\Lambda(f + g) = \Lambda(f) + \Lambda(g)$

ii. $\Lambda(0) = 0$

A Cartesian difference category $\mathbf{C}$ is a **difference λ-category** if it Cartesian closed left-additive and satisfies the following additional axioms:

[$\partial\lambda$**C.1**] $\partial[\Lambda(f)] = \Lambda\left(\partial[f] \circ \langle(\pi_1 \times \mathbf{id}), (\pi_2 \times 0)\rangle\right)$

[$\partial\lambda$**C.2**] $\Lambda(\varepsilon(f)) = \varepsilon\left(\Lambda(f)\right)$

Remark 6.4.1. Let $\mathbf{sw}$ denote the composite

$$\langle\langle\pi_{11}, \pi_2\rangle, \pi_{21}\rangle : (A \times B) \times C \to (A \times C) \times B$$

Then the derivative $\partial[\Lambda(f)]$ can be equivalently written in terms of $\mathbf{sw}$ as:

$$\partial[\Lambda(f)] \coloneqq \Lambda\left(\partial[f] \circ (\mathbf{id} \times \langle\mathbf{id}, 0\rangle) \circ \mathbf{sw}\right)$$

Axiom [$\partial\lambda$**C.1**] is identical to its differential analogue [**DλC.1**], and the previous remark shows that it follows the same broad intuition. Given a map $f : A \times B \to C$, we usually understand the composite $\partial[f] \circ (\mathbf{id}_{A \times B} \times (\mathbf{id}_A \times 0_B)) : (A \times B) \times A \to C$ as a partial derivative of f with respect to its first argument. Hence, just as it was with differential λ-categories, axiom [$\partial\lambda$**C.1**] states that the derivative of a curried function is precisely the derivative of the uncurried function with respect to its first argument (compare with the much weaker Theorem 5.1.3, which merely states that both expressions act on $A \Rightarrow B$ in the same way).

Example 6.4.1. Let $\mathbf{C}$ be a differential λ-category. Then the corresponding Cartesian difference category (setting $\varepsilon = 0$) is a difference λ-category.

Example 6.4.2. The category $\mathbf{Ab}_\star$ is a difference λ-category. Given groups G, H, the exponential $G \Rightarrow H$ is defined pointwise as in Theorem 5.2.3. Evidently the exponential respects the monoidal structure and the infinitesimal extension (trivial). We check that it also verifies axiom [$\partial\lambda$**C.1**]:

$$\begin{aligned}\partial[\Lambda(f)](x, u)(y) &= \Lambda(f)(x + u)(y) - \Lambda(f)(x)(y)\\ &= f(x + u, y) - f(x, y)\\ &= \Lambda(\partial[f] \circ (\mathbf{id} \times \langle\mathbf{id}, 0\rangle) \circ \mathbf{sw})(x, u)(y)\end{aligned}$$

As is the case in differential λ-categories, we can define a "differential substitution" operator on the semantic side. This operator is akin to post-composition with a partial derivative, and can be defined as follows.

Definition 6.4.2. Given morphisms $s : A \times B \to C, u : A \to B$, we define their **differential composition** $s \star u : A \times B \to C$ by:

$$s \star u \coloneqq \partial[s] \circ \langle\mathbf{id}_{A \times B}, \langle 0_A, u \circ \pi_1\rangle\rangle$$

We should understand the morphism $s \star u$ as the partial derivative of s in its second argument, pre-composed with the morphism u. When we develop our calculus, this will correspond precisely to the differential substitution of the top variable in s by the term u.

Some technical results follow that will be of use later. These all correspond to well-known properties of differential λ-categories (see e.g. [22, Lemma 4.8]) and their proofs are identical, but we reproduce them here for reference, and to show that their statements do not differ in our setting (as opposed to some other technical lemmas in that work which hinge on derivatives being additive).

Lemma 6.4.1. Let $f : A \to B, g : A \to C, h : (A \times B) \times E \to F$ be arbitrary $\mathbf{C}$-morphisms. Then the following properties hold:

i. $\partial[\mathbf{sw}] = \mathbf{sw} \circ \pi_2$

ii. $(g \circ \pi_1) \star f = 0$

iii. $\Lambda(h) \star f = \Lambda(((h \circ \mathbf{sw}) \star (f \circ \pi_1)) \circ \mathbf{sw})$

iv. $\Lambda^-(\Lambda(h) \star f) = ((h \circ \mathbf{sw}) \star (f \circ \pi_1)) \circ \mathbf{sw}$

Proof.

i.
$$\pi_2 \star f = \mathrm{D}(\pi_2) \circ \langle \mathbf{id}, \langle 0, f \circ \pi_1 \rangle \rangle = \pi_{22} \circ \langle \mathbf{id}, \langle 0, f \circ \pi_1 \rangle \rangle = f \circ \pi_1$$

ii.
$$\begin{aligned}(g \circ \pi_1) \star f &= \partial[g \circ \pi_1] \circ \langle \mathbf{id}, \langle 0, f \circ \pi_1 \rangle \rangle \\ &= \partial[g] \circ \langle \pi_{11}, \pi_{12} \rangle \circ \langle \mathbf{id}, \langle 0, f \circ \pi_1 \rangle \rangle \\ &= \partial[g] \circ \langle \pi_1, 0 \rangle \\ &= 0\end{aligned}$$

iii.
$$\begin{aligned}\Lambda(h) \star f &= \partial[\Lambda(h)] \circ \langle \mathbf{id}, \langle 0, f \circ \pi_1 \rangle \rangle \\ &= \Lambda\Big(\partial[h] \circ (\mathbf{id} \times \langle \mathbf{id}, 0 \rangle) \circ \mathbf{sw}\Big) \circ \langle \mathbf{id}, \langle 0, f \circ \pi_1 \rangle \rangle \\ &= \Lambda\Big(\partial[h] \circ (\mathbf{id} \times \langle \mathbf{id}, 0 \rangle) \circ \mathbf{sw} \circ \big(\langle \mathbf{id}, \langle 0, f \circ \pi_1 \rangle \rangle \times \mathbf{id}\big)\Big) \\ &= \Lambda\Big(\partial[h] \circ (\mathbf{id} \times \langle \mathbf{id}, 0 \rangle) \circ \langle \mathbf{id}, \langle 0, f \circ \pi_{11} \rangle \rangle \Big) \\ &= \Lambda\Big(\partial[h] \circ \langle \mathbf{id}, \langle \langle 0, f \circ \pi_{11} \rangle, 0 \rangle \rangle \Big) \\ &= \Lambda\Big(\partial[h] \circ \langle \mathbf{sw} \circ \pi_1, \mathbf{sw} \circ \pi_2 \rangle \circ \langle \mathbf{sw}, \langle 0, f \circ \pi_{11} \rangle \rangle \Big) \\ &= \Lambda\Big(\partial[h \circ \mathbf{sw}] \circ \langle \mathbf{id}, \langle 0, (f \circ \pi_1) \circ \pi_1 \rangle \rangle \circ \mathbf{sw}\Big) \\ &= \Lambda\Big(((h \circ \mathbf{sw}) \star (f \circ \pi_1)) \circ \mathbf{sw}\Big)\end{aligned}$$

iv. Immediate consequence of *iii*. ☐

A central property of differential λ-categories is a deep correspondence between differentiation and the evaluation map. As one would expect, the partial derivative of the evaluation map gives one a first-class derivative operator (see, for example, [22][Lemma 4.5], which provides an interpretation for the differential substitution operator in the differential λ-calculus). This property still holds in difference categories, although its formulation is somewhat more involved.

Lemma 6.4.2. For any **C**-morphisms $\Lambda(f) : A \to (B \Rightarrow C), e : A \to B$, the following identities hold:

i. $\partial[\mathbf{ev} \circ \langle \Lambda(f), e\rangle] = \mathbf{ev} \circ \langle \partial[\Lambda(f)], e \circ \pi_1\rangle + \partial[f] \circ \langle\langle \pi_1 + \varepsilon(\pi_2), e \circ \pi_1\rangle, \langle 0, \partial[e]\rangle\rangle$

ii. $\partial[\mathbf{ev} \circ \langle \Lambda(f), e\rangle] = \mathbf{ev} \circ \langle \partial[\Lambda(f)], e \circ \pi_1 + \varepsilon(\partial[e])\rangle + \partial[f] \circ \langle\langle \pi_1, e \circ \pi_1\rangle, \langle 0, \partial[e]\rangle\rangle$

Proof. We prove *i.* explicitly. The proof for *ii.* follows the same structure but applies regularity of $\partial[f]$ in the opposite direction.

$$\begin{aligned}
&\partial[\mathbf{ev} \circ \langle \Lambda(f), e\rangle]\\
&\quad= \partial[f \circ \langle \mathbf{id}, e\rangle]\\
&\quad= \partial[f] \circ \langle\langle \mathbf{id}, e\rangle \circ \pi_1, \langle \pi_2, \partial[e]\rangle\rangle\\
&\quad= \partial[f] \circ \langle\langle \pi_1, e \circ \pi_1\rangle, \langle \pi_2, 0\rangle + \langle 0, \partial[e]\rangle\rangle\\
&\quad= \partial[f] \circ \langle\langle \pi_1, e \circ \pi_1\rangle, \langle \pi_2, 0\rangle\rangle + \partial[f] \circ \langle\langle \pi_1 + \varepsilon(\pi_2), e \circ \pi_1\rangle, \langle 0, \partial[e]\rangle\rangle\\
&\quad= \partial[f] \circ \langle\langle \pi_1, e \circ \pi_1\rangle, \langle \pi_2, 0\rangle\rangle + \partial[f] \circ \langle\langle \pi_1 + \varepsilon(\pi_2), e \circ \pi_1\rangle, \langle 0, \partial[e]\rangle\rangle\\
&\quad= \partial[f] \circ \langle (\pi_1 \times \mathbf{id}), (\pi_2 \times 0)\rangle \circ \langle \mathbf{id}, e \circ \pi_1\rangle + \partial[f] \circ \langle\langle \pi_1 + \varepsilon(\pi_2), e \circ \pi_1\rangle, \langle 0, \partial[e]\rangle\rangle\\
&\quad= \mathbf{ev} \circ \langle \Lambda(\partial[f] \circ \langle (\pi_1 \times \mathbf{id}), (\pi_2 \times 0)\rangle), e \circ \pi_1\rangle\\
&\qquad\quad + \partial[f] \circ \langle\langle \pi_1 + \varepsilon(\pi_2), e \circ \pi_1\rangle, \langle 0, \partial[e]\rangle\rangle\\
&\quad= \mathbf{ev} \circ \langle \partial[\Lambda(f)], e \circ \pi_1\rangle + \partial[f] \circ \langle\langle \pi_1 + \varepsilon(\pi_2), e \circ \pi_1\rangle, \langle 0, \partial[e]\rangle\rangle
\end{aligned}$$
□

The results below also have rough analogues in the theory of Cartesian differential categories, but the correspondence starts growing a bit more distant as any result that hinges on derivatives being additive will, in general, only hold up to some second-order term in the theory of difference categories.

Lemma 6.4.3. Let $f : A \times B \times C \to D, g : A \to B, g' : A \times B \to B, e : A \times B \to C$ be arbitrary **C**-morphisms. Then the following identities hold:

i. $(\mathbf{ev} \circ \langle \Lambda(f), e\rangle) \star g = \mathbf{ev} \circ \langle \Lambda(f \star (e \star g)), e\rangle + \mathbf{ev} \circ \langle \Lambda(f) \star g, e \circ \langle \pi_1, \pi_2 + \varepsilon(g) \circ \pi_1\rangle\rangle$

ii. $\Lambda(f \star e) \star g = \Lambda\Big[\Lambda^-(\Lambda(f) \star g) \star (e \circ (\mathbf{id} + \langle 0, \varepsilon(g)\rangle))) + \varepsilon(f \star e) \star (e \star g) + (f \star (e \star g))\Big]$

iii. $\Lambda(f \star e) \circ \langle \pi_1, g'\rangle = \Lambda(\Lambda^-(\Lambda(f) \circ \langle \pi_1, g'\rangle) \star (e \circ \langle \pi_1, g'\rangle))$

Proof. The proofs follow the same structure as those in [22, Lemma 4.9], with the added caveat that derivatives in difference categories are not additive.

i. For readability we will abbreviate $\langle \mathbf{id}, \langle 0, g \circ \pi_1 \rangle\rangle$ as ψ, and remark that, for any s, $\partial[s] \circ \psi = f \star g$. The proof then proceeds by straightforward calculation, applying Lemma 6.4.2[ii.]:

$$\begin{aligned}
&(\mathbf{ev} \circ \langle \Lambda(f), e\rangle) \star g \\
&= \partial[\mathbf{ev} \circ \langle \Lambda(f), e\rangle] \circ \psi \\
&= \Big(\mathbf{ev} \circ \langle \partial[\Lambda(f)], e \circ \pi_1 + \varepsilon(\partial[e])\rangle + \partial[f] \circ \langle\langle \pi_1, e \circ \pi_1\rangle, \langle 0, \partial[e]\rangle\rangle\Big) \circ \psi \\
&= \Big(\mathbf{ev} \circ \langle \partial[\Lambda(f)] \circ \psi, (e \circ \pi_1 + \varepsilon(\partial[e])) \circ \psi\rangle\Big) \\
&\qquad + \Big(\partial[f] \circ \langle\langle \mathbf{id}, e\rangle, \langle 0, e \star g\rangle\rangle\Big) \\
&= \Big(\mathbf{ev} \circ \langle \Lambda(f) \star g, e \circ (\mathbf{id} + \langle 0, \varepsilon(g) \circ \pi_1\rangle)\rangle\Big) \\
&\qquad + \Big(\partial[f] \circ \langle \mathbf{id}, \langle 0, (e \star g) \circ \pi_1\rangle\rangle \circ \langle \mathbf{id}, e\rangle\Big) \\
&= \Big(\mathbf{ev} \circ \langle \Lambda(f) \star g, e \circ \langle \pi_1, \pi_2 + \varepsilon(g) \circ \pi_1\rangle\rangle\Big) \\
&\qquad + \Big(\mathbf{ev} \circ \langle \Lambda(f \star (e \star g)), e\rangle\Big)
\end{aligned}$$

ii. Before proceeding with the proof we will first eliminate the abstractions on both sides of the goal. To this end, we note we can rewrite the LHS (the only term where the abstraction is not the outermost construct) by applying Lemma 6.4.1:

$$\begin{aligned}
\Lambda(f \star e) \star g &= \Lambda\Big((((f \star e) \circ \mathbf{sw}) \star (g \circ \pi_1)) \circ \mathbf{sw}\Big) \\
&= \Lambda\Big(\partial[(f \star e) \circ \mathbf{sw}] \circ \langle \mathbf{sw}, \langle 0, g \circ \pi_{11}\rangle\rangle\Big)
\end{aligned}$$

We abbreviate $\langle \mathbf{sw}, \langle 0, g \circ \pi_{11}\rangle\rangle$ as φ. Then we seek to expand the term $\partial[(f \star e) \circ \mathbf{sw}] \circ \varphi$ and show how it can be decomposed into the following three summands:

$$\Lambda^-(\Lambda(f) \star g) \star (e \circ (\mathbf{id} + \langle 0, \varepsilon(g)\rangle))$$

$$\varepsilon(f \star e) \star (e \star g)$$

$$f \star (e \star g)$$

Throughout the proof we will colour some terms to indicate into which of the three terms above they are intended to expand.

$$\begin{aligned}
&\partial[(f \star e) \circ \mathbf{sw}] \circ \varphi \\
&= \partial[\partial[f] \circ \langle \mathbf{id}, \langle 0, e \circ \pi_1\rangle\rangle\, \mathbf{sw}] \circ \varphi
\end{aligned}$$

$$
\begin{aligned}
&= \partial[\partial[f] \circ \langle \mathbf{sw}, \langle 0, e \circ \langle \pi_{11}, \pi_2 \rangle \rangle \rangle] \circ \varphi \\
&= \partial[\partial[f]] \circ \langle \langle \mathbf{sw}, \langle 0, e \circ \langle \pi_{11}, \pi_2 \rangle \rangle \rangle \circ \pi_1, \langle \partial[\mathbf{sw}], \langle 0, \partial[e \circ \langle \pi_{11}, \pi_2 \rangle] \rangle \rangle \rangle \circ \varphi \\
&= \partial[\partial[f]] \circ \langle \langle \mathbf{sw}, \langle 0, e \circ \langle \pi_{11}, \pi_2 \rangle \rangle \rangle \circ \mathbf{sw}, \langle \partial[\mathbf{sw}], \langle 0, \partial[e \circ \langle \pi_{11}, \pi_2 \rangle] \rangle \rangle \circ \varphi \rangle \\
&= \partial[\partial[f]] \circ \langle \langle \mathbf{sw}, \langle 0, e \circ \langle \pi_{11}, \pi_2 \rangle \rangle \rangle \circ \mathbf{sw}, \langle \partial[\mathbf{sw}] \circ \varphi, \langle 0, \partial[e \circ \langle \pi_{11}, \pi_2 \rangle] \circ \varphi \rangle \rangle \rangle \\
&= \partial[\partial[f]] \circ \langle \langle \mathbf{id}, \langle 0, e \circ \langle \pi_{11}, \pi_{21} \rangle \rangle \rangle, \langle \partial[\mathbf{sw}] \circ \varphi, \langle 0, \partial[e] \circ \mathbf{T}(\langle \pi_{11}, \pi_2 \rangle) \circ \varphi \rangle \rangle \rangle \\
&= \partial[\partial[f]] \circ \langle \langle \mathbf{id}, \langle 0, e \circ \pi_1 + \varepsilon(\partial[e]) \circ \mathbf{T}(\langle \pi_{11}, \pi_2 \rangle) \circ \varphi \rangle \rangle, \langle \partial[\mathbf{sw}] \circ \varphi, 0 \rangle \rangle \\
&\qquad + \partial[\partial[f]] \circ \langle \langle \mathbf{id}, \langle 0, e \circ \pi_1 \rangle \rangle, \langle 0, \langle 0, \partial[e] \circ \langle \pi_{11}, \pi_2 \rangle \circ \varphi \rangle \rangle \rangle
\end{aligned}
$$

First we simplify the second term (in sand), and will return to the first term (in green) later.

$$
\begin{aligned}
&\partial[\partial[f]] \circ \langle \langle \mathbf{id}, \langle 0, e \circ \pi_1 \rangle \rangle, \langle 0, \langle 0, \partial[e] \circ \mathbf{T}(\langle \pi_{11}, \pi_2 \rangle) \circ \varphi \rangle \rangle \rangle \\
&\quad = \partial[f] \circ \langle \mathbf{id} + \langle 0, \varepsilon(e) \circ \pi_1 \rangle, \langle 0, \partial[e] \circ \mathbf{T}(\langle \pi_{11}, \pi_2 \rangle) \circ \varphi \rangle \rangle \\
&\quad = \partial[f] \circ \langle \mathbf{id}, \langle 0, \partial[e] \circ \mathbf{T}(\langle \pi_{11}, \pi_2 \rangle) \circ \varphi \rangle \rangle \\
&\qquad\quad + \varepsilon\partial[\partial[f]] \circ \langle \langle \mathbf{id}, \langle 0, \partial[e] \circ \mathbf{T}(\langle \pi_{11}, \pi_2 \rangle) \circ \varphi \rangle \rangle, \langle \langle 0, e \circ \pi_1 \rangle, 0 \rangle \rangle
\end{aligned}
$$

We now focus on the first summand, in indigo ink.

$$
\begin{aligned}
&\partial[f] \circ \langle \mathbf{id}, \langle 0, \partial[e] \circ \mathbf{T}(\langle \pi_{11}, \pi_2 \rangle) \circ \varphi \rangle \rangle \\
&\quad = \partial[f] \circ \langle \mathbf{id}, \langle 0, \partial[e] \circ \langle \langle \pi_{11}, \pi_2 \rangle \circ \pi_1, \langle \pi_{11}\pi_2, \circ \rangle\, \pi_2 \rangle \circ \varphi \rangle \rangle \\
&\quad = \partial[f] \circ \langle \mathbf{id}, \langle 0, \partial[e] \circ \langle \langle \pi_{111}, \pi_{21} \rangle, \langle \pi_{112}, \pi_{22} \rangle \rangle \circ \varphi \rangle \rangle \\
&\quad = \partial[f] \circ \langle \mathbf{id}, \langle 0, \partial[e] \circ \langle \langle \pi_{11}, \pi_2 \rangle \circ \mathbf{sw}, \langle \pi_{11}, \pi_2 \rangle \circ \langle 0, g \circ \pi_{11} \rangle \rangle \rangle \rangle \\
&\quad = \partial[f] \circ \langle \mathbf{id}, \langle 0, \partial[e] \circ \langle \langle \pi_{11}, \pi_{21} \rangle, \langle 0, g \circ \pi_{11} \rangle \rangle \rangle \rangle \\
&\quad = \partial[f] \circ \langle \mathbf{id}, \langle 0, \partial[e] \circ \langle \pi_1, \langle 0, g \circ \pi_{11} \rangle \rangle \rangle \rangle \\
&\quad = \partial[f] \circ \langle \mathbf{id}, \langle 0, \partial[e] \circ \langle \mathbf{id}, \langle 0, g \circ \pi_1 \rangle \rangle \rangle \circ \pi_1 \rangle \\
&\quad = \partial[f] \circ \langle \mathbf{id}, \langle 0, (e \star g) \circ \pi_1 \rangle \rangle \\
&\quad = f \star (e \star g)
\end{aligned}
$$

To simplify the rose term, we proceed as follows:

$$
\begin{aligned}
&\varepsilon\partial[\partial[f]] \circ \langle \langle \mathbf{id}, \langle 0, \partial[e] \circ \mathbf{T}(\langle \pi_{11}, \pi_2 \rangle) \circ \varphi \rangle \rangle, \langle \langle 0, e \circ \pi_1 \rangle, 0 \rangle \rangle \\
&\quad = \varepsilon\partial[\partial[f]] \circ \langle \langle \mathbf{id}, \langle 0, (e \star g) \circ \pi_1 \rangle \rangle, \langle \langle 0, e \circ \pi_1 \rangle, 0 \rangle \rangle \\
&\quad = \varepsilon\partial[\partial[f]] \circ \langle \langle \mathbf{id}, \langle 0, e \circ \pi_1 \rangle \rangle, \langle \langle 0, (e \star g) \circ \pi_1 \rangle, 0 \rangle \rangle \\
&\quad = \varepsilon\partial[\partial[f]] \circ \langle \langle \mathbf{id}, \langle 0, e \circ \pi_1 \rangle \rangle, \langle \langle 0, (e \star g) \circ \pi_1 \rangle, \langle 0, \partial[e] \circ \langle \pi_{11}, 0 \rangle \rangle \rangle \rangle \\
&\quad = \varepsilon\partial[\partial[f]] \circ \langle \langle \mathbf{id}, \langle 0, e \circ \pi_1 \rangle \rangle \circ \pi_1, \langle \pi_2, \langle 0, \partial[e] \circ (\pi_1 \times \pi_1) \rangle \rangle \rangle \circ \langle \mathbf{id}, \langle 0, (e \star g) \circ \pi_1 \rangle \rangle \\
&\quad = \varepsilon\partial[\partial[f] \circ \langle \mathbf{id}, \langle 0, e \circ \pi_1 \rangle \rangle] \circ \langle \mathbf{id}, \langle 0, (e \star g) \circ \pi_1 \rangle \rangle \\
&\quad = \varepsilon(f \star e) \star (e \star g)
\end{aligned}
$$

It only remains to simplify the green term. This follows mostly the same process as the rose term, applying axiom **[C∂C.0]** to fold the derivative of e. We will write e' for the resulting term $e \circ (\mathbf{id} + \langle 0, \varepsilon(g) \circ \pi_1 \rangle)$.

$$
\partial[\partial[f]] \circ \langle \langle \mathbf{id}, \langle 0, e \circ \pi_1 + \varepsilon(\partial[e]) \circ \mathbf{T}(\langle \pi_{11}, \pi_2 \rangle) \circ \varphi \rangle \rangle, \langle \partial[\mathbf{sw}] \circ \varphi, 0 \rangle \rangle
$$

$$
\begin{aligned}
&= \partial[\partial[f]] \circ \langle\langle \mathbf{id}, \langle 0, e \circ \pi_1 + \varepsilon(\partial[e]) \circ \mathbf{T}(\langle \pi_{11}, \pi_2 \rangle) \circ \varphi \rangle\rangle, \langle \partial[\mathbf{sw}] \circ \varphi, 0 \rangle\rangle \\
&= \partial[\partial[f]] \circ \langle\langle \mathbf{id}, \langle 0, e \circ \pi_1 + \varepsilon(\partial[e]) \circ \langle \pi_1, \langle 0, g \circ \pi_{11} \rangle\rangle\rangle\rangle, \langle \partial[\mathbf{sw}] \circ \varphi, 0 \rangle\rangle \\
&= \partial[\partial[f]] \circ \langle\langle \mathbf{id}, \langle 0, (e + \varepsilon(\partial[e]) \circ \langle \mathbf{id}, \langle 0, g \circ \pi_1 \rangle\rangle) \circ \pi_1 \rangle\rangle, \langle \mathbf{sw} \circ \pi_2 \circ \varphi, 0 \rangle\rangle \\
&= \partial[\partial[f]] \circ \big\langle \big\langle \mathbf{id}, \big\langle 0, e' \circ \pi_1 \big\rangle\big\rangle, \langle\langle\langle 0, g \circ \pi_{11} \rangle, 0 \rangle, 0 \rangle\big\rangle \\
&= \partial[\partial[f]] \circ \big\langle \langle \mathbf{id}, \langle\langle 0, g \circ \pi_{11} \rangle, 0 \rangle\rangle, \big\langle \big\langle 0, e' \circ \pi_1 \big\rangle, 0 \big\rangle\big\rangle \\
&= \partial[\partial[f]] \circ \langle\langle \mathbf{id}, \langle\langle 0, g \circ \pi_{11} \rangle, 0 \rangle\rangle \circ \pi_1, \langle \pi_2, \langle\langle 0, \partial[g] \circ (\pi_{11} \times \pi_{11}) \rangle, 0 \rangle\rangle\rangle \circ \big\langle \mathbf{id}, \big\langle 0, e' \circ \pi_1 \big\rangle\big\rangle \\
&= \partial[\partial[f] \circ \langle \mathbf{id}, \langle\langle 0, g \circ \pi_{11} \rangle, 0 \rangle\rangle] \circ \big\langle \mathbf{id}, \big\langle 0, e' \circ \pi_1 \big\rangle\big\rangle \\
&= \partial[\partial[f] \circ \langle \mathbf{sw}, \langle\langle 0, g \circ \pi_{11} \rangle, 0 \rangle\rangle \circ \mathbf{sw}] \circ \big\langle \mathbf{id}, \big\langle 0, e' \circ \pi_1 \big\rangle\big\rangle \\
&= \partial[\partial[f \circ \mathbf{sw}] \circ \langle \mathbf{id}, \langle 0, g \circ \pi_{11} \rangle\rangle \circ \mathbf{sw}] \circ \big\langle \mathbf{id}, \big\langle 0, e' \circ \pi_1 \big\rangle\big\rangle \\
&= (((f \circ \mathbf{sw}) \star (g \circ \pi_1)) \circ \mathbf{sw}) \star e' \\
&= (\Lambda^{-}(\Lambda(f) \star g)) \star e'
\end{aligned}
$$

iii. The proof for *iii.* is simpler and in fact identical to the one in [22], as additivity of the derivative is never assumed. We reproduce it here for completeness and to account for the differences in notation.

First, since $\Lambda(f \star e) \circ \langle \pi_1, g' \rangle$ is precisely $\Lambda((f \star e) \circ (\langle \pi_1, g' \rangle \times \mathbf{id}))$ it suffices to show that $(f \star e) \circ (\langle \pi_1, g' \rangle \times \mathbf{id}) = \Lambda^{-}(\Lambda(f) \circ \langle \pi_1, g' \rangle) \star (e \circ \langle \pi_1, g' \rangle)$, which we do by straightforward calculation.

$$
\begin{aligned}
&(f \star e) \circ (\big\langle \pi_1, g' \big\rangle \times \mathbf{id}) \\
&\quad= \partial[f] \circ \langle \mathbf{id}, \langle 0, e \circ \pi_1 \rangle\rangle \circ (\big\langle \pi_1, g' \big\rangle \times \mathbf{id}) \\
&\quad= \partial[f] \circ \big\langle (\big\langle \pi_1, g' \big\rangle \times \mathbf{id}), \big\langle 0, e \circ \big\langle \pi_1, g' \big\rangle \circ \pi_1 \big\rangle\big\rangle \\
&\quad= \partial[f] \circ \big\langle \big\langle \big\langle \pi_{11}, g' \circ \pi_1 \big\rangle, \pi_2 \big\rangle, \big\langle \big\langle 0, \partial[g'] \circ (\pi_1 \times 0) \big\rangle, e \circ \big\langle \pi_1, g' \big\rangle \circ \pi_1 \big\rangle\big\rangle \\
&\quad= \partial[f] \circ \langle\langle\langle \pi_{111}, g \circ \pi_{11} \rangle, \pi_{21} \rangle, \langle\langle \pi_{112}, \partial[g] \circ (\pi_1 \times \pi_1) \rangle, \pi_{22} \rangle\rangle \circ \big\langle \mathbf{id}, \big\langle 0, e \circ \big\langle \pi_1, g' \big\rangle \circ \pi_1 \big\rangle\big\rangle \\
&\quad= (f \circ \langle\langle \pi_{11}, g \circ \pi_1 \rangle, \pi_2 \rangle) \star (e \circ \big\langle \pi_1, g' \big\rangle) \\
&\quad= \Lambda^{-}(\Lambda(f \circ (\big\langle \pi_1, g' \big\rangle \times \mathbf{id}))) \star (e \circ \big\langle \pi_1, g' \big\rangle) \\
&\quad= \Lambda^{-}(\Lambda(f) \circ \big\langle \pi_1, g' \big\rangle) \star (e \circ \big\langle \pi_1, g' \big\rangle)
\end{aligned}
$$

$\square$

Chapter 7

A calculus of finite differences

So far our treatment of change actions and difference categories has been mostly semantic. Change actions may arise and be applied in the setting of incremental computation, but they exist merely as a theoretical model for reasoning about it and deriving algorithms, rather than an executable entity. This is in contrast with Cartesian differential categories and, in particular, differential λ-categories, for which the differential λ-calculus is known to be an internal language.

It is reasonable to ask, then, whether there is a language of change actions and differential maps – especially since, as we have shown, difference categories subsume differential categories. The issue is far from trivial, as many of the properties of the differential λ-calculus crucially hinge on derivatives being linear.

Through this chapter we will provide an affirmative answer to this question: in Section 7.1 we develop λ_ε, an untyped calculus in the spirit of the differential λ-calculus which adds infinitesimal extensions and relaxes the linearity requirement. In Section 7.2 we then introduce a type system for this calculus and prove a strong normalisation theorem. Finally, in Section 7.3, we show how λ_ε-terms can be soundly interpreted in any difference λ-category.

7.1 An Untyped Calculus of Differences

We proceed in a manner similar to Vaux [108] in his treatment of the algebraic λ-calculus; that is, we will first define a set of "unrestricted" terms Λ_ε which we will later consider up to an equivalence relation arising from the theory of difference categories.

Definition 7.1.1. The set Λ_ε of **unrestricted terms** of the λ_ε-calculus is given by the following inductive definition:

$$\text{Terms:} \quad s, t, e \;:=\; x \mid \lambda x.t \mid (s\ t) \mid \mathrm{D}(s) \cdot t \mid \varepsilon t \mid s + t \mid 0$$

assuming a countably infinite set of variables $x, y, z \dots$ is given.

Henceforth we will only consider the above terms up to α-equivalence. Since the only binder in the λ_ε-calculus is the usual λ-abstraction, the definition of free and bound variables is straightforward.

Definition 7.1.2. The set of **free variables** $\mathrm{FV}(t)$ of a term $t \in \Lambda_\varepsilon$ is defined by induction on the structure of t as follows:

$$\begin{array}{rcl}
\mathrm{FV}(x) & := & \{x\} \\
\mathrm{FV}(\lambda x.t) & := & \mathrm{FV}(t) \setminus \{x\} \\
\mathrm{FV}(s\ t) & := & \mathrm{FV}(s) \cup \mathrm{FV}(t) \\
\mathrm{FV}(\mathrm{D}(s) \cdot t) & := & \mathrm{FV}(s) \cup \mathrm{FV}(t) \\
\mathrm{FV}(\varepsilon t) & := & \mathrm{FV}(t) \\
\mathrm{FV}(s + t) & := & \mathrm{FV}(s) \cup \mathrm{FV}(t) \\
\mathrm{FV}(0) & := & \emptyset
\end{array}$$

As usual, a variable x is **free** in t whenever $x \in \mathrm{FV}(t)$. An occurrence of a variable x in some term t is said to be **bound** whenever it appears in some subterm t' of t with $x \notin \mathrm{FV}(t)$.

Two terms are said to be α-equivalent if they are identical up to a renaming of all their bound variables.

In what follows, we will speak of terms only up to α-equivalence. That is, we consider the terms $\lambda x.x$ and $\lambda y.y$ to be identical for all intents and purposes. Since this means we can rename bound variables freely, we will assume by convention that all bound variables appearing in any term $t \in \Lambda_\varepsilon$ are different from its free variables.

7.1.1 Differential Equivalence

Further to α-equivalence, we introduce here the notion of Euclidean equivalence of terms. The role of this relation is, as in [105], to enforce that the elementary algebraic properties of sums and actions are preserved. For example, we wish to treat the terms $\lambda x.(0 + \varepsilon(s + t))$ and $(\lambda x.\varepsilon t) + (\lambda x.\varepsilon s)$ as if they were equivalent (as it will be the case in the models). This equivalence relation also has the role of ensuring that the axioms of a Cartesian difference category are verified, especially regularity of derivatives.

Definition 7.1.3. A binary relation $\sim\ \subseteq \Lambda_\varepsilon \times \Lambda_\varepsilon$ is **contextual** whenever it satisfies the conditions in Figure 7.1 below.

$$\begin{array}{rcc}
t \sim t' & \Rightarrow & \lambda x.t \sim \lambda x.t' \\
t \sim t' & \Rightarrow & \varepsilon t \sim \varepsilon t' \\
s \sim s' \wedge t \sim t' & \Rightarrow & s\ t \sim s'\ t' \\
s \sim s' \wedge t \sim t' & \Rightarrow & \mathrm{D}(s) \cdot t \sim \mathrm{D}(s') \cdot t' \\
s \sim s' \wedge t \sim t' & \Rightarrow & s + t \sim s' + t'
\end{array}$$

Figure 7.1: Contextuality on unrestricted Λ_ε-terms

Lemma 7.1.1. Whenever $\sim$ is contextual, if t is a subterm of s and $t \sim t'$ then $s \sim s'$, where s' is the term resulting from substituting the occurrence of t in s for t'.

Definition 7.1.4. Differential equivalence $\sim_\varepsilon \subseteq \Lambda_\varepsilon \times \Lambda_\varepsilon$ is the least equivalence relation which is contextual and contains the relation $\sim^1_\varepsilon$ below Figure 7.2 below.

$$\begin{array}{rcl}
(s+t)+e & \sim^1_\varepsilon & s+(t+e) \\
s+0 & \sim^1_\varepsilon & s \\
s+t & \sim^1_\varepsilon & t+s \\
\varepsilon 0 & \sim^1_\varepsilon & 0 \\
\varepsilon(s+t) & \sim^1_\varepsilon & \varepsilon s + \varepsilon t
\end{array}
\qquad
\begin{array}{rcl}
\lambda x.0 & \sim^1_\varepsilon & 0 \\
\lambda x.(s+t) & \sim^1_\varepsilon & (\lambda x.s) + (\lambda x.t) \\
\lambda x.\varepsilon t & \sim^1_\varepsilon & \varepsilon(\lambda x.t) \\
0\ s & \sim^1_\varepsilon & 0 \\
(s+t)\ e & \sim^1_\varepsilon & (s\ e) + (t\ e) \\
(\varepsilon s)\ t & \sim^1_\varepsilon & \varepsilon(s\ t)
\end{array}$$

$$\begin{array}{rcl}
\mathrm{D}(0) \cdot e & \sim^1_\varepsilon & 0 \\
\mathrm{D}(s+t) \cdot e & \sim^1_\varepsilon & (\mathrm{D}(s) \cdot e) + (\mathrm{D}(t) \cdot e) \\
\mathrm{D}(\varepsilon t) \cdot e & \sim^1_\varepsilon & \varepsilon(\mathrm{D}(t) \cdot e)
\end{array}$$

$$\begin{array}{rcl}
\mathrm{D}(s) \cdot 0 & \sim^1_\varepsilon & 0 \\
\mathrm{D}(s) \cdot (t+e) & \sim^1_\varepsilon & \mathrm{D}(s) \cdot t + \mathrm{D}(s) \cdot e + \varepsilon(\mathrm{D}(\mathrm{D}(s) \cdot t) \cdot e) \\
\mathrm{D}(s) \cdot (\varepsilon t) & \sim^1_\varepsilon & \varepsilon(\mathrm{D}(s) \cdot t) \\
\mathrm{D}(\mathrm{D}(s) \cdot t) \cdot e & \sim^1_\varepsilon & \mathrm{D}(\mathrm{D}(s) \cdot e) \cdot t \\
\varepsilon^2 \mathrm{D}(\mathrm{D}(s) \cdot t) \cdot e & \sim^1_\varepsilon & \varepsilon \mathrm{D}(\mathrm{D}(s) \cdot t) \cdot e \\
s\ (t + \varepsilon e) & \sim^1_\varepsilon & (s\ t) + \varepsilon((\mathrm{D}(s) \cdot e)\ t)
\end{array}$$

Figure 7.2: Differential equivalence on unrestricted Λ_ε-terms

The above conditions can be separated in a number of conceptually distinct groups corresponding to their purpose. These are as follows:

- The first block of equations simply states that $+, 0$ define a commutative monoid and that ε defines a monoid homomorphism.
- The second block of equations amounts to stating that the monoid and infinitesimal extension structure on functions is pointwise.

