# Automatic Differentiation in PCF

DAMIANO MAZZA, CNRS, France
MICHELE PAGANI, Université de Paris, France

We study the correctness of automatic differentiation (AD) in the context of a higher-order, Turing-complete language (PCF with real numbers), both in forward and reverse mode. Our main result is that, under mild hypotheses on the primitive functions included in the language, AD is almost everywhere correct, that is, it computes the derivative or gradient of the program under consideration *except* for a set of Lebesgue measure zero. Stated otherwise, there are inputs on which AD is incorrect, but the probability of randomly choosing one such input is zero. Our result is in fact more precise, in that the set of failure points admits a more explicit description: for example, in case the primitive functions are just constants, addition and multiplication, the set of points where AD fails is contained in a countable union of zero sets of polynomials.

CCS Concepts: • **Theory of computation → Program semantics**; *Theory and algorithms for application domains.*

Additional Key Words and Phrases: Differentiable Programming, Lambda-Calculus, Linear Logic

**ACM Reference Format:**
Damiano Mazza and Michele Pagani. 2021. Automatic Differentiation in PCF. *Proc. ACM Program. Lang.* 5, POPL, Article 28 (January 2021), 27 pages. https://doi.org/10.1145/3434309



## 1 INTRODUCTION

Automatic differentiation (AD) provides efficient methods for computing the derivative (or, more generally, the gradient or Jacobian) of a function specified by a computer program. Since computing derivatives is a key ingredient in the resolution of all sorts of optimization problems, it is not surprising that AD grew into a large field with applications to a host of scientific domains, most notably machine learning [Baydin et al. 2018].

Traditionally, AD focused on first-order imperative programs and, although its techniques allow the presence of flow control instructions and loops [Beck and Fischer 1994; Joss 1976; Speelpenning 1980], its scope was often limited to straight-line programs (also know as *computational graphs* [Goodfellow et al. 2016]), which were enough for most practical purposes, such as expressing neural networks. After the advances in deep learning of the last years, this is no longer the case: neural network architectures are now "dynamic", in the sense that the input may influence the shape of the net, and expressing such architectures requires resorting a priori to the full power of a modern programming language, yielding what some have called *differentiable programming* [LeCun 2018]. This evolution of deep learning spurred the rapid development of differentiable programming frameworks [Abadi et al. 2016; Paszke et al. 2017] and, at the same time, received much attention in programming languages (PL) research for establishing its theoretical foundations [Abadi and Plotkin 2020; Brunel et al. 2020; Elliott 2018; Huot et al. 2020; Shaikhha et al. 2019; Wang et al. 2019].

Authors' addresses: Damiano Mazza, CNRS, UMR 7030, LIPN, Université Sorbonne Paris Nord, France, Damiano.Mazza@lipn.univ-paris13.fr; Michele Pagani, IRIF UMR CNRS 8243, Université de Paris, France, pagani@irif.fr.


2475-1421/2021/1-ART28
https://doi.org/10.1145/3434309

From the viewpoint of PL theory, AD methods boil down to *program transformations*: writing $\mathbb{R}$ and R for the set and type of real numbers, respectively, we receive as input a program $M : \mathsf{R}^n \to \mathsf{R}$ computing a (possibly partial) function $[\![M]\!] : \mathbb{R}^n \rightharpoonup \mathbb{R}$ whose gradient $\nabla[\![M]\!]$ exists in a set $\mathrm{d}(M) \subseteq \mathbb{R}^n$ (called the *domain of differentiability*), and we must output another program $grad(M)$ computing $\nabla[\![M]\!]$. The crucial features that one typically asks of such transformations are:

(i) **efficiency**: asymptotically, evaluating $grad(M)$ is not more costly than evaluating $M$;
(ii) **soundness**: $grad(M)(\mathbf{r})$ evaluates to $\nabla[\![M]\!](\mathbf{r})$ for all $\mathbf{r} \in \mathrm{d}(M)$.