- The third block of equations implies (and is equivalent to) stating that addition and infinitesimal extension are "linear", in the sense that they are equal to their own derivatives (that is, $\partial\,[+] = + \circ \pi_2$ and $\partial\,[\varepsilon] = \varepsilon$).

- The fourth block of equations states structural properties of the derivative, such as the derivative conditions and the commutativity of second derivatives. Similar equations are also present in the differential λ-calculus, where they merely state that the derivative is additive (as opposed to regular).

Most of these equations correspond directly to properties of Cartesian difference categories, with the only exception being the requirement that $\mathrm{D}(s) \cdot (\varepsilon t) \sim_\varepsilon \varepsilon(\mathrm{D}(s) \cdot t)$ and the "duplication of infinitesimals" in $\varepsilon\mathrm{D}(\mathrm{D}(s) \cdot t) \cdot e = \varepsilon^2\mathrm{D}(\mathrm{D}(s) \cdot t) \cdot e$. To understand the logic of the first equation, consider that the term $\mathrm{D}(\lambda x.s) \cdot \varepsilon t$ should roughly correspond to $\partial\,[\![s]\!]\,(x, \varepsilon\,[\![t]\!])$ which expands to:

$$\begin{aligned}\partial\,[\![s]\!]\,(x, \varepsilon\,[\![t]\!]) &= \partial\,[\![s]\!]\,(x, 0 + \varepsilon\,[\![t]\!]) \\ &= \partial\,[\![s]\!]\,(x, 0) + \varepsilon\partial^2\,[\![s]\!]\,(x, 0, 0, [\![t]\!]) \\ &= \varepsilon\partial^2\,[\![s]\!]\,(x, 0, 0, [\![t]\!])\end{aligned}$$

In the particular setting of Cartesian difference categories, the last term boils down to $\varepsilon\partial\,[\![s]\!]\,(x, [\![t]\!])$ by axiom **[C∂C.6]**, hence the equality $\mathrm{D}(s) \cdot \varepsilon t \sim_\varepsilon \varepsilon(\mathrm{D}(s) \cdot t)$ is justified, allowing infinitesimal extensions to be "pulled out" of differential application. The duplication of infinitesimals is simply a syntactic version of Lemma 6.1.6.iii.

Definition 7.1.5. The set λ_ε of **well-formed terms**, or simply **terms**, of the λ_ε-calculus is defined as the quotient set $\lambda_\varepsilon := \Lambda_\varepsilon / \sim_\varepsilon$. Whenever t is an unrestricted term, we write $\underline{t}$ to refer to the well-formed term represented by t, that is to say, the $\sim_\varepsilon$-equivalence class of t.

The notion of differential equivalence allows us to ensure that our calculus reflects the laws of the underlying models, but has the unintended consequence that our λ_ε-terms are equivalence classes, rather than purely syntactic objects. We will proceed by defining a notion of canonical form of a term and a canonicalization algorithm which explicitly constructs the canonical form of any given term, thus proving that $\sim_\varepsilon$ is decidable.

Definition 7.1.6. We define the sets $\mathrm{B}_\varepsilon \subset \mathrm{B}_\varepsilon^+ \subset \mathrm{B}_\varepsilon^* \subset \mathrm{C}_\varepsilon^+ \subset \mathrm{C}_\varepsilon (\subset \Lambda_\varepsilon)$ of **basic**, **positive**, **additive**, **positive canonical** and **canonical** terms according to the following grammar:

$$\begin{array}{rrcl}
\text{Basic terms:} & s^{\mathbf{b}}, t^{\mathbf{b}}, e^{\mathbf{b}} \in \mathrm{B}_\varepsilon & := & x \mid \lambda x.t^{\mathbf{b}} \mid (s^{\mathbf{b}}\ t^*) \mid \mathrm{D}(s^{\mathbf{b}}) \cdot t^{\mathbf{b}} \\
\text{Positive terms:} & s^+, t^+, e^+ \in \mathrm{B}_\varepsilon^+ & := & s^{\mathbf{b}} \mid s^{\mathbf{b}} + (t^+) \\
\text{Additive terms:} & s^*, t^*, e^* \in \mathrm{B}_\varepsilon^* & := & 0 \mid s^+ \\
\text{Positive canonical terms:} & S^+, T^+ \in \mathrm{C}_\varepsilon & := & \varepsilon^k s^{\mathbf{b}} \mid \varepsilon^k s^{\mathbf{b}} + (S^+) \\
\text{Canonical terms:} & S, T \in \mathrm{C}_\varepsilon & := & 0 \mid S^+
\end{array}$$

We will sometimes abuse the notation and write $\underline{t^*}$ or $\underline{t^{\mathbf{b}}}$ to denote well-formed terms whose canonical form is an additive or basic term respectively.

The above definition is somewhat technical, so a more informal description of canonical forms should be helpful. We observe that all the rules of differential equivalence can be oriented from left to right to obtain a rewrite system - intuitively, this rewrite system operates by pulling all the instances of addition and infinitesimal extension to the outermost layers of a term. Since every syntactic construct is additive except for application, basic terms may only contain additive terms as the arguments to a function application.

As infinitesimal extensions are themselves additive, we also want to disallow terms such as $\varepsilon(s+t)$, instead factoring out the extension into $\varepsilon s + \varepsilon t)$. A general canonical term $T \in \mathrm{C}_\varepsilon$ then has the form:

$$T = \varepsilon^{k_1} t_1^{\mathbf{b}} + (\varepsilon^{k_2} t_2 + (\ldots + \varepsilon^{k_n} t_n^{\mathbf{b}}) \ldots)$$

That is to say, a canonical term is similar to a polynomial with coefficients in the set of basic terms and a variable ε (but note that canonical terms are always written in their "fully distributed" form, that is, we write $\varepsilon s + (\varepsilon t + \varepsilon^2 e)$ rather than $\varepsilon((s+t) + \varepsilon e))$.

We will freely abuse the notation and write $\sum_{i=1}^n \varepsilon^{k_i} t_i^{\mathbf{b}}$ to denote a general canonical term, as this form is easier to manipulate in many cases. In particular, the canonical term 0 is precisely the sum of zero terms.

Definition 7.1.7. Given unrestricted terms s, t, we define their **canonical sum** $s \boxplus t$ by induction as follows:

$$0 \boxplus t := t$$

$$s \boxplus 0 := s$$

$$(s + s') \boxplus t := s + (s' \boxplus t)$$

Lemma 7.1.2. The canonical sum $S \boxplus T$ of any two canonical terms is a canonical term. Furthermore $\boxplus$ is associative and has 0 as an identity element. When S_i are canonical terms, we will write $\boxplus_{i=1}^n S_i$ for the canonical term $S_1 \boxplus (S_2 \boxplus \ldots S_n) \ldots)$.

Definition 7.1.8. Given an unrestricted λ_ε-term $t \in \Lambda_\varepsilon$, we define its **canonical form** $\mathbf{can}\,(t)$ by structural induction on t as follows:

- $\mathbf{can}\,(0) := 0$
- $\mathbf{can}\,(x) := x$
- $\mathbf{can}\,(s+t) := \mathbf{can}\,(s) \boxplus \mathbf{can}\,(t)$
- $\mathbf{can}\,(\varepsilon t) := \varepsilon^* \mathbf{can}\,(t)$, where

$$\varepsilon^* T := \begin{cases} \sum_{i=1}^n \varepsilon^* \varepsilon^{k_i} t_i^{\mathbf{b}} & \text{if } T = \sum_{i=1}^n \varepsilon^{k_i} t_i^{\mathbf{b}} \\ T & \text{if } T = \varepsilon \mathrm{D}(\mathrm{D}(e) \cdot u) \cdot v \\ \varepsilon T & \text{otherwise} \end{cases}$$

- If $\mathbf{can}\,(t) = \sum_{i=1}^n \varepsilon^{k_i} t_i^{\mathbf{b}}$ then

$$\mathbf{can}\,(\lambda x.t) := \sum_{i=1}^n \varepsilon^{k_i} (\lambda^* x. t_i^{\mathbf{b}})$$

- If $\mathbf{can}\,(s) = \sum_{i=1}^n \varepsilon^{k_i} s_i^{\mathbf{b}}$ and $\mathbf{can}\,(t) = T$ then

$$\mathbf{can}\,(\mathrm{D}(s) \cdot t) := \boxplus_{i=1}^{n} ((\varepsilon^*)^{k_i} \mathbf{reg}\,(s_i^{\mathbf{b}}, T))$$

where the **regularization** $\mathbf{reg}\,(s, T)$ is defined by structural induction on T:

$$\begin{aligned} \mathbf{reg}\,(s, 0) &:= 0 \\ \mathbf{reg}\,(s, \varepsilon^k t^{\mathbf{b}} + T') &:= \left[(\varepsilon^*)^k \, \mathrm{D}\,(s) \cdot t^{\mathbf{b}} \right] \boxplus [\mathbf{reg}\,(s, T')] \\ &\qquad \boxplus \left[(\varepsilon^*)^{k+1} \, \mathrm{D}^* \, (\mathbf{reg}\,(s, T')) \cdot t^{\mathbf{b}} \right] \end{aligned}$$

and D^* denotes the extension of D by additivity in its first argument, that is to say:

$$\mathrm{D}^* \left(\sum_{i=1}^n \varepsilon^{k_i} s_i^{\mathbf{b}} \right) \cdot t^{\mathbf{b}} := \sum_{i=1}^n \varepsilon^{k_i} \left(s_i^{\mathbf{b}} \ t^{\mathbf{b}} \right)$$

Observe that, whenever S is canonical and $t^{\mathbf{b}}$ is basic, the term $\mathrm{D}^*(S) \cdot t^{\mathbf{b}}$ is also canonical. Therefore, by induction, the regularization $\mathbf{reg}\,(s^{\mathbf{b}}, T)$ is indeed a canonical term term, since canonicity is preserved by $\varepsilon^*, \boxplus$.

- If $\mathbf{can}\,(s) = \sum_{i=1}^{n} \varepsilon^{k_i} s_i^{\mathbf{b}}$ and $\mathbf{can}\,(t) = T$, then

$$\mathbf{can}\,(s\ t) := \left[\sum_{i=1}^{n} \varepsilon^{k_i} \left(s_i^{\mathbf{b}}\ \mathbf{pri}(T)\right)\right] \boxplus \left[\varepsilon^{*} \left(\sum_{i=1}^{n} \mathbf{ap}(\mathbf{reg}\left(s_i^{\mathbf{b}}, \mathbf{tan}(T)\right), \mathbf{pri}(T))\right)\right]$$

where the primal **pri** and tangent **tan** components of a canonical term T correspond respectively to the basic terms with zero and non-zero ε coefficients, and **ap** is the additive extension of application.

$$\begin{array}{rcl@{\qquad}rcl}
\mathbf{pri}\,(0) & := & 0 & \mathbf{tan}\,(0) & := & 0 \\
\mathbf{pri}\left(\varepsilon^{k+1}t^{\mathbf{b}} + T'\right) & := & \mathbf{pri}\,(T') & \mathbf{tan}\left(\varepsilon^{k+1}t^{\mathbf{b}} + T'\right) & := & \varepsilon^{k+1}t^{\mathbf{b}} + \mathbf{tan}\,(T') \\
\mathbf{pri}\left(\varepsilon^{0}t^{\mathbf{b}} + T'\right) & := & t^{\mathbf{b}} + \mathbf{pri}\,(T') & \mathbf{tan}\left(\varepsilon^{0}t^{\mathbf{b}} + T'\right) & := & \mathbf{tan}\,(T')
\end{array}$$

$$\mathbf{ap}\left(\textstyle\sum_{i=1}^{n} \varepsilon^{k_i} s_i^{\mathbf{b}}, t^{\mathbf{b}}\right) \ := \ \textstyle\sum_{i=1}^{n} \varepsilon^{k_i}(s_i^{\mathbf{b}}\ t^{\mathbf{b}})$$

Example 7.1.1. The canonicalization algorithm is mostly straightforward, with only the cases for differential and standard application being of any interest. Since this is the part of the system where we diverge most from the differential λ-calculus, it is worth examining an example of regularization. Consider the following unrestricted term on free variables u, x, y, z:

$$t := \mathrm{D}(u) \cdot (x + y + \varepsilon z)$$

Applying the algorithm above, we compute its canonical form to be:

$$\begin{aligned}
&\mathbf{can}\,(\mathrm{D}(u) \cdot (x + y + \varepsilon z)) \\
&\quad = \mathbf{reg}\,(u, x + y + \varepsilon z) \\
&\quad = (\mathrm{D}(u) \cdot x) \boxplus \mathbf{reg}\,(u, y + \varepsilon z) \boxplus (\varepsilon^{*}\mathrm{D}^{*}(\mathbf{reg}\,(u, y + \varepsilon z)) \cdot x) \\
&\quad = (\mathrm{D}(u) \cdot x) \boxplus \Big[(\mathrm{D}(u) \cdot y) \boxplus \mathbf{reg}\,(u, \varepsilon z) \boxplus \varepsilon^{*}(\mathrm{D}^{*}(\mathbf{reg}\,(u, \varepsilon z)) \cdot y)\Big] \\
&\qquad\quad \boxplus (\varepsilon^{*}\mathrm{D}^{*}(\mathbf{reg}\,(u, y + \varepsilon z)) \cdot x) \\
&\quad = (\mathrm{D}(u) \cdot x) \boxplus \Big[(\mathrm{D}(u) \cdot y) \boxplus (\varepsilon\mathrm{D}(u) \cdot z) \boxplus \varepsilon^{*}(\mathrm{D}^{*}(\varepsilon\mathrm{D}(u) \cdot z) \cdot y))\Big] \\
&\qquad\quad \boxplus (\varepsilon^{*}\mathrm{D}^{*}(\mathbf{reg}\,(u, y + \varepsilon z)) \cdot x) \\
&\quad = (\mathrm{D}(u) \cdot x)) \boxplus \Big[\mathrm{D}(u) \cdot y + \varepsilon(\mathrm{D}(u) \cdot z) + \varepsilon(\mathrm{D}(\mathrm{D}(u) \cdot z) \cdot y)\Big] \\
&\qquad\quad \boxplus (\varepsilon^{*}\mathrm{D}^{*}(\mathbf{reg}\,(u, y + \varepsilon z)) \cdot x) \\
&\quad = \Big[\mathrm{D}(u) \cdot x + \mathrm{D}(u) \cdot y + \varepsilon\,(\mathrm{D}(u) \cdot z) + \varepsilon\,(\mathrm{D}(\mathrm{D}(u) \cdot z) \cdot y))\Big] \\
&\qquad\quad \boxplus (\varepsilon\,(\mathrm{D}(\mathrm{D}(u) \cdot y) \cdot x + \varepsilon\,(\mathrm{D}(\mathrm{D}(u) \cdot z) \cdot x + \varepsilon\,(\mathrm{D}(\mathrm{D}(\mathrm{D}(u) \cdot z) \cdot y) \cdot x)))) \\
&\quad = \mathrm{D}(u) \cdot y + \mathrm{D}(u) \cdot x \\
&\qquad\quad + \varepsilon\Big[\mathrm{D}(u) \cdot z + \mathrm{D}(\mathrm{D}(u) \cdot y) \cdot x \\
&\qquad\qquad\qquad + \varepsilon(\mathrm{D}(\mathrm{D}(u) \cdot z) \cdot y + \mathrm{D}(\mathrm{D}(u) \cdot z) \cdot x + \varepsilon\,(\mathrm{D}(\mathrm{D}(\mathrm{D}(u) \cdot z) \cdot y) \cdot x))\Big]
\end{aligned}$$

This is precisely the result we would expect from fully unfolding the expression $\partial(u, x + y + \varepsilon(z))$ in a Cartesian difference category and repeatedly applying regularity of the derivative!

Theorem 7.1.1. Every unrestricted λ_ε-term is differentially equivalent to its canonical form. That is to say, for all $t \in \Lambda_\varepsilon$, we have $t \sim_\varepsilon \mathbf{can}\,(t)$.

Proof. The proof proceeds by straightforward induction on t. We explicitly prove the case for the canonicalization of differential and standard application, as these are the only nontrivial cases. For this we will make use of the following results:

Lemma 7.1.3. Given canonical terms S, T, we have:

$$\begin{aligned}
\varepsilon^* S &\sim_\varepsilon \varepsilon S \\
S \boxplus T &\sim_\varepsilon S + T \\
\mathrm{D}^*(S) \cdot T &\sim_\varepsilon \mathrm{D}(S) \cdot T \\
\mathbf{ap}\,(S, T) &\sim_\varepsilon S\ T
\end{aligned}$$

Proof. All four results follow by straightforward structural induction on S. □

Lemma 7.1.4. Given a basic term $s^{\mathbf{b}}$ and a canonical term T the differential application $\mathrm{D}(s^{\mathbf{b}}) \cdot T$ is differentially equivalent to its regularized version $\mathbf{reg}\left(s^{\mathbf{b}}, T\right)$.

Proof. The proof follows by induction on the structure of T.

- When $T = 0$ we have $\mathbf{reg}\left(s^{\mathbf{b}}, 0\right) = 0 \sim_\varepsilon \mathrm{D}(s^*) \cdot 0$
- When $T = \varepsilon^k t^{\mathbf{b}} + T'$ we have:

$$\begin{aligned}
\mathbf{reg}\left(s^{\mathbf{b}}, \varepsilon^k t^{\mathbf{b}} + T'\right) &= \left[(\varepsilon^*)^k \mathrm{D}(s^{\mathbf{b}}) \cdot t^{\mathbf{b}}\right] \boxplus \left[\mathbf{reg}\left(s^{\mathbf{b}}, T'\right)\right] \\
&\quad \boxplus \left[(\varepsilon^*)^{k+1} \mathrm{D}^*(\mathbf{reg}\left(s^{\mathbf{b}}, T'\right)) \cdot t^{\mathbf{b}}\right] \\
&\sim_\varepsilon \varepsilon^k \mathrm{D}(s^{\mathbf{b}}) \cdot t^{\mathbf{b}} + \mathrm{D}(s^{\mathbf{b}}) \cdot T' + \varepsilon^{k+1} \mathrm{D}(\mathrm{D}(s^{\mathbf{b}}) \cdot T') \cdot t^{\mathbf{b}}
\end{aligned}$$ □

Going back to the proof of Theorem 7.1.1, the case for differential application is obtained as a straightforward corollary of Lemma 7.1.4. For conventional application, consider terms s, t, and note that if $\mathbf{can}\,(t) = T$ then $t \sim_\varepsilon \mathbf{pri}(T) + \varepsilon\mathbf{tan}(T)$. Then, for any basic term $s^{\mathbf{b}}$, we obtain:

$$\begin{aligned}
\mathbf{can}\left(s^{\mathbf{b}}\ t\right) &= \left(s^{\mathbf{b}}\ \mathbf{pri}(T)\right) \boxplus \left[\varepsilon^* \mathbf{ap}\left(\mathbf{reg}\left(s^{\mathbf{b}}, \mathbf{tan}(T)\right), \mathbf{pri}(T)\right)\right] \\
&\sim_\varepsilon \left(s^{\mathbf{b}}\ \mathbf{pri}(T)\right) + \varepsilon\left[(\mathrm{D}(s^{\mathbf{b}}) \cdot \mathbf{tan}(T))\ \mathbf{pri}(T)\right] \\
&\sim_\varepsilon s^{\mathbf{b}}\ [\mathbf{pri}(T) + \varepsilon\mathbf{tan}(T)] \\
&\sim_\varepsilon s^{\mathbf{b}}\ t
\end{aligned}$$ □

Our canonicalization algorithm is a result of orienting the equations in Figure 7.2. Note, however, that while most of these equivalences have a “natural” orientation to them, two of them are entirely symmetrical: those being commutativity of the sum and the derivative. Barring the imposition of some arbitrary total ordering on terms which would allow us to prefer the term $x+y$ over $y+x$ (or vice versa), we settle for our canonical forms to be unique “up to” these commutativity conditions.

Definition 7.1.9. Permutative equivalence $\sim_+ \ \subseteq \Lambda_\varepsilon \times \Lambda_\varepsilon$ is the least equivalence relation which is contextual and satisfies the properties in Figure 7.3 below.

$$\begin{aligned} s + (t + e) &\sim_+ (s + t) + e \\ s + t &\sim_+ t + s \\ \mathrm{D}(\mathrm{D}(s) \cdot t) \cdot e &\sim_+ \mathrm{D}(\mathrm{D}(s) \cdot e) \cdot t \end{aligned}$$

Figure 7.3: Permutative equivalence on unrestricted Λ_ε-terms

We need to include associativity in the definition of permutative equality, as otherwise the canonical term $x + (y + z)$ would not be permutatively equivalent to $y + (x + z)$.

Theorem 7.1.2. Given unrestricted terms $s, t \in \Lambda_\varepsilon$, they are differentially equivalent if and only if their canonical forms are permutatively equivalent. More succinctly, $s \sim_\varepsilon t$ if and only if $\mathbf{can}\,(s) \sim_+ \mathbf{can}\,(t)$

Proof. As an immediate consequence of Theorem 7.1.1, we know that $s \sim_\varepsilon t$ if and only if $\mathbf{can}\,(s) \sim_\varepsilon \mathbf{can}\,(t)$. The desired result will then follow from the fact that differential equivalence is precisely equivalent to permutative equivalence on canonical terms.

Before proving this, we remark that $\sim_{\mathbf{can}}$ is reflexive, transitive and symmetric, which follows immediately from its definition and reflexivity, transitivity and symmetry of $\sim_+$. We also prove the following two helpful results:

Lemma 7.1.5. For any unrestricted terms s, t, e, the following equalities hold:

$$\begin{aligned} \mathbf{can}\,(\mathbf{can}\,(s) + \mathbf{can}\,(t)) &= \mathbf{can}\,(s + t) \\ \mathbf{can}\,(\varepsilon \mathbf{can}\,(x)) &= \mathbf{can}\,(\varepsilon(x)) \\ \mathbf{can}\,(\mathbf{can}\,(s)\ \ \mathbf{can}\,(t)) &= \mathbf{can}\,(s\ t) \\ \mathbf{can}\,(\mathrm{D}(\mathbf{can}\,(s)) \cdot \mathbf{can}\,(t)) &= \mathbf{can}\,(\mathrm{D}(s) \cdot t) \end{aligned}$$

As a consequence, the relation $\sim_{\mathbf{can}}$ is contextual.

Proof. Follows immediately from the definition of $\mathbf{can}()$ and observing that it only depends on the canonicalization of the subterms of the outermost syntactic form. $\square$

Lemma 7.1.6. Whenever S, T are canonical terms, $S \sim_\varepsilon T$ if and only if $S \sim_+ T$.

Proof. Since $\sim_\varepsilon$ is the least reflexive, transitive, symmetric and contextual relation that verifies the conditions in Figure 7.2, it follows that whenever $S \sim_\varepsilon T$ it must be the case that $S = S_1 \sim_\varepsilon^\star S_2 \sim_\varepsilon^\star \ldots \sim_\varepsilon^\star S_n = T$, where $\sim_\varepsilon^\star$ denotes the contextual, symmetric (but not transitive) closure of $\sim_\varepsilon^1$.

But for each such condition, we have $\mathbf{can}(\text{LHS}) = \mathbf{can}(\text{RHS})$, except for the two commutativity conditions, and in all cases we have $\mathbf{can}(\text{LHS}) \sim_+ \mathbf{can}(\text{RHS})$. Since $\sim_{\mathbf{can}}$ is contextual, we have that $S = S_1 \sim_{\mathbf{can}} S_2 \sim_{\mathbf{can}} \ldots \sim_{\mathbf{can}} S_n = T$, and thus by transitivity we obtain $S \sim_{\mathbf{can}} T$. $\square$

Theorem 7.1.2 is then an immediate consequence of Lemma 7.1.6 and Theorem 7.1.1. $\square$

Corollary 7.1.1. Differential equivalence of Λ_ε-terms is decidable.

Proof. Permutative equivalence of two terms is decidable, since the set of $\sim_+$-equivalence classes of any term is finite and can be enumerated easily. Canonicalization is also decidable, since it is defined as a clearly well-founded recursion. $\square$

Corollary 7.1.2. The set λ_ε of well-formed terms corresponds precisely to the set of canonical terms up to permutative equivalence $\mathrm{C}_\varepsilon / \sim_+$.

7.1.2 Substitution

As is usual, our calculus features two different kinds of application: standard function application, represented as $(s\ t)$; and differential application, represented as $\mathrm{D}(s) \cdot t$. These two give rise to two different notions of substitution. The first is, of course, the usual capture-avoiding substitution. The second, differential substitution, is similar to the equivalent notion in the differential λ-calculus, as it arises from the same chain rule that is verified in both Cartesian differential categories and change action models.

Definition 7.1.10. Given terms $s, t \in \Lambda_\varepsilon$ and a variable x, the **capture-avoiding substitution** of s for x in t (which we write as $t\,[s/x]$) is defined by induction on the structure of t as in Figure 7.4 below.

$$\begin{array}{rcll}
x\,[s/x] & := & s & \\
y\,[s/x] & := & y & \text{if } x \neq y \\
(\lambda y.t)\,[s/x] & := & \lambda y.(t\,[s/x]) & \text{if } y \notin \mathrm{FV}(s) \\
(t\ e)\,[s/x] & := & (t\,[s/x])(e\,[s/x]) & \\
(\mathrm{D}(t)\cdot e)\,[s/x] & := & \mathrm{D}(t\,[s/x])\cdot(e\,[s/x]) & \\
(\varepsilon t)\,[s/x] & := & \varepsilon(t\,[s/x]) & \\
(t+e)\,[s/x] & := & (t\,[s/x]) + (e\,[s/x]) & \\
0\,[s/x] & := & 0 &
\end{array}$$

Figure 7.4: Capture-avoiding substitution in λ_ε

Proposition 7.1.1. Capture-avoiding substitution respects differential equivalence. That is to say, whenever $s \sim_\varepsilon s'$ and $t \sim_\varepsilon t'$, it is the case that $t\,[s/x] \sim_\varepsilon t'\,[s'/x]$.

Proof. It suffices to show that $t\,[s/x] \sim_\varepsilon^1 t'\,[s/x]$ (the full result will then follow from transitivity and contextuality of $\sim_\varepsilon$), which can be proven by straightforward structural induction on t. $\square$

Definition 7.1.11. Given terms $s, t \in \Lambda_\varepsilon$ and a variable x *which is not free in* s[1], the **differential substitution** of s for x in t, which we write as $\frac{\partial t}{\partial x}(s)$, is defined by induction on the structure of t as in Figure 7.5. We write

$$\frac{\partial^k t}{\partial(x_1, \ldots, x_k)}(u_1, \ldots, u_k)$$

to denote the sequence of nested differential substitutions

$$(\partial(\ldots((\partial t/\partial x_1)(u_1))\ldots)/\partial x_k)(u_k)$$

Most of the cases of differential substitution are identical to those in the differential λ-calculus (compare the above with Figure 2.1). There are, however, a number of notable differences which stem from our more general setting. First, we must point out that this definition in fact coincides exactly with the original notion of differential substitution in e.g [37], provided that one assumes the identity $\varepsilon t = 0$ for all terms. This reflects the fact that every Cartesian differential category is in fact a Cartesian difference category with trivial infinitesimal extension.

All the differences in this definition stem from the failure of derivatives to be additive in the setting of Cartesian difference categories. Consider the case for $\frac{\partial \mathrm{D}(t)\cdot s}{\partial x}(e)$,

[1]One should emphasise this constraint. Differential substitution appears in the reduction of $\mathrm{D}(\lambda x.t)\cdot s$ into $\lambda x.\frac{\partial t}{\partial x}(s)$, and so if x were free in s it would become bound by the enclosing λ-abstraction.

$$\begin{array}{rcll}
\frac{\partial x}{\partial x}(s) & := & s & \\
\frac{\partial y}{\partial x}(s) & := & 0 & \text{if } x \neq y \\
\frac{\partial(\lambda y.t)}{\partial x}(s) & := & \lambda y.\left(\frac{\partial t}{\partial x}(s)\right) & \text{if } y \notin \mathrm{FV}(t) \\
\frac{\partial(t\ e)}{\partial x}(s) & := & \left[\mathrm{D}(t)\cdot\left(\frac{\partial e}{\partial x}(s)\right)\ e\right] + \left[\frac{\partial t}{\partial x}(s)\ \left(e\left[(x+\varepsilon s)/x\right]\right)\right] & \\
\frac{\partial(\mathrm{D}(t)\cdot e)}{\partial x}(s) & := & \mathrm{D}(t)\cdot\left(\frac{\partial e}{\partial x}(s)\right) + \mathrm{D}\left(\frac{\partial t}{\partial x}(s)\right)\cdot\left(e\left[(x+\varepsilon s)/x\right]\right) & \\
& & + \varepsilon\mathrm{D}(\mathrm{D}(t)\cdot e)\cdot\left(\frac{\partial e}{\partial x}(s)\right) & \\
\frac{\partial(\varepsilon t)}{\partial x}(s) & := & \varepsilon\left(\frac{\partial t}{\partial x}(s)\right) & \\
\frac{\partial(t+e)}{\partial x}(s) & := & \left(\frac{\partial t}{\partial x}(s)\right) + \left(\frac{\partial e}{\partial x}(s)\right) & \\
\frac{\partial 0}{\partial x}(s) & := & 0 &
\end{array}$$

Figure 7.5: Differential substitution in λ_ε

and remember that the "essence" of a derivative in our setting lies in the derivative condition, that is to say, if $t(x)$ is a term with a free variable x, we seek our notion of differential substitution to satisfy a condition akin to Taylor's formula:

$$t(x+\varepsilon y) \sim_\varepsilon t(x) + \varepsilon\frac{\partial t}{\partial x}(y)$$

When the term t is a differential application, and assuming the above "Taylor's formula" holds for all of its subterms (which we will show later), this leads us to the following informal argument:

$$\begin{aligned}
\mathrm{D}(t(x+\varepsilon y))\cdot(s(x+\varepsilon y)) &\sim_\varepsilon \mathrm{D}\left(t(x)+\varepsilon\frac{\partial t}{\partial x}(y)\right)\cdot(s(x+\varepsilon y)) \\
&\sim_\varepsilon \mathrm{D}(t(x))\cdot(s(x+\varepsilon y)) + \varepsilon\mathrm{D}\left(\frac{\partial t}{\partial x}(y)\right)\cdot(s(x+\varepsilon y)) \\
&\sim_\varepsilon \mathrm{D}(t(x))\cdot(s(x)) \\
&\qquad + \varepsilon\mathrm{D}(t(x))\cdot\left(\frac{\partial s}{\partial x}(y)\right) \\
&\qquad + \varepsilon\mathrm{D}\left(\frac{\partial t}{\partial x}(y)\right)\cdot(s(x+\varepsilon y)) \\
&\qquad + \varepsilon^2\mathrm{D}(\mathrm{D}(t(x))\cdot(s(x)))\cdot\left(\frac{\partial s}{\partial x}(y)\right)
\end{aligned}$$

From this calculation, the differential substitution for this case arises naturally as it results from factoring out the ε and noticing that the resulting expression has precisely the correct shape to be Taylor's formula for the case of differential application. The case for standard application can be derived similarly, although the involved terms are simpler. Differential substitution verifies some useful properties, which we state below (mechanised proofs are available, although the details are more cumbersome than enlightening).

Proposition 7.1.2. Differential substitution respects differential equivalence. That is to say, whenever $s \sim_\varepsilon s'$ and $t \sim_\varepsilon t'$, it is the case that $\frac{\partial t}{\partial x}(s) \sim_\varepsilon \frac{\partial t'}{\partial x}(s')$.

Proof. See `Lemma dsubst_diff`. □

Proposition 7.1.3. Whenever x is not free in t, then $\frac{\partial t}{\partial x}(u) \sim_\varepsilon 0$.

Proof. See `Lemma dsubst_empty`. □

Proposition 7.1.4. Whenever x is not free in u, v, then:

$$\frac{\partial^2 t}{\partial x^2}(u, v) \sim_\varepsilon \frac{\partial^2 t}{\partial x^2}(v, u)$$

Proof. See `Lemma dsubst_commute` □

As we have previously mentioned, the rationale behind our specific definition of differential substitution is that it should verify some sort of "Taylor's formula" (or rather, Kock-Lawvere formula), in the following sense:

Theorem 7.1.3. For any unrestricted terms s, t, e and any variable x which does not appear free in e, we have

$$s\left[t + \varepsilon e / x\right] \sim_\varepsilon s\left[t/x\right] + \varepsilon\left(\left(\frac{\partial s}{\partial x}(e)\right)\left[t/x\right]\right)$$

We will often refer to the right-hand side of the above equivalence as the **Taylor expansion** of the corresponding term in the left-hand side.

Proof. The proof follows by induction on s and some involved calculations. We also provide a mechanised version in `Theorem Taylor`.

- When $s = x$ we have $\frac{\partial x}{\partial x}(e) = e$ and so:

$$x\left[t + \varepsilon e/x\right] = t + \varepsilon e = x\left[t/x\right] + \varepsilon\left(\frac{\partial x}{\partial x}(e)\right)\left[t/x\right]$$

- When $s = y \neq x$ we have $\frac{\partial y}{\partial x}(e) = 0$ and so:

$$y\left[t + \varepsilon e/x\right] = y \sim_\varepsilon y + 0 = y\left[y/x\right] + \varepsilon\left(\frac{\partial y}{\partial x}(e)\right)\left[t/x\right]$$

- When $s = 0$ we have LHS $\sim_\varepsilon 0 \sim_\varepsilon$ RHS.
- When $s = s' + s''$ we have:

$$\begin{aligned}
(s' + s'')\,[t + \varepsilon e/x] &= s'\,[t + \varepsilon e/x] + s''\,[t + \varepsilon e/x] \\
&\sim_\varepsilon \left(s'\,[t/x] + \varepsilon\left(\left(\frac{\partial s'}{\partial x}\,(e)\right)[t/x]\right)\right) + \left(s''\,[t/x] + \varepsilon\left(\left(\frac{\partial s''}{\partial x}\,(e)\right)[t/x]\right)\right) \\
&\sim_\varepsilon \left(s'\,[t/x] + s''\,[t/x]\right) + \left(\left(\frac{\partial s'}{\partial x}\,(e)\right)[t/x] + \left(\frac{\partial s''}{\partial x}\,(e)\right)[t/x]\right) \\
&= (s' + s'')\,[t/x] + \left(\frac{\partial (s' + s'')}{\partial x}\,(e)\right)[t/x]
\end{aligned}$$

- When $s = \varepsilon s'$ we have:

$$\begin{aligned}
(\varepsilon s')\,[t + \varepsilon e/x] &= \varepsilon s'\,[t + \varepsilon e/x] \\
&\sim_\varepsilon \varepsilon\left(s'\,[t/x] + \varepsilon\left(\left(\frac{\partial s'}{\partial x}\,(e)\right)[t/x]\right)\right) \\
&\sim_\varepsilon \varepsilon\left(s'\,[t/x]\right) + \varepsilon\left(\varepsilon\left(\left(\frac{\partial s'}{\partial x}\,(e)\right)[t/x]\right)\right) \\
&\sim_\varepsilon (\varepsilon s')\,[t/x] + \varepsilon\left(\left(\frac{\partial (\varepsilon s')}{\partial x}\,(e)\right)[t/x]\right)
\end{aligned}$$

- When $s = s'\ s''$ we have:

$$\begin{aligned}
(s'\ s'')\,[t + \varepsilon e/x] &= (s'\,[t + \varepsilon e/x])\ (s''\,[t + \varepsilon e/x]) \\
&\sim_\varepsilon \left[s'\,[t/x] + \varepsilon\left(\left(\frac{\partial s'}{\partial x}\,(e)\right)[t/x]\right)\right]\ \left[s''\,[t/x] + \varepsilon\left(\left(\frac{\partial s''}{\partial x}\,(e)\right)[t/x]\right)\right] \\
&\sim_\varepsilon \left[s'\,[t/x]\ \left(s''\,[t/x] + \varepsilon\left(\left(\frac{\partial s''}{\partial x}\,(e)\right)[t/x]\right)\right)\right] \\
&\qquad + \varepsilon\left[\left(\left(\frac{\partial s'}{\partial x}\,(e)\right)[t/x]\right)\ \left(s''\,[t/x] + \varepsilon\left(\left(\frac{\partial s''}{\partial x}\,(e)\right)[t/x]\right)\right)\right] \\
&\sim_\varepsilon \left(s'\,[t/x]\ \left(s''\,[t/x]\right)\right) \\
&\qquad + \varepsilon\left[\left(\mathrm{D}(s'\,[t/x]) \cdot \left(\left(\frac{\partial s''}{\partial x}\,(e)\right)[t/x]\right)\right)\ \left(s''\,[t/x]\right)\right] \\
&\qquad + \varepsilon\left[\left(\left(\frac{\partial s'}{\partial x}\,(e)\right)[t/x]\right)\ \left(s''\,[t + \varepsilon e/x]\right)\right] \\
&= (s'\ s'')\,[t/x] \\
&\qquad + \varepsilon\left[\left(\mathrm{D}(s') \cdot \left(\frac{\partial s''}{\partial x}\,(e)\right)\right)\ s''\right][t/x] \\
&\qquad + \varepsilon\left[\left(\frac{\partial s'}{\partial x}\,(e)\right)\ \left(s''\,[(x + \varepsilon e)/x]\right)\right][t/x] \\
&\sim_\varepsilon (s'\ s'')\,[t/x] + \varepsilon\left(\left(\frac{\partial (s' + s'')}{\partial x}\,(e)\right)[t/x]\right)
\end{aligned}$$

- When $s = \mathrm{D}(s') \cdot s''$ we have:

$$\begin{aligned}
(\mathrm{D}(s') \cdot s'')\,[t + \varepsilon e/x] &= \mathrm{D}(s'\,[t+\varepsilon e/x]) \cdot (s''\,[t+\varepsilon e/x]) \\
&\sim_\varepsilon \mathrm{D}\left(s'\,[t/x] + \varepsilon\left(\left(\frac{\partial s'}{\partial x}(e)\right)[t/x]\right)\right) \cdot (s''\,[t+\varepsilon e/x]) \\
&\sim_\varepsilon \mathrm{D}(s'\,[t/x]) \cdot (s''\,[t+\varepsilon e/x]) + \varepsilon\left(\mathrm{D}\left(\left(\frac{\partial s'}{\partial x}(e)\right)[t/x]\right) \cdot (s''\,[t+\varepsilon e/x])\right) \\
&\sim_\varepsilon \mathrm{D}(s'\,[t/x]) \cdot \left(s''\,[t/x] + \varepsilon\left(\frac{\partial s''}{\partial x}(e)\right)[t/x]\right) \\
&\qquad + \varepsilon\left(\mathrm{D}\left(\left(\frac{\partial s'}{\partial x}(e)\right)[t/x]\right) \cdot (s''\,[t+\varepsilon e/x])\right) \\
&\sim_\varepsilon \mathrm{D}(s'\,[t/x]) \cdot (s''\,[t/x]) + \varepsilon\left(\mathrm{D}(s'\,[t/x]) \cdot \left(\left(\frac{\partial s''}{\partial x}(e)\right)[t/x]\right)\right) \\
&\qquad + \varepsilon^2\left(\mathrm{D}(\mathrm{D}(s'\,[t/x]) \cdot (s''\,[t/x])) \cdot \left(\left(\frac{\partial s''}{\partial x}(e)\right)[t/x]\right)\right) \\
&\qquad + \varepsilon\left(\mathrm{D}\left(\left(\frac{\partial s'}{\partial x}(e)\right)[t/x]\right) \cdot (s''\,[t+\varepsilon e/x])\right) \\
&\sim_\varepsilon (\mathrm{D}(s') \cdot s'')\,[t/x] + \varepsilon\left(\mathrm{D}(s') \cdot \left(\frac{\partial s''}{\partial x}(e)\right)\right)[t/x] \\
&\qquad + \varepsilon^2\left(\mathrm{D}(\mathrm{D}(s') \cdot s'') \cdot \left(\frac{\partial s''}{\partial x}(e)\right)\right)[t/x] \\
&\qquad + \varepsilon\left(\mathrm{D}\left(\frac{\partial s'}{\partial x}(e)\right) \cdot (s''\,[x+\varepsilon e/x])\right)[t/x] \\
&\sim_\varepsilon (\mathrm{D}(s') \cdot s'')\,[t/x] + \varepsilon\left(\left(\frac{\partial(\mathrm{D}(s') \cdot s'')}{\partial x}(e)\right)[t/x]\right)
\end{aligned}$$

□

The following lemmas relate differential substitution and standard substitution, and will be of much use later.

Lemma 7.1.7. Whenever x, y are (distinct) variables then for any unrestricted terms t, u, v where x is not free in v we have:

$$\left(\frac{\partial t}{\partial x}(u)\right)[v/y] = \frac{\partial t\,[v/y]}{\partial x}(u\,[v/y])$$

Proof. See `Lemma replace_dsubst`. □

Lemma 7.1.8. Whenever x, y are (distinct) variables, with y not free in either u, v, we have:

$$\frac{\partial t\,[v/y]}{\partial x}(u) \sim_\varepsilon \left(\frac{\partial t}{\partial x}(u)\right)[(v\,[x+\varepsilon u/x])/y] + \left(\frac{\partial t}{\partial y}\left(\frac{\partial v}{\partial x}(u)\right)\right)[v/y]$$

One consequence of this “syntactic Taylor’s formula” is that derivatives in the difference λ-calculus can be computed by a sort of quasi-AD algorithm: given an expression of the form $\lambda x.t$, its derivative at point s along u can be computed by reducing the differential application $(\mathrm{D}(\lambda x.t) \cdot (u))\ s$ which, as we shall see later, reduces (by definition) to $\left(\frac{\partial t}{\partial x}(u)\right)[s/x]$. Alternatively, we can simply evaluate $(\lambda x.t)\ (s + \varepsilon(u))$ to compute $t\,[s + \varepsilon(u)/x]$ which, by Theorem 7.1.3 and Lemma 7.1.7, is equivalent to $t\,[s/x] + \varepsilon(\frac{\partial t}{\partial x}(u)\,[s/x])$. In an appropriate setting (i.e. one where subtraction of terms is allowed and ε admits an inverse) the derivative can then be extracted from this result by extracting the term under the ε. This process is remarkably similar to forward-mode automatic differentiation, where derivatives are computed by adding "perturbations" to the program input.

Theorem 7.1.3 also allows us to unpack all of the substitutions in the definition of differential substitution. For example, the term $\frac{\partial (t\ e)}{\partial x}(u)$ can be expanded to:

$$\begin{aligned}\frac{\partial(t\ e)}{\partial x}(u) &= \left[\mathrm{D}(t)\cdot\left(\frac{\partial e}{\partial x}(u)\right)\ e\right] + \left[\frac{\partial t}{\partial x}(u)\ \ (e\,[(x+\varepsilon u)/x])\right] \\ &\sim_\varepsilon \left[\mathrm{D}(t)\cdot\left(\frac{\partial e}{\partial x}(u)\right)\ e\right] + \left[\frac{\partial t}{\partial x}(u)\ \ \left(e + \varepsilon\frac{\partial e}{\partial x}(u)\right)\right] \\ &\sim_\varepsilon \left[\mathrm{D}(t)\cdot\left(\frac{\partial e}{\partial x}(u)\right)\ e\right] + \left[\frac{\partial t}{\partial x}(u)\ e\right] + \varepsilon\left[\mathrm{D}\left(\frac{\partial t}{\partial x}(u)\right)\cdot\frac{\partial e}{\partial x}(u)\ e\right]\end{aligned}$$

We can generalise this procedure to arbitrary sequences of differential substitutions, although the terms involved are too complex to give a simple account.

Lemma 7.1.9. For any basic terms $s^{\mathbf{b}}, t^{\mathbf{b}}$, variables $x_1, \ldots, x_n$ and basic terms $u_1^{\mathbf{b}}, \ldots, u_n^{\mathbf{b}}$ such that none of the x_i appear free in the u_i, the differential substitution

$$\frac{\partial^k(\mathrm{D}(s^{\mathbf{b}})\cdot t^{\mathbf{b}})}{\partial(x_1,\ldots,x_n)}(u_1^{\mathbf{b}},\ldots,u_n^{\mathbf{b}})$$

is differentially equivalent to a sum of terms of the form

$$\varepsilon^z \mathrm{D}^l(v)\cdot(w_1,\ldots,w_l)$$

where v is of the form

$$\frac{\partial^p t^{\mathbf{b}}}{\partial(x_1^{(t)},\ldots,x_p^{(t)})}(u_1^{(t)},\ldots,u_p^{(t)})$$

and every w_j is of the form

$$\frac{\partial^{q_j} e^{\mathbf{b}}}{\partial(x_1^{(w_j)},\ldots,x_{q_j}^{(w_j)})}(u_1^{(w_j)},\ldots,u_{q_j}^{(w_j)})$$

where $1 \leq l \leq 2^n$ and each pair of sequences $x_i^{(t)}, u_i^{(t)}$ corresponds to a reordering of some subsequence of the $x_i, u_i^{\mathbf{b}}$.

Proof. Straightforward induction on k and applying Theorem 7.1.3. $\square$

Lemma 7.1.10. For any basic term $s^{\mathbf{b}}$ and additive term t^*, variables $x_1, \ldots, x_n$ and basic terms $u_1^{\mathbf{b}}, \ldots, u_n^{\mathbf{b}}$ such that none of the x_i appear free in the $u_i^{\mathbf{b}}$, the differential substitution

$$\frac{\partial^k (s^{\mathbf{b}}\ t^*)}{\partial(x_1, \ldots, x_n)}(u_1^{\mathbf{b}}, \ldots, u_n^{\mathbf{b}})$$

is differentially equivalent to a sum of terms of the form

$$\varepsilon^z \mathrm{D}^l(v) \cdot (w_1, \ldots, w_l)$$

where v is of the form

$$\frac{\partial^p t^{\mathbf{b}}}{\partial(x_1^{(t)}, \ldots, x_p^{(t)})}(u_1^{(t)}, \ldots, u_p^{(t)})$$

and every w_j is of the form

$$\frac{\partial^{q_j} e^{\mathbf{b}}}{\partial(x_1^{(w_j)}, \ldots, x_{q_j}^{(w_j)})}(u_1^{(w_j)}, \ldots, u_{q_j}^{(w_j)})$$

where $1 \leq l \leq 2^n$ and each pair of sequences $x_i^{(t)}, u_i^{(t)}$ corresponds to a reordering of some subsequence of the $x_i, u_i^{\mathbf{b}}$.

The above results may seem overly weak and arcane, but at its core they make a very simple statement: if one applies any number of differential substitutions to the term $\mathrm{D}(s) \cdot t$ (or $s\ t$) and "cranks the lever", pushing the substitutions as far down the term as possible, then all the differential substitutions in the resulting term are applied to either s or t, and their arguments are a reordering of some subsequence of the arguments to the initial differential substitution.

Theorem 7.1.4. Differential substitution is regular, that is, for any unrestricted terms s, u, v where x does not appear free in either u or v, we have:

$$\frac{\partial s}{\partial x}(0) \sim_\varepsilon 0$$
$$\frac{\partial s}{\partial x}(u + v) \sim_\varepsilon \frac{\partial s}{\partial x}(u) + \left(\frac{\partial s}{\partial x}(v)\right)[x + \varepsilon u/x]$$

Proof. Both properties follow by induction on s. As the proof involves immense amounts of tedious calculations, we refer the reader to `Theorem Regularity` calculations, We omit the details for the first one, and show only the non-trivial inductive cases for the second.

- The case $s = \mathrm{D}(t) \cdot e$ is rather mechanically involved. We first expand the right-hand side of the desired result and group the expansion into pieces. We will later expand the left-hand side and reconstruct each piece from this expansion. We use colourful notation to help the reader follow the structure of the proof: terms arising from expanding right-hand side of the equivalence will be coloured, whereas terms arising from expanding the left-hand side will be highlighted. Whenever a left-hand term and a right-hand term match each other exactly, we will use the same colour for both and consider them henceforth "discharged".

$$\begin{aligned}
&\frac{\partial s}{\partial x}(u) + \left(\frac{\partial s}{\partial x}(v)\right)[x + \varepsilon u/x] \\
&= \frac{\partial(\mathrm{D}(t)\cdot e)}{\partial x}(u) + \left(\frac{\partial(\mathrm{D}(t)\cdot e)}{\partial x}(v)\right)[x + \varepsilon u/x] \\
&= \left[\mathrm{D}(t)\cdot\left(\frac{\partial e}{\partial x}(u)\right) + \mathrm{D}\left(\frac{\partial t}{\partial x}(u)\right)\cdot(e\,[x+\varepsilon u/x]) + \varepsilon \mathrm{D}(\mathrm{D}(t)\cdot e)\cdot\left(\frac{\partial e}{\partial x}(u)\right)\right] \\
&+ \left[\mathrm{D}(t)\cdot\left(\frac{\partial e}{\partial x}(v)\right) + \mathrm{D}\left(\frac{\partial t}{\partial x}(v)\right)\cdot(e\,[x+\varepsilon v/x]) + \varepsilon \mathrm{D}(\mathrm{D}(t)\cdot e)\cdot\left(\frac{\partial e}{\partial x}(v)\right)\right][x+\varepsilon u/x] \\
&\sim_\varepsilon \left[\mathrm{D}(t)\cdot\left(\frac{\partial e}{\partial x}(u)\right) + \left(\mathrm{D}(t)\cdot\left(\frac{\partial e}{\partial x}(v)\right)\right)[x+\varepsilon u/x]\right] \\
&+ \left[\mathrm{D}\left(\frac{\partial t}{\partial x}(u)\right)\cdot(e\,[x+\varepsilon u/x]) + \left(\mathrm{D}\left(\frac{\partial t}{\partial x}(v)\right)\cdot(e\,[x+\varepsilon v/x])\right)[x+\varepsilon u/x]\right] \\
&+ \varepsilon\left[\mathrm{D}(\mathrm{D}(t)\cdot e)\cdot\left(\frac{\partial e}{\partial x}(u)\right) + \left(\mathrm{D}(\mathrm{D}(t)\cdot e)\cdot\left(\frac{\partial e}{\partial x}(v)\right)\right)[x+\varepsilon u/x]\right]
\end{aligned}$$

Before we continue, we expand the rose summand above by applying Theorem 7.1.3, as we will be cancelling these terms later:

$$\begin{aligned}
&\left[\mathrm{D}(t)\cdot\left(\frac{\partial e}{\partial x}(u)\right) + \left(\mathrm{D}(t)\cdot\left(\frac{\partial e}{\partial x}(v)\right)\right)[x+\varepsilon u/x]\right] \\
&\sim_\varepsilon \left[\mathrm{D}(t)\cdot\left(\frac{\partial e}{\partial x}(u)\right) + \mathrm{D}(t)\cdot\left(\frac{\partial e}{\partial x}(v)\right)\right] + \varepsilon\frac{\partial\big(\mathrm{D}(t)\cdot(\frac{\partial e}{\partial x}(v))\big)}{\partial x}(u) \\
&\sim_\varepsilon \left[\mathrm{D}(t)\cdot\left(\frac{\partial e}{\partial x}(u)\right) + \mathrm{D}(t)\cdot\left(\frac{\partial e}{\partial x}(v)\right)\right] \\
&\qquad + \varepsilon\left[\mathrm{D}(t)\cdot\left(\frac{\partial^2 e}{\partial x^2}(v,u)\right)\right] \\
&\qquad + \varepsilon\left[\mathrm{D}\left(\frac{\partial t}{\partial x}(u)\right)\cdot\left(\frac{\partial(e\,[x+\varepsilon u/x])}{\partial x}(v)\right)\right] \\
&\qquad + \varepsilon\left[\varepsilon \mathrm{D}\left(\mathrm{D}(t)\cdot\left(\frac{\partial e}{\partial x}(v)\right)\right)\right]
\end{aligned}$$

We now expand the left-hand side.

$$\frac{\partial s}{\partial x}(u+v) = \frac{\partial \mathrm{D}(t)\cdot e}{\partial x}(u+v)$$

$$
\begin{aligned}
&= \mathrm{D}(t) \cdot \left(\frac{\partial e}{\partial x}(u+v)\right) + \mathrm{D}\left(\frac{\partial t}{\partial x}(u+v)\right) \cdot \left(e\left[(x+\varepsilon(u+v))/x\right]\right) \\
&+ \varepsilon \mathrm{D}(\mathrm{D}(t) \cdot e) \cdot \left(\frac{\partial e}{\partial x}(u+v)\right) \\
&\sim_\varepsilon \left[\mathrm{D}(t) \cdot \left(\frac{\partial e}{\partial x}(u) + \left(\frac{\partial e}{\partial x}(v)\right)\left[(x+\varepsilon u)/x\right]\right)\right] \\
&+ \left[\mathrm{D}\left(\frac{\partial t}{\partial x}(u) + \left(\frac{\partial t}{\partial x}(v)\right)\left[(x+\varepsilon u)/x\right]\right) \cdot \left(e\left[(x+\varepsilon(u+v))/x\right]\right)\right] \\
&+ \left[\varepsilon \mathrm{D}(\mathrm{D}(t) \cdot e) \cdot \left(\frac{\partial e}{\partial x}(u) + \left(\frac{\partial e}{\partial x}(v)\right)\left[(x+\varepsilon u)/x\right]\right)\right]
\end{aligned}
$$

Now the second summand of the above expression, highlighted in sand, can be expanded even further:

$$
\begin{aligned}
&\mathrm{D}\left(\frac{\partial t}{\partial x}(u) + \left(\frac{\partial t}{\partial x}(v)\right)\left[(x+\varepsilon u)/x\right]\right) \cdot \left(e\left[(x+\varepsilon(u+v))/x\right]\right) \\
&\sim_\varepsilon \left[\mathrm{D}\left(\frac{\partial t}{\partial x}(u)\right) \cdot \left(e\left[(x+\varepsilon u)/x\right]\left[(x+\varepsilon v)/x\right]\right)\right] \\
&+ \left[\mathrm{D}\left(\frac{\partial t}{\partial x}(v)\right) \cdot \left(e\left[(x+\varepsilon v)/x\right]\right)\right]\left[(x+\varepsilon u)/x\right] \\
&\sim_\varepsilon \left[\mathrm{D}\left(\frac{\partial t}{\partial x}(u)\right) \cdot \left(e\left[(x+\varepsilon u)/x\right] + \varepsilon \frac{\partial(e\left[(x+\varepsilon u)/x\right])}{\partial x}(v)\right)\right] \\
&+ \left[\mathrm{D}\left(\frac{\partial t}{\partial x}(v)\right) \cdot \left(e\left[(x+\varepsilon v)/x\right]\right)\right]\left[(x+\varepsilon u)/x\right] \\
&\sim_\varepsilon \left[\mathrm{D}\left(\frac{\partial t}{\partial x}(u)\right) \cdot \left(e\left[x+\varepsilon u/x\right]\right) + \left(\mathrm{D}\left(\frac{\partial t}{\partial x}(v)\right) \cdot \left(e\left[x+\varepsilon v/x\right]\right)\right)\left[x+\varepsilon u/x\right]\right] \\
&+ \left[\mathrm{D}\left(\frac{\partial t}{\partial x}(u)\right) \cdot \left(\varepsilon \frac{\partial(e\left[(x+\varepsilon u)/x\right])}{\partial x}(v)\right)\right] \\
&+ \left[\varepsilon \mathrm{D}\left(\mathrm{D}\left(\frac{\partial t}{\partial x}(u)\right) \cdot \left(e\left[(x+\varepsilon u)/x\right]\right)\right) \cdot \left(\varepsilon \frac{\partial(e\left[(x+\varepsilon u)/x\right])}{\partial x}(v)\right)\right]
\end{aligned}
$$

At this point, we note that the summands highlighted in indigo correspond precisely to the indigo summand in the expansion of the right-hand side of our original equivalence. Now joining the teal blocks together we obtain:

$$
\mathrm{D}(t) \cdot \left(\frac{\partial e}{\partial x}(u) + \left(\frac{\partial e}{\partial x}(v)\left[x+\varepsilon u/x\right]\right)\right) + \mathrm{D}\left(\frac{\partial t}{\partial x}(u)\right) \cdot \left(\varepsilon \frac{\partial(e\left[x+\varepsilon u/x\right])}{\partial x}(v)\right)
$$

$$
\begin{aligned}
&\sim_\varepsilon \left[\mathrm{D}(t)\cdot\left(\frac{\partial e}{\partial x}\,(u)\right) + \mathrm{D}(t)\cdot\left(\left(\frac{\partial e}{\partial x}\,(v)\right)[x+\varepsilon u/x]\right)\right]\\
&+\varepsilon\left[\mathrm{D}\left(\mathrm{D}(t)\cdot\left(\frac{\partial e}{\partial x}\,(u)\right)\right)\cdot\left(\frac{\partial e}{\partial x}\,(v)\right)[x+\varepsilon u/x]\right]\\
&+\varepsilon\left[\mathrm{D}\left(\frac{\partial t}{\partial x}\,(u)\right)\cdot\left(\frac{\partial(e\,[x+\varepsilon u/x])}{\partial x}\,(v)\right)\right]\\
&\sim_\varepsilon \left[\mathrm{D}(t)\cdot\left(\frac{\partial e}{\partial x}\,(u)\right) + \mathrm{D}(t)\cdot\left(\frac{\partial e}{\partial x}\,(u)+\varepsilon\left(\frac{\partial^2 e}{\partial x^2}(v,u)\right)\right)\right]\\
&+\varepsilon\left[\mathrm{D}\left(\mathrm{D}(t)\cdot\left(\frac{\partial e}{\partial x}\,(u)\right)\right)\cdot\left(\frac{\partial e}{\partial x}\,(v)\right)[x+\varepsilon u/x]\right]\\
&+\varepsilon\left[\mathrm{D}\left(\frac{\partial t}{\partial x}\,(u)\right)\cdot\left(\frac{\partial(e\,[x+\varepsilon u/x])}{\partial x}\,(v)\right)\right]\\
&\sim_\varepsilon \left[\mathrm{D}(t)\cdot\left(\frac{\partial e}{\partial x}\,(u)\right) + \mathrm{D}(t)\cdot\left(\frac{\partial e}{\partial x}\,(u)\right)\right]\\
&+\left[\varepsilon\mathrm{D}(t)\cdot\left(\frac{\partial^2 e}{\partial x^2}(v,u)\right)+\varepsilon^2\mathrm{D}\left(\mathrm{D}(t)\cdot\left(\frac{\partial e}{\partial x}\,(u)\right)\right)\cdot\left(\frac{\partial^2 e}{\partial x^2}(v,u)\right)\right]\\
&+\varepsilon\left[\mathrm{D}\left(\mathrm{D}(t)\cdot\left(\frac{\partial e}{\partial x}\,(u)\right)\right)\cdot\left(\left(\frac{\partial e}{\partial x}\,(v)\right)[x+\varepsilon u/x]\right)\right]\\
&+\varepsilon\left[\mathrm{D}\left(\frac{\partial t}{\partial x}\,(u)\right)\cdot\left(\frac{\partial(e\,[x+\varepsilon u/x])}{\partial x}\,(v)\right)\right]
\end{aligned}
$$

The terms highlighted in rose match the Taylor expansion of the rose summand in the right-hand side. It remains to assemble the fragments highlighted in wine and show that they are equivalent to the wine summand of in the right-hand side. To simplify this arduous process, we expand the first wine block and decompose it into three pieces, which will later discharge some of our remaining equivalences.

$$
\begin{aligned}
&\varepsilon\mathrm{D}(\mathrm{D}(t)\cdot e)\cdot\left(\frac{\partial e}{\partial x}\,(u)+\left(\frac{\partial e}{\partial x}\,(v)\right)[x+\varepsilon u/x]\right)\\
&\sim_\varepsilon \varepsilon\mathrm{D}(\mathrm{D}(t)\cdot e)\cdot\left(\frac{\partial e}{\partial x}\,(u)\right)+\varepsilon\mathrm{D}(\mathrm{D}(t)\cdot e)\cdot\left(\left(\frac{\partial e}{\partial x}\,(v)\right)[x+\varepsilon u/x]\right)\\
&+\varepsilon^2\mathrm{D}\left(\mathrm{D}\left(\mathrm{D}(t)\cdot e\right)\cdot\left(\frac{\partial e}{\partial x}\,(u)\right)\right)\cdot\left(\frac{\partial e}{\partial x}\,(v)\,[x+\varepsilon u/x]\right)
\end{aligned}
$$

The term highlighted in green already appears in the green summand of the original right-hand expansion and so we will henceforth consider it discharged. The remainder of the green summand Taylor-expands as follows:

$$
\varepsilon\left(\mathrm{D}(\mathrm{D}(t)\cdot e)\cdot\left(\frac{\partial e}{\partial x}\,(v)\right)\right)[x+\varepsilon u/x]
$$

$$
\begin{aligned}
&\sim_\varepsilon \varepsilon \left[\mathrm{D}(\mathrm{D}(t) \cdot e) \cdot \left(\frac{\partial e}{\partial x}(v) \right) + \varepsilon \frac{\partial \big(\mathrm{D}(\mathrm{D}(t) \cdot e) \cdot \big(\frac{\partial e}{\partial x}(v)\big)\big)}{\partial x}(u) \right] \\
&\sim_\varepsilon \varepsilon \mathrm{D}(\mathrm{D}(t) \cdot e) \cdot \left(\frac{\partial e}{\partial x}(v) \right) + \varepsilon^2 \Bigg[\\
&\qquad \mathrm{D}(\mathrm{D}(t) \cdot e) \cdot \left(\frac{\partial^2 e}{\partial x^2}(v,u) \right) \\
&\qquad + \mathrm{D}\left(\frac{\partial \mathrm{D}(t) \cdot e}{\partial x}(u) \right) \cdot \left(\left(\frac{\partial e}{\partial x}(v) \right) [x + \varepsilon u / x] \right) \\
&\qquad + \varepsilon \mathrm{D}\left(\mathrm{D}(\mathrm{D}(t) \cdot e) \cdot \frac{\partial e}{\partial x}(v) \right) \cdot \left(\frac{\partial^2 e}{\partial x^2}(v,u) \right) \\
&\Bigg] \\
&\sim_\varepsilon \varepsilon \mathrm{D}(\mathrm{D}(t) \cdot e) \cdot \left(\frac{\partial e}{\partial x}(v) \right) \\
&\qquad \varepsilon^2 \mathrm{D}(\mathrm{D}(t) \cdot e) \cdot \left(\frac{\partial^2 e}{\partial x^2}(v,u) \right) \\
&\qquad + \varepsilon^2 \mathrm{D}\left(\mathrm{D}(t) \cdot \left(\frac{\partial e}{\partial x}(u) \right) \right) \cdot \left(\left(\frac{\partial e}{\partial x}(v) \right) [x + \varepsilon u / x] \right) \\
&\qquad + \varepsilon^2 \mathrm{D}\left(\mathrm{D}\left(\frac{\partial t}{\partial x}(u) \right) \cdot (e\,[x + \varepsilon u / x]) \right) \cdot \left(\left(\frac{\partial e}{\partial x}(v) \right) [x + \varepsilon u / x] \right) \\
&\qquad + \varepsilon^3 \mathrm{D}\left(\mathrm{D}\left(\mathrm{D}(t) \cdot e\right)) \cdot \left(\frac{\partial e}{\partial x}(u) \right) \right) \cdot \left(\left(\frac{\partial e}{\partial x}(v) \right) [x + \varepsilon u / x] \right) \\
&\qquad + \varepsilon^3 \mathrm{D}\left(\mathrm{D}(\mathrm{D}(t) \cdot e) \cdot \frac{\partial e}{\partial x}(v) \right) \cdot \left(\frac{\partial^2 e}{\partial x^2}(v,u) \right)
\end{aligned}
$$

The cyan summands are neutralised with the Taylor expansion of the term highlighted in cyan (in the expansion of the second wine block), whereas the wine summand is exactly the wine-highlighted term that remained from the teal blocks. Due to infinitesimal duplication, the olive summand is equivalent to the term highlighted in olive, and the term in purple ink is equivalent to the remaining purple-highlighted term.

- The case $s = t\ e$ is similar to the previous case, although simpler. For brevity's sake, we omit it here, but a fully mechanised version of the proof using the `Coq` proof assistant is available. □

7.1.3 The Operational Semantics of λ_ε

With the substitution operations we have introduced so far, we can now proceed to give a small-step operational semantics as a reduction system.

Definition 7.1.12. The **one-step reduction relation** $\rightsquigarrow \; \subseteq \Lambda_\varepsilon \times \Lambda_\varepsilon$ is the least contextual relation satisfying the reduction rules in Figure 7.6 below.

$$\begin{array}{rcl} (\lambda x.t)\ s & \rightsquigarrow_\beta & t\,[s/x] \\ \mathrm{D}(\lambda x.t) \cdot s & \rightsquigarrow_\partial & \lambda x.\left(\frac{\partial t}{\partial x}\,(s)\right) \end{array}$$

Figure 7.6: One-step reduction rules for λ_ε

We write $\rightsquigarrow^+$ to denote the transitive closure of $\rightsquigarrow$, and $\rightsquigarrow^*$ to denote its transitive, reflexive closure.

While the one-step reduction rules for λ_ε may seem identical to those in the differential λ-calculus (see Figure 2.2), they are in fact not equivalent, as our notions of differential substitution and term equivalence differ substantially.

The above one-step reduction is defined as a relation from unrestricted terms to unrestricted terms, but it is not compatible with differential equivalence. That is to say, there may be differentially equivalent terms $t \sim_\varepsilon t'$ such that t' can be reduced but t cannot. For example, consider the term $(\lambda x.x + 0)\ 0$, which contains no β-redexes that can be reduced. This term is, however, equivalent to $(\lambda x.x)\ 0$, which clearly reduces to 0.

We could lift the one-step reduction relation to well-formed terms by setting $\underline{t} \rightsquigarrow \underline{t'}$ whenever there exist s, s' such that $t \sim_\varepsilon s, t' \sim_\varepsilon s'$ and $s \rightsquigarrow s'$. This is not very satisfactory, however, as it would make one-step reduction undecidable. Indeed, in order to check whether $\underline{s} \rightsquigarrow \underline{s'}$ it would be necessary to check whether $s \rightsquigarrow s'$ for all their (infinitely many) representatives s, s'!

Another problem with this definition lies in the fact that the term $\underline{0}$ (ostensibly a value which should not reduce) can also be written as $\underline{0\ t}$ for any term t. Whenever t reduces to t' in one step, then according to the previous definition so does $\underline{0\ t}$ reduce to $\underline{0\ t'}$, which is equivalent to $\underline{0}$. Hence zero reduces to itself, rather than being a normal form!

Fortunately the canonical form of a term t gives us a representative of $\underline{t}$ which is "maximally reducible", that is to say, whenever any representative of $\underline{t}$ can be reduced, then so can $\mathbf{can}\,(t)$, possibly in zero steps.

Theorem 7.1.5. Reduction is compatible with canonicalization. That is to say, if $s \rightsquigarrow s'$, then $\mathbf{can}\,(s) \rightsquigarrow^* s''$ for some $s'' \sim_\varepsilon s'$.

Proof. We prove the theorem by induction on the depth at which reduction happens and the number of non-canonical β-redexes in the term (that is, the number of redexes of the form $(\lambda x.s)\ (t+\varepsilon u)$). The most important cases are the ones where it happens at the outermost level, but we explicitly show some of the other cases. Before we do so, however, we state the following auxiliary properties:

Lemma 7.1.11. Whenever $T \sim_\varepsilon \lambda x.t$ where T is a canonical term and t is an unrestricted term, then T is of the form $\sum_{i=1}^n \varepsilon^{k_i}(\lambda x.t_i^{\mathbf{b}})$, and additionally $t \sim_\varepsilon \sum_{i=1}^n \varepsilon^{k_i} t_i^{\mathbf{b}}$.

Lemma 7.1.12. Whenever $s + t \rightsquigarrow^* e$ then $e = s' + t'$ with $s \rightsquigarrow^* s'$ and $t \rightsquigarrow^* t'$. Whenever $\varepsilon s \rightsquigarrow^* e$ then $e = \varepsilon s'$ with $s \rightsquigarrow^* s'$. In particular, whenever $\mathbf{can}\,(t) \rightsquigarrow^* t'$ then $t' = \sum_{i=1}^n \varepsilon^{k_i} t_i'$, where $\mathbf{can}\,(t) = \sum_{i=1}^n \varepsilon^{k_i} t_i^{\mathbf{b}}$ and $t_i^{\mathbf{b}} \rightsquigarrow^* t_i'$. Note that the t_i' may not be basic terms and thus t' may not be canonical.

Lemma 7.1.13. Whenever $s \rightsquigarrow^* s'$ and $t \rightsquigarrow^* t'$, then $s \boxplus t \rightsquigarrow^* s' \boxplus t'$.

Lemma 7.1.14. Whenever $s \rightsquigarrow^* s'$ and $\mathbf{can}\,(t) \rightsquigarrow^* t'$, then $\mathbf{reg}\,(s, \mathbf{can}\,(t)) \rightsquigarrow^* \mathbf{reg}\,(s', t')$.

We proceed now to prove one of the cases where reduction happens in a subterm of s, which illustrates the ideas for the other cases:

- Let $s = \mathrm{D}(t) \cdot u$ and $s' = \mathrm{D}(t') \cdot u$, with $t \rightsquigarrow t'$. By Lemma 7.1.5, we can write $\mathbf{can}\,(s)$ as $\mathbf{can}\,(\mathrm{D}(\mathbf{can}\,(t)) \cdot \mathbf{can}\,(u))$. Let $\mathbf{can}\,(t) = \sum_{i=1}^n \varepsilon^{k_i} t_i^{\mathbf{b}}$. By induction and Lemma 7.1.12, we have $t_i^{\mathbf{b}} \rightsquigarrow^* t_i''$ and $t' \sim_\varepsilon \sum_{i=1}^n \varepsilon^{k_i} t_i''$. Applying the previous auxiliary lemmas, we obtain:

$$\begin{aligned}
\mathbf{can}\,(s) &= \mathbf{can}\,(\mathrm{D}(\mathbf{can}\,(t)) \cdot u) \\
&= \underset{i=1}{\overset{n}{\boxplus}} (\varepsilon^*)^{k_i} \mathbf{reg}\left(t_i^{\mathbf{b}}, \mathbf{can}\,(u)\right) \\
&\rightsquigarrow^* \underset{i=1}{\overset{n}{\boxplus}} (\varepsilon^*)^{k_i} \mathbf{reg}\left(t_i'', \mathbf{can}\,(u)\right) \\
&\sim_\varepsilon \mathrm{D}\left(\sum_{i=1}^n \varepsilon^{k_i} t_i''\right) \cdot u \\
&\sim_\varepsilon \mathrm{D}(t') \cdot u
\end{aligned}$$

- Let $s = (t\ e)$ and $s' = (t\ e')$, with $e \rightsquigarrow e'$. The result follows from the previous auxiliary lemmas and the fact that the primal **pri** and tangent **tan** components commute with reduction.

- Every other case is either immediate or follows from similar arguments.

The more involved cases are those when reduction happens at the outermost level of s. For brevity we will focus on the non-trivial cases where the underlying λ-abstraction involves only basic terms, as the more general cases follow by unfolding the λ-abstraction into a canonical sum and applying the primitive cases below to each summand separately.

- Let $s = \mathrm{D}(\lambda x.s^{\mathbf{b}}) \cdot t$ and $s' = \lambda x.\frac{\partial s^{\mathbf{b}}}{\partial x}\left(t\right)$, and write T for $\mathbf{can}\left(t\right)$. The proof proceeds then by induction on the number of summands of T.