Notice that there is a tension between efficiency and soundness: implementing the definition of derivative as a limit gives a trivially sound (to an arbitrary degree of precision) but unacceptably inefficient method. Conversely, more efficient transformations tend to be more complex (*e.g.*, reverse mode is more complex than forward mode, see below) and their soundness more difficult to prove. Also observe that we are only interested in the correctness of the result when $\nabla[\![M]\!]$ is defined; in case $\mathbf{r} \notin \mathrm{d}(M)$, the evaluation of $grad(M)(\mathbf{r})$ may give anything, including (but not necessarily!) divergence.

Another highly desirable feature of $grad$ is *modularity*: if $P$ is a subprogram of $M$ then $grad(P)$ is a subprogram of $grad(M)$ or, if this is not literally the case, the computation of the former may be reused in computing the latter. Indeed, it has been known from the early days of AD that modularity offers a path to attaining both efficiency and soundness: the program $M$ is decomposed into elementary blocks whose gradients are immediately computable, and $grad(M)$ is obtained by assembling these transformed blocks following the structure of $M$. In this way, the execution of $grad(M)$ mimics that of $M$, yielding efficiency, and soundness relies on the so-called *chain rule* of calculus, which assures us that the derivative of a compound function may be expressed in terms of the derivative of its components. Furthermore, one sees that there are two "dual" ways of assembling the transformed blocks to form $grad(M)$: a covariant way, yielding *forward mode* AD, and a contravariant way, yielding *reverse mode* AD (this will be explained in Sect. 2.2).

The theory of AD transformations has by now been developed to considerable depth by several authors: Pearlmutter and Siskind first pointed out that reverse mode AD, commonly known as *backpropagation*, may be naturally expressed in terms of higher-order programs, and used this idea to develop a differentiable variant of Scheme [Pearlmutter and Siskind 2008]; more recently, Elliott emphasized functoriality as a systematic way of understanding the modular nature of AD transformations [Elliott 2018]; the work [Wang et al. 2019] introduced Lantern, a fully general differentiable programming framework in which the notion of delimited continuation is used to correctly handle memory updates during backpropagation; finally, Brunel, Mazza and Pagani showed that the continuation-passing machinery at work in [Wang et al. 2019] (and, implicitly, in [Pearlmutter and Siskind 2008]) may be understood in terms of linear negation (in the sense of Girard's linear logic), giving a purely functional transformation for reverse mode AD and a conceptually clean analysis of its efficiency in terms of a "linear factoring" evaluation rule [Brunel et al. 2020]. On the semantics side, Abadi and Plotkin studied denotational semantics for a first order differentiable language [Abadi and Plotkin 2020] and Huot, Staton and Vákár gave a uniform approach to proving soundness of AD (forward and reverse) for simply-typed programs based on a diffeology semantics [Huot et al. 2020].

Nevertheless, an analysis of soundness of AD transformations for a fully general programming language is currently missing: the above-mentioned work is either fully general but lacks soundness proofs [Pearlmutter and Siskind 2008; Wang et al. 2019] or proves soundness in a restricted setting (first order [Abadi and Plotkin 2020] or simply-typed $\lambda$-calculi [Barthe et al. 2020; Brunel et al. 2020; Huot et al. 2020]). Filling this gap is precisely the contribution of the present paper: we study the soundness of AD transformations in the setting of the (idealized) functional programming language

$\mathsf{PCF}_{\mathsf{R}}$, a variant with real numbers of Plotkin's famous Turing-complete language [Plotkin 1977]. For forward mode, we use the standard transformation described for instance in [Wang et al. 2019]. For reverse mode, since $\mathsf{PCF}_{\mathsf{R}}$ is purely functional, we consider an extension of the transformation introduced in [Brunel et al. 2020], albeit simplified in that we leave linearity aside, since that is only needed for efficiency and here we are merely interested in soundness. The extension, which is a contribution of this paper in its own right, concerns conditional statements and fixpoints, which are not dealt with in *loc. cit.*