  - When $T = 0$ we have $\mathbf{can}\left(s\right) = 0$. On the other hand:

$$s' = \lambda x.\frac{\partial s^{\mathbf{b}}}{\partial x}\left(t\right) \sim_{\varepsilon} \lambda x.\frac{\partial s^{\mathbf{b}}}{\partial x}\left(0\right) \sim_{\varepsilon} 0$$

  - When $T = \varepsilon^k t^{\mathbf{b}} + T'$, we apply the induction hypothesis and Lemma 7.1.11 to obtain

$$\mathbf{can}\left(\mathrm{D}(\lambda x.s^{\mathbf{b}}) \cdot T'\right) \rightsquigarrow^* \sum_{i=1}^{n} \varepsilon^{k_i}(\lambda x.w_i) \sim_{\varepsilon} \lambda x.\frac{\partial s^{\mathbf{b}}}{\partial x}\left(T'\right)$$

    Then the canonical form $\mathbf{can}\left(s\right)$ reduces as follows:

$$\begin{aligned}
&\mathbf{can}\left(s\right)\\
&= (\varepsilon^*)^k \mathrm{D}(\lambda x.s^{\mathbf{b}}) \cdot t^{\mathbf{b}}\\
&\quad + \left[\mathbf{reg}\left(\mathrm{D}(\lambda x.s^{\mathbf{b}}) \cdot T'\right) \boxplus \varepsilon^* \mathrm{D}^*\left(\mathbf{reg}\left(\mathrm{D}(\lambda x.s^{\mathbf{b}}) \cdot T'\right)\right) \cdot t^{\mathbf{b}}\right]\\
&= (\varepsilon^*)^k \mathrm{D}(\lambda x.s^{\mathbf{b}}) \cdot t^{\mathbf{b}}\\
&\quad + \left[\mathbf{can}\left(\mathrm{D}(\lambda x.s^{\mathbf{b}}) \cdot T'\right) \boxplus \varepsilon^* \mathrm{D}^*\left(\mathbf{can}\left(\mathrm{D}(\lambda x.s^{\mathbf{b}}) \cdot T'\right)\right) \cdot t^{\mathbf{b}}\right]\\
&\rightsquigarrow^* (\varepsilon^*)^k \left(\lambda x.\frac{\partial s^{\mathbf{b}}}{\partial x}\left(t^{\mathbf{b}}\right)\right)\\
&\quad + \left[\left(\sum_{i=1}^{n} \varepsilon^{k_i}(\lambda x.w_i)\right) \boxplus \varepsilon^* \mathrm{D}^*\left(\sum_{i=1}^{n} \varepsilon^{k_i}(\lambda x.w_i)\right) \cdot t^{\mathbf{b}}\right]\\
&= (\varepsilon^*)^k \left(\lambda x.\frac{\partial s^{\mathbf{b}}}{\partial x}\left(t^{\mathbf{b}}\right)\right)\\
&\quad + \left[\left(\sum_{i=1}^{n} \varepsilon^{k_i}(\lambda x.w_i)\right) \boxplus \varepsilon^*\left(\sum_{i=1}^{n} \varepsilon^{k_i} \mathrm{D}(\lambda x.w_i) \cdot t^{\mathbf{b}}\right)\right]\\
&\rightsquigarrow^* (\varepsilon^*)^k \left(\lambda x.\frac{\partial s^{\mathbf{b}}}{\partial x}\left(t^{\mathbf{b}}\right)\right)\\
&\quad + \left[\left(\sum_{i=1}^{n} \varepsilon^{k_i}(\lambda x.w_i)\right) \boxplus \varepsilon^*\left(\sum_{i=1}^{n} \varepsilon^{k_i} \lambda x.\frac{\partial w_i}{\partial x}\left(t^{\mathbf{b}}\right)\right)\right]\\
&\sim_{\varepsilon} \lambda x.\left(\frac{\partial s^{\mathbf{b}}}{\partial x}\left(\varepsilon^k t^{\mathbf{b}}\right)\right) + \lambda x.\left(\frac{\partial s^{\mathbf{b}}}{\partial x}\left(T'\right)\right) + \varepsilon\left(\lambda x.\frac{\partial\left(\sum_{i=1}^{n} \varepsilon^{k_i} w_i\right)}{\partial x}\left(t^{\mathbf{b}}\right)\right)
\end{aligned}$$

$$\sim_\varepsilon \lambda x.\left(\frac{\partial s^{\mathbf{b}}}{\partial x}\left(\varepsilon^k t^{\mathbf{b}}\right)\right) + \lambda x.\left(\frac{\partial s^{\mathbf{b}}}{\partial x}\left(T'\right)\right) + \varepsilon\left(\lambda x.\frac{\partial \frac{\partial s^{\mathbf{b}}}{\partial x}\left(T'\right)}{\partial x}\left(t^{\mathbf{b}}\right)\right)$$
$$\sim_\varepsilon \lambda x.\left(\frac{\partial s^{\mathbf{b}}}{\partial x}\left(\varepsilon^k t^{\mathbf{b}}\right) + \left(\frac{\partial s^{\mathbf{b}}}{\partial x}\left(T'\right)\right)\left[x + \varepsilon t^{\mathbf{b}}/x\right]\right)$$

By Theorem 7.1.4, this last term is precisely the term $\lambda x.\frac{\partial s^{\mathbf{b}}}{\partial x}\left(\varepsilon^k t^{\mathbf{b}} + T'\right)$, which is equivalent to s' and thus the proof is concluded.

- Let $s = (\lambda x.s^{\mathbf{b}})\ t$ and $s' = s^{\mathbf{b}}\left[t/x\right]$, and suppose $\mathbf{can}\left(t\right) = t^* + \varepsilon^* T$ (that is, $\mathbf{pri}(\mathbf{can}\left(t\right)) = t^*$ and $\mathbf{tan}(\mathbf{can}\left(t\right)) = T$, with t^* additive and T canonical).

$$\mathbf{can}\left(s\right) = (\lambda x.s^{\mathbf{b}})\ t^* \boxplus \varepsilon^* \mathbf{ap}\left(\mathbf{reg}\left(\lambda x.s^{\mathbf{b}}, T\right), t^*\right)$$

Since the term $(\lambda x.s^{\mathbf{b}}) \cdot T$ contains one less non-canonical β-redex than the term s, we apply our induction hypothesis to obtain

$$\mathbf{reg}\left(\lambda x.s^{\mathbf{b}}, T\right) = \mathbf{can}\left((\lambda x.s^{\mathbf{b}}) \cdot T\right) \rightsquigarrow^* \sum_{i=1}^{n} \varepsilon^{k_i}(\lambda x.w_i) \sim_\varepsilon \lambda x.\frac{\partial s^{\mathbf{b}}}{\partial x}\left(T\right)$$

and therefore $\frac{\partial s^{\mathbf{b}}}{\partial x}\left(T\right) \sim_\varepsilon \sum_{i=1}^{n} \varepsilon^{k_i} w_i$. With this in mind we continue to reduce the previous equation:

$$\begin{aligned}
&(\lambda x.s^{\mathbf{b}})\ t^* \boxplus \varepsilon^* \mathbf{ap}\left(\mathbf{reg}\left(\lambda x.s^{\mathbf{b}}, T\right), t^*\right)\\
&\quad\rightsquigarrow^* s^{\mathbf{b}}\left[t^*/x\right] \boxplus \varepsilon^* \mathbf{ap}\left(\sum_{i=1}^{n} \varepsilon^{k_i}(\lambda x.w_i), t^*\right)\\
&\quad= s^{\mathbf{b}}\left[t^*/x\right] \boxplus \varepsilon^* \left(\sum_{i=1}^{n} \varepsilon^{k_i}(\lambda x.w_i)\ t^*\right)\\
&\quad\rightsquigarrow^* s^{\mathbf{b}}\left[t^*/x\right] \boxplus \varepsilon^* \left(\sum_{i=1}^{n} \varepsilon^{k_i}(w_i\left[t^*/x\right])\right)\\
&\quad= s^{\mathbf{b}}\left[t^*/x\right] \boxplus \varepsilon^* \left[\left(\sum_{i=1}^{n} \varepsilon^{k_i} w_i\right)\left[t^*/x\right])\right]\\
&\quad\sim_\varepsilon s^{\mathbf{b}}\left[t^*/x\right] + \varepsilon\left(\frac{\partial s^{\mathbf{b}}}{\partial x}\left(T\right)\left[t^*/x\right]\right)\\
&\quad\sim_\varepsilon s^{\mathbf{b}}\left[t^* + \varepsilon T/x\right]\\
&\quad\sim_\varepsilon s^{\mathbf{b}}\left[t/x\right]
\end{aligned}$$

□

The above result then legitimises our proposed "existential" definition of reduction of well-formed terms, as it shows that, in order to reduce a given term, it suffices to reduce its canonical form. It also gets rid of the "reducing zero" problem, as canonical forms do not contain "spurious" representations of zero.

Definition 7.1.13. Given well-formed terms $\underline{s}, \underline{s'}$, we say that $\underline{s}$ **reduces to** $\underline{s'}$ **in one step**, and write $\underline{s} \mathrel{\underline{\rightsquigarrow}} \underline{t}$, whenever $\mathbf{can}\,(s) \rightsquigarrow s''$ and $s'' \sim_\varepsilon s'$, for some canonical form $\mathbf{can}\,(s)$ of $\underline{s}$.

Proposition 7.1.5. Whenever $\underline{s} \mathrel{\underline{\rightsquigarrow}} \underline{s'}$ then for any term $\underline{t}$ we have $\underline{s + t} \mathrel{\underline{\rightsquigarrow}} \underline{s' + t}$.

If $t = t^*$ is an additive term, then additionally $\underline{s\ t^*} \mathrel{\underline{\rightsquigarrow}}^+ \underline{s'\ t^*}$.

Furthermore, when $t = t^{\mathbf{b}}$ is a basic term (in particular $t^{\mathbf{b}}$ is not differentially equivalent to zero), we also have $\underline{\mathrm{D}(s) \cdot t} \mathrel{\underline{\rightsquigarrow}}^+ \underline{\mathrm{D}(s') \cdot t}$.

Conversely, whenever s is not differentially equivalent to zero and $\underline{t} \mathrel{\underline{\rightsquigarrow}} \underline{t'}$, then $\underline{s\ t} \mathrel{\underline{\rightsquigarrow}}^+ \underline{s\ t'}$ and $\underline{\mathrm{D}(s) \cdot t} \mathrel{\underline{\rightsquigarrow}}^+ \underline{\mathrm{D}(s) \cdot t'}$.

The wording of the above definition specifies that a well-formed term reduces to another whenever *any* of its canonical forms reduces. As we have shown before, canonical forms are in fact only unique up to commutativity of addition and derivatives. Addition is not problematic, since it respects reduction; that is to say, if a sum $s + t$ reduces to $s' + t'$ then its permutation $t + s$ also reduces to $t' + s'$. Symmetry of derivatives raises a more significant issue: consider the following diagram, which does *not* commute:

$$\begin{tikzcd}
\mathrm{D}(\mathrm{D}(\lambda x.t) \cdot u) \cdot v \arrow[r, phantom, "\sim_\varepsilon"] \arrow[d, squiggly] & \mathrm{D}(\mathrm{D}(\lambda x.t) \cdot v) \cdot u \arrow[d, squiggly] \\
\mathrm{D}\left(\lambda x. \frac{\partial t}{\partial x}(u)\right) \cdot v \arrow[r, phantom, "\not\sim_\varepsilon"] & \mathrm{D}\left(\lambda x. \frac{\partial t}{\partial x}(v)\right) \cdot u
\end{tikzcd}$$

One-step reduction is still computable, since the set of canonical forms of any given term is finite, but we will have to keep this behaviour in mind when showing confluence.

A proof of confluence for λ_ε will proceed by the standard Tait/Martin-Löf method by introducing a notion of parallel reduction on terms.

Definition 7.1.14. The **parallel reduction** relation between (unrestricted) terms is defined according to the deduction rules in Figure 7.7.

The parallel reduction relation can be extended to well-formed terms by setting $\underline{t} \mathrel{\underline{\rightrightarrows}} \underline{t'}$ whenever $\mathbf{can}\,(t) \rightrightarrows t''$ with $t'' \sim_\varepsilon t'$ for some canonical form of $\underline{t}$.

Remark 7.1.1. Our definition of parallel reduction differs slightly from the usual in the rule $(\rightrightarrows_\beta)$, which allows reducing a newly-formed λ-abstraction. This is necessary because our calculus contains terms of the shape $(\mathrm{D}(\lambda x.s) \cdot u)\ t$, which we need to parallel reduce in a single step to $\left(\frac{\partial s}{\partial x}(u)\right)[t/x]$. The original presentation of the

$$(\substack{\rightsquigarrow\\\rightsquigarrow}_x)\ \frac{}{x \mathrel{\substack{\rightsquigarrow\\\rightsquigarrow}} x} \qquad (\substack{\rightsquigarrow\\\rightsquigarrow}_0)\ \frac{}{0 \mathrel{\substack{\rightsquigarrow\\\rightsquigarrow}} 0} \qquad (\substack{\rightsquigarrow\\\rightsquigarrow}_\lambda)\ \frac{t \mathrel{\substack{\rightsquigarrow\\\rightsquigarrow}} t'}{\lambda x.t \mathrel{\substack{\rightsquigarrow\\\rightsquigarrow}} \lambda x.t'}$$

$$(\substack{\rightsquigarrow\\\rightsquigarrow}_\varepsilon)\ \frac{t \mathrel{\substack{\rightsquigarrow\\\rightsquigarrow}} t'}{\varepsilon t \mathrel{\substack{\rightsquigarrow\\\rightsquigarrow}} \varepsilon t'} \qquad (\substack{\rightsquigarrow\\\rightsquigarrow}_+)\ \frac{s \mathrel{\substack{\rightsquigarrow\\\rightsquigarrow}} s' \qquad t \mathrel{\substack{\rightsquigarrow\\\rightsquigarrow}} t'}{s + t \mathrel{\substack{\rightsquigarrow\\\rightsquigarrow}} s' + t'}$$

$$(\substack{\rightsquigarrow\\\rightsquigarrow}_{\mathbf{ap}})\ \frac{s \mathrel{\substack{\rightsquigarrow\\\rightsquigarrow}} s' \qquad t \mathrel{\substack{\rightsquigarrow\\\rightsquigarrow}} t'}{s\ t \mathrel{\substack{\rightsquigarrow\\\rightsquigarrow}} s'\ t'} \qquad (\substack{\rightsquigarrow\\\rightsquigarrow}_{\mathrm{D}})\ \frac{s \mathrel{\substack{\rightsquigarrow\\\rightsquigarrow}} s' \qquad t \mathrel{\substack{\rightsquigarrow\\\rightsquigarrow}} t'}{\mathrm{D}(s) \cdot t \mathrel{\substack{\rightsquigarrow\\\rightsquigarrow}} \mathrm{D}(s') \cdot t'}$$

$$(\substack{\rightsquigarrow\\\rightsquigarrow}_\beta)\ \frac{s \mathrel{\substack{\rightsquigarrow\\\rightsquigarrow}} \lambda x.s' \qquad t \mathrel{\substack{\rightsquigarrow\\\rightsquigarrow}} t'}{s\ t \mathrel{\substack{\rightsquigarrow\\\rightsquigarrow}} s'\,[t'/x]} \qquad (\substack{\rightsquigarrow\\\rightsquigarrow}_\partial)\ \frac{s \mathrel{\substack{\rightsquigarrow\\\rightsquigarrow}} \lambda x.s' \qquad t \mathrel{\substack{\rightsquigarrow\\\rightsquigarrow}} t'}{\mathrm{D}(s) \cdot t \mathrel{\substack{\rightsquigarrow\\\rightsquigarrow}} \lambda x.\frac{\partial s'}{\partial x}\,(t')}$$

Figure 7.7: Parallel reduction rules for λ_ε

differential λ-calculus opted instead for adding an extra parallel reduction rule to allow for the case of reducing an abstraction under a differential application. Similarly, our rule $(\substack{\rightsquigarrow\\\rightsquigarrow}_\partial)$ allows reducing terms of the form $\mathrm{D}(\mathrm{D}(\lambda x.s) \cdot u) \cdot v$ in a single step.

One convenient property of the parallel reduction relation lies in its relation to canonical forms. As we saw in Theorem 7.1.5, canonical forms are "maximally reducible", but don't respect the number of reduction steps. This is no longer the case for parallel reduction: the process of canonicalization only duplicates regexes "in parallel" (that is, by copying them onto multiple separate summands) or in a "parallelizable series" (i.e. a differential application may be regularized into a term of the form $\mathrm{D}(\mathrm{D}(\ldots) \cdot u) \cdot v$, which can be entirely reduced in a single parallel reduction step).

Theorem 7.1.6. Whenever $s \mathrel{\substack{\rightsquigarrow\\\rightsquigarrow}} s'$, then $\mathbf{can}\,(s) \mathrel{\substack{\rightsquigarrow\\\rightsquigarrow}} s''$ for some $s'' \sim_\varepsilon s'$.

Proof. It suffices to inspect the proof of Theorem 7.1.5 and convince oneself that all of the reductions introduced by the proof can be lifted into a single instance of parallel reduction. □

We also state the following standard properties of parallel reduction, all of which can be proven by straightforward induction on the term.

Lemma 7.1.15. Parallel reduction sits between one-step and many-step reduction. That is to say: $\rightsquigarrow\ \subseteq\ \substack{\rightsquigarrow\\\rightsquigarrow}\ \subseteq\ \rightsquigarrow^*$, and furthermore $\underline{\rightsquigarrow}\ \subseteq\ \underline{\substack{\rightsquigarrow\\\rightsquigarrow}}\ \subseteq\ \underline{\rightsquigarrow}^*$.

Lemma 7.1.16. The parallel reduction relation is contextual. In particular, every term parallel-reduces to itself.

Lemma 7.1.17. Parallel reduction cannot introduce free variables. That is to say: whenever $t \Rrightarrow t'$, we have $\mathrm{FV}(t') \subseteq \mathrm{FV}(t)$.

Lemma 7.1.18. Whenever $\lambda x.t \Rrightarrow u$, it must be the case that $u = \lambda x.t'$ and $t \Rrightarrow t'$.

Lemma 7.1.19. Whenever $s \Rrightarrow s'$ and $t \Rrightarrow t'$ then $s\,[t/x] \Rrightarrow s'\,[t'/x]$, and furthermore there is some w with $\frac{\partial s}{\partial x}\,(t) \Rrightarrow w \sim_\varepsilon \frac{\partial s'}{\partial x}\,(t')$.

Proof. Both proofs follow by induction on the derivation applied to obtain $s \Rrightarrow s'$. We explicitly prove some non-trivial cases. First, for standard substitution.

- When the last rule applied is $(\Rrightarrow_\beta)$, that is: $s = e\ u, s' = e'\,[u'/y]$, with $e \Rrightarrow \lambda y.e', u \Rrightarrow u'$. By the induction hypothesis, we have $e\,[t/x] \Rrightarrow \lambda y.(e'\,[t'/x])$ and $u\,[t/x] \Rrightarrow u'\,[t'/x]$, hence:
$$s\,[t/x] = e\,[t/x]\ \ (u\,[t/x]) \Rrightarrow (e'\,[t'/x])\,[(u'\,[t'/x])/y] = (e'\,[u'/y])\,[t'/x]$$
- When the last rule applied is $(\Rrightarrow_\partial)$, that is: $s = \mathrm{D}(e)\cdot u$, $s' = \lambda y.\frac{\partial e'}{\partial x}\,(u')$, with $e \Rrightarrow \lambda y.e', u \Rrightarrow u'$. As before, we apply the induction hypothesis and obtain:
$$\begin{aligned} s\,[t/x] &= D(e\,[t/x])\cdot(u\,[t/x]) \\ &\Rrightarrow \lambda y.\left(\frac{\partial(e'\,[t'/x])}{\partial y}\,(u'\,[t'/x])\right) = \left(\lambda y.\frac{\partial e'}{\partial y}\,(u')\right)[t'/x] \end{aligned}$$

The corresponding cases for differential substitution are slightly more technically involved.

- When the last rule applied is $(\Rrightarrow_\beta)$, that is: $s = (e\ u), s' = e'\,[u'/y]$, with $e \Rrightarrow \lambda y.e', u \Rrightarrow u'$. By the induction hypothesis and applying the definition of differential substitution, we have $\frac{\partial e}{\partial x}\,(t) \Rrightarrow \lambda y.\frac{\partial e'}{\partial x}\,(t')$ and $\frac{\partial u}{\partial x}\,(t) \Rrightarrow \frac{\partial u'}{\partial x}\,(t')$. By applying the previous proof we also obtain $u\,[x+\varepsilon t/x] \Rrightarrow u'\,[x+\varepsilon t'/x]$ hence[2]:
$$\begin{aligned} \frac{\partial s}{\partial x}\,(t) &= \left[\left(\mathrm{D}(e)\cdot\left(\frac{\partial u}{\partial x}\,(t)\right)\right)\ u\right] + \left[\frac{\partial e}{\partial x}\,(t)\ \ (u\,[x+\varepsilon t/x])\right] \\ &\Rrightarrow \left[\frac{\partial e'}{\partial y}\left(\frac{\partial u'}{\partial x}\,(t')\right)\right][u'/y] + \left(\frac{\partial e'}{\partial x}\,(t')\right)[(u'\,[x+\varepsilon t'/x])/y] \end{aligned}$$

  On the other hand, since y is not free in either u' or t', applying Lemma 7.1.8, we obtain:
$$\begin{aligned} \frac{\partial s'}{\partial x}\,(t') &= \frac{\partial(e'\,[u'/y])}{\partial x}\,(t') \\ &\sim_\varepsilon \left(\frac{\partial e'}{\partial x}\,(t')\right)[(u'\,[x+\varepsilon t'/x])/y] + \left(\frac{\partial e'}{\partial y}\left(\frac{\partial u'}{\partial x}\,(t')\right)\right)[u'/y] \end{aligned}$$

[2]Observe that the reasoning here would not hold if we had opted to define parallel reduction in the "standard" way, as differential substitution may "unfold" an application into a differential application followed by a standard one.

- When the last rule applied is $(\ggg_\partial)$, that is: $s = \mathrm{D}(\lambda y.e) \cdot u$, $s' = \lambda y.\frac{\partial e'}{\partial x}(u')$, with $e \ggg e', u \ggg u'$. As before, we apply the induction hypothesis and obtain:

$$\begin{aligned} s\,[t/x] &= D(\lambda y.e\,[t/x]) \cdot (u\,[t/x]) \\ &\ggg \lambda y.\left(\frac{\partial(e'\,[t'/x])}{\partial y}\,(u'\,[t'/x])\right) \\ &= \left(\lambda y.\frac{\partial e'}{\partial y}\,(u')\right)[t'/x] \end{aligned}$$

□

We first prove that parallel reduction has the diamond property when applied to canonical terms, taking care that it holds up to differential equivalence (note that, much like one-step reduction, the result of parallel-reducing a canonical term need not be canonical). For this, we introduce the usual notion of a full parallel reduct of a term.

Definition 7.1.15. Given an unrestricted term t, its **full parallel reduct** $t_\downarrow$ is defined inductively by:

$$\begin{aligned} x_\downarrow &:= x \\ (\varepsilon t)_\downarrow &:= \varepsilon(t_\downarrow) \\ (s+t)_\downarrow &:= (s_\downarrow) + (t_\downarrow) \\ 0_\downarrow &:= 0 \\ (\lambda x.t)_\downarrow &:= \lambda x.(t_\downarrow) \\ (s\ t)_\downarrow &:= \begin{cases} e\,[t_\downarrow/x] & \text{if } s_\downarrow = \lambda x.e \\ (s_\downarrow)\ (t_\downarrow) & \text{otherwise} \end{cases} \\ (\mathrm{D}(s)\cdot t)_\downarrow &:= \begin{cases} \lambda x.\frac{\partial e}{\partial x}\,(t_\downarrow) & \text{if } t_\downarrow = \lambda x.e \\ D(s_\downarrow)\cdot(t_\downarrow) & \text{otherwise} \end{cases} \end{aligned}$$

Lemma 7.1.20. Whenever $s \ggg \lambda x.v$, then $s_\downarrow$ is of the form $\lambda x.w$, for some term w.

Proof. The proof follows by inspection of the parallel reduction rules. Consider a derivation of $s \ggg s'$, which will be of the form $\lambda x_1.\lambda x_2.\ldots\lambda x_n.t$ (with n possibly equal to 0). In general the amount of abstractions at the outermost level of the term depends on our choice of a derivation for $s \ggg s'$. Suppose then that we pick a derivation for which n is maximal. If this derivation does not use the rules $\ggg_{\mathbf{ap}}$ or $\ggg_{\mathrm{D}}$, then it is already the case that $s' = s_\downarrow$. On the other hand, if it contains either rule, it is straightforward to see that replacing the last application of $\ggg_{\mathbf{ap}}$ or $\ggg_{\mathrm{D}}$ by $\ggg_\beta$ or $\ggg_\partial$ respectively the resulting term has at least as many λ-abstractions at the outermost level as the previous one. Iterating this process, we obtain that the number of outermost abstractions is maximised precisely whenever $s' = s_\downarrow$. □

Theorem 7.1.7. For any unrestricted terms s, s' such that $s \rightsquigarrow s'$, there is an unrestricted term w such that $s' \rightsquigarrow w$ and $w \sim_\varepsilon s_\downarrow$.

Proof. The proof follows by induction on the derivation of $s \rightsquigarrow s'$. Most cases are straightforward, and the rest follow as a corollary of Lemma 7.1.19, as we now show.

- The last applied rule is $\rightsquigarrow_\beta$, that is, $s = (t\ e)$, $s' = t'\,[e'/x]$, with $t \rightsquigarrow \lambda x.t', e \rightsquigarrow e'$.

  By the induction hypothesis we have $e' \rightsquigarrow w_e \sim_\varepsilon e_\downarrow$ and $(\lambda x.t') \rightsquigarrow (\lambda x.w_t) \sim_\varepsilon t_\downarrow$. By Lemma 7.1.20, it follows that $t_\downarrow$ has the form $\lambda x.v_t$, and since $(\lambda x.w_t) \sim_\varepsilon (\lambda x.v_t)$ we also have $w_t \sim_\varepsilon v_t$. Then:

  $$s' = t'\,[e'/x] \rightsquigarrow w_t\,[w_e/x] \sim_\varepsilon v_t\,[e_\downarrow/x] = (t\ e)_\downarrow$$

- The last applied rule is $\rightsquigarrow_\partial$, that is, $s = (\mathrm{D}(t) \cdot u), s' = \lambda x.\frac{\partial t'}{\partial x}\,(u')$, with $t \rightsquigarrow \lambda x.t', u \rightsquigarrow u'$.

  By the induction hypothesis we have $u' \rightsquigarrow w_u \sim_\varepsilon u_\downarrow$ and $\lambda x.t' \rightsquigarrow \lambda x.w_t \sim_\varepsilon t_\downarrow$. Again we must have $t_\downarrow = \lambda x.v_t$ with $w_t \sim_\varepsilon v_t$. Then:

  $$s' = \frac{\partial t'}{\partial x}\,(u') \rightsquigarrow \frac{\partial w_t}{\partial x}\,(w_u) \sim_\varepsilon \frac{\partial v_t}{\partial x}\,(u_\downarrow) = (\mathrm{D}(t) \cdot u)_\downarrow \qquad \square$$

Corollary 7.1.3. Parallel reduction has the diamond property up to differential equivalence. That is to say, for any unrestricted term t and terms t_1, t_2 such that $t \rightsquigarrow t_1$ and $t \rightsquigarrow t_2$, there are terms u, v making the following diagram commute:

$$\begin{tikzcd}
 & T \arrow[dl, rightsquigarrow] \arrow[dr, rightsquigarrow] & \\
t_1 \arrow[d, rightsquigarrow] & & t_2 \arrow[d, rightsquigarrow] \\
u \arrow[rr, phantom, "\sim_\varepsilon"] & & v
\end{tikzcd}$$

Lemma 7.1.21. Given unrestricted terms $s \sim_+ s'$ which are permutatively equivalent, that is, which differ only up to a reordering of their additions and differential applications, their full parallel reducts are differentially equivalent.

Proof. The proof follows by straightforward induction on the proof of permutative equivalence. The only involved case is $s = \mathrm{D}(\mathrm{D}(\lambda x.t) \cdot u) \cdot v \sim_+ \mathrm{D}(\mathrm{D}(\lambda x.t) \cdot v) \cdot u = s'$. But in this case the result follows as a corollary of Proposition 7.1.4

$$\begin{aligned} s_{\downarrow} &= (\mathrm{D}(\mathrm{D}(\lambda x.t) \cdot u) \cdot v)_{\downarrow} \\ &= \lambda x.\frac{\partial^2 t}{\partial x^2}(u, v) \\ &\sim_{\varepsilon} \lambda x.\frac{\partial^2 t}{\partial x^2}(v)u \\ &= s'_{\downarrow} \end{aligned}$$

□

Theorem 7.1.8. The reduction relation $\underline{\overset{\rightsquigarrow}{\rightsquigarrow}}$ has the diamond property. That is, whenever $\underline{s} \,\underline{\overset{\rightsquigarrow}{\rightsquigarrow}}\, \underline{u}$ and $\underline{s} \,\underline{\overset{\rightsquigarrow}{\rightsquigarrow}}\, \underline{v}$ there is a term $\underline{c}$ such that $\underline{u} \,\underline{\overset{\rightsquigarrow}{\rightsquigarrow}}\, \underline{c}$ and $\underline{v} \,\underline{\overset{\rightsquigarrow}{\rightsquigarrow}}\, \underline{c}$.

Proof. Consider a well-formed term $\underline{s}$, and suppose that $\underline{s} \,\underline{\overset{\rightsquigarrow}{\rightsquigarrow}}\, \underline{u}$ and $\underline{s} \,\underline{\overset{\rightsquigarrow}{\rightsquigarrow}}\, \underline{v}$. In particular, this means there are two canonical forms $\mathbf{can}\,(s)_1\,, \mathbf{can}\,(s)_2$ of s such that $\mathbf{can}\,(s)_1 \overset{\rightsquigarrow}{\rightsquigarrow} u$ and $\mathbf{can}\,(s)_2 \overset{\rightsquigarrow}{\rightsquigarrow} v$. These canonical forms $\mathbf{can}\,(s)_1\,, \mathbf{can}\,(s)_2$ are equivalent up to permutative equivalence, and so their full parallel reducts are differentially equivalent as per Lemma 7.1.21. Denote their $\sim_{\varepsilon}$-equivalence class by $\underline{c}$. Therefore since $\mathbf{can}\,(s)_1 \overset{\rightsquigarrow}{\rightsquigarrow} \mathbf{can}\,(s)_{1\downarrow} \sim_{\varepsilon} c$ and $\mathbf{can}\,(s)_2 \overset{\rightsquigarrow}{\rightsquigarrow} \mathbf{can}\,(s)_{2\downarrow} \sim_{\varepsilon} c$ it follows that $\underline{u} \,\underline{\overset{\rightsquigarrow}{\rightsquigarrow}}\, \underline{c}$ and $\underline{v} \,\underline{\overset{\rightsquigarrow}{\rightsquigarrow}}\, \underline{c}$. □

Corollary 7.1.4. The reduction relation $\underline{\rightsquigarrow}$ is confluent.

7.1.4 Encoding the Differential λ-Calculus

It is immediately clear, from simply inspecting the operational semantics for λ_{ε}, that it is closely related to the differential λ-calculus – indeed, every Cartesian differential category is a Cartesian difference category, and this connection should also be reflected in the syntax.

As it turns out, there is a clean translation that embeds λ_{ε} into the differential λ-calculus, which proceeds by deleting every term that contains an ε. The intuition behind this scheme should be apparent: every single differential substitution rule in λ_{ε} is identical to the corresponding case for the differential λ-calculus, once all the ε terms are cancelled out.

Definition 7.1.16. Given an unrestricted λ_{ε} term t, its **ε-erasure** is the differential λ-term $\lceil t \rceil$ defined as follows:

$$\begin{array}{rcl}
\lceil x \rceil & := & x \\
\lceil 0 \rceil & := & 0 \\
\lceil s + t \rceil & := & \lceil s \rceil + \lceil t \rceil \\
\lceil \varepsilon t \rceil & := & 0 \\
\lceil s\ t \rceil & := & \lceil s \rceil\ \lceil t \rceil \\
\lceil \mathrm{D}(s) \cdot t \rceil & := & \mathrm{D}(\lceil s \rceil) \cdot \lceil t \rceil
\end{array} \quad rcl$$

Figure 7.8: ε-erasure of a term t

Proposition 7.1.6. The erasure $\lceil t \rceil$ is invariant under differential equivalence. That is to say, whenever $t \sim_\varepsilon t'$, it is the case that $\lceil t \rceil = \lceil t' \rceil$.

Proof. Follows immediately from inspecting the differential equivalence rules in Figure 7.2 and noticing that the erasure of both sides coincides.

Note that the standard presentation of the differential λ-calculus does not distinguish between equivalent terms, so terms like $s + t$ and $t + s$ are not merely equivalent but in fact identical. □

Proposition 7.1.7. Erasure is compatible with standard and differential substitution. That is to say, for any terms s, t and a variable x, we have:

$$\lceil s\,[t/x] \rceil = \lceil s \rceil\,[\lceil t \rceil / x]\, \lceil \frac{\partial s}{\partial x}\,(t) \rceil = \frac{\partial \lceil s \rceil}{\partial x}\,(\lceil t \rceil)$$

Corollary 7.1.5. Whenever $s \rightsquigarrow s'$, then $\lceil s \rceil \rightsquigarrow^* \lceil s' \rceil$.

These results form the syntactic obverse to the purely semantic Theorem 6.1.1: the former exhibits the differential λ-calculus as an instance of λ_ε where the ε operator is "degenerate", whereas the later shows that every Cartesian differential category can be understood as a "degenerate" Cartesian difference category, in the same sense that the corresponding infinitesimal extension is just the zero map.

7.2 Simple Types for λ_ε

Much like the differential λ-calculus, λ_ε can be endowed with a system of simple types, built from a set of basic types using the usual function type constructor.

Definition 7.2.1. The set of **types** and **contexts** of the λ_ε-calculus is given by the following inductive definition:

$$\begin{array}{rrcl}
\text{Types:} & \sigma, \tau & := & \mathbf{t} \mid \sigma \Rightarrow \tau \\
\text{Contexts:} & \Gamma & := & \emptyset \mid \Gamma, x : \tau
\end{array}$$

assuming a countably infinite set of basic types $\mathbf{t}, \mathbf{s} \ldots$ is given.

The typing rules for the λ_ε-calculus are given in Figure 7.9 below, and should not be in the least surprising, as they are identical to the typing rules for the differential λ-calculus, with the addition of a typing rule for the infinitesimal extension of a term. As one would expect, our type system enjoys all the "usual" structural properties and their proofs follow by straightforward induction on the typing derivation. Note, however, that all of these typing rules operate on unrestricted terms, rather than on well-formed terms, for reasons that we will clarify later.

$$\frac{}{\Gamma, x : \tau \vdash x : \tau} \qquad \frac{\Gamma \vdash s : \tau \Rightarrow \sigma \qquad \Gamma \vdash t : \tau}{\Gamma \vdash (s\ t) : \sigma} \qquad \frac{\Gamma, x : \tau \vdash t : \sigma}{\Gamma \vdash \lambda x.t : \sigma \Rightarrow \tau}$$

$$\frac{}{\Gamma \vdash 0 : \tau} \qquad \frac{\Gamma \vdash s : \tau \qquad \Gamma \vdash t : \tau}{\Gamma \vdash s + t : \tau} \qquad \frac{\Gamma \vdash t : \tau}{\Gamma \vdash \varepsilon t : \tau}$$

$$\frac{\Gamma \vdash s : \tau \Rightarrow \sigma \qquad \Gamma \vdash t : \tau}{\Gamma \vdash \mathrm{D}(s) \cdot t : \tau \Rightarrow \sigma}$$

Figure 7.9: Simple types for λ_ε

According to the above rules, typing derivations are *invertible*, that is to say, whenever $\Gamma \vdash t : \tau$ and t is of the form $s + e$, then it must be the case that $\Gamma \vdash s : \tau$ and $\Gamma \vdash e : \tau$, and so on. One property that fails to hold is uniqueness of typings: indeed the term 0 admits any type, as do terms such as $0 + 0$ or $(\lambda x.0)\ y$.

The following "standard" properties also hold, and can be proven by straightforward induction on the relevant typing derivation.

Proposition 7.2.1 (Weakening)**.** Whenever $\Gamma \vdash t : \tau$, then for any context Σ which is disjoint with Γ it is also the case that $\Gamma, \Sigma \vdash t : \tau$.

Proposition 7.2.2 (Substitution)**.** Whenever $\Gamma, x : \tau \vdash s : \sigma$ and $\Gamma \vdash t : \tau$, we have:

(i) $\Gamma \vdash s\,[t/x] : \sigma$

(ii) $\Gamma, x : \tau \vdash \frac{\partial s}{\partial x}(t) : \sigma$

Theorem 7.2.1 (Subject reduction)**.** Whenever $\Gamma \vdash t : \tau$ and $t \rightsquigarrow t'$ then $\Gamma \vdash t' : \tau$.

Since we have defined well-formed terms as equivalence classes of unrestricted terms, we might ask if typing is compatible with this equivalence relation. The answer is unfortunately no, that is to say, there are ill-typed terms that are differentially equivalent to well-typed terms. In particular, the term $(0\ t)$ is differentially equivalent to the term 0, but while the later is trivially well-typed, the former will not be typable for many choices of t (for example, whenever $t = (x\ x)$). A weaker version of this property does hold, however, that makes use of canonicity.

Proposition 7.2.3. Whenever $\Gamma \vdash t : \tau$, then $\Gamma \vdash \mathbf{can}(t) : \tau$, and furthermore whenever $\Gamma \vdash \mathbf{can}(t) : \tau$ then every canonical form of t admits the same type.

Proof. The proof proceeds by induction on the typing derivation, by noting that every operation involved in canonicalization respects the typing rules. □

Remark 7.2.1. The above issue could have been entirely avoided by circumventing the untyped calculus altogether and instead defining and operating on well-typed (unrestricted) terms directly. We have preferred to work out the untyped case first for two reasons: first, to mimic the development of the differential λ-calculus. Second, since differentiation of control and fixpoint operators is suspect (in that there is not an "obvious" choice of a derivative for them), we hope that working in an untyped calculus featuring Church encodings and a Y combinator can illustrate what the "natural" choice for their derivatives should be.

Before stating a progress theorem for λ_ε, we must point out one small subtlety, as the definition of reduction of unrestricted terms depends on the particular representation chosen for the term. For example, the terms $((\lambda x.x) + 0)\ 0$ and $(\lambda x.x)\ 0$ are equivalent, but the first one contains no β-redexes, whereas the second one reduces to 0 in one step. We can prove that progress holds for canonical terms, however, as those are "maximally reducible".

Definition 7.2.2. A canonical term T is a **canonical value** whenever it is of the form

$$T = \sum_{i=1}^{i} \varepsilon^{k_i}(\lambda x_i.t_i)$$

Theorem 7.2.2 (Progress)**.** Whenever a canonical term T admits a typing derivation $\vdash T : \tau$, then either T is a canonical value or there is some term t' with $T \rightsquigarrow t'$.

Proof. The proof proceeds by induction on the structure of T.

- When $T = 0$ then T is trivially a canonical value.
- When $T = \varepsilon^k s^{\mathbf{b}}$, then $s^{\mathbf{b}}$ has the form $\lambda x.e^{\mathbf{b}}$ or $\mathrm{D}(e^{\mathbf{b}}) \cdot u^{\mathbf{b}}$. In the first case T is already a canonical value. In the second case, note that the term $e^{\mathbf{b}}$ is itself a canonical term and a strict subterm of T. By inversion of the typing rules, we have that $\vdash e^{\mathbf{b}} : \tau$ (note that the type of a differential application is the same as the type of its body, unlike the case of standard application). Hence either $e^{\mathbf{b}}$*reduces* (and therefore so does T), or it is a canonical value, i.e. $e^{\mathbf{b}}$ is of the form $\lambda x.w^{\mathbf{b}}$. But in the last case then $T = \varepsilon^k \mathrm{D}(\lambda x.w^{\mathbf{b}}) \cdot s^{\mathbf{b}}$ and therefore $T \rightsquigarrow \varepsilon^k \lambda x.\frac{\partial w^{\mathbf{b}}}{\partial x}\left(s^{\mathbf{b}}\right)$.

- When $T = T_1 + T_2$, then either both T_1, T_2 are canonical values, and then so is T, or one of T_1, T_2 reduces, in which case so does T. $\square$

Definition 7.2.3. We extend typing judgements to well-formed terms by setting $\Gamma \underline{\vdash} \underline{t} : \tau$ whenever $\Gamma \vdash \mathbf{can}\,(t) : \tau$.

Corollary 7.2.1 (Subject reduction for well-formed terms)**.** Whenever $\Gamma \underline{\vdash} \underline{t} : \tau$ and $\underline{t} \underline{\rightsquigarrow} \underline{t'}$, then $\Gamma \underline{\vdash} \underline{t'} : \tau$.

Proof. Since $\underline{t} \underline{\rightsquigarrow} \underline{t'}$, there is some canonical form $T = \mathbf{can}\,(\underline{t})$ such that $T \rightsquigarrow t'' \sim_\varepsilon t'$, and furthermore $\Gamma \vdash T : \tau$. By definition of Theorem 7.2.1, we have that $\Gamma \vdash t'' : \tau$ and therefore by Proposition 7.2.3 $\Gamma \vdash \mathbf{can}\,(t'') : \tau$, from which it follows that $\Gamma \underline{\vdash} \underline{t'} : \tau$. $\square$

Corollary 7.2.2 (Progress for well-formed terms)**.** Whenever $\Gamma \underline{\vdash} \underline{t} : \tau$ then either $\underline{t} \underline{\rightsquigarrow} \underline{t'}$ or every canonical form $\mathbf{can}\,(\underline{t})$ is a canonical value.

7.2.1 Strong Normalisation

With our typing rules in place, we set out to show that λ_ε is strongly normalising. Our proof follows the structure of Ehrhard and Regnier's [37] and Vaux's[105], which use an adaptation of the well-known argument by reducibility candidates. Our proof will be somewhat simpler, however, due to two main reasons: first, we are not concerning ourselves with terms with coefficients on some general rig; and second, we have defined unrestricted and canonical terms as inductive types, and so we can freely use induction on the syntax of our terms. We will need some auxiliary results, which we prove now.

Lemma 7.2.1. Given an unrestricted term t, there are only finitely many terms t' such that $t \rightsquigarrow t'$.

Proof. Since our reduction relation is defined by simple induction on the syntax of t, it suffices to observe that any term t will only contain a finite number of applications where reduction may take place. $\square$

Lemma 7.2.2. Given a well-formed term $\underline{t}$, there are only finitely many canonical terms T such that $T \sim_\varepsilon \underline{t}$.

Proof. By Theorem 7.1.2, we know that any two canonical forms for $\underline{t}$ must be permutatively equivalent. But any term has a finite number of permutative equivalence classes, hence a term $\underline{t}$ only has finitely many canonical forms. $\square$

As a corollary of the two previous results, whenever a well-formed term $\underline{t}$ is strongly normalising, by König's lemma there is a longest sequence of well-formed terms $\underline{t} = \underline{t_1}, \underline{t_2}, \ldots, \underline{t_n}$ such that $\mathbf{can}\,(t_i) \rightsquigarrow t_i' \sim_\varepsilon t_{i+1}$. We write $|\underline{t}|$ to indicate the length of this sequence. The following result is then immediate.

Lemma 7.2.3. Whenever $\underline{t}$ is strongly normalising and $\underline{t} \rightsquigarrow \underline{t'}$, we have $|\underline{t}| > |\underline{t'}|$.

Lemma 7.2.4. A term $\underline{s+t}$ is strongly normalising if and only if $\underline{s}, \underline{t}$ are strongly normalising.

Proof. The proof in the first direction proceeds by induction on $|s+t|$.

Suppose $\underline{s+t}$ is strongly normalising. and suppose S, T are canonical forms for $\underline{s}, \underline{t}$ respectively. Then $S+T$ is a canonical form for $\underline{s+t}$ (up to associativity of addition), and any sequence of reductions

If $|\underline{s+t}| = 0$ it follows that its canonical form $S+T$ does not reduce, and hence neither do the separate $S+T$, thus $\underline{s}, \underline{t}$ are normal forms.

On the other hand, suppose $S \rightsquigarrow s', T \rightsquigarrow t'$ then $\underline{s+t} \underline{\rightsquigarrow} \underline{s'+t'}$ which is therefore strongly normalising, with $|\underline{s'+t'}| < |t|$. Hence, by induction, $\underline{s'}$ and $\underline{t'}$ are strongly normalising for any choices of canonical forms S, T and reducts s', t'.

In the opposite direction, the proof follows similarly by induction on $|s|+|t|$. The base case is equally trivial. For the inductive step, since any canonical form for $\underline{s+t}$ is (up to commutativity of addition) of the form $S+T$, with S, T being canonical forms for $\underline{s}, \underline{t}$ respectively, it follows that if $\underline{s+t} \underline{\rightsquigarrow} \underline{e}$. Without loss of generality, we assume that $S \underline{\rightsquigarrow} S'$ with $\mathbf{can}\,(\underline{e}) = S' + T$. Then $\underline{s} \underline{\rightsquigarrow} \underline{S'}$ and $\underline{e} = \underline{S'+t}$. Now $|\underline{S'}| < |\underline{s}|$, and so we apply the induction hypothesis and obtain that $\underline{e}$ must be strongly normalising, and therefore so is $\underline{s+t}$. ☐

Lemma 7.2.5. A term $\underline{\varepsilon s}$ is strongly normalising if and only if $\underline{s}$ is.

Definition 7.2.4. For every type τ we introduce a set $\mathcal{R}_\tau$ of well-formed terms of type τ. We do so by induction on τ.

- Whenever $\tau = \mathbf{t}$ is a primitive type, $\underline{s} \in \mathcal{R}_{\mathbf{t}}$ if and only if $\underline{s}$ is strongly normalising.

- Whenever $\tau = \sigma_1 \Rightarrow \sigma_2$, $\underline{s} \in \mathcal{R}_{\sigma_1 \Rightarrow \sigma_2}$ if and only if for any additive term $\underline{t^*} \in \mathcal{R}_{\sigma_1}$ and for any sequence $\underline{v_1^{\mathbf{b}}}, \ldots, \underline{v_n^{\mathbf{b}}}$ of basic terms $\underline{v_i^{\mathbf{b}}} \in \mathcal{R}_{\sigma_1}$ of length $n \geq 0$ we have $\underline{\left(\mathrm{D}^n(s) \cdot (v_1^{\mathbf{b}}, \ldots, v_i^{\mathbf{b}})\right)\ t^*} \in \mathcal{R}_{\sigma_2}$

If $\underline{t} \in \mathcal{R}_\tau$ we will often just say that $\underline{t}$ is **reducible** if the choice of τ is clear from the context.

Lemma 7.2.6. Whenever $\underline{t} \in \mathcal{R}_\tau$, then for any two distinct variables x, y the renaming $\underline{t\,[y/x]}$ is also in $\mathcal{R}_\tau$.

Proof. Straightforward induction on τ. □

Lemma 7.2.7. Whenever $\underline{t} \in \mathcal{R}_\tau$, then $\underline{t}$ is strongly normalising.

Proof. By induction on τ. When τ is a primitive type, the result follows trivially. Let $\tau = \sigma_1 \Rightarrow \sigma_2$ and $\underline{t} \in \mathcal{R}_\tau$. By the induction hypothesis we know that for all $\underline{u} \in \mathcal{R}_{\sigma_2}$ the application $\underline{t\ u}$ is strongly normalising.

Now suppose some canonical form of $\underline{t}$ reduces, that is, $T \rightsquigarrow t'$. Since $\mathbf{can}\,(t\ u^*) = \mathbf{ap}(\mathbf{can}\,(t)\,, \mathbf{pri}(u^*)) = \mathbf{ap}(T, \mathbf{can}\,(u^*))$, it follows that $\mathbf{can}\,(t\ u) \rightsquigarrow^+ \mathbf{ap}(t', \mathbf{pri}(u))$. Hence if there were any infinite sequence of reductions starting from $\underline{t}$, so would there be an infinite sequence of reductions starting from $\underline{t\ u}$. Since $\underline{t\ u}$ is strongly normalising, this must be impossible and so $\underline{t}$ must be strongly normalising as well. □

Lemma 7.2.8. Whenever $\underline{s}, \underline{t} \in \mathcal{R}_\tau$, then both $\underline{s+t}, \underline{\varepsilon s}$ are in $\mathcal{R}_\tau$. Conversely, whenever $\underline{s+t}$ is in $\mathcal{R}_\tau$ then so are $\underline{s}, \underline{t}$.

Proof. When τ is a primitive type the proof is a straightforward corollary of Lemmas 7.2.4 and 7.2.5.

When $\tau = \sigma_1 \Rightarrow \sigma_2$, consider an additive term $\underline{u^*} \in \mathcal{R}_{\sigma_1}$. We ask whether the application $\underline{(s+t)\ u^*}$ is in $\mathcal{R}_{\sigma_2}$. But note that $\mathbf{can}\,((s+t)\ u^*)$ is equal to $\mathbf{can}\,(s\ u^*) + \mathbf{can}\,(t\ u^*)$ (modulo commutativity of the sum) and therefore $\underline{(s+t)\ u^*} = \underline{s\ u^*} + \underline{t\ u^*}$. Since $\underline{s\ u^*}$ and $\underline{t\ u^*}$ are both in $\mathcal{R}_{\sigma_2}$, it follows by the induction hypothesis that so is $\underline{(s+t)\ u^*}$. The same reasoning shows that $\underline{(\mathrm{D}(s+t) \cdot v^{\mathbf{b}})\ u^*}$ is in $\mathcal{R}_{\sigma_2}$.

The proof for ε follows by a simpler but otherwise identical procedure.

On the opposite direction, consider a sum $\underline{s+t} \in \mathcal{R}_\tau$. When τ is a primitive type then $\underline{s+t}$ is strongly normalising and therefore so are $\underline{s}, \underline{t}$, hence they are regular. On the other hand, if $\tau = \sigma_1 \Rightarrow \sigma_2$, we have that for any $e^* \in \mathcal{R}_{\sigma_1}$ the reducible $\underline{(s+t)\ e^*}$ is equal to $\underline{(s\ e^*) + (t\ e^*)}$. By the induction hypothesis, both $(s\ e^*)$ and $(t\ e^*)$ are regular. A similar argument proves that differential applications of s and t are also regular, thus $s, t \in \mathcal{R}_{\sigma_1 \Rightarrow \sigma_2}$. □

Lemma 7.2.9. Whenever $\underline{s} \in \mathcal{R}_{\sigma \Rightarrow \tau}$ and $\underline{t} \in \mathcal{R}_\sigma$ then $\underline{\mathrm{D}(s) \cdot t} \in \mathcal{R}_{\sigma \Rightarrow \tau}$.

Proof. Pick a canonical form T of t. The proof proceeds by induction on the number of summands of T. If $T = 0$ or $T = \varepsilon^k t^{\mathbf{b}}$ the result follows directly by definition of $\mathcal{R}_{\sigma \Rightarrow \tau}$.

Now suppose $T = \varepsilon^k t^{\mathbf{b}} + T'$. Then

$$\begin{aligned} \mathrm{D}(s) \cdot t &\sim_\varepsilon \mathrm{D}(s) \cdot (t^{\mathbf{b}} + T') \\ &\sim_\varepsilon \mathrm{D}(s) \cdot t^{\mathbf{b}} + \mathrm{D}(s) \cdot T' + \varepsilon \mathrm{D}(\mathrm{D}(s) \cdot T') \cdot t^{\mathbf{b}} \end{aligned}$$

Now $\mathrm{D}(s) \cdot t^{\mathbf{b}}$ is evidently reducible (as s is reducible by hypothesis and, by Lemma 7.2.8, $t^{\mathbf{b}}$ is also reducible), and by the induction hypothesis so is $\mathrm{D}(s) \cdot T'$, from which also follows that $\mathrm{D}(\mathrm{D}(s) \cdot T') \cdot t^{\mathbf{b}}$ is reducible as well. By Lemma 7.2.8 it follows that $\underline{t}$ is reducible. □

Corollary 7.2.3. A well-formed term $\underline{t}$ is in $\mathcal{R}_\tau$ if and only if some canonical form $T = \mathbf{can}\,(\underline{t})$ is of the form $\sum_{i=1}^{n} \varepsilon^{k_i} t_i^{\mathbf{b}}$ with $\underline{t_i^{\mathbf{b}}} \in \mathcal{R}_\tau$ for each $1 \leq i \leq n$.

Lemma 7.2.10. Whenever $\underline{t} \in \mathcal{R}_\tau, \underline{t} \rightsquigarrow^+ \underline{t'}$, then $\underline{t'} \in \mathcal{R}_\tau$.

Proof. We proceed by induction on τ. When τ is a primitive type, we have that $\underline{t}$ is strongly normalising and therefore so is $\underline{t'}$, hence $\underline{t'} \in \mathcal{R}_\tau$.

When $\tau = \sigma_1 \Rightarrow \sigma_2$, we pick some additive reducible term $\underline{e^*}$ and a sequence of reducible basic terms $\underline{u_1^{\mathbf{b}}}, \ldots, \underline{u_k^{\mathbf{b}}}$, and establish that $\left(\mathrm{D}^k(t') \cdot (u_1^{\mathbf{b}}, \ldots, u_k^{\mathbf{b}})\right)\ e^*$ is reducible. But this is immediate: since $\underline{t}$ reduces to $\underline{t'}$, then so does $\underline{\mathrm{D}^k(t) \cdot (u_1^{\mathbf{b}}, \ldots, u_k^{\mathbf{b}})\ e^*}$ reduce to $\underline{\mathrm{D}^k(t') \cdot (u_1^{\mathbf{b}}, \ldots, u_k^{\mathbf{b}})\ e^*}$ and, by induction, $\underline{\mathrm{D}^k(t') \cdot (u_1^{\mathbf{b}}, \ldots, u_k^{\mathbf{b}})\ e^*}$ is reducible. □

Definition 7.2.5. A basic term $t^{\mathbf{b}}$ is **neutral** whenever it is not a λ-abstraction. In other words, a basic term is neutral whenever it is of the form $x, (s\ t)$ or $\mathrm{D}(s) \cdot u$.

A canonical term T is neutral whenever it is of the form $\sum_{i=1}^{n} \varepsilon^{k_i} s_i^{\mathbf{b}}$, where each of the $s_i^{\mathbf{b}}$ are neutral. In particular, 0 is a neutral term.

A well-formed term $\underline{t}$ is neutral whenever some canonical form (and therefore all of its canonical forms) is neutral.

Lemma 7.2.11. Whenever $\underline{t}$ is neutral and every $\underline{t'}$ such that $\underline{t} \rightsquigarrow^+ \underline{t'}$ is in $\mathcal{R}_\tau$, then so is $\underline{t}$.

Proof. When τ is a primitive type the proof is immediate, as $\underline{t'} \in \mathcal{R}_\tau$ implies $\underline{t'}$ is strongly normalising and therefore so is $\underline{t}$.

When $\tau = \sigma_1 \Rightarrow \sigma_2$, we show the reasoning for standard application first. We select arbitrary $\underline{e^*}, \underline{u_1^{\mathbf{b}}}, \ldots, \underline{u_k^{\mathbf{b}}} \in \mathcal{R}_{\sigma_1}$ and show that whenever $\left(\mathrm{D}^k(t) \cdot (u_1^{\mathbf{b}}, \ldots, u_k^{\mathbf{b}})\right)\ e^*$ reduces then it reduces to a reducible term (hence our desired result will follow by induction on τ).

We prove this property by induction on $Q := |\underline{e^*}| + |\underline{u_1^{\mathbf{b}}}| + \ldots + |\underline{u_k^{\mathbf{b}}}|$ which is well-defined since, by hypothesis, all of the involved terms are strongly normalising.

When $Q = 0$ then all of our chosen terms are normal, and so, since t is neutral, if $\underline{\left(\mathrm{D}^k(t) \cdot (u_1^{\mathbf{b}}, \ldots, u_k^{\mathbf{b}})\right)\ e^*}$ reduces it must be that $\underline{t} \rightsquigarrow^+ \underline{t'}$. By hypothesis, $\underline{t'} \in \mathcal{R}_{\sigma_1 \Rightarrow \sigma_2}$ and therefore $\underline{\left(\mathrm{D}^k(t') \cdot (u_1^{\mathbf{b}}, \ldots, u_k^{\mathbf{b}})\right)\ e^*}$ is reducible.

When $Q > 0$, then a reduction may occur in t, in which case we apply the previous reasoning, or in one of the applied terms, in which case we apply the induction hypothesis on Q. induction hypothesis. $\square$

Lemma 7.2.12. If, for all $\underline{t^*} \in \mathcal{R}_{\sigma_1}$ where x does not appear free, the term $\underline{s\,[t^*/x]}$ is in $\mathcal{R}_{\sigma_2}$ and, for all $\underline{u^{\mathbf{b}}}$ where x does not appear free, the term $\underline{\left(\frac{\partial s}{\partial x}\left(u^{\mathbf{b}}\right)\right)[t^*/x]}$ is in $\mathcal{R}_{\sigma_2}$, then the term $\underline{\lambda x.s}$ is in $\mathcal{R}_{\sigma_1 \Rightarrow \sigma_2}$.

Proof. As a corollary of Lemma 7.2.8, it suffices to check the case when s is some basic term $s = s^{\mathbf{b}}$.

Pick any variable $y \neq$ z. Since the variable y itself is an additive term in $\mathcal{R}_{\sigma_1}$, by hypothesis we have $\underline{s\,[y/x]} \in \mathcal{R}_{\sigma_2}$. But since reducible terms are closed under renaming as per Lemma 7.2.6, this means that so is $\underline{s}$.

Now consider an arbitrary $\underline{e^*} \in \mathcal{R}_{\sigma_1}$. We show that the application $\underline{(\lambda x.s^{\mathbf{b}})\ e^*}$ is in $\mathcal{R}_{\sigma_2}$. As it is a neutral term, by Lemma 7.2.11 it suffices to prove that every one-step reduct of $(\lambda x.s^{\mathbf{b}})\ e^*$ is in $\mathcal{R}_{\sigma_2}$. We do so by induction on $|s^{\mathbf{b}}| + |e^*|$. The term $(\lambda x.s^{\mathbf{b}})\ e^*$ reduces to one of:

- $s^{\mathbf{b}}\,[e^*/x]$, which by hypothesis is a representative of a term in $\mathcal{R}_{\sigma_2}$.
- $(\lambda x.s')\ e^*$ with $s^{\mathbf{b}} \rightsquigarrow s'$. Then $s' \in \mathcal{R}_{\sigma_2}$ and since $|s'| < |s^{\mathbf{b}}|$ we apply our induction on $|s^{\mathbf{b}}|$ to obtain that $(\lambda x.s')\ e^* \in \mathcal{R}_{\sigma_2}$.
- $(\lambda x.s^{\mathbf{b}})\ e'$ with $e^* \rightsquigarrow e'$. Then $e' \in \mathcal{R}_{\sigma_1}$ and since $|e'| < |e^*|$ we apply our induction on $|e^*|$ to obtain that $(\lambda x.s^{\mathbf{b}})\ e' \in \mathcal{R}_{\sigma_2}$.

By a similar argument we can show that $\underline{\left(\mathrm{D}^k(\lambda x.s^{\mathbf{b}}) \cdot (u_1^{\mathbf{b}}, \ldots, u_k^{\mathbf{b}})\right)\ e^*}$ is reducible, applying Lemma 7.2.11 and using induction in $|e^*| + |u_1^{\mathbf{b}}| + \ldots + |u_k^{\mathbf{b}}|$. $\square$

Theorem 7.2.3. Consider a well-formed term $\underline{t}$ which admits a typing of the form $x_1 : \sigma_1, \ldots, x_n : \sigma_n \vdash \underline{t} : \tau$ and assume given the following data:

- A sequence of basic terms $\underline{d_1^{\mathbf{b}}} \in \mathcal{R}_{\sigma_1}, \ldots, \underline{d_n^{\mathbf{b}}} \in \mathcal{R}_{\sigma_n}$.
- An arbitrary sequence of indices $i_1, \ldots, i_k \in \{1, \ldots, n\}$ (possibly with repetitions).
- A sequence of additive terms $\underline{s_1^*} \in \mathcal{R}_{\sigma_{i_1}}, \ldots, \underline{s_k^*} \in \mathcal{R}_{\sigma_{i_k}}$.

such that none of the variables $x_1, \dots, x_i$ appear free in the $d_i^{\mathbf{b}}, s_i^*$. Then the term

$$\underline{t'} = \underline{\left(\frac{\partial^k t}{\partial(x_{i_1}, \dots, x_{i_k})}(d_1^{\mathbf{b}}, \dots, d_k^{\mathbf{b}})\right)[s_1^*, \dots, s_n^*/x_1, \dots, x_n]}$$

is in $\mathcal{R}_\tau$.

Proof. Throughout the proof we will write $\overline{x_i}, \overline{s_i^*}, \overline{d_i^{\mathbf{b}}}$ as a shorthand for the corresponding sequences $x_1, \dots, x_n$, etc.

By definition of $\underline{\vdash}$, we know that there is some canonical form T (in fact any canonical form) of $\underline{t}$ such that $x_1 : \sigma_1, \dots, x_n : \sigma_n \vdash T : \tau$. We prove our property holds by induction on this typing derivation. Furthermore, by Lemma 7.2.8, it suffices to consider the case when T is in fact some basic term $t^{\mathbf{b}}$. We proceed now by case analysis on the last rule of the typing derivation:

- $t^{\mathbf{b}} = x_i$ (and therefore $\tau = \sigma_i$)

  If the sequence of indices $i_1, \dots, i_k$ is empty then the substitution $\underline{t'}$ is exactly equal to $\underline{s_i^*}$ and therefore $\underline{t'} \in \mathcal{R}_{\sigma_i}$.

  If the sequence of indices $i_1, \dots, i_k$ is exactly the sequence containing only i then since x_i does not appear free in the substituted term $d^{\mathbf{b}}$ then t' is differentially equivalent to $\left(\frac{\partial x_i}{\partial x_i}\left(d_i^{\mathbf{b}}\right)\right)$ and therefore $\underline{t'} = \underline{d_i^{\mathbf{b}}} \in \mathcal{R}_{\sigma_i}$.

  If the list of indices contains two or more indices, or does not contain i, then $t' \sim_\varepsilon 0$ and therefore $\underline{t'}$ is trivially in $\in \mathcal{R}_{\sigma_i}$ (either the derivative)

- $t^{\mathbf{b}} = \mathrm{D}(s^{\mathbf{b}}) \cdot e^{\mathbf{b}}$

  Applying Lemma 7.1.9, we know that the term

  $$\left(\left(\partial^k t / \partial x_{i_1} \dots \partial x_{i_k}\right)\left(d_1^{\mathbf{b}}, \dots, d_k^{\mathbf{b}}\right)\right)[s_1^*, \dots, s_n^*/x_1, \dots, x_n]$$

  is equivalent to a sum of terms of the form

  $$\left(\varepsilon^z \mathrm{D}^l\left(v\left[\overline{s_i^*}/\overline{x_i}\right]\right) \cdot \left(w_1\left[\overline{s_i^*}/\overline{x_i}\right], \dots, w_l\left[\overline{s_i^*}/\overline{x_i}\right]\right)\right)$$

  Again by Lemmas 7.2.8 and 7.2.9, it suffices to show that each of the $v\left[\overline{s_i^*}/\overline{x_i}\right]$, $w_j\left[\overline{s_i^*}/\overline{x_i}\right]$ are reducible. But by Lemma 7.1.10, we know that v has the form

  $$\frac{\partial^p s^{\mathbf{b}}}{\partial(x_{j_1}, \dots, x_{j_m})}\left(d_{j_1}^{\mathbf{b}}, \dots, d_{j_m}^{\mathbf{b}}\right)$$

  Since $s^{\mathbf{b}}$ is a subterm of $t^{\mathbf{b}}$ its typing derivation is therefore a sub-derivation of the one for $t^{\mathbf{b}}$. We apply the induction hypothesis, obtaining that $\underline{v\left[\overline{s_i^*}/\overline{x_i}\right]}$ is reducible (as each of the $d_i^{\mathbf{b}}$ are reducible). By a similar argument, each of the $\underline{w_j\left[\overline{s_i^*}/\overline{x_i}\right]}$ are reducible as well, and therefore so is $\underline{t^{\mathbf{b}}}$.

- $t^{\mathbf{b}} = (s^{\mathbf{b}}\ e^*)$

  Applying Lemma 7.1.9, we know that the term

  $$\left(\left(\partial^k t/\partial x_{i_1} \dots \partial x_{i_k}\right)\left(d_1^{\mathbf{b}}, \dots, d_k^{\mathbf{b}}\right)\right)[s_1^*, \dots, s_n^*/x_1, \dots, x_n]$$

  is equivalent to a sum of terms of the form

  $$\left(\varepsilon^z \mathrm{D}^l \left(v\left[\overline{s_i^*}/\overline{x_i}\right]\right) \cdot \left(w_1\left[\overline{s_i^*}/\overline{x_i}\right], \dots, w_l\left[\overline{s_i^*}/\overline{x_i}\right]\right)\right)\ \left(e^*\left[\overline{s_i^*}/\overline{x_i}\right]\right)$$

  Again by Lemma 7.2.8 it suffices to show that every such term is reducible. First, by Lemma 7.1.10, we know that v has the form

  $$\frac{\partial^p s^{\mathbf{b}}}{\partial(x_{j_1}, \dots, x_{j_m})}\left(d_{j_1}^{\mathbf{b}}, \dots, d_{j_m}^{\mathbf{b}}\right)$$

  Since $s^{\mathbf{b}}$ is a subterm of $t^{\mathbf{b}}$ its typing derivation is therefore a sub-derivation of the one for $t^{\mathbf{b}}$. We apply the induction hypothesis, obtaining that $\underline{v\left[\overline{s_i^*}/\overline{x_i}\right]}$ is reducible (as each of the $d_i^{\mathbf{b}}$ are reducible). By a similar argument, each of the $\underline{w_j\left[\overline{s_i^*}/\overline{x_i}\right]}$ are reducible as well, as is $\underline{e^*\left[\overline{s_i^*}/\overline{x_i}\right]}$. Thus, by Lemma 7.2.9, the entire differential application is reducible.

  But since $t^{\mathbf{b}}$ is an application of the reducible term

  $$\varepsilon^z \mathrm{D}^l \left(v\left[\overline{s_i^*}/\overline{x_i}\right]\right) \cdot \left(w_1\left[\overline{s_i^*}/\overline{x_i}\right], \dots, w_l\left[s_1^*, \dots, s_n^*/x_1, \dots, x_n\right]\right)$$

  to the reducible term $e^*[s_1^*, \dots, s_n^*/x_1, \dots, x_n]$, it follows then that $\underline{t^{\mathbf{b}}}$ is itself reducible.

- $t^{\mathbf{b}} = \lambda y.s^{\mathbf{b}}$

  Pick fresh variables $z_1, \dots, z_n$. By the induction hypothesis, we know that both of the following substitutions are reducible:

  $$\left(t^{\mathbf{b}}\right)[z_1, \dots, z_n, e^*/x_1, \dots, x_n, y]$$
  $$\left(\frac{\partial t^{\mathbf{b}}}{\partial y}\left(d^{\mathbf{b}}\right)\right)[z_1, \dots, z_n, e^*/x_1, \dots, x_n, y]$$

  But since reducible terms are closed under renaming, then the terms $t^{\mathbf{b}}[e^*/y]$ and $\left(\frac{\partial t^{\mathbf{b}}}{\partial y}\left(d^{\mathbf{b}}\right)\right)[e^*/y]$ are also reducible. Hence, by Lemma 7.2.12, the term $\lambda y.s^{\mathbf{b}}$ is reducible. □

Corollary 7.2.4 (Strong normalisation)**.** Whenever a closed well-formed term is typable with type $\vdash \underline{t} : \tau$, it is strongly normalising.

Proof. By the previous result, we know that any such term satisfies $\underline{t} \in \mathcal{R}_\tau$, and therefore $\underline{t}$ is strongly normalising. □

7.3 Semantics

It is a well-known result that the differential λ-calculus can be soundly interpreted in any differential λ-category, that is to say, any Cartesian differential category where differentiation "commutes with" abstraction (in the sense of [22, Definition 4.4]).

The exact same result holds for the difference λ-calculus and difference λ-categories. In what follows we will consider a fixed difference λ-category $\mathbf{C}$, and proceed to define interpretations for the types, contexts and terms of the simply-typed λ_ε-calculus.

Definition 7.3.1. Given a $\mathbf{t}$-indexed family of objects $O_\mathbf{t}$, we define the interpretation $[\![\tau]\!]$ of a type τ by induction on its structure by setting $[\![\mathbf{t}]\!] \coloneqq O_\mathbf{t}$, $[\![\sigma \Rightarrow \tau]\!] \coloneqq [\![\sigma]\!] \Rightarrow [\![\tau]\!]$. We lift the interpretation of types to contexts in the usual way. Or, more formally, we have: $[\![\cdot]\!] \coloneqq \mathbf{1}$, $[\![\Gamma, x : \tau]\!] \coloneqq [\![\Gamma]\!] \times [\![\tau]\!]$.

Definition 7.3.2. Given a well-typed unrestricted λ_ε-term $\Gamma \vdash t : \tau$, we define its interpretation $[\![t]\!] : [\![\Gamma]\!] \to [\![\tau]\!]$ inductively as in Figure 7.10 below. When Γ and τ are irrelevant or can be inferred from the context, we will simply write $[\![t]\!]$.

$$\begin{array}{rcll}
[\![(x_i : \tau_i)_{i=1}^n \vdash x_k : \tau_k]\!] & \coloneqq & \pi_2 \circ \pi_1^{n-k} & : \prod_{i=1}^n [\![\tau_i]\!] \to [\![\tau_k]\!] \\
[\![\Gamma \vdash 0 : \tau]\!] & \coloneqq & 0 & : [\![\Gamma]\!] \to [\![\tau]\!] \\
[\![\Gamma \vdash s + t : \tau]\!] & \coloneqq & [\![s]\!] + [\![t]\!] & : [\![\Gamma]\!] \to [\![\tau]\!] \\
[\![\Gamma \vdash \varepsilon t : \tau]\!] & \coloneqq & \varepsilon\, [\![t]\!] & : [\![\Gamma]\!] \to [\![\tau]\!] \\
[\![\Gamma \vdash \lambda x.t : \sigma \Rightarrow \tau]\!] & \coloneqq & \Lambda\, [\![t]\!] & : [\![\Gamma]\!] \to [\![\sigma]\!] \Rightarrow [\![\tau]\!] \\
[\![\Gamma \vdash (s\ t) : \tau]\!] & \coloneqq & \mathbf{ev} \circ \langle [\![s]\!], [\![t]\!] \rangle & : [\![\Gamma]\!] \to [\![\tau]\!] \\
[\![\Gamma \vdash \mathrm{D}(s) \cdot t : \sigma \Rightarrow \tau]\!] & \coloneqq & \Lambda(\partial[\Lambda^-([\![s]\!])] \circ \langle \mathbf{id}, \langle 0, [\![t]\!] \circ \pi_1 \rangle\rangle) & : [\![\Gamma]\!] \to [\![\tau]\!]
\end{array}$$

Figure 7.10: Interpreting λ_ε in $\mathbf{C}$

Lemma 7.3.1.

$$\partial[\Lambda^- f] \circ \langle \mathbf{id}, \langle 0, g \circ \pi_1 \rangle\rangle = \partial[\mathbf{ev}] \circ \langle , \rangle \circ$$

Lemma 7.3.2. Define the relation $\sim_{[\![]\!]} \subseteq \Lambda_\varepsilon$ by letting $s \sim_{[\![]\!]} t$ whenever there exist Γ, τ such that $[\![\Gamma \vdash s : \tau]\!] = [\![\Gamma \vdash t : \tau]\!]$. Then the relation $\sim_{[\![]\!]}$ is contextual.

Theorem 7.3.1. Whenever $s \sim_\varepsilon t$ are equivalent unrestricted terms that admit typing derivations $\Gamma \vdash s : \tau, \Gamma \vdash t : \tau$, then their interpretations are identical, that is to say:

$$[\![\Gamma \vdash s : \tau]\!] = [\![\Gamma \vdash t : \tau]\!]$$

Proof. The result can be equivalently stated as "$\sim_{[\![]\!]}$ contains $\sim_\varepsilon$". Since $\sim_\varepsilon$ is defined as the least contextual equivalence relation that contains $\sim^1_\varepsilon$, and $\sim_{[\![]\!]}$ is both contextual and an equivalence relation, it suffices to prove that it contains $\sim^1_\varepsilon$.