The first relevant observation is that, in presence of conditionals, soundness in the sense of statement (ii) above actually fails. Consider the program

$$\mathsf{SillyId} \quad := \quad \lambda x^{\mathsf{R}}.\mathsf{if}\, x = 0\, \mathsf{then}\, 0\, \mathsf{else}\, x.$$

We clearly have that $\mathsf{SillyId} : \mathsf{R} \to \mathsf{R}$ and that $[\![\mathsf{SillyId}]\!]$ is the identity function. We therefore expect $grad(\mathsf{SillyId})$ to compute the constant function 1. And yet, by modularity/functoriality, AD transformations will give something like

$$grad(\mathsf{SillyId}) \quad = \quad \lambda x^{\mathsf{R}}.\mathsf{if}\, x = 0\, \mathsf{then}\, 0\, \mathsf{else}\, 1,$$

which obviously gives the wrong result for $x = 0$. This phenomenon, which is well known in the AD community [Beck and Fischer 1994], is due to functoriality turning a syntactic discontinuity into a semantic one. Notice that, although the above example is indeed quite silly, similar situations may happen in a non-trivial neural network with rectified linear unit activation: if

$$\mathsf{ReLU} \quad := \quad \lambda x^{\mathsf{R}}.\mathsf{if}\, x \leq 0\, \mathsf{then}\, 0\, \mathsf{else}\, x,$$

then $\mathsf{ReLU}(x) - \mathsf{ReLU}(-x)$ behaves exactly as $\mathsf{SillyId}(x)$. Also, using recursive definitions, it is easy to obtain programs on which AD fails on infinitely many inputs, even uncountably many in case of programs of type $\mathsf{R}^n \to \mathsf{R}$ with $n > 1$ (simply consider $\lambda x^{\mathsf{R}}.\lambda y^{\mathsf{R}}.\mathsf{if}\, x \cdot y = 0\, \mathsf{then}\, 0\, \mathsf{else}\, x \cdot y$).

So, to each given $\mathsf{PCF}_{\mathsf{R}}$ program $M : \mathsf{R}^n \to \mathsf{R}$, we may assign a set $\mathrm{Fail}(M) \subseteq \mathrm{d}(M)$ of points on which AD is unsound. Notice once again that we disregard what lies outside of $\mathrm{d}(M)$, where $grad(M)$ is free to behave arbitrarily. For instance, $grad(\mathsf{ReLU})$ is something like $\lambda x^{\mathsf{R}}.\mathsf{if}\, x \leq 0\, \mathsf{then}\, 0\, \mathsf{else}\, 1$, which evaluates to 0 when $x = 0$, even though $[\![\mathsf{ReLU}]\!]$ is not differentiable in 0. Morally, we cannot say that AD is "wrong" when there is no "right" value to compare it to.

After toying with more examples, one is led to conjecture that $\mathrm{Fail}(M)$ is always of measure zero (in the sense of the standard Lebesgue measure on $\mathbb{R}^n$), so one may hope to establish an "almost-everywhere" relaxation of (ii):

(ii') **ae-soundness:** $grad(M)(\mathbf{r})$ evaluates to $\nabla[\![M]\!](\mathbf{r})$ for all $\mathbf{r} \in \mathrm{d}(M)$ *except* on a set of measure zero.

This is exactly the main result of our paper. Let us stress that, considering that "full" soundness is impossible, such a result is quite meaningful in practice because of the link between the Lebesgue measure and the standard understanding of randomness on $\mathbb{R}^n$. In typical deep learning applications, weights are initialized "at random" and later updated via gradient descent in order to minimize a loss function. Technically, this means that the weights evolve following some standard probability distribution, which always arises from integrating a probability density function with respect to the Lebesgue measure. So, an informal way of stating (ii') is that *AD almost never fails*, in the sense that, according to the standard definition of probability, the likelihood of computing wrong derivatives during gradient descent is zero.

The main result itself is articulated in Theorem 33 and Theorem 42. The first result takes care of soundness proper: we define, for any given $\mathsf{PCF}_{\mathsf{R}}$ program $M : \mathsf{R}^n \to \mathsf{R}$ whose domain (*i.e.*, the inputs on which it converges) is $\Downarrow\!M$, a set $\mathrm{S}(M) \subseteq \Downarrow\!M$ of *stable points*, and Theorem 33 affirms

that statement (ii) holds on $\mathrm{d}(M) \cap \mathrm{S}(M)$. Then, we establish in Theorem 42 that the set $\Downarrow M \setminus \mathrm{S}(M)$ of *unstable points* of $M$ is of measure zero. Since $\mathrm{d}(M) \subseteq \Downarrow M$, this proves statement (ii').

The intuition behind stable points is the following. Take $M : \mathsf{R}^n \to \mathsf{R}$, $\mathbf{r} \in \mathbb{R}^n$ and trace the execution of $M(\mathbf{r})$. This means, in particular, unfolding the recursive definitions in $M$ and choosing, for each instance of a conditional statement of $M$, the "then" or "else" branch. One obtains thus a program with no conditionals and no fixpoints, *i.e.*, a *simply-typed $\lambda$-term*, which is said to *trace* $M(\mathbf{r})$. Such a program depends of course on $\mathbf{r}$. If, however, there exists a simply-typed $\lambda$-term $t$ and an open neighborhood $U \subseteq \mathbb{R}^n$ of $\mathbf{r}$ (in the standard topology) such that $t(\mathbf{r}')$ traces $M(\mathbf{r}')$ for all $\mathbf{r}' \in U$, then $\mathbf{r}$ is *stable* (Definition 26). For instance, any $r \neq 0$ is stable for the ReLU program given above: there always exists an open interval $I$ around $r$ such that either $\mathsf{Zero} := \lambda x^{\mathsf{R}}.0$ or $\mathsf{Id} := \lambda x^{\mathsf{R}}.x$ traces ReLU on $I$, depending on whether $r < 0$ or $r > 0$, respectively. On the other hand, 0 is an unstable point of ReLU: any open interval around 0 must contain negative points, on which ReLU is traced by Zero, and positive points, on which ReLU is traced by Id, and of course $\mathsf{Zero} \neq \mathsf{Id}$.

The proof of Theorem 33 uses stability to reduce the soundness of AD on $\mathsf{PCF}_{\mathsf{R}}$ to the soundness of AD on simply-typed $\lambda$-terms, which may be established in various ways [Brunel et al. 2020; Huot et al. 2020]. The idea is simple: if $\mathbf{r} \in \mathrm{S}(M)$, then $M$ "behaves like" a simply-typed $\lambda$-term $t$ in an open neighborhood of $\mathbf{r}$, and we know that AD works for $t$ everywhere, so it "must" work for $M$ on $\mathbf{r}$. Although intuitively clear, the actual argument is surprisingly subtle. First, the definition of trace (Definition 25) is not obvious, due to non-uniformity issues introduced by higher types: two copies of the same higher-order subterm may be traced in different ways, as explained in the example given at the beginning of Sect. 3.1. Second, knowledge of the correctness of $grad(t)(\mathbf{r})$ does not immediately imply the correctness of $grad(M)(\mathbf{r})$ and some non-trivial work is needed to show that they behave similarly.

The measure-zero bound on unstable points (Theorem 42), albeit obtained via a standard logical predicate argument, also requires non-trivial elements, most notably the notion of *complete quasi-continuity*, which is needed to account for the behavior of unstable points under composition, and the related notion of *quasivariety*. Although the exact meaning of these notions depends on the choice of primitives of $\mathsf{PCF}_{\mathsf{R}}$ (*i.e.*, the basic real functions included in the language), under mild assumptions a quasivariety is always of measure zero, and Theorem 42 states precisely that $\mathrm{Fail}(M)$ is a quasivariety. For example, when the primitives are just constants, addition and multiplication, quasivarieties are arbitrary subsets of countable unions of zero sets of polynomials. Furthermore, our Lemma 41 implies properties of $\mathsf{PCF}_{\mathsf{R}}$-definable functions which, as far as we can tell, were previously unknown. For example, if $f : \mathbb{R}^n \rightharpoonup \mathbb{R}$ is definable in $\mathsf{PCF}_{\mathsf{R}}$, and if $U \subseteq \mathbb{R}$ is open, then in general the border of $f^{-1}(U)$ (*i.e.*, $f^{-1}(U)$ minus its interior) is not empty because conditionals introduce discontinuities, but *it is always a quasivariety*. Similarly, *the set of zeros of $f$ is always the disjoint union of an open set and a quasivariety*.