We examine the rules in Figure 7.2 and show that they all hold, by checking that each rule verifies $[\![\mathrm{LHS}]\!] = [\![\mathrm{RHS}]\!]$. The first and second blocks are trivial, as the rules correspond precisely to stating that $\mathbf{C}$ is a Cartesian closed left-additive category with an infinitesimal extension which is compatible with the Cartesian structure. The third block is a trivial consequence of [$\mathbf{C\partial C.1}$] and so we will omit it as well, but we give explicit proofs for some of the properties of the fourth block.

First we show a general equivalence between syntactic and semantic second derivatives which will simplify the task considerably.

$$\begin{aligned}
&\Lambda^-([\![\mathrm{D}(\mathrm{D}(s)\cdot u)\cdot v]\!])\\
&\quad= \partial[\Lambda^-([\![\mathrm{D}(s)\cdot u]\!])] \circ \langle \mathbf{id}, \langle 0, [\![v]\!]\circ\pi_1\rangle\rangle\\
&\quad= \partial[\partial[\Lambda^-([\![s]\!])] \circ \langle \mathbf{id}, \langle 0, [\![u]\!]\circ\pi_1\rangle\rangle]\, \langle \mathbf{id}, \langle 0, [\![v]\!]\circ\pi_1\rangle\rangle\\
&\quad= \partial[\partial[\Lambda^-([\![s]\!])]] \circ \langle\langle \mathbf{id}, \langle 0, [\![u]\!]\circ\pi_1\rangle\rangle\circ\pi_1, \partial[\langle \mathbf{id}, \langle 0, [\![u]\!]\circ\pi_1\rangle\rangle]\rangle \circ \langle \mathbf{id}, \langle 0, [\![v]\!]\circ\pi_1\rangle\rangle\\
&\quad= \partial[\partial[\Lambda^-([\![s]\!])]]\\
&\qquad\circ \langle\langle \mathbf{id}, \langle 0, [\![u]\!]\circ\pi_1\rangle\rangle, \langle \partial[\mathbf{id}] \circ \langle \mathbf{id}, \langle 0, [\![v]\!]\circ\pi_1\rangle\rangle, \partial[[\![v]\!]\circ\pi_1]\circ\langle \mathbf{id}, \langle 0, [\![v]\!]\circ\pi_1\rangle\rangle\rangle\rangle\\
&\quad= \partial[\partial[\Lambda^-([\![s]\!])]] \circ \langle\langle \mathbf{id}, \langle 0, [\![u]\!]\circ\pi_1\rangle\rangle, \langle\langle 0, [\![v]\!]\circ\pi_1\rangle, \partial[[\![v]\!]]\circ\langle\pi_1, 0\rangle\rangle\rangle\\
&\quad= \partial[\partial[\Lambda^-([\![s]\!])]] \circ \langle\langle \mathbf{id}, \langle 0, [\![u]\!]\circ\pi_1\rangle\rangle, \langle\langle 0, [\![v]\!]\circ\pi_1\rangle, 0\rangle\rangle
\end{aligned}$$

With the above, most of the conditions become trivial. For example, we can prove that regularity of the syntactic derivative follows from semantic regularity (that is to say, [$\mathbf{C\partial C.2}$]) with the following calculation:

- $\mathrm{D}(s)\cdot(u+v) \sim^1_\varepsilon \mathrm{D}(s)\cdot u + \mathrm{D}(s)\cdot v + \varepsilon(\mathrm{D}(\mathrm{D}(s)\cdot u)\cdot v)$

$$\begin{aligned}
[\![\mathrm{D}(s)\cdot(u+v)]\!] &= \Lambda(\partial[f]\circ\langle \mathbf{id}, \langle 0, [\![u]\!]\circ\pi_1 + [\![v]\!]\circ\pi_1\rangle\rangle)\\
&= \Lambda(\partial[f]\circ\langle \mathbf{id}, \langle 0, [\![u]\!]\circ\pi_1\rangle + \langle 0, [\![v]\!]\circ\pi_1\rangle\rangle)\\
&= \Lambda\big[(\partial[f]\circ\langle \mathbf{id}, \langle 0, [\![u]\!]\circ\pi_1\rangle\rangle)\\
&\quad + (\partial[f]\circ\langle \mathbf{id} + \varepsilon(\langle 0, [\![u]\!]\circ\pi_1\rangle), \langle 0, [\![v]\!]\circ\pi_1\rangle\rangle)\big]\\
&= \Lambda\left[\partial[f]\circ\langle \mathbf{id}, \langle 0, [\![u]\!]\circ\pi_1\rangle\rangle\right]\\
&\quad + \Lambda\left[(\partial[f]\circ\langle \mathbf{id}, \langle 0, [\![v]\!]\circ\pi_1\rangle\rangle)\right]\\
&\quad + \varepsilon\Lambda\left[\partial[\partial[f]]\circ\langle\langle \mathbf{id}, \langle 0, [\![v]\!]\circ\pi_1\rangle\rangle, \langle\langle 0, [\![u]\!]\circ\pi_1\rangle, 0\rangle\rangle\right]\,\big]\\
&= \Lambda\left[\partial[f]\circ\langle \mathbf{id}, \langle 0, [\![u]\!]\circ\pi_1\rangle\rangle\right]\\
&\quad + \Lambda\left[(\partial[f]\circ\langle \mathbf{id}, \langle 0, [\![v]\!]\circ\pi_1\rangle\rangle)\right]\\
&\quad + \varepsilon\Lambda\left[\partial[\partial[f]]\circ\langle\langle \mathbf{id}, \langle 0, [\![u]\!]\circ\pi_1\rangle\rangle, \langle\langle 0, [\![v]\!]\circ\pi_1\rangle, 0\rangle\rangle\right]\,\big]\\
&= [\![\mathrm{D}(s)\cdot u)]\!] + [\![\mathrm{D}(s)\cdot v]\!] + [\![\varepsilon(\mathrm{D}(\mathrm{D}(s)\cdot u)\cdot v)]\!]
\end{aligned}$$

Most of the other conditions follow from similar arguments. For example, commutativity of the syntactic second derivative follows from commutativity of the semantic second derivative. The third and fifth conditions, dealing with infinitesimal extensions, may seem harder to prove, but they are both corollaries of axiom **[C∂C.6]**, as we showed in Lemma 6.1.6. It remains to show that the syntactic derivative condition holds; this is not hard, but we do it explicitly as the derivative condition is such a central notion.

- $s\ (t + \varepsilon e) \sim^1_\varepsilon (s\ t) + \varepsilon((\mathrm{D}(s) \cdot e)\ t)$

$$\begin{aligned}
&[\![s\ (t+\varepsilon e)]\!] \\
&\quad = \mathbf{ev} \circ \langle [\![s]\!], [\![t]\!] + \varepsilon\, [\![e]\!] \rangle \\
&\quad = \Lambda^-\, [\![s]\!] \circ \langle \mathbf{id}, [\![t]\!] + \varepsilon\, [\![e]\!] \rangle \\
&\quad = \left(\Lambda^-\, [\![s]\!] \circ \langle \mathbf{id}, [\![t]\!] \rangle\right) + \varepsilon \left(\partial[\Lambda^-\, [\![s]\!]] \circ \langle \langle \mathbf{id}, [\![t]\!] \rangle, \langle 0, [\![e]\!] \rangle \rangle\right) \\
&\quad = [\![s\ t]\!] + \varepsilon \left[\left(\partial[\Lambda^-\, [\![s]\!]] \circ \langle \mathbf{id}, \langle 0, [\![e]\!] \circ \pi_1 \rangle \rangle\right) \circ \langle \mathbf{id}, [\![t]\!] \rangle\right] \\
&\quad = [\![s\ t]\!] + \varepsilon \left[\mathbf{ev} \circ \left\langle \Lambda \left(\partial[\Lambda^-\, [\![s]\!]] \circ \langle \mathbf{id}, \langle 0, [\![e]\!] \circ \pi_1 \rangle \rangle\right), [\![t]\!] \right\rangle\right] \\
&\quad = [\![s\ t]\!] + \varepsilon \left[\mathbf{ev} \circ \langle [\![\mathrm{D}(s) \cdot e]\!], [\![t]\!] \rangle\right] \\
&\quad = [\![s\ t]\!] + \varepsilon\, [\![(\mathrm{D}(s) \cdot e)\ t]\!] \qquad \square
\end{aligned}$$

Lemma 7.3.3. Let t be some unrestricted λ_ε-term. The following properties hold:

i. If $\Gamma \vdash t : \tau$ and x does not appear in Γ then $[\![\Gamma, x : \sigma \vdash t : \tau]\!] = [\![\Gamma \vdash t : \tau]\!] \circ \pi_1$

ii. If $\Gamma, x : \sigma_1, y : \sigma_2 \vdash t : \tau$ then $[\![\Gamma, y : \sigma_2, x : \sigma_1 \vdash t : \tau]\!] = [\![\Gamma, x : \sigma_1, y : \sigma_2 \vdash t : \tau]\!] \circ$ **sw**

The morphism **sw** above is the obvious isomorphism between $(A \times B) \times C$ and $(A \times C) \times B$, which we can define explicitly by:

$$\mathbf{sw} \coloneqq \langle \langle \pi_{11}, \pi_2 \rangle, \pi_{21} \rangle : (A \times B) \times C \to (A \times C) \times B$$

Lemma 7.3.4. Let $\Gamma, x : \tau \vdash s : \sigma$, with s some unrestricted λ_ε-term. Then:

i. Whenever $\Gamma, x : \tau \vdash t : \tau$, then $[\![s\,[t/x]]\!]_\Gamma = [\![s]\!]_{\Gamma, x:\tau} \circ \left\langle \pi_1, [\![t]\!]_{\Gamma, x:\tau} \right\rangle$

ii. Whenever $\Gamma \vdash t : \tau$, then $\left[\!\left[\frac{\partial s}{\partial x}\,(t)\right]\!\right]_{\Gamma, x:\tau} = \partial[[\![s]\!]_{\Gamma, x:\tau}] \circ \langle \mathbf{id}, \langle 0, [\![t]\!]_\Gamma \circ \pi_1 \rangle \rangle$. Or, using the notation in Definition 6.4.2, $\left[\!\left[\frac{\partial s}{\partial x}\,(t)\right]\!\right] = [\![s]\!] \star [\![t]\!]$.

Proof. The proof follows roughly the structure of [22, Theorem 4.11], taking into account the differences in our notion of differential substitution. Note also that we prove substitution in the case that the variable x is not free in t. This is because we require this (stronger) form of substitution to write $e\,[x + \varepsilon(t)/x]$ in some cases of differential substitution.

Both properties will follow by induction on the typing derivation of s. The only non-trivial case for the first one is differential application. For this, we must show that $[\![\mathrm{D}(s\,[t/x])\cdot(u\,[t/x])]\!]$ is equal to $[\![\mathrm{D}(s)\cdot u]\!]\circ\langle\mathbf{id},[\![t]\!]\rangle$. Expanding the term we obtain:

$$\begin{aligned}[\![\mathrm{D}(s\,[t/x])\cdot(u\,[t/x])]\!] &= \Lambda\Big(\Lambda^-([\![s\,[t/x]]\!])\star[\![u\,[t/x]]\!]\Big)\\ &= \Lambda\Big(\Lambda^-([\![s]\!]\circ\langle\pi_1,\langle 0,[\![u]\!]\rangle\rangle)\star([\![u]\!]\circ\langle\pi_1,\langle 0,[\![u]\!]\rangle\rangle)\Big)\end{aligned}$$

By Lemma 6.4.3(iii.), the above expression can be written as:

$$\Lambda\Big((\Lambda^-\,[\![s]\!])\star[\![u]\!]\Big)\circ\langle\pi_1,[\![t]\!]\rangle$$

which concludes the proof.

We show now the cases for differential substitution.

- $s=x$ Then $[\![s]\!]=\pi_2$ and

$$\partial[[\![s]\!]]\circ\langle\mathbf{id},\langle 0,[\![t]\!]\circ\pi_1\rangle\rangle=\pi_{22}\circ\langle\mathbf{id},\langle 0,[\![t]\!]\circ\pi_1\rangle\rangle=[\![t]\!]\circ\pi_1=[\![\Gamma,x:\tau\vdash t:\tau]\!]$$

- $s=y\neq x$

  Then $[\![s]\!]=\pi_2\circ\pi_1^n\circ\pi_1$ and

$$\partial[[\![s]\!]]\circ\langle\mathbf{id},\langle 0,[\![t]\!]\circ\pi_1\rangle\rangle=\pi_2\circ\pi_1^n\circ\pi_1\circ\pi_2\circ\langle\mathbf{id},\langle 0,[\![t]\!]\circ\pi_1\rangle\rangle=0=[\![\Gamma,x:\tau\vdash 0:\tau]\!]$$

- $s=\varepsilon s_1$

  Then $[\![s]\!]=\varepsilon\,[\![s_1]\!]$ and

$$\partial[[\![s]\!]]\circ\langle\mathbf{id},\langle 0,[\![t]\!]\circ\pi_1\rangle\rangle=\varepsilon(\partial[[\![s_1]\!]]\circ\langle\mathbf{id},\langle 0,[\![t]\!]\circ\pi_1\rangle\rangle)=\varepsilon\left[\!\!\left[\frac{\partial s_1}{\partial x}(t)\right]\!\!\right]=\left[\!\!\left[\frac{\partial(\varepsilon s_1)}{\partial x}(t)\right]\!\!\right]$$

- The case $s=\sum_{i=1}^n s_i$ follows by a similar argument as the previous one.

- $s=\lambda y.s_1:\sigma_1\Rightarrow\sigma_2$

  Then $\Gamma,x:\tau,y:\sigma_1\vdash s_1:\sigma_2$ and therefore, by the induction hypothesis, we know that:

$$\begin{aligned}\left[\!\!\left[\frac{\partial s_1}{\partial x}(t)\right]\!\!\right]_{\Gamma,x:\tau,y:\sigma_1} &= \left[\!\!\left[\frac{\partial s_1}{\partial x}(t)\right]\!\!\right]_{\Gamma,y:\sigma_1,x:\tau}\circ\mathbf{sw}\\ &= \partial[[\![s_1]\!]_{\Gamma,y:\sigma_1,x:\tau}]\circ\Big\langle\mathbf{id},\Big\langle 0,[\![t]\!]_{\Gamma,y:\sigma_1,x:\tau}\Big\rangle\Big\rangle\circ\mathbf{sw}\\ &= \partial[[\![s_1]\!]_{\Gamma,x:\sigma_1,y:\tau}]\circ(\mathbf{sw}\times\mathbf{sw})\circ\langle\mathbf{id},\langle 0,[\![t]\!]_\Gamma\circ\pi_{11}\rangle\rangle\circ\mathbf{sw}\\ &= \partial[[\![s_1]\!]_{\Gamma,x:\sigma_1,y:\tau}]\circ\langle\mathbf{sw},\langle\langle 0,[\![t]\!]_\Gamma\circ\pi_{11}\rangle,0\rangle\rangle\circ\mathbf{sw}\end{aligned}$$

$$= \partial[[\![s_1]\!]_{\Gamma,x:\sigma_1,y:\tau}] \circ \langle \mathbf{id}, \langle\langle 0, [\![t]\!]_\Gamma \circ \pi_{11}\rangle, 0\rangle\rangle$$

Obtaining the final result is just a matter of applying this identity and Remark 6.4.1.

$$\begin{aligned}
\partial[[\![s]\!]] \circ \langle \mathbf{id}, \langle 0, [\![t]\!] \circ \pi_1\rangle\rangle &= \partial[\Lambda([\![s_1]\!])] \circ \langle \mathbf{id}, \langle 0, [\![t]\!] \circ \pi_1\rangle\rangle \\
&= \Lambda\Big[\partial[[\![s_1]\!]] \circ (\mathbf{id} \times \langle \mathbf{id}, 0\rangle) \circ \mathbf{sw}\Big] \circ \langle \mathbf{id}, \langle 0, [\![t]\!] \circ \pi_1\rangle\rangle \\
&= \Lambda\Big[\partial[[\![s_1]\!]] \circ (\mathbf{id} \times \langle \mathbf{id}, 0\rangle) \circ \mathbf{sw} \circ \langle\langle \mathbf{id}, \langle 0, [\![t]\!] \circ \pi_1\rangle\rangle \circ \pi_1, \pi_2\rangle\Big] \\
&= \Lambda\Big[\partial[[\![s_1]\!]] \circ (\mathbf{id} \times \langle \mathbf{id}, 0\rangle) \circ \mathbf{sw} \circ \langle\langle \pi_1, \langle 0, [\![t]\!] \circ \pi_{11}\rangle\rangle, \pi_2\rangle\Big] \\
&= \Lambda\Big[\partial[[\![s_1]\!]] \circ (\mathbf{id} \times \langle \mathbf{id}, 0\rangle) \circ \langle \mathbf{id}, \langle 0, [\![t]\!] \circ \pi_{11}\rangle\rangle\Big] \\
&= \Lambda\Big[\partial[[\![s_1]\!]] \circ \langle \mathbf{id}, \langle\langle 0, [\![t]\!] \circ \pi_{11}\rangle, 0\rangle\rangle\Big] \\
&= \Lambda\Big[\partial[[\![s_1]\!]] \circ (\mathbf{sw} \times \mathbf{sw}) \circ \langle \mathbf{id}, \langle 0, [\![t]\!] \circ \pi_{11}\rangle\rangle\Big] \\
&= \Lambda\Big[\partial[[\![s_1]\!]] \circ \langle \mathbf{sw}, \langle\langle 0, [\![t]\!] \circ \pi_{11}\rangle, 0\rangle\rangle\Big] \\
&= \Lambda\Big[\partial[[\![s_1]\!]] \circ \langle \mathbf{id}, \langle\langle 0, [\![t]\!] \circ \pi_{11}\rangle, 0\rangle\rangle \circ \mathbf{sw}\Big] \\
&= \Lambda \left[\!\!\left[\Gamma, x : \tau, y : \sigma_1 \vdash \frac{\partial s_1}{\partial x}(t) : \sigma_2\right]\!\!\right] \\
&= \left[\!\!\left[\Gamma, x : \tau \vdash \lambda y. \frac{\partial s_1}{\partial x}(t) : \sigma_1 \Rightarrow \sigma_2\right]\!\!\right]
\end{aligned}$$

- $s = s_1\ e$ To simplify the calculations, we write $\mathfrak{s}$ for $\Lambda^-([\![s_1]\!])$. The result follows as a consequence of Lemma 6.4.3[iii.].

$$\begin{aligned}
&\left[\!\!\left[\frac{\partial(s_1\ e)}{\partial x}(t)\right]\!\!\right] \\
&= \left[\!\!\left[\mathrm{D}(s_1) \cdot \left(\frac{\partial e}{\partial x}(t)\right)\ e\right]\!\!\right] + \left[\!\!\left[\left(\frac{\partial s_1}{\partial x}(t)\right)\ (e\,[x + \varepsilon t/x])\right]\!\!\right] \\
&= \mathbf{ev} \circ \langle \Lambda\,(\mathfrak{s} \star ([\![e]\!] \star [\![t]\!])), [\![e]\!]\rangle \\
&\qquad + \mathbf{ev} \circ \langle [\![s_1]\!] \star [\![t]\!], [\![e]\!] \circ \langle \pi_1, \pi_2 + \varepsilon([\![t]\!]) \circ \pi_1\rangle\rangle \\
&= (\mathbf{ev} \circ \langle [\![s_1]\!], [\![e]\!]\rangle) \star [\![t]\!] \\
&= [\![s_1\ e]\!] \star [\![t]\!]
\end{aligned}$$

- $s = \mathrm{D}(s_1) \cdot u$

We will again abbreviate $\Lambda^-([\![s_1]\!])$ as $\mathfrak{s}$ to make the subsequent calculations more readable. The result then follows by applying of Lemma 6.4.3[ii.].

$$\begin{aligned}
&\left[\!\!\left[\frac{\partial \mathrm{D}(s_1) \cdot u}{\partial x}(t)\right]\!\!\right] \\
&= \left[\!\!\left[\mathrm{D}(s_1) \cdot \frac{\partial u}{\partial x}(t)\right]\!\!\right] + \left[\!\!\left[\mathrm{D}\left(\frac{\partial s_1}{\partial x}(t)\right) \cdot (u\,[x + \varepsilon(t)/x])\right]\!\!\right] \\
&\qquad + \varepsilon \left[\!\!\left[\left(\mathrm{D}(\mathrm{D}(s_1) \cdot u) \cdot \left(\frac{\partial u}{\partial x}(t)\right)\right)\right]\!\!\right]
\end{aligned}$$

$$
\begin{aligned}
&= \Lambda\left(\mathfrak{s}\star\left[\!\!\left[\frac{\partial u}{\partial x}(t)\right]\!\!\right]\right) + \Lambda\left(\left(\Lambda^{-}\left[\!\!\left[\frac{\partial s_1}{\partial x}(t)\right]\!\!\right]\right)\star[\![u\,[x+\varepsilon(t)/x]]\!]\right)\\
&\qquad + \varepsilon\Lambda\left(\left(\Lambda^{-}[\![\mathrm{D}(s_1)\cdot u]\!]\right)\star\left[\!\!\left[\frac{\partial u}{\partial x}(t)\right]\!\!\right]\right)\\
&= \Lambda\left(\mathfrak{s}\star([\![u]\!]\star[\![t]\!])\right) + \Lambda\left(\left(\Lambda^{-}(\Lambda([\![s_1]\!])\star[\![t]\!])\right)\star[\![u\,[x+\varepsilon(t)/x]]\!]\right)\\
&\qquad + \varepsilon\Lambda\left(\left(\Lambda^{-}(\mathfrak{s}\star[\![u]\!])\right)\star([\![u]\!]\star[\![t]\!])\right)\\
&= \Lambda\left(\mathfrak{s}\star([\![u]\!]\star[\![t]\!])\right) + \Lambda\left(\left(\Lambda^{-}(\Lambda([\![s_1]\!])\star[\![t]\!])\right)\star([\![u]\!]\circ\langle\pi_1,\pi_2+\varepsilon([\![t]\!])\circ\pi_1\rangle)\right)\\
&\qquad + \varepsilon\Lambda\left(\left(\Lambda^{-}(\mathfrak{s}\star[\![u]\!])\right)\star([\![u]\!]\star[\![t]\!])\right)\\
&= \left(\Lambda\left(\mathfrak{s}\star[\![u]\!]\right)\right)\star[\![t]\!]\\
&= \left(\Lambda\left(\Lambda^{-}([\![s_1]\!])\star[\![u]\!]\right)\right)\star[\![t]\!]\\
&= \left([\![\mathrm{D}(s_1)\cdot u]\!]\right)\star[\![t]\!]
\end{aligned}
$$

□

Definition 7.3.3. Given well-formed terms $\underline{s}, \underline{s'}$, we define the equivalence relation $\sim_{\beta\partial}$ as the least contextual equivalence relation that contains the one-step reduction relation $\underline{\rightsquigarrow}$.

Corollary 7.3.1. The interpretation $[\![\cdot]\!]$ is *sound*, that is to say, whenever $\underline{s} \sim_{\beta\partial} \underline{s'}$ then $[\![s]\!] = [\![s']\!]$, independently of the choice of representatives s, s'.

Chapter 8

Conclusions

8.1 Summary of Results

The main contribution of this thesis is the notion of a differential map between change actions. Initially motivated by their applications to incremental computation, we believe differential maps are an important generalisation of many natural definitions of derivative. We have shown that they arise from, and connect, many seemingly disparate areas, providing a formal basis for the idea that incrementalisation is indeed a sort of differentiation.

We believe change actions compare favorably to Cai and Giarrusso's change structures, having three main advantages: first, the formulation of a change action is not dependently-typed, and so it can be applied in many more settings, e.g. categories where one might not be able to interpret dependent type theory. Second, the presence of the difference operator $\ominus$ means that the theory of change structures is, in a sense, "trivial": every function into a change structure always admits a derivative. This also rules out interesting models, such as the ones arising from Cartesian differential categories. Finally, the requirement that the change space ΔA be a monoid, together with the regularity property of derivatives, allow for a rich theory (for example, the proof that a multivariate function is differentiable if and only if partial derivatives exist crucially hinges on the monoid structure of ΔA).

On the other hand, change actions, through difference categories, are a strict generalisation of Cartesian difference categories which can account for important notions of derivative that do not fit in the Cartesian differential mold. As we have shown, many theorems about Cartesian differential categories also hold in the difference setting, and one can likewise construct a calculus out of difference categories.

8.2 Future Work

The writing of this thesis has been an exercise in restraint, with many research leads having been postponed due to time constraints. We compile here some of the most promising ones, in the hopes that they will serve as a suggestion for the interested reader.

8.2.1 The Higher-Categorical Story

As we showed in Section 3.3, a change action is nothing more and nothing less than a particular kind of internal category, with the category $\mathrm{CAct}(\mathbf{C})$ of $\mathbf{C}$-change actions embedding fully and faithfully into the category $\mathrm{Cat}(\mathbf{C})$ of $\mathbf{C}$-categories. A natural question arising from this result is: exactly what sort of categories correspond to change actions? Or, in other words, is there a meaningful property characterising those categories that are in the image of the previous embedding?

The above statement also has interesting implications for the higher-order setting: taking it into consideration, a change action model $\alpha : \mathbf{C} \to \mathrm{CAct}(\mathbf{C})$ is simply a particular kind of functor $\alpha : \mathbf{C} \to \mathrm{Cat}(\mathbf{C})$, and thus every object in a change action model on $\mathbf{C}$ is a category internal to $\mathbf{C}$ – but, by iterating α, a much stronger statement is also true: every object in $\mathbf{C}$ is a particularly well-behaved internal ∞-fold category (see [80] for a discussion of n-fold categories, and [79] for the 2-fold case). In particular, every object is an ∞-fold category where the horizontal and vertical morphisms coincide, which leads us to conjecture that change action models are in fact strict ω-categories. The significance and implications of these observations are yet to be understood.

8.2.2 Iteration and Integration

Through this work we have focused on differentiation, while ommitting any mentions of integrals or differential equations. The categorical theory of integration is, in general, much less developed than that of differentiation, only recently having received more attention [27, 66].

Similarly, we have developed a simply-typed calculus which does not include any mechanisms for iteration. Iteration is itself problematic, as it is not immediately clear that iteration combinators are differentiable (although some of the results in this thesis already indicate that fixed-point combinators can be differentiated under reasonable conditions, see Theorem 4.5.4).

These two missing pieces are more closely related than one might imagine. Indeed, consider a hypothetical extension of the difference λ-calculus equipped with a type of natural numbers (with the identity as its corresponding infinitesimal extension, that is to say, $\varepsilon_{\mathbb{N}} = \mathbf{id}_{\mathbb{N}}$). How should an iteration operator **iter** be defined? The straightforward option would be to give it the usual behavior, that is to say:

$$\begin{aligned} \mathbf{iter}\ \mathbf{Z}\ z\ s &\rightsquigarrow z \\ \mathbf{iter}\ (\mathbf{S}\ n)\ z\ s &\rightsquigarrow s\ (\mathbf{iter}\ n\ z\ s) \end{aligned}$$

While this is not unreasonable, an unexpected consequence of this reduction rule is that it implies that every change action involved must be complete, that is to say, for every $s, t : A$, there must be a change $u : \Delta A$ with $x + \varepsilon(u) = y$ – such a change can be obtained by the term $((\mathrm{D}(\lambda n.\mathbf{iter}\ n\ s\ (\lambda x.t)) \cdot (\mathbf{S}\ \mathbf{Z}))\ \mathbf{Z})$, which has the property that:

$$\begin{aligned} t &\approx \mathbf{iter}\ (\mathbf{S}\ \mathbf{Z})\ s\ (\lambda x.t) \\ &\approx (\lambda n.\mathbf{iter}\ n\ s\ (\lambda x.t))\ (\mathbf{S}\ \mathbf{Z}) \\ &\approx [(\lambda n.\mathbf{iter}\ n\ s\ (\lambda x.t))\ \mathbf{Z}] + \varepsilon\,[((\mathrm{D}(\lambda n.\mathbf{iter}\ n\ s\ (\lambda x.t)) \cdot (\mathbf{S}\ \mathbf{Z}))\ \mathbf{Z})] \\ &\approx s + \varepsilon\,[((\mathrm{D}(\lambda n.\mathbf{iter}\ n\ s\ (\lambda x.t)) \cdot (\mathbf{S}\ \mathbf{Z}))\ \mathbf{Z})] \end{aligned}$$

This would rule out a number of interesting models – especially ones where the set ΔA is to be interpreted as a set of infinitesimals – and so it seems rather unsatisfactory. An alternative is to define the iteration operator by:

$$\begin{aligned} \mathbf{iter}\ \mathbf{Z}\ z\ s &\rightsquigarrow z \\ \mathbf{iter}\ (\mathbf{S}\ n)\ z\ s &\rightsquigarrow (\mathbf{iter}\ n\ z\ s) + \varepsilon(s\ (\mathbf{iter}\ n\ z\ s)) \end{aligned}$$

or, in terms of actions rather than the associated infinitesimal extensions:

$$\begin{aligned} \mathbf{iter}\ \mathbf{Z}\ z\ s &\rightsquigarrow z \\ \mathbf{iter}\ (\mathbf{S}\ n)\ z\ s &\rightsquigarrow (\mathbf{iter}\ n\ z\ s) \oplus (s\ (\mathbf{iter}\ n\ z\ s)) \end{aligned}$$

where $s : A \to \Delta A$. If one extends the Cartesian difference category $\mathbf{Ab}_\star$ to cover all commutative monoids where addition is injective (which includes monoids such as $(\mathbb{N}, +, 0)$) and all infinitely differentiable maps, it turns out that this **iter** operator is itself an arrow in this difference category, that is, the above definition of iteration is always smooth, without requiring that every change action be complete. Fixed z, s, define the map $\mu(n) \coloneqq \mathbf{iter}\ n\ z\ s$. What is the derivative $\mathrm{D}[\mu](n, \mathbf{S}\ \mathbf{Z})$? Nothing more and nothing less than $s(\mu(n))$. Then the function $\mu : \mathbb{N} \to A$ defines a "curve"[1] which starts at z and whose derivative at a given point n is $s(\mu(n))$ – if one squints, this is

[1]It seems adequate that curves in difference categories should be maps from $\mathbb{N}$, rather than $\mathbb{R}$, as befits a discrete setting.

nothing more and nothing less than the requirement that the curve μ be an integral curve for the smooth vector field s satisfying the initial condition $\mu(\mathbf{Z}) = z$! Hence one can think of iteration as a discrete counterpart of the Picard-Lindelöf theorem, which states that such integral curves always exist (locally).

It would be of great interest, then, to extend λ_ε with an interation operator and give its semantics in terms of differential (or difference) equations. Studying recurrence equations using the language of differential equations is a very useful tool in discrete analysis; for example, one can treat the recursive definition of the Fibonacci sequence as a discrete ODE and use differential equation methods to find a closed-form solution. We believe that in a language which frames iteration in such terms may be amenable to optimisation by similar analytic methods.

8.2.3 Gradients and Coderivatives

One of the main applications of automatic differentiation nowadays is computing gradients for the purpose of optimising functions by gradient descent. This immediately raises the question of whether a similar notion of gradient can be defined "synthetically" for change actions, and whether it leads to optimisation algorithms for discrete spaces.

An initial attempt at this task led us to proposing the notion of *coderivatives.*

Definition 8.2.1. Given a map $f : A \to B$ and change actions $\overline{A}, \overline{B}$, a map $\partial^\dagger f : A \times \Delta B \to \Delta A$ is a *coderivative* for f whenever it satisfies the following condition:

$$f \circ \oplus_A \circ \langle \pi_1, \partial^\dagger f \rangle = \oplus_B \circ (f \times \mathbf{id}_{\Delta B})$$

We also require that coderivatives preserve the zero change[2], i.e.

$$\partial^\dagger f \circ \langle \mathbf{id}_A, 0_B \rangle = 0_A$$

Coderivatives do not quite correspond to gradients – in fact, it can be shown that, whenever a map f admits an invertible derivative, its inverse is a coderivative for f. They are, however, not without importance: a generalisation of Newton's method to change actions can be framed in terms of coderivatives, and it can be shown that a specific kind of codifferential maps corresponds to lenses [18, 53], a well-known concept in the functional programming community where they provide a simple and elegant solution to the problem of updating deeply nested data-structures.

[2]One may further require that coderivatives preserve sums in a similar sense to regularity but the usefulness of this is debatable.

Furthermore, the following theorem shows that coderivatives satisfy a reverse version of the chain rule, where changes are propagated backwards and values are propagated forwards.

Theorem 8.2.1. Let $\overline{A}, \overline{B}, \overline{C}$ be change actions and consider maps $f : A \to B, g : B \to C$ with coderivatives $\partial^\dagger f, \partial^\dagger g$ respectively. Then the map

$$\partial^\dagger (g \circ f) \coloneqq \partial^\dagger f \circ \left\langle \pi_1, \partial^\dagger g \circ (f \times \mathbf{id}_{\Delta C}) \right\rangle$$

or, more intuitively,

$$\partial^\dagger (g \circ f)(a, \delta c) \coloneqq \partial^\dagger f(a, \partial^\dagger g(f(a), \delta c))$$

is a coderivative for $g \circ f$.

The above chain rule looks strikingly similar to the operational behavior of reverse-mode automatic differentiation, where computation is first performed in the usual fashion and perturbations are then propagated backwards through a stored trace of the computation, and one hopes that a theory of coderivatives could likewise lead to a theory and calculus of reverse-mode automatic differentiation.