*Related work.* We already mentioned some of the relevant previous work at the interface between AD and PL theory and stressed that our contribution here is to study the soundness of AD in a fully general setting (higher-order, Turing-complete language), something which, as far as we know, was lacking.

We also mentioned that the unsoundness of AD in presence of conditional statements is well known [Beck and Fischer 1994]. Surprisingly, though, recent PL work on the subject acknowledges this problem only sporadically, *e.g.* [Abadi and Plotkin 2020]. The solution proposed therein is restricting to *continuous* Boolean conditions, meaning that the inverse images of the two Boolean values along such conditions are open. For example, testing for zero or for non-positivity are not continuous, because the inverse image of true is the singleton $\{0\}$ or $]-\infty, 0]$, respectively, which

are not open. With this limitation, the authors prove statement (ii) for a Turing-complete first-order language. The benefit of Abadi and Plotkin's approach is allowing a denotational semantics modeling the *grad* operator, but it has the drawback of representing standard total functions with programs diverging on singularities (for example, ReLU yields a program diverging in 0), which is somewhat unexpected. We discuss this further at the end of Sect. 2.2 and in Sect. 5.

Our approach, in the wake of a large part of the AD literature, is to stick to the standard semantics and provide a bound on the unsoundness of AD, as precise as possible. This approach dates back to the Seventies, as far as we know to Joss's Ph.D. thesis [Joss 1976], who proved statement (ii') for forward mode AD in the context of an imperative language with variable assignments, basic arithmetic functions (sum, multiplication, division), conditional statements and gotos. So our result may be seen as an extension Joss's theorem in several directions: to a higher-order language; to a wider set of primitive functions (as long as they form an *admissible clone*, Definition 11); and to reverse mode AD. Additionally, Theorem 42 is more precise than (ii'), because it characterizes the set of failure points $\mathrm{Fail}(M)$ as a quasivariety, which is a rather special example of negligible set. Some discussion about this point is given in Sect. 5, in particular the proof of Theorem 42 hints to methods for automatically computing at least some overapproximation of $\mathrm{Fail}(M)$ statically from the structure of $M$.

From a broader perspective, variants of PCF with real numbers similar to the one studied here have been considered in the literature, *e.g.* [Escardó 1996] and [Di Gianantonio and Edalat 2013]. The latter actually also considers AD, but it is not about correctness and is quite different in spirit, being more focused on denotational semantics. There is also a recent line of work whose goal is to understand the non-differentiable points of program-defined functions, such as [Mak et al. 2020; Zhou et al. 2019], including in the context of AD [Lee et al. 2020], where it is a natural and important question [Griewank and Walther 2008]. Non-differentiability and unsoundness of AD have an important point in common: they are both introduced by conditionals. This explains why some notions used in our paper also crop up in the study of non-differentiability, such as zero sets of analytic functions [Zhou et al. 2019]. However, let us underline that the two issues are orthogonal: from our perspective, PCF-definable functions might as well have been differentiable *everywhere* (as in the SillyId example), what matters is that AD still makes mistakes and we wish to understand them. Whether our techniques also yield tools for describing the set of non-differentiable points of PCF-definable functions and, in that case, exactly how they relate to the above-mentioned work is an interesting question which we leave for the future.

Finally, let us mention that our notion of stable point (Definition 26), which is new as far as we know, is based on a concept of trace (Definition 25) belonging to the same circle of ideas as Ehrhard and Regnier's Taylor expansion [Ehrhard and Regnier 2006, 2008] and the modern understanding of intersection types in the spirit of Mazza, Pellissier and Vial's work [Mazza 2017; Mazza et al. 2018]. This extremely general perspective allows the definition of finitary approximations of programs at the level of the operational semantics, rather than denotational, as was the case traditionally. Our proof techniques are therefore not *ad hoc* for our current purposes and may be expected to have applications beyond the present paper.