Bibliography

[1] Abiteboul, S., Hull, R., Vianu, V.: Foundations of databases: the logical level. Addison-Wesley Longman Publishing Co., Inc. (1995)

[2] Abramsky, S., Jung, A.: Domain theory. In: Handbook of logic in computer science. Oxford University Press (1994)

[3] Acar, U.A.: Self-adjusting computation:(an overview). In: Proceedings of the 2009 ACM SIGPLAN workshop on Partial evaluation and program manipulation. pp. 1–6. ACM (2009)

[4] Alvarez-Picallo, M., Lemay, J.S.P.: Cartesian difference categories. In: International Conference on Foundations of Software Science and Computation Structures. p. to appear. Springer (2020)

[5] Alvarez-Picallo, M., Ong, C.H.L.: Change actions: models of generalised differentiation. In: International Conference on Foundations of Software Science and Computation Structures. pp. 45–61. Springer (2019)

[6] Alvarez-Picallo, M., Peyton-Jones, M., Eyers-Taylor, A., Ong, C.H.L.: Fixing incremental computation. In: European Symposium on Programming. Springer (2019), in press

[7] Arntzenius, M., Krishnaswami, N.R.: Datafun: a functional datalog. In: Proceedings of the 21st ACM SIGPLAN International Conference on Functional Programming. pp. 214–227. ACM (2016)

[8] Arrighi, P., Dowek, G.: Linear-algebraic lambda-calculus: higher-order, encodings and confluence in A. Voronkov, Rewriting techniques and applications. Lecture Notes in Computer Science, Springer-Verlag (2008)

[9] Arrighi, P., Dowek, G.: A computational definition of the notion of vectorial space. Electronic Notes in Theoretical Computer Science **117**, 249–261 (2005)

[10] Arrighi, P., Dowek, G.: Linear-algebraic lambda-calculus. arXiv preprint quant-ph/0501150 (2005)

[11] Avgustinov, P., de Moor, O., Jones, M.P., Schäfer, M.: Ql: Object-oriented queries on relational data. In: LIPIcs-Leibniz International Proceedings in Informatics. vol. 56. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik (2016)

[12] Bancilhon, F.: Naive evaluation of recursively defined relations. In: On Knowledge Base Management Systems, pp. 165–178. Springer (1986)

[13] Bancilhon, F., Ramakrishnan, R.: An amateur's introduction to recursive query processing strategies. ACM SIGMOD Record **15**(2), 16–52 (1986)

[14] Blute, R., Ehrhard, T., Tasson, C.: A convenient differential category. arXiv preprint arXiv:1006.3140 (2010)

[15] Blute, R., Panangaden, P., Seely, R.: Fock space: a model of linear exponential types. Manuscript (1994)

[16] Blute, R.F., Cockett, J.R.B., Seely, R.A.G.: Differential categories. Mathematical structures in computer science **16**(06), 1049–1083 (2006)

[17] Blute, R.F., Cockett, J.R.B., Seely, R.A.G.: Cartesian differential categories. Theory and Applications of Categories **22**(23), 622–672 (2009)

[18] Bohannon, A., Pierce, B.C., Vaughan, J.A.: Relational lenses: a language for updatable views. In: Proceedings of the twenty-fifth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems. pp. 338–347. ACM (2006)

[19] Boudol, G.: The lambda-calculus with multiplicities. In: International Conference on Concurrency Theory. pp. 1–6. Springer (1993)

[20] Bradet-Legris, J., Reid, H.: Differential forms in non-linear cartesian differential categories (2018), Foundational Methods in Computer Science

[21] Brzozowski, J.A.: Derivatives of regular expressions. J. ACM **11**(4), 481–494 (1964). https://doi.org/10.1145/321239.321249

[22] Bucciarelli, A., Ehrhard, T., Manzonetto, G.: Categorical models for simply typed resource calculi. Electronic Notes in Theoretical Computer Science **265**, 213–230 (2010)

[23] Cai, Y., Giarrusso, P.G., Rendel, T., Ostermann, K.: A theory of changes for higher-order languages: Incrementalizing λ-calculi by static differentiation. ACM SIGPLAN Notices **49**(6), 145–155 (2014)

[24] Carlsson, M.: Monads for incremental computing. ACM SIGPLAN Notices **37**(9), 26–35 (2002)

[25] Cockett, J.R.B., Cruttwell, G.S.H.: Differential structure, tangent structure, and sdg. Applied Categorical Structures **22**(2), 331–417 (2014)

[26] Cockett, J.R.B., Cruttwell, G.S.H.: Differential bundles and fibrations for tangent categories. arXiv preprint arXiv:1606.08379 (2016)

[27] Cockett, J.R.B., Lemay, J.S.: Integral categories and calculus categories. Mathematical Structures in Computer Science **29**(2), 243–308 (2019)

[28] Compton, K.J.: Stratified least fixpoint logic. Theoretical Computer Science **131**(1), 95–120 (1994)

[29] Conway, J.H.: Regular Algebra and Finite Machines. Chapman and Hall (1971)

[30] Cruttwell, G.S.H.: Cartesian differential categories revisited. Mathematical Structures in Computer Science **27**(1), 70–91 (2017)

[31] Dalrymple, D.: Differentiable programming. `https://www.edge.org/response-detail/26794` (2016)

[32] Danos, V., Herbelin, H., Regnier, L.: Game Semantics & Abstract Machines. In: LICS. vol. 96, pp. 394–405 (1996)

[33] Datomic website, `https://www.datomic.com`, accessed: 2018-01-01

[34] Dieudonné, J.: Sur les espaces de Köthe. Journal d'Analyse Mathématique **1**(1), 81–115 (1951)

[35] Ehrhard, T.: On Köthe sequence spaces and linear logic. Mathematical Structures in Computer Science **12**(05), 579–623 (2002)

[36] Ehrhard, T.: Finiteness spaces. Mathematical Structures in Computer Science **15**(04), 615–646 (2005)

[37] Ehrhard, T., Regnier, L.: The differential lambda-calculus. Theoretical Computer Science **309**(1-3), 1–41 (2003). https://doi.org/10.1016/S0304-3975(03)00392-X

[38] Ehrhard, T., Regnier, L.: Differential interaction nets. Theoretical Computer Science **364**(2), 166–195 (2006)

[39] Ehrhard, T., Regnier, L.: Uniformity and the Taylor expansion of ordinary lambda-terms. Theoretical Computer Science **403**(2), 347–372 (2008)

[40] Esparza, J., Kiefer, S., Luttenberger, M.: Newtonian program analysis. J. ACM **57**(6), 33:1–33:47 (2010). https://doi.org/10.1145/1857914.1857917

[41] Frolicher, A.: Linear spaces and differentiation theory. Pure and Applied Mathematics (1988)

[42] Geroch, R.: Mathematical physics. Chicago lectures in physics, University of Chicago Press (1985)

[43] Giarrusso, P.G., Régis-Gianas, Y., Schuster, P.: Incremental *lambda*-calculus in cache-transfer style. In: European Symposium on Programming. pp. 553–580. Springer (2019)

[44] Girard, J.Y.: Linear logic. Theoretical Computer Science **50**(1), 1 – 101 (1987). https://doi.org/http://dx.doi.org/10.1016/0304-3975(87)90045-4, `http://www.sciencedirect.com/science/article/pii/0304397587900454`

[45] Girard, J.: Linear logic: Its syntax and semantics. In: Girard, J., Lafont, Y., Regnier, L. (eds.) Advances in Linear Logic, pp. 222–1. Cambridge University Press (1995)

[46] Girard, J.Y.: Proof-nets: the parallel syntax for proof-theory. Lecture Notes in Pure and Applied Mathematics pp. 97–124 (1996)

[47] Girard, J.Y.: Coherent Banach spaces: A continuous denotational semantics. Theoretical Computer Science **227**(1-2), 275–297 (Sep 1999). https://doi.org/10.1016/S0304-3975(99)00056-0, `http://dx.doi.org/10.1016/S0304-3975(99)00056-0`

[48] Gleich, D.: Finite calculus: A tutorial for solving nasty sums. Stanford University (2005)

[49] Griewank, A., Walther, A.: Evaluating derivatives: principles and techniques of algorithmic differentiation, vol. 105. Siam (2008)

[50] Gupta, A., Mumick, I.S., Subrahmanian, V.S.: Maintaining views incrementally. ACM SIGMOD Record **22**(2), 157–166 (1993)

[51] Gupta, A., Mumick, I.S., et al.: Maintenance of materialized views: Problems, techniques, and applications. IEEE Data Eng. Bull. **18**(2), 3–18 (1995)

[52] Halpin, T., Rugaber, S.: LogiQL: A Query Language for Smart Databases. CRC Press (2014)

[53] Hofmann, M., Pierce, B., Wagner, D.: Symmetric lenses. ACM SIGPLAN Notices **46**(1), 371–384 (2011)

[54] Hopkins, M.W., Kozen, D.: Parikh's theorem in commutative Kleene algebra. In: 14th Annual IEEE Symposium on Logic in Computer Science, Trento, Italy, July 2-5, 1999. pp. 394–401 (1999). https://doi.org/10.1109/LICS.1999.782634

[55] Jacobs, B.: Categorical logic and type theory, Studies in Logic and the Foundations of Mathematics, vol. 141. Elsevier (1999)

[56] Jordan, C.: Calculus of finite differences, vol. 33. American Mathematical Soc. (1965)

[57] Kelly, R., Pearlmutter, B.A., Siskind, J.M.: Evolving the incremental λ calculus into a model of forward automatic differentiation (ad). arXiv preprint arXiv:1611.03429 (2016)

[58] Kerjean, M., Tasson, C.: Mackey-complete spaces and power series–a topological model of differential linear logic. Mathematical Structures in Computer Science pp. 1–36 (2016)

[59] Kock, A.: Synthetic Differential Geometry. Cambridge University Press, 2nd edn. (2006)

[60] Kock, A.: Synthetic Geometry of Manifolds. No. 180 in Cambridge Tracts in Mathematics, Cambridge University Press (2009)

[61] Köthe, G.: Topologische lineare räume. In: Topologische Lineare Räume I, pp. 127–204. Springer (1960)

[62] Kriegl, A., Michor, P.W.: The convenient setting of global analysis, vol. 53. American Mathematical Soc. (1997)

[63] Kupke, C., Niqui, M., Rutten, J.: Stream differential equations: concrete formats for coinductive definitions (2011)

[64] Lafont, Y.: From proof nets to interaction nets. London Mathematical Society Lecture Note Series pp. 225–248 (1995)

[65] Lambek, J.: Deductive systems and categories III. Cartesian closed categories, intuitionist propositional calculus, and combinatory logic. In: Toposes, algebraic geometry and logic, pp. 57–82. Springer (1972)

[66] Lemay, J.S.P.: Exponential functions in cartesian differential categories. arXiv preprint arXiv:1911.04790 (2019)

[67] Liao, Q., Sun, L., Du, H., Yang, Y.: An incremental algorithm for estimating average clustering coefficient based on random walk. In: Asia-Pacific Web (APWeb) and Web-Age Information Management (WAIM) Joint Conference on Web and Big Data. pp. 158–165. Springer (2017)

[68] Logicblox inc. website, `http://www.logicblox.com`, accessed: 2018-01-01

[69] Lombardy, S., Sakarovitch, J.: How expressions can code for automata. In: LATIN 2004: Theoretical Informatics, 6th Latin American Symposium, Buenos Aires, Argentina, April 5-8, 2004, Proceedings. pp. 242–251 (2004). https://doi.org/10.1007/978-3-540-24698-5_28

[70] Mak, C.: Simply-typed Differential and Resource λ-calculus. Master's thesis, University of Oxford (2016)

[71] Manzonetto, G.: What is a Categorical Model of the Differential and the Resource λ-Calculi? Mathematical Structures in Computer Science **22**(03), 451–520 (2012)

[72] Manzyuk, O.: Non-Confluence of the Perturbative λ-Calculus. `https://oleksandrmanzyuk.wordpress.com/2012/07/01/non-confluence-of-the-perturbative-%CE%BB-calculus/` (2012)

[73] Manzyuk, O.: A simply typed λ-calculus of forward automatic differentiation. Electronic Notes in Theoretical Computer Science **286**, 257–272 (2012)

[74] Manzyuk, O.: Tangent bundles in differential lambda-categories. arXiv preprint arXiv:1202.0411 (2012)

[75] Megginson, R.E.: An Introduction to Banach Space Theory, Graduate Texts in Mathematics, vol. 183. Springer (1998)

[76] Mihaylov, S.R., Ives, Z.G., Guha, S.: Rex: recursive, delta-based data-centric computation. Proceedings of the VLDB Endowment **5**(11), 1280–1291 (2012)

[77] Milne-Thomson, L.M.: The calculus of finite differences. American Mathematical Soc. (2000)

[78] de Moor, O., Baars, A.: Doing a doaitse: Simple recursive aggregates in datalog (2013), `http://www.staff.science.uu.nl/~hage0101/liberdoaitseswierstra.pdf`, accessed: 2018-01-01

[79] nLab authors: double category. `http://ncatlab.org/nlab/show/double%20category` (Jan 2020), Revision 60

[80] nLab authors: n-fold category. `http://ncatlab.org/nlab/show/n-fold%20category` (Jan 2020), Revision 26

[81] Olah, C.: Neural networks, types, and functional programming. `http://colah.github.io/posts/2015-09-NN-Types-FP/` (2015)

[82] Paige, L., et al.: Formal differentiation: A program synthesis technique. UMI Research Press (1981)

[83] Parigot, M.: $\lambda\mu$-calculus: an algorithmic interpretation of classical natural deduction. In: International Conference on Logic for Programming Artificial Intelligence and Reasoning. pp. 190–201. Springer (1992)

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

[85] Pilling, D.L.: Commutative regular equations and Parikh's theorem. J. London Math. Soc. **6**, 663–666 (1973)

[86] Ramalingam, G., Reps, T.: A categorized bibliography on incremental computation. In: Proceedings of the 20th ACM SIGPLAN-SIGACT symposium on Principles of programming languages. pp. 502–510. ACM (1993)

[87] Rosickỳ, J.: Abstract tangent functors. Diagrammes **12**, 1–11 (1984)

[88] Rumelhart, D.E., Hinton, G.E., Williams, R.J.: Learning representations by back-propagating errors. Cognitive modeling **5**(3), 1 (1988)

[89] Rutten, J.: Behavioural differential equations: a coinductive calculus of streams, automata, and power series. Theoretical Computer Science **308**(1-3), 1–53 (2003)

[90] Rutten, J.: A coinductive calculus of streams. Mathematical Structures in Computer Science **15**(1), 93–147 (2005)

[91] Sáenz-Pérez, F.: Des: A deductive database system. Electronic notes in theoretical computer science **271**, 63–78 (2011)

[92] Schäfer, M., de Moor, O.: Type inference for datalog with complex type hierarchies. ACM Sigplan Notices **45**(1), 145–156 (2010)

[93] Scholz, B., Jordan, H., Subotić, P., Westmann, T.: On fast large-scale program analysis in datalog. In: Proceedings of the 25th International Conference on Compiler Construction. pp. 196–206. ACM (2016)

[94] Semmle ltd. website, `https://semmle.com`, accessed: 2018-01-01

[95] Sereni, D., Avgustinov, P., De Moor, O.: Adding magic to an optimising datalog compiler. In: Proceedings of the 2008 ACM SIGMOD international conference on Management of data. pp. 553–566. ACM (2008)

[96] Siskind, J.M., Pearlmutter, B.A.: Using polyvariant union-free flow analysis to compile a higher-order functional-programming language with a first-class derivative operator to efficient fortran-like code. ECE Technical Reports p. 367 (2008)

[97] Siskind, J.M., Pearlmutter, B.A.: First-class nonstandard interpretations by opening closures. ACM SIGPLAN Notices **42**(1), 71–76 (2007)

[98] Siskind, J.M., Pearlmutter, B.A.: Nesting forward-mode AD in a functional framework. Higher-Order and Symbolic Computation **21**(4), 361–376 (2008)

[99] Souffle language website, `http://souffle-lang.org`, accessed: 2018-01-01

[100] Sprunger, D., Jacobs, B.: The differential calculus of causal functions. arXiv preprint arXiv:1904.10611 (2019)

[101] Sprunger, D., Katsumata, S.y.: Differentiable causal computations via delayed trace. arXiv preprint arXiv:1903.01093 (2019)

[102] Steinbach, B., Posthoff, C.: Boolean differential calculus. Synthesis Lectures on Digital Circuits and Systems **12**(1), 1–215 (2017)

[103] Thayse, A.: Boolean calculus of differences. No. 101 in Lecture Notes in Computer Science, Springer Science & Business Media (1981)

[104] Tranquilli, P.: Intuitionistic differential nets and lambda-calculus. Theoretical Computer Science **412**(20), 1979–1997 (2011)

[105] Vaux, L.: λ-calculus in an algebraic setting. unpublished note (2006)

[106] Vaux, L.: On linear combinations of λ-terms. In: International Conference on Rewriting Techniques and Applications. pp. 374–388. Springer (2007)

[107] Vaux, L.: The differential $\lambda\mu$-calculus. Theoretical Computer Science **379**(1), 166–209 (2007)

[108] Vaux, L.: The algebraic lambda calculus. Mathematical Structures in Computer Science **19**(05), 1029–1059 (2009)

[109] Winskel, G.: The formal semantics of programming languages: an introduction. MIT press (1993)

Appendix A

Mechanised Proofs

A.1 Module **Definitions.v**

```
Require Import Unicode.Utf8.
Require Import Coq.Program.Equality.
Require Import Coq.Relations.Relation_Operators.
Require Import Coq.Relations.Relation_Definitions.
Require Import Coq.Arith.EqNat.
Require Import Coq.Arith.Compare_dec.

(* The type of (unrestricted) terms of the difference lambda-calculus *)
Inductive Term :=
| Var  : nat → Term
| Λ    : Term → Term
| App  : Term → Term → Term
| Zero : Term
| Add  : Term → Term → Term
| ε    : Term → Term
| Dif  : Term → Term → Term
.

Notation "0" := Zero.
Notation "s + t" := (Add s t) (at level 50, left associativity).
Notation "'D' s · t" := (Dif s t) (at level 40).
Notation "{ s | t }" := (App s t).

Section Contextual_Closure.

  Variable ρ : relation Term.

  Inductive Ctx : relation Term :=
  | CVar  n     : Ctx (Var n) (Var n)
  | CZero       : Ctx 0 0
  | CAbs  t t' : Ctx t t' → Ctx (Λ t) (Λ t')
  | CE    t t' : Ctx t t' → Ctx (ε t) (ε t')

  | CApp  s s' t t' : Ctx s s' → Ctx t t' → Ctx {s|t} { s' | t' }
  | CAdd  s s' t t' : Ctx s s' → Ctx t t' → Ctx (s + t) (s' + t')
  | CD    s s' t t' : Ctx s s' → Ctx t t' → Ctx (D s · t) (D s' · t')

  | CRho  t t' : ρ t t' → Ctx t t'
  .
  Hint Constructors Ctx : diff.

  Theorem Ctx_refl : forall t, Ctx t t.
  Proof. intros t; induction t; eauto with diff.
  Qed.
End Contextual_Closure.
Hint Constructors Ctx : diff.
Hint Resolve Ctx_refl : diff.

(* Contextual relations on terms *)
Definition Contextual ρ := forall s t, Ctx ρ s t → ρ s t.

(* One-step differential equivalence on terms *)
Reserved Notation "s ~1 t" (at level 90).
Inductive Equiv1 : Term → Term → Prop :=
(* Commutative monoid conditions *)
| Eq_add_0 t          : t + 0 ~1 t
| Eq_add_comm  s t    : s + t ~1 t + s
| Eq_add_assoc s t e : (s + t) + e ~1 s + (t + e)
| Eq_add_E s t        : ε (s + t) ~1 ε s + ε t
| Eq_E_0              : ε 0 ~1 0

(* Linearity conditions *)
| Eq_app_0   t      : {0 | t} ~1 0
| Eq_app_add s t e : {s + t | e} ~1 {s|e} + {t|e}
| Eq_app_E s t      : {ε s | t} ~1 ε {s | t}
| Eq_lam_0          : Λ 0 ~1 0
| Eq_lam_add s t    : Λ (s + t) ~1 Λ s + Λ t
| Eq_lam_E s        : Λ (ε s) ~1 ε (Λ s)
| Eq_D_0 u          : D 0 · u ~1 0
| Eq_D_add s t e    : D (s + t) · e ~1 D s · e + D t · e
| Eq_D_E s t        : D (ε s) · t ~1 ε (D s · t)

| Eq_D_0_r t   : D t · 0 ~1 0
| Eq_D_E_r t u : D t · (ε u) ~1 ε (D t · u)

(* Regularity *)
| Eq_regularity s u v : D s · (u + v) ~1 D s · u + D s · v + ε (D (D s · v) · u)

(* Derivative condition *)
| Eq_derivative s t u : {s | t + ε u} ~1 {s | t} + ε {D s · u | t}

(* Commutativity of derivatives *)
| Eq_D_comm s u v : D (D s · u) · v ~1 D (D s · v) · u

(* Infinitesimal cancellation *)
| Eq_D_inf s u v : ε (ε (D (D s · u) · v)) ~1 ε (D (D s · u) · v)
where "s ~1 t" := (Equiv1 s t).
Hint Constructors Equiv1 : diff.
```

```
(* Differential equivalence of terms *)
Definition Equiv (s : Term) (t : Term) :=
  clos_refl_sym_trans Term (Ctx Equiv1) s t.
Notation "s ≈ t" := (Equiv s t) (at level 90).

(* Avoid using rst_* as that replaces ≈ by clos_refl_sym_trans... and messes
with the pattern matching in tactics *)
Definition eq_step : forall s t, Ctx Equiv1 s t → s ≈ t.
Proof. constructor; trivial. Qed.
Definition eq_trans : forall x y z, x ≈ y → y ≈ z → x ≈ z.
Proof. apply rst_trans. Qed.
Definition eq_sym : forall x y, x ≈ y → y ≈ x.
Proof. apply rst_sym. Qed.
Definition eq_refl : forall x, x ≈ x.
Proof. apply rst_refl. Qed.
Opaque Equiv.

Hint Resolve eq_step eq_trans eq_sym eq_refl : diff.

(* Differential equivalence is contextual *)
Theorem eq_contextual :
  Contextual Equiv.
Proof with (auto with diff).
  intros s s' cx; induction cx; auto.
  constructor...
  constructor...
  - induction IHcx...
    + apply eq_trans with (y := Λ y)...
  - induction IHcx...
    + apply eq_trans with (y := ε y)...
  - apply eq_trans with (y := { s' | t }).
    + induction IHcx1...
      * eapply eq_trans...
    + induction IHcx2...
      * eapply eq_trans...
  - apply eq_trans with (y := s' + t).
    + induction IHcx1...
      * eapply eq_trans...
    + induction IHcx2...
      * eapply eq_trans...
  - apply eq_trans with (y := D (s') · t).
    + induction IHcx1...
      * eapply eq_trans...
    + induction IHcx2...
      * eapply eq_trans...
Qed.
Hint Resolve eq_contextual : diff.

(*
  The tactic Cx tries to prove a goal of the form Rho t1 t2,
  where t1, t2 are terms and Rho is a contextual relation.
  It does so by applying contextuality and adding the resulting
  subcomponents as goals.
*)
```

```
Ltac Cx :=
  let cx := fresh "cx" in
  lazymatch goal with
  | |- ρ? (ε ?u) (ε ?v) ⇒
      assert (Contextual ρ) as cx;
      [auto with diff | apply cx; apply (CE ρ u v)]

  | |- ρ? (Λ ?u) (Λ ?v) ⇒
      assert (Contextual ρ) as cx;
      [auto with diff | apply cx; apply (CAbs ρ u v)]

  | |- ρ? 0 0 ⇒
      assert (Contextual ρ) as cx;
      [auto with diff | apply cx; apply (CZero ρ)]

  | |- ρ? (Var ?n) (Var ?m) ⇒
      assert (Contextual ρ) as cx;
      [auto with diff | apply cx; apply (CVar ρ n)]

  | |- ρ? (?s + ?t) (?s' + ?t') ⇒
      assert (Contextual ρ) as cx;
      [auto with diff | apply cx; apply (CAdd ρ s s' t t')]

  | |- ρ? { ?s | ?t } { ?s' | ?t' } ⇒
      assert (Contextual ρ) as cx;
      [auto with diff | apply cx; apply (CApp ρ s s' t t')]

  | |- ρ? (D ?s · ?t) (D ?s' · ?t') ⇒
      assert (Contextual ρ) as cx;
      [auto with diff | apply cx; apply (CD ρ s s' t t')]

  | |- ?g ⇒ fail "Not a contextual goal " g
  end; clear cx; try apply CRho.

Ltac Cxx := repeat Cx.

Ltac eq_step ctor := (
  eapply eq_trans; [| apply eq_step; apply CRho; apply ctor]
).

Ltac eq_rstep ctor := (
  apply eq_sym; eq_step ctor; apply eq_sym
).

(*
  Term rewriting modulo differential equivalence.
  This should probably be moved to setoid rewriting.

  This tactic should never be called. Instead prefer the "local rewrite"
  and "local rewrite←" macros defined later.
*)
Ltac local_rewrite u v eqn := match goal with
| [|- ?t1 ≈ ?t2] ⇒
  lazymatch t1 with
```

```
    | context tm [u] ⇒
      let qu := context tm[u] in
      let qv := context tm[v] in
      let lhs := fresh "lhs" in
      let rhs := fresh "rhs" in
      let eq := fresh "eq" in
      pose (lhs := u);
      pose (rhs := v);
      assert (u ≈ v → qu ≈ qv) as eq by (
        let hyp := fresh "hyp" in
        fold rhs lhs;
        intro hyp; Cxx; try apply eq_refl;
        try exact hyp;
        match goal with
        | |- ?H ⇒ idtac "I can't prove contextuality in local rewrite: " H;
          idtac "Falling back on evars, I hope someone can prove this later";
          let hack := fresh "hack" in
          evar (hack : H); exact hack
        end);
      apply eq_trans with (x := qu) (y := qv);
      fold lhs rhs;
      [apply eq; apply eqn|];
      unfold lhs, rhs;
      (*try (solve [ auto with diff | try apply eq_refl]; fail);*)
      clear lhs rhs eq
    | _ ⇒
      lazymatch t2 with
      | context tm [u] ⇒ apply eq_sym; local_rewrite u v eqn; apply eq_sym
      | _ ⇒ fail "I can't find lhs" u "in goal" t1 "≈" t2
      end
    end
  end.

Ltac local_rewrite_eq eq :=
  lazymatch type of eq with
  | ?u ≈ ?v ⇒ (local_rewrite u v eq)
  | clos_refl_sym_trans Term (Ctx Equiv1) ?u ?v
    ⇒ local_rewrite u v (eq_step u v eq)
  | Equiv1 ?u ?v
    ⇒ (local_rewrite u v (eq_step u v (CRho Equiv1 u v eq)))
  | ?u ~1 ?v
    ⇒ (local_rewrite u v (eq_step u v (CRho Equiv1 u v eq)))
  | forall y, ·?w y ⇒ (
      let eqname := fresh "eq" in
      epose (eq _) as eqname; simpl in eqname;
      let the_expr := (eval hnf in eqname) in
      local_rewrite_eq the_expr;
      clear eqname
    )
  | _ ⇒ fail "I do not understand " eq " as an equation"
  end.

Ltac local_rewrite_rev eq :=
  lazymatch type of eq with
  | ?u ≈ ?v ⇒ (local_rewrite v u eq)
  | clos_refl_sym_trans Term (Ctx Equiv1) ?u ?v
    ⇒ local_rewrite v u (eq_sym u v (eq_step u v eq))
  | Equiv1 ?u ?v
    ⇒ local_rewrite v u (eq_sym u v (eq_step u v (CRho Equiv1 u v eq)))
  | forall y, ·?w y ⇒ (
     let eqname := fresh "eq" in
     epose (eq _) as eqname; simpl in eqname;
     let the_expr := (eval hnf in eqname) in
     local_rewrite_rev the_expr;
     clear eqname
    )
  | _ ⇒ fail "I do not understand " eq " as an equation"
  end.

Tactic Notation "local" "rewrite" constr(eq) :=
  local_rewrite_eq eq.

Tactic Notation "local" "rewrite" "←" constr(eq) :=
  local_rewrite_rev eq.

Lemma Eq_0_add : forall t, 0 + t ≈ t.
Proof. intro.
  local rewrite Eq_add_comm.
  local rewrite Eq_add_0.
  apply eq_refl.
Qed.

(* Simplifying a term by applying every equivalence rule once *)
Ltac simplify :=
  try local rewrite Eq_add_0;
  try local rewrite Eq_0_add;
  try local rewrite Eq_add_assoc;
  try local rewrite Eq_add_E;
  try local rewrite Eq_E_0;
  try local rewrite Eq_app_0;
  try local rewrite Eq_app_add;
  try local rewrite Eq_app_E;
  try local rewrite Eq_lam_0;
  try local rewrite Eq_lam_add;
  try local rewrite Eq_lam_E;
  try local rewrite Eq_D_0;
  try local rewrite Eq_D_add;
  try local rewrite Eq_D_E;
  try local rewrite Eq_D_0_r;
  try local rewrite Eq_D_E_r;
  try local rewrite Eq_regularity;
  try local rewrite Eq_derivative;
  try local rewrite Eq_D_inf.

Ltac simplify2 := try simplify; apply eq_sym; try simplify; apply eq_sym.

(*
  Reduce a term to a (near) canonical form by iterating
  one-step simplification. This is VERY SLOW. Also note
  that sometimes it won't get to a canonical form,
  depending on how reduction happens.
*)
Ltac can := repeat simplify; apply eq_sym; repeat simplify; apply eq_sym.

(*
  Right-associate all the additions in a term.
*)
Ltac associate := repeat local rewrite Eq_add_assoc.

(*
  Turn a term of the form u + (v + w) into v + (w + u).
  Usually Cx gets called immediately after this.
*)
Ltac rotate :=
match goal with
| [|- ?u + ?v ≈ _] ⇒ local rewrite (Eq_add_comm u v); repeat associate
end.

Tactic Notation "rot" integer(i) := do i rotate.

Tactic Notation "r" tactic(t) := apply eq_sym; t; apply eq_sym.

(*
  Rotate the term in the left-hand side until the leftmost
  summand of both terms is the same. Note: very expensive.
  Prefer the listify combinators in the next file.
*)
Ltac align n :=
match eval compute in n with
| O ⇒ idtac
| S ?n' ⇒ match goal with
          | [|- ?u + _ ≈ ?u + _ ] ⇒ idtac
          | [|- _ ] ⇒ rotate; align n'
          end
end.

Example test_cx : forall t, (t + t + ε ({t | t})) ≈ t + t + ε {t | t + 0}.
intro.
  local rewrite (Eq_add_comm t 0).
  local rewrite (Eq_0_add).
  Cxx; apply eq_refl.
Qed.
```

A.2 Module **Substitution.v**

```
Require Import Unicode.Utf8.
Require Import Coq.Relations.Relation_Operators.
Require Import Coq.Relations.Relation_Definitions.
Require Import Coq.Arith.EqNat.
Require Import Coq.Arith.Compare_dec.
Require Import Coq.Arith.PeanoNat.
Require Import Omega.

Add LoadPath "./".
Load Definitions.

Lemma lt_leb : forall m n, m < n → m <=? n = true.
Proof. intros; assert (m ≤ n) by omega; now apply leb_correct. Qed.

Lemma lt_sym : forall m n, m > n → n < m.
Proof. intros; omega. Qed.

(*
  A tactic for turning inequalities in Prop to
  equalities in bool
*)
Ltac bool_from_nat eq :=
  lazymatch type of eq with
  | ?m < ?n ⇒
    lazymatch goal with
    | |- ?m <=? ?n = true ⇒ apply lt_leb
    | |- ?m <? ?n = true ⇒ apply Nat.ltb_lt
    | |- ?m ?= ?n = Lt ⇒ apply nat_compare_lt
    | |- ?n <=? ?m = false ⇒ apply leb_iff_conv
    | |- ?n ?= ?m = Gt ⇒ apply nat_compare_gt; apply lt_sym
    end; exact eq
  | ?m = ?n ⇒
    lazymatch goal with
    | |- ?m <=? ?n = true ⇒ (rewrite eq; apply Nat.leb_refl)
    | |- ?n <=? ?m = true ⇒ (rewrite eq; apply Nat.leb_refl)
    | |- ?m ?= ?n = Eq ⇒ apply Nat.compare_eq_iff; exact eq
    end
  end.

Tactic Notation "bool" constr(target) "from" constr(eq) "as" ident(eqname) :=
  assert target as eqname by bool_from_nat eq.
Tactic Notation "bool" "from" constr(eq) := bool_from_nat eq.
Tactic Notation "bool" constr(target) "from" constr(eq) :=
  let eqname := fresh "beq" in bool target from eq as eqname.

Tactic Notation "bool" "rewrite" constr(target) "from" constr(eq)
                "in" hyp_list(HS) "|-" "*" :=
  let beq := fresh "beq" in bool target from eq as beq;
  repeat try rewrite beq in HS|-; repeat try rewrite beq; clear beq.
Tactic Notation "bool" "rewrite" constr(target) "from" constr(eq)
                "in" "*" :=
  let beq := fresh "beq" in bool target from eq as beq;
  repeat try rewrite beq in *|-*; repeat try rewrite beq; clear beq.
Tactic Notation "bool" "rewrite" constr(target) "from" constr(eq) :=
  bool rewrite target from eq in |-*.

(*
  Prove a boolean goal whenever the propositional version
  can be proven by Omega.
  Slow and overkill, prefer bool beq from eq
*)
Ltac decide_nat :=
  match goal with
  | [|- ?m <=? ?n = true ] ⇒
    assert (m <= n) by omega; now apply leb_correct
  | [|- ?m <=? ?n = false ] ⇒
    assert (m > n) by omega; now apply leb_correct_conv
  | [|- ?m <? ?n = true ] ⇒
    assert (m < n) by omega; now apply Nat.ltb_lt
  | [|- ?m <? ?n = false ] ⇒
    assert (n <= m) by omega; now apply Nat.ltb_ge
  | [|- ?m =? ?n = true ] ⇒
    assert (m = n) by omega; now apply Nat.eqb_eq
  | [|- ?m =? ?n = false ] ⇒
    assert (m ≠ n) by omega; now apply Nat.eqb_neq
  | [|- ?m ?= ?n = Lt ] ⇒
    assert (m < n) by omega; now apply nat_compare_lt
  | [|- ?m ?= ?n = Eq ] ⇒
    assert (m = n) by omega; now apply Nat.compare_eq_iff
  | [|- ?m ?= ?n = Gt ] ⇒
    assert (m > n) by omega; now apply nat_compare_gt
  end.

(* Try to prove any goal from an absurd arithmetic premise *)
Ltac prune eq :=
  lazymatch type of eq with
  | ?m < 0 ⇒ solve [exfalso; apply (Nat.nlt_0_r m eq)]
  | ?m < ?m ⇒ solve [exfalso; apply (Nat.lt_irrefl m eq)]
  | ?m < ?n ⇒
    lazymatch goal with
    | [h:(n < m) |- _] ⇒
      solve [exfalso; apply (Nat.lt_asymm n m h eq)]
    | [h:(m > n) |- _] ⇒
      solve [exfalso; apply (Nat.lt_asymm n m (lt_sym m n h) eq)]
    | [h:(n = m) |- _] ⇒
      solve [exfalso; apply (Nat.lt_neq m n eq); rewrite h; auto]
    | [h:(m = n) |- _] ⇒
      solve [exfalso; apply (Nat.lt_neq m n eq); rewrite h; auto]
    | _ ⇒ idtac
    end
  | ?n = ?m ⇒
    lazymatch goal with
    | [h:(n < m) |- _] ⇒
      solve [exfalso; apply (Nat.lt_neq n m h); rewrite eq; auto]
    | [h:(n > m) |- _] ⇒
```

```
      solve [exfalso; apply (Nat.lt_neq m n (lt_sym _ _ h)); rewrite eq; auto]
    | [h:(m > n) |- _] ⇒
      solve [exfalso; apply (Nat.lt_neq n m (lt_sym _ _ h)); rewrite eq; auto]
    | [h:(m < n) |- _] ⇒
      solve [exfalso; apply (Nat.lt_neq m n h); rewrite eq; auto]
    | _ ⇒ idtac
    end
  end.

(*
  Compare two naturals and generate one goal for each reasonable case.
  Uses already-existing information to prune absurd branches.
*)
Ltac compare m n :=
  let name := fresh "name" in
  let cmp := fresh "cmp" in
  let eq := fresh "eq" in
  destruct (lt_eq_lt_dec m n) as [cmp|eq];
  [destruct cmp as [eq|eq];
    [ bool rewrite (m <=? n = true) from eq in *;
      bool rewrite (m ?= n = Lt) from eq in *;
      prune eq
    | try bool rewrite (m <=? n = true) from eq in *;
      try bool rewrite (m ?= n = Eq) from eq in *;
      prune eq
  ] | bool rewrite (m <=? n = false) from eq in *;
      bool rewrite (m ?= n = Gt) from eq in *;
      prune eq
  ]; simpl.

(*
  Unfold every arithmetic comparison in the goal into its
  corresponding branches
*)
Ltac trivial_arithmetic :=
  match goal with
  | [|- ?p ] ⇒
    match p with
    | context foo [?u ?= ?v] ⇒ compare u v
    | context foo [?u <=? ?v] ⇒ compare u v
    end
  | [ H:?h |- _] ⇒
    match h with
    | context foo [?u ?= ?v] ⇒ compare u v
    | context foo [?u <=? ?v] ⇒ compare u v
    end
  end; try (omega).

Arguments leb n m : simpl never.
Arguments Nat.compare n m : simpl never.

(*
  Increase every DeBruijn index above n by one.
*)
```

```
Fixpoint open (n : nat) (u : Term) : Term :=
match u with
| Var m   ⇒ Var (if n <=? m then S m else m)
| 0       ⇒ 0
| {s|t}   ⇒ {open n s|open n t}
| s + t   ⇒ open n s + open n t
| D s · t ⇒ D (open n s) · (open n t)
| ε t     ⇒ ε (open n t)
| Λ t     ⇒ Λ (open (S n) t)
end.

Lemma open_open_le
  :  forall t x y
  ,  x ≤ y
  → open x (open y t) = open (S y) (open x t).
Proof with auto with arith.
  intro t; induction t; intros x y le; simpl;
  try rewrite IHt; try rewrite IHt1; try rewrite IHt2...
  - repeat trivial_arithmetic...
Qed.
Hint Resolve open_open_le : diff.

(*
  Decrease every DeBruijn index above n by one.
*)
Fixpoint close (n : nat) (u : Term) : Term :=
match u with
| Var m   ⇒ Var (if n <=? m then pred m else m)
| 0       ⇒ 0
| {s|t}   ⇒ {close n s|close n t}
| s + t   ⇒ close n s + close n t
| D s · t ⇒ D (close n s) · (close n t)
| ε t     ⇒ ε (close n t)
| Λ t     ⇒ Λ (close (S n) t)
end.

Lemma close_close_le
  :  forall t x y
  ,  x ≤ y
  → close x (close (S y) t) = close y (close x t).
Proof with auto with arith.
  intro t; induction t; intros x y le; simpl;
  try rewrite IHt; try rewrite IHt1; try rewrite IHt2...
  - repeat trivial_arithmetic... f_equal. omega.
Qed.

Lemma close_open_eq
  :  forall t x
  ,  close x (open x t) = t.
Proof with auto with arith.
  intro t; induction t; intro x; simpl;
  try rewrite IHt; try rewrite IHt1; try rewrite IHt2...
  - repeat trivial_arithmetic...
Qed.
```

```
Lemma close_open_eq_S
  :  forall t x
  ,  close (S x) (open x t) = t.
Proof with auto with arith.
  intro t; induction t; intro x; simpl;
  try rewrite IHt; try rewrite IHt1; try rewrite IHt2...
  - repeat trivial_arithmetic...
Qed.

Lemma close_open_lt
  :  forall t x y
  ,  x < y
  → close x (open (S y) t) = open y (close x t).
Proof with auto with arith.
  intro t; induction t; intros x y lt; simpl;
  try rewrite IHt; try rewrite IHt1; try rewrite IHt2...
  - repeat trivial_arithmetic...
    f_equal; omega.
    f_equal; omega.
Qed.

Lemma close_open_gt
  :  forall t x y
  ,  x > y
  → close (S x) (open y t) = open y (close x t).
Proof with auto with arith.
  intro t; induction t; intros x y ge; simpl;
  try rewrite IHt; try rewrite IHt1; try rewrite IHt2...
  - repeat trivial_arithmetic... all: f_equal; try omega.
Qed.

(*
  Shifting DeBruijn indices is compatible with differential equivalence
*)
Lemma open_diff
  :  forall t t'
  ,  t ≈ t'
  → forall x
  ,  open x t ≈ open x t'.
Proof with (auto with diff).
  intros t t' eq; induction eq...
  - induction H; intros; simpl...
    induction H; simpl...
  - intro; eapply eq_trans...
Qed.
Hint Resolve open_diff : diff.

Lemma close_diff
  :  forall t t'
  ,  t ≈ t'
  → forall x
  ,  close x t ≈ close x t'.
Proof with (auto with diff).
```

```
  intros t t' eq; induction eq...
  - induction H; intros; simpl...
    induction H; simpl...
  - intro; eapply eq_trans...
Qed.
Hint Resolve close_diff : diff.

(* Like substitution, except it doesn't actually bind the variable *)
Fixpoint replace (t : Term) (x : nat) (u : Term) : Term :=
match t with
| Var m   ⇒
  (match Nat.compare x m with
  | Lt ⇒ Var m
  | Eq ⇒ u
  | Gt ⇒ Var m
  end)
| 0       ⇒ 0
| {s | t} ⇒ { replace s x u | replace t x u }
| D s · t ⇒ D (replace s x u) · (replace t x u)
| s + t   ⇒ (replace s x u) + (replace t x u)
| ε t     ⇒ ε (replace t x u)
| Λ t     ⇒ Λ (replace t (S x) (open 0 u))
end.

Lemma replace_open_eq
  :  forall t
  ,  forall x u
  ,  replace (open x t) x u = (open x t).
Proof with auto.
  intro t; induction t; intros; simpl; auto;
  try rewrite IHt; try rewrite IHt1; try rewrite IHt2...
  - repeat trivial_arithmetic...
Qed.

Lemma replace_open_le
  :  forall t
  ,  forall x y u
  ,  x ≤ y
  → replace (open x t) (S y) (open x u) = open x (replace t y u).
Proof with auto.
  intro t; induction t; intros; simpl; auto;
  try rewrite IHt; try rewrite IHt1; try rewrite IHt2...
  - repeat trivial_arithmetic...
  - f_equal; rewrite open_open_le; auto with arith; rewrite IHt.
Qed.

Lemma replace_replace
  :  forall t x y u v
  ,  x ≠ y
  → replace (replace t x u) y (open x v)
     = replace (replace t y (open x v)) x (replace u y (open x v)).
Proof with auto with diff arith.
  intro t; induction t; intros x y u v ne; simpl in *|-*;
  try rewrite IHt1; try rewrite IHt2; try rewrite IHt...
  - repeat trivial_arithmetic...
    all: rewrite replace_open_eq...
  - repeat rewrite ← replace_open_le...
    repeat rewrite (open_open_le _ 0)...
    rewrite IHt...
Qed.

Lemma replace_diff_new
  :  forall t
  ,  forall x s s'
  ,  s ≈ s'
  → replace t x s ≈ replace t x s'.
Proof with (auto with diff).
  intro t; induction t; intros x s s' eq; simpl...
  compare x n... Cx...
Qed.

Lemma replace_diff_base
  :  forall t t'
  ,  t ≈ t'
  → forall x s
  ,  replace t x s ≈ replace t' x s.
Proof with (auto with diff).
  intros t t' eq; induction eq...
  - induction H; intros; simpl; Cxx...
    induction H; simpl...
  - intros; eauto using eq_trans with diff.
Qed.

(* Replacing a variable by a term respects differential equivalence *)
Lemma replace_diff
  :  forall t t'
  ,  t ≈ t'
  → forall s s'
  ,  s ≈ s'
  → forall x
  ,  replace t x s ≈ replace t' x s'.
Proof with (auto with diff).
  intros.
  local rewrite replace_diff_base.
  local rewrite replace_diff_new.
  apply eq_refl.
Unshelve. all: auto.
Qed.
Hint Resolve replace_diff : diff.

(*
  Standard, capture-avoiding substitution -- we make sure that the
  variable that we are substituting does not appear free in the
  new sub-term by opening it beforehand.
*)
Definition subst t x u := close (S x) (replace t x (open x u)).
Notation "t [ x := u ]" := (subst t x u) (at level 85).
Reserved Notation "t [ u / x ]" (at level 90).
```

```
Hint Unfold subst : diff.

Theorem subst_diff
  :  forall s s'
  ,  s ≈ s'
  → forall t t'
  ,  t ≈ t'
  → forall x
  ,  (s[x := t]) ≈ (s'[x := t']).
Proof with (auto with diff).
  intros; simpl; unfold subst...
Qed.

(*
  Differential substitution
*)
Reserved Notation "( ∂ t // ∂ x ) [ u ]" (at level 85).
Fixpoint dsubst (t : Term) (x : nat) (u : Term) : Term :=
match t with
| Var m   ⇒
  (match Nat.compare x m with
  | Lt ⇒ 0
  | Eq ⇒ u
  | Gt ⇒ 0
  end)
| 0       ⇒ 0
| {t | e} ⇒
    { D t · ∂((e // ∂x)[u]) | e }
    + { ∂(t // ∂x)[u] | (replace e x (Var x + ε u)) }
| D t · e ⇒
    D (t) · ∂((e // ∂x)[u])
    + D∂((t // ∂x)[u]) · (replace e x (Var x + ε u))
    + ε(D (D t · e) · ∂((e // ∂x)[u]))
| s + t   ⇒ ∂(( s // ∂ x)[u]) + ∂(( t // ∂ x)[u])
| ε t     ⇒ ε ∂((t // ∂x)[u])
| Λ t     ⇒ Λ ∂(( t // ∂ (S x))[open 0 u])
end
where "( ∂ t // ∂ x ) [ u ]" := (dsubst t x u).

Lemma open_injective
  :  forall s t
  ,  forall m
  ,  open m s = open m t
  → s = t.
Proof with auto.
  intro s; induction s; intro t; destruct t; intros m eq;
  simpl in eq; try inversion eq; try f_equal; eauto.
  - compare m n; compare m n0; auto; try omega.
Qed.

Lemma open_subst_ge
  :  forall t x y u
  ,  x ≥ y
  → (open (S x) t) [y := open x u] = open x (t[y := u]).
```

```
Proof with (auto with arith).
  intro t; induction t; intros x y u le; simpl; [idtac|f_equal..];
  unfold subst in *|-*; simpl in *|-*;
  try (rewrite IHt1); try (rewrite IHt2); try rewrite (IHt)...
  - destruct n.
    + repeat trivial_arithmetic; auto;
      do 2 rewrite close_open_eq_S...
    + destruct n; repeat trivial_arithmetic; auto;
      do 2 rewrite close_open_eq_S...
  - f_equal; repeat rewrite (open_open_le _ 0 )...
Qed.

Tactic Notation "and" tactic(t) := [>  t|..].
Tactic Notation "and" "finally" tactic(t) := [> t.. ] .

Lemma subst_subst_le
  :  forall t x u y v
  ,  x ≤ y
  → (t [x := u])[y := v]
     = ((t[S y := open x v])[x := u[y:=v]]).
Proof with (auto with arith).
  intro t; induction t; intros;
  unfold subst in *|-*; simpl in *|-*; auto with diff;
  try rewrite IHt1; try rewrite IHt2; try rewrite IHt...
  - repeat trivial_arithmetic...
    all: do 2 rewrite close_open_eq_S; auto;
      rewrite replace_open_eq;
      rewrite close_open_eq_S...
  - f_equal. repeat rewrite (open_open_le _ 0); auto with arith.
    rewrite (IHt (S x));(
    and do 3 f_equal;
    and (rewrite← (open_open_le _ 0));
    and rewrite replace_open_le;
    and rewrite close_open_gt)...
Qed.

Lemma subst_subst_gt
  :  forall t x u y v
  ,  x > y
  → (t [S x := u])[y := v] = ((t[y := open x v])[x := u[y := v]]).
Proof with auto with arith.
  intro t; induction t; intros; simpl; unfold subst in *|-*; simpl in *|-*;
  try rewrite IHt; try rewrite IHt1; try rewrite IHt2...
  - repeat trivial_arithmetic...
    all: do 2 rewrite close_open_eq_S...
      rewrite replace_open_eq; rewrite close_open_eq_S...
  - f_equal; repeat rewrite (open_open_le _ 0); auto with arith; (
      rewrite (IHt (S x));
      and do 3 f_equal;
      and rewrite← close_open_gt;
      and rewrite← open_open_le;
      and rewrite replace_open_le)...
Qed.
```

```
Lemma replace_E : forall t x u, ε (replace t x u) = replace (ε t) x u.
Proof. now simpl. Qed.
Lemma replace_0 : forall x u, 0 = replace 0 x u.
Proof. now simpl. Qed.
Lemma replace_add : forall s t x u,
  replace s x u + replace t x u = replace (s + t) x u.
Proof. now simpl. Qed.
Lemma replace_app : forall s t x u,
  { replace s x u | replace t x u } = replace { s | t } x u.
Proof. now simpl. Qed.
Lemma replace_D : forall s t x u,
  D (replace s x u) · (replace t x u) = replace (D s · t) x u.
Proof. now simpl. Qed.

(*
  A tactic for automatically folding calls to replace
  when simpl or unfold go too far.
*)
Ltac fold_replace :=
  repeat (progress (
      try rewrite replace_E;
      try rewrite replace_0;
      try rewrite replace_add;
      try rewrite replace_app;
      try rewrite replace_D
    )
  ).

Lemma open_E : forall x t, ε (open x t) = open x (ε t).
Proof. now simpl. Qed.
Lemma open_0 : forall x, 0 = open x 0.
Proof. now simpl. Qed.
Lemma open_add : forall x s t, (open x s) + (open x t) = open x (s + t).
Proof. now simpl. Qed.
Lemma open_app : forall x s t, ({open x s | open x t}) =  (open x {s | t}).
Proof. now simpl. Qed.
Lemma open_D : forall x s t, D (open x s) · (open x t) = open x (D s · t).
Proof. now simpl. Qed.

(*
  A tactic for automatically folding calls to open
  when simpl or unfold go too far.
*)
Ltac fold_open :=
  progress (
    try rewrite open_E;
    try rewrite open_0;
    try rewrite open_add;
    try rewrite open_app;
    try rewrite open_D
  ).

(* Replacing a variable by itself is the identity *)
Lemma replace_trivial
```

```
  :  forall t x
  ,  t = replace t x (Var x).
Proof with (auto with diff).
  intro t; induction t; intros; simpl; f_equal...
  trivial_arithmetic...
Qed.

Lemma replace_update
  :  forall t x u v
  ,  replace (replace t x u) x v = replace t x (replace u x v).
Proof with auto with diff. intro t; induction t; intros; simpl; f_equal...
  - repeat trivial_arithmetic; auto; rewrite replace_open_eq...
  - rewrite IHt; rewrite replace_open_le; auto with arith.
Qed.


(*
  Differential substitution of a variable that does not appear free
  in a term is zero.
*)
Lemma dsubst_empty
  :  forall t x u
  ,  ∂( (open x t) // ∂ x)[u] ≈ 0.
Proof with (auto with diff).
  intro t; induction t; intros; simpl;
  try local rewrite IHt; try local rewrite IHt1;
  repeat local rewrite IHt2; can...
  - repeat trivial_arithmetic...
Qed.

(*
  A handy tactic for introducing shorthands in proofs with
  lots of partial derivatives.
*)
Tactic Notation "differential" ident(name) ":="
  "d" constr(t) "/" "d" ident(x) "[" constr(u) "]"
  := pose (name := ∂( t // ∂ x)[open x u]); fold name.

(*
  A tactic for commuting the arguments to a differential
  application.
*)
Ltac drot i expr :=
  lazymatch expr with
  | D (D ?t · ?u) · ?v ⇒
    match eval compute in i with
    | O ⇒ local rewrite (Eq_D_comm t u v)
    | S ?i' ⇒ drot i' (D t · u)
    end
  | _ ⇒ fail "Not a differential goal"
  end.

Tactic Notation "d" "rot" constr(pos) :=
  match goal with
  | |- ?w ≈ _ ⇒ drot pos w
  end.

(*
  Handling raw terms is slow because of how Coq handles pattern
  matching. The combinators below allow for splitting a term into
  a list of its summands and operating on that instead. Avoids a lot
  of canonicalisation headaches.
*)
Require Import List.

Definition sum (l : list Term) := fold_right (fun s t ⇒ s + t) 0 l.

Definition eqs (l1 l2 : list Term) := sum l1 ≈ sum l2.

Notation "l1 ≈[] l2" := (eqs l1 l2) (at level 80).

Lemma eqs_trans : forall l1 l2 l3, l1 ≈[] l2 → l2 ≈[] l3 → l1 ≈[] l3.
intros; eapply eq_trans; eauto.
Qed.
Lemma eqs_sym : forall l1 l2, l1 ≈[] l2 → l2 ≈[] l1.
intros; apply eq_sym; auto.
Qed.

Lemma eqs_start : forall t t',  (t :: nil) ≈[] (t' :: nil) → t ≈ t'.
unfold eqs; unfold sum; intros; simpl in H; local rewrite ← Eq_add_0.
r local rewrite ← Eq_add_0. auto.
Qed.

Ltac local_rewrite_eqs eq prf :=
  lazymatch type of eq with
  | ?ll1 ≈[] ?ll2 ⇒
    eapply eqs_trans with (l2 := ll2); [eapply prf|]
  | forall y, ·?w y ⇒ (
      let eqname := fresh "eq" in
      epose (eq _) as eqname; simpl in eqname;
      let the_expr := (eval hnf in eqname) in
      local_rewrite_eqs the_expr prf;
      clear eqname
    )
  end.

Tactic Notation "[rewrite]" constr(eq) :=
  local_rewrite_eqs eq eq; simpl "++".

Tactic Notation "[l]" tactic(tac) :=
  tac.

Tactic Notation "[r]" tactic(tac) :=
  apply eqs_sym; tac; apply eqs_sym.

Lemma sum_diff : forall l a a', a ≈ a' →
  fold_right (fun s t ⇒ s + t) a l ≈ fold_right (fun s t ⇒ s + t) a' l.
Proof with auto with diff. intro l; induction l; intros; unfold sum; simpl...
```

```
Qed.

Lemma sum_extra : forall l s t,
  fold_right (fun s t ⇒ s + t) (s + t) l
  ≈ fold_right (fun s t ⇒ s + t) s l + t.
Proof with auto with diff. intro l; induction l; intros; unfold sum; simpl...
  - can; Cx...
Qed.

Lemma fold_append : forall (l1 : list Term) l2 f (x : Term),
  fold_right f x (l1 ++ l2) = fold_right f (fold_right f x l2) l1.
Proof with auto. intro l1; induction l1; intros; simpl...
  - rewrite IHl1...
Qed.

Lemma split_eqs : forall l1 l2, sum (l1 ++ l2) ≈ (sum l1) + (sum l2).
Proof with auto with diff.
  intro l1; induction l1; intros; unfold sum; simpl.
  - fold (sum l2); can; apply eq_refl.
  - fold (sum (l1 ++ l2)) (sum l1) (sum l2); can; Cx...
Qed.

Theorem rots : forall l t, t :: l ≈[] l ++ (t :: nil).
Proof with auto with diff.
  intros; unfold eqs; local rewrite (split_eqs l (t :: nil));
  unfold sum; simpl... can; local rewrite Eq_add_comm; Cx...
Qed.

Tactic Notation "[rot]" integer(i) := do i ([rewrite] rots).

Theorem splits : forall l t1 t2, (t1 + t2) :: l ≈[] t1 :: t2 :: l.
Proof with auto with diff. intros; unfold eqs; unfold sum; simpl...
Qed.

Theorem eqs_diff : forall t t', t ≈ t' → forall l, t :: l ≈[] t' :: l.
intros; unfold eqs; unfold sum; simpl; local rewrite H; apply eq_refl.
Qed.

Tactic Notation "[*]" tactic(tac) := [rewrite] eqs_diff;
  and tac; and eapply eq_refl.

Ltac splits :=
  match goal with
  | |- (?s + ?t) :: ?l ≈[] _ ⇒ [rewrite] (splits l s t)
  end.

Tactic Notation "[can]" := [*] can.

Ltac stomp := [can];
  ([*] repeat local rewrite ← Eq_add_assoc);
  repeat splits.
Lemma eqs_refl : forall l, l ≈[] l.
Proof with auto with diff.
  induction l; unfold eqs; unfold sum; simpl...
```

```
Qed.
Tactic Notation "[refl]" := apply eqs_refl.

Ltac listify := apply eqs_start.

Tactic Notation "[aligns]" :=
  timeout 1 repeat match goal with
  | |- ?t :: _ ≈[] ?t :: _ ⇒ idtac "found" t
  | |- ?u :: _ ≈[] ?v :: _ ⇒ [rot] 1
  end.

Lemma pop : forall t t' l l', t ≈ t' → l ≈[] l' → (t :: l) ≈[] (t' :: l').
intros; unfold eqs; unfold sum; simpl; Cx; auto using eq_refl.
Qed.
Ltac pop := apply pop.

Lemma replace_open_lt
  :  forall t x y u
  ,  x < y
  → replace (open y t) x (open y u) = open y (replace t x u).
Proof with auto with diff.
  intro t; induction t; intros; simpl; [|repeat f_equal..]...
  - repeat trivial_arithmetic...
  - rewrite open_open_le; and rewrite IHt... all: auto with arith.
Qed.

Lemma replace_open_ge
  :  forall t x y u
  ,  x ≥ y
  → replace (open y t) (S x) (open y u) = open y (replace t x u).
Proof with auto with diff.
  intro t; induction t; intros; simpl; [|repeat f_equal..]...
  - repeat trivial_arithmetic...
  - rewrite open_open_le; and rewrite IHt... all: auto with arith.
Qed.

Lemma dsubst_open_lt
  :  forall t x y u
  ,  x < y
  →  dsubst (open y t) x (open y u) = open y (dsubst t x u).
Proof with auto with diff arith.
  intro t; induction t; intros; simpl...
  - repeat trivial_arithmetic...
  - f_equal; rewrite open_open_le; and rewrite IHt...
  - f_equal.
    + rewrite IHt2...
    + rewrite IHt1... rewrite ← replace_open_lt; simpl...
      trivial_arithmetic...
  - f_equal; auto with diff; rewrite ← replace_open_lt; simpl;
    trivial_arithmetic...
  - f_equal; rewrite IHt...
  - f_equal; and f_equal.
    + rewrite IHt2...
    + rewrite IHt1... rewrite ← replace_open_lt; simpl...
```

```
      trivial_arithmetic...
    + rewrite IHt2...
Qed.

Lemma dsubst_open_ge
  :  forall t x y u
  ,  x ≥ y
  → dsubst (open y t) (S x) (open y u) = open y (dsubst t x u).
Proof with auto with diff arith.
  intro t; induction t; intros; simpl;
  try f_equal; try rewrite IHt; try rewrite IHt1; try rewrite IHt2...
  - repeat trivial_arithmetic...
  - rewrite open_open_le; and rewrite IHt...
  - rewrite ← replace_open_ge; simpl... trivial_arithmetic...
  - rewrite ← replace_open_ge; simpl... trivial_arithmetic...
Qed.
```

A.3 Module **Theorems.v**

```
(*
  Mechanically checked proofs of Taylor's theorem, regularity of
  differential substitution, and other "hard" results.
*)
Require Import Unicode.Utf8.
Require Import Coq.Relations.Relation_Operators.
Require Import Coq.Relations.Relation_Definitions.
Require Import Coq.Arith.EqNat.
Require Import Coq.Arith.Compare_dec.
Require Import Coq.Arith.PeanoNat.
Require Import Omega.

Add LoadPath "./".

Require Import Substitution.

(* Syntactic Taylor's Theorem *)
Theorem Taylor
  :  forall s x t u
  ,  replace s x (t + ε (open x u))
     ≈ replace (s + ε ∂(( s // ∂ x)[ open x u ])) x t.
Proof with (auto with diff).
  intro s; induction s; intros; simpl...
  - trivial_arithmetic; simplify2...
    rewrite replace_open_eq...
  - rewrite open_open_le; and (local rewrite IHs; simpl; can)...
    auto with arith.
  - local rewrite IHs1; local rewrite IHs2; simpl; can... Cx...
    fold_replace; apply replace_diff_base; local rewrite IHs2.
    simpl; can; repeat rewrite← replace_trivial; Cxx...
  - can...
  - local rewrite IHs1; local rewrite IHs2; simpl; can.
    Cx... rot 1; Cx...
  - local rewrite IHs; simpl...
  - local rewrite IHs1; local rewrite IHs2; simpl; can; Cx...
    Cx... r rot 1; Cx...
    local rewrite IHs2; simpl; can; repeat rewrite← replace_trivial.
    Cxx...
Unshelve. apply replace_diff...
Qed.

Lemma dsubst_congruent_base
  :  forall t t'
  ,  t ≈ t'
  → forall x u
  ,  ∂( t // ∂ x)[open x u] ≈ ∂( t' // ∂ x)[open x u].
Proof with auto with diff.
  intros t t' eq; induction eq.
  - induction H; try solve [intros; simpl; Cxx; auto with diff].
    + intros; simpl; Cxx; rewrite open_open_le; and apply IHCtx.
      auto with arith.
    + induction H; intros; try solve [simpl; auto with diff];
```

```
      simpl; can; try solve [Cx ; [ auto with diff ..]].
      * repeat (align 5; Cx; try apply eq_refl).
      * repeat (align 5; Cx; try apply eq_refl).
      * differential du := d u / d x [u0].
        differential dv := d v / d x [u0].
        differential ds := d s / d x [u0].
        listify; stomp; [r] stomp.
        repeat ([aligns]; pop; and apply eq_refl).
        ([r] [rot] 1); [aligns]. pop...
        ([r] [rot] 1); pop...
        apply eqs_sym.
        [*] local rewrite Taylor; rewrite← replace_trivial.
        [rot] 1; [*] local rewrite Taylor; rewrite← replace_trivial.
        stomp. fold du dv ds. [rot] 3; stomp.
        pop... [rot] 2; pop... [r] [rot] 1. pop...
        pop... Cx; d rot 0... pop... pop; and Cx; d rot 0; Cx...
        [refl].
      * differential dt := d t / d x [u0].
        differential ds := d s / d x [u0].
        differential du := d u / d x [u0].
        repeat (align 10; Cx; [solve [apply eq_refl]|]).
        r rot 6; Cx...
        rot 2; r rot 1; Cx; and do 2 Cx...
        repeat local rewrite Taylor; simpl; repeat rewrite← replace_trivial;
        can. fold dt ds du.
        rot 2; Cx... Cx... Cx.
        do 3 Cx; and d rot 0...
        do 4 Cx; and d rot 0...
      * differential du := d u / d x [u0];
        differential dv := d v / d x [u0];
        differential ds := d s / d x [u0].
        rot 2; r rot 2; Cx...
        repeat local rewrite Taylor; simpl; repeat rewrite← replace_trivial.
        fold du dv ds; can.
        r rot 3; Cx. Cx; d rot 0...
        r rot 3; Cx; and Cx; and d rot 0; and d rot 1...
        r rot 2; Cx; and Cx; and d rot 2; and d rot 1; and d rot 0; and Cx;
        and d rot 1; and d rot 0; and Cx...
        r rot 3; Cx; and (Cx; and d rot 0; and Cx)...
        r rot 2; Cx...
        rot 1; Cx... Cx... Cx; d rot 1; d rot 0...
  - intros...
  - intros...
  - intros... eapply eq_trans; [eapply IHeq1 | eapply IHeq2].
Unshelve. all: try apply eq_refl.
Qed.

Theorem dsubst_congruent_new
  :  forall s
  ,  forall t t'
  ,  t ≈ t'
  → forall x
  ,  ∂(s // ∂x)[t] ≈ ∂(s // ∂x)[t'].
Proof with (auto with diff).
```

```
  intro s; induction s; intros; simpl; try Cxx...
  - destruct (Nat.compare x n)...
  - apply replace_diff...
  - apply replace_diff...
Qed.

(* Differential substitution respects differential equivalence *)
Lemma dsubst_diff
  :  forall t t'
  ,  t ≈ t'
  → forall x u u'
  ,  u ≈ u'
  → ∂( t // ∂ x)[open x u] ≈ ∂( t' // ∂ x)[open x u'].
Proof with auto with diff.
  intros; eapply eq_trans.
  apply dsubst_congruent_base; and exact H.
  apply dsubst_congruent_new; and apply open_diff; and exact H0.
Qed.

Lemma dsubst_commute
  :  forall t x u v
  ,  dsubst (dsubst t x (open x u)) x (open x v)
     ≈ dsubst (dsubst t x (open x v)) x (open x u).
Proof with auto with diff arith.
  intro t; induction t; intros; simpl...
  - repeat trivial_arithmetic... repeat local rewrite dsubst_empty...
  - Cx. repeat rewrite (open_open_le _ 0)...
  - can. repeat local rewrite Taylor. simpl. repeat rewrite← replace_trivial.

    specialize (IHt1 x u v); specialize (IHt2 x u v).
    differential d2u := d t2 / d x [u].
    differential d2v := d t2 / d x [v].
    differential d1u := d t1 / d x [u].
    differential d1v := d t1 / d x [v].
    differential d1uv := d d1u / d x [v].
    differential d1vu := d d1v / d x [u].
    differential d2uv := d d2u / d x [v].
    differential d2vu := d d2v / d x [u].
    fold d1u d1v in IHt1; fold d1uv d1vu in IHt1.
    fold d2u d2v in IHt2; fold d2uv d2vu in IHt2.
    repeat local rewrite IHt1. repeat local rewrite IHt2. Cx...
    listify.
    stomp; [r] stomp.
    [aligns]; pop...
    [r] [rot] 1. [aligns]; pop...
    repeat ([aligns]; pop; [auto with diff|])...
    pop; and repeat local rewrite← Eq_app_E; can...
    repeat ([aligns]; pop; [auto with diff|])...
    [r] [rot] 8. pop... Cx... Cx... repeat local rewrite Eq_add_assoc; Cx...
    [r] [rot] 1. pop... [r] [rot] 1.
    repeat (pop; [auto with diff|])...
    + repeat local rewrite← Eq_app_E; repeat local rewrite Eq_D_inf; can...
    + do 3 Cx... d rot 1; d rot 0...
    + [refl].
    Optimize Proof.
  - can. repeat local rewrite Taylor. simpl. repeat rewrite ← replace_trivial.
    specialize (IHt1 x u v); specialize (IHt2 x u v).
    differential d2u := d t2 / d x [u];
    differential d2v := d t2 / d x [v];
    differential d1u := d t1 / d x [u];
    differential d1v := d t1 / d x [v];
    differential d2uv := d d2u / d x [v];
    differential d2vu := d d2v / d x [u];
    differential d1uv := d d1u / d x [v];
    differential d1vu := d d1v / d x [u].
    fold d1u d1v in IHt1; fold d1uv d1vu in IHt1.
    fold d2u d2v in IHt2; fold d2uv d2vu in IHt2.
    repeat local rewrite IHt1. repeat local rewrite IHt2.
    listify. Optimize Proof.
    [*] repeat local rewrite ← Eq_add_assoc. repeat splits.
    [r] ([*] repeat local rewrite ← Eq_add_assoc); repeat splits.
    pop... [rot] 2; pop... ([r] [rot] 2); pop...
    Cx... repeat local rewrite Eq_add_assoc. Cx...
    repeat local rewrite Eq_add_E. repeat local rewrite ← Eq_add_assoc; Cx...
    repeat ([aligns]; pop; [auto with diff|]).
    stomp; [r] stomp. pop... [rot] 1; [r] [rot] 1. pop...
    Cx; and d rot 1; and d rot 0; Cx...
    [rot] 3; [r] [rot] 5. pop...
    [rot] 4; [r] [rot] 4.
    pop... pop... [r] [rot] 2. pop...
    stomp; [r] stomp.
    pop; and Cx; and d rot 0...
    [rot] 2; pop... [rot] 1; pop... Cx; and d rot 1; and d rot 0...
    pop... [refl].
Unshelve.
  all: try apply eq_refl.
  all: try apply dsubst_diff...
  all:
    apply replace_diff; auto with diff;
    Cx; [apply replace_diff|]; auto with diff;
    Cx; apply dsubst_diff...
Qed.

(* Regularity of differential substitution *)
Theorem Regularity
  :  forall s x u v
  ,  ∂( s // ∂ x)[ open x (u + v) ] ≈
      ∂(( s // ∂ x)[ open x u ]
      + ∂(( s // ∂ x)[ open x v ]
      + ε ∂(( ∂(( s // ∂ x)[ open x v ]) // ∂ x)[ open x u ]))).
Proof with (auto with diff).
  intro s; induction s; intros; simpl.
  - compare x n; can; Cx...
    local rewrite dsubst_empty; can...
  - repeat rewrite (open_open_le _ 0); try auto with arith.
    fold_open;
    local rewrite IHs.
    local rewrite ← Eq_lam_E.
    repeat local rewrite ← Eq_lam_add...
  - fold_open; repeat local rewrite IHs1; repeat local rewrite IHs2;
    repeat local rewrite Taylor; simpl; repeat rewrite ← replace_trivial;
    repeat local rewrite Taylor; simpl; repeat rewrite ← replace_trivial.
    fold_open; local rewrite IHs2.
    differential d2u := d s2 / d x [u].
    differential d2v := d s2 / d x [v].
    differential d1u := d s1 / d x [u].
    differential d1v := d s1 / d x [v].
    differential d2vu := d d2v / d x [u].
    differential d1vu := d d1v / d x [u].
    listify; stomp;
    [r] stomp.
    pop... [r] [aligns]. pop...
    [r] [aligns]. pop...
    [aligns]. pop... [r] [rot] 7. pop; and do 4 Cx...
    [rot] 3. [r] [rot] 25. pop; and do 3 Cx...
    [r] [rot] 18. pop...
    [*] repeat local rewrite Eq_derivative. stomp.
    repeat (([r] [aligns]); pop; [auto with diff|]).
    [*] repeat local rewrite Eq_derivative. stomp.
    repeat (([r] [aligns]); pop; [auto with diff|]).
    pop; and do 3 Cx...
    pop; and do 3 Cx...
    [rot] 1; pop...
    ([r] [rot] 1); pop.
    repeat local rewrite ← Eq_app_E;
    repeat local rewrite Eq_D_inf; Cx...
    [rot] 1. [r] [rot] 8. pop.
    repeat local rewrite ← Eq_app_E;
    repeat local rewrite Eq_D_inf; Cx...
    [*] repeat local rewrite Eq_derivative. stomp.
    repeat ([aligns]; pop; [auto with diff|]).
    [refl].
  - can...
  - fold_open; repeat local rewrite IHs1; repeat local rewrite IHs2;
    repeat local rewrite Taylor; simpl; repeat rewrite ← replace_trivial.
    differential d1u := d s1 / d x [u].
    differential d1v := d s1 / d x [v].
    differential d2u := d s2 / d x [u].
    differential d2v := d s2 / d x [v].
    can; repeat (align 10; Cx; [auto with diff|])...
  - fold_open; repeat local rewrite IHs; can; Cxx...
  - fold_open. repeat local rewrite IHs1; repeat local rewrite IHs2;
    repeat local rewrite Taylor.
    differential d1u := d s1 / d x [u].
    differential d1v := d s1 / d x [v].
    differential d2u := d s2 / d x [u].
    differential d2v := d s2 / d x [v].
    differential d1uv := d d1u / d x [v].
    differential d1vu := d d1v / d x [u].
    differential d2uv := d d2u / d x [v].
    differential d2vu := d d2v / d x [u].
    repeat rewrite ← replace_trivial.
```

```
simpl; repeat local rewrite Taylor.
fold_open; local rewrite IHs2.
fold d1u d1v d2u d2v. fold d1uv d1vu d2uv d2vu.
repeat rewrite← replace_trivial.
listify. repeat splits.
[r] ([*] repeat local rewrite← Eq_add_assoc); repeat splits.
[*] can; repeat local rewrite← Eq_add_assoc.
repeat splits.
repeat ([aligns];pop;[auto with diff|]).
repeat ([r][aligns]; pop; [auto with diff|]).
[rot] 6. stomp.
[r] [rot] 2. [r] ([*] do 10 simplify; repeat local rewrite← Eq_add_assoc).
[r] repeat splits.
repeat ([aligns];pop;[auto with diff|]).
[r] [rot] 3. pop...
[rot] 4; [r] [aligns]. pop...
[r] [aligns]. pop...
[rot] 8. [*] do 10 simplify; repeat local rewrite← Eq_add_assoc.
repeat splits.
repeat ([aligns]; pop;[auto with diff|]).
pop. repeat local rewrite Eq_D_inf...
[rot] 8; pop...
Optimize Proof.
[rot] 1; pop... can...
[rot] 1; stomp. [rot] 1; pop...
[r] [rot] 2; [*] repeat local rewrite Eq_add_E. [r] repeat splits.
[r] [rot] 6. [r] repeat splits.
[rot] 4. [r] [rot] 4. pop; and can...
[r] [rot] 10. [rot] 22. pop...
[rot] 3. [r] stomp. pop... pop... pop... Cx; d rot 0...
[r] [rot] 2; stomp. pop... pop...
[rot] 7. [r] [rot] 1; stomp. pop... pop...
[r] [rot] 4. [rot] 1; pop; and can...
[r] [rot] 1; stomp. [rot] 1; stomp.
[rot] 2; pop...
[r] [rot] 1. pop; and can; and Cx; and d rot 1; and d rot 0...
[rot] 10. [r] [rot] 3. pop; and can...
[rot] 6; pop...
[rot] 15; ([r] [rot] 10); pop; and can...
pop; and can...
[rot] 12; pop; and can...
[r] [rot] 2; stomp.
[rot] 4; pop...
pop...
pop; and Cx; and d rot 0...
[r] [rot] 3; stomp.
[*] can. [r] [rot] 1. pop...
[*] can. [r] [rot] 2. pop...
[r] [rot] 2. pop; and can; and Cx; and d rot 0...
[rot] 2; stomp. [rot] 2; pop; and Cx; d rot 1; d rot 0; Cx; and r d rot 0...
[r] [rot] 2; stomp.
[rot] 1; pop; and Cx; and d rot 0...
([r] [rot] 1); pop; and Cx; and d rot 0; and Cx; and r d rot 0...
[rot] 1; stomp. pop... [rot] 2; pop; and Cx; and r d rot 0...
```

```
    [r] [rot] 1. pop; and Cx; and r d rot 0...
    [r] [rot] 3. pop...
    [rot] 1; pop...
    [*] can. pop; and Cx; and d rot 0...
    [*] do 10 simplify; repeat local rewrite ← Eq_add_assoc. repeat splits.
    [r] ([*] do 10 simplify; repeat local rewrite ← Eq_add_assoc); repeat
    splits.
    pop; and can...
    repeat (pop; [solve [auto with diff]|]).
    pop; and can...
    Optimize Proof.
    stomp. [r] stomp. repeat (pop; [solve [auto with diff]|]).
    stomp. repeat (pop; [solve [auto with diff]|]).
    [refl].
Unshelve.
  (* There is no reason this should have to be done manually*)
  all: try apply eq_refl.
  all: try apply dsubst_diff...
Qed.

Lemma replace_dsubst
  :  forall t x e y v
  ,  x ≠ y
  → replace (dsubst t x e) y (open x v)
     ≈ dsubst (replace t y (open x v)) x (replace e y (open x v)).
Proof with auto with arith diff.
  intro t; induction t; intros x e y v le; simpl in *|-*;
  try rewrite IHt1; try rewrite IHt2; try rewrite IHt...
  - repeat trivial_arithmetic...
    all: local rewrite dsubst_empty...
  - Cx; repeat rewrite (open_open_le _ 0)...
    local rewrite IHt;
    rewrite ← replace_open_le...
    repeat rewrite (open_open_le _ 0)...
  - Cxx...
    + rewrite replace_replace; simpl...
      trivial_arithmetic...
  - Cxx...
    + rewrite replace_replace; simpl...
      trivial_arithmetic...
Unshelve.
  auto with arith.
Qed.
```