*Contents of the paper.* Sect. 2 introduces the language $\mathrm{PCF}_{\mathrm{R}}$ with its rewriting relation (Fig. 1) and the AD transformations (Fig. 2 and Equations (9), (10)). Our results are quite general and do not depend on a specific operational semantics but apply to a wide family of them (Proposition 10), including the standard ones (call-by-value, call-by-name, etc.). We also introduce admissibility of primitives (Definition 11) and the associated topological notions, among which quasivarieties (Definition 13). Sect. 3 and Sect. 4 are the heart of the paper, containing the proofs of Theorem 33 and Theorem 42, respectively, as described above. Sect. 5 concludes the paper, discussing the results.

*Notations.* We write $f : A \rightharpoonup B$ to say that $f$ is a partial function from a set $A$ to a set $B$. In that case, $\Downarrow f$ will denote the subset of $A$ on which $f$ is defined. Given two partial functions $f, g : A \rightharpoonup B$ and $a \in A$, the equality $f(a) = g(a)$ means that either both $f(a)$ and $g(a)$ are defined and equal, or that both $f(a)$ and $g(a)$ are undefined. The notation $f(a) \neq g(a)$ of course is understood as the logical negation of that. Given a subset $A' \subseteq A$, we write $f|_{A'}$ for the restriction of $f$ to $A'$. If $A$ is endowed with a complete measure $\lambda$ (typically $A$ is $\mathbb{R}$ and $\lambda$ is the Lebesgue measure), then we say that $f$ and $g$ are *almost everywhere equal*, and we write $f \sim g$, if $\lambda(\{a \in A \mid f(a) \neq f(b)\}) = 0$. This is equivalent to the existence of two subsets $A', Z$ of $A$ such that $\Downarrow f \cup \Downarrow g \subseteq A' \cup Z$, $\lambda(Z) = 0$ and $f|_{A'} = g|_{A'}$, a fact which is implicitly used in the proof of Proposition 10.[1]

We write $B_\varepsilon(\mathbf{r})$ for the open ball of $\mathbb{R}^n$ of radius $\varepsilon$ centered at $\mathbf{r} \in \mathbb{R}^n$. Given $f : \mathbb{R}^n \rightharpoonup \mathbb{R}^m$, we denote by $\mathrm{d}(f)$ the *domain of differentiability* of $f$, defined to be the set of all $\mathbf{r} \in \mathbb{R}^n$ where $f$ is differentiable in the sense that the total derivative of $f$ at $\mathbf{r}$ exists, *i.e.*, $f$ admits a best linear approximation at $\mathbf{r}$. We denote by $\mathcal{J}f$ the *Jacobian* of $f$. We recall that, if $\partial_i f_j$ denotes the partial derivative of the $j$-th component of $f$ with respect to its $i$-th parameter, then within $\mathrm{d}(f)$ the Jacobian is equal to the $m \times n$ matrix $(\partial_i f_j)$. If $m = 1$, the Jacobian is called *gradient* and denoted by $\nabla f$. As a special case of the above, within $\mathrm{d}(f)$ we have $\nabla f = (\partial_1 f, \dots, \partial_n f)$. Although it may happen in general that all partial derivatives exist without the Jacobian/gradient being defined, it will never be the case in what follows because *we will always work within* $\mathrm{d}(f)$.

## 2 PCF WITH REAL NUMBERS

### 2.1 Terms and Semantics

The programming language we use, called $\mathsf{PCF}_\mathsf{R}$, is introduced in Fig. 1. There is only one base type, R, for real numbers. We use $n$-ary products for convenience. If one prefers, these may be seen as syntactic sugar defined from nullary and binary products. Throughout the paper, we stipulate that unary products are just identities: $\langle M \rangle$ and $\pi^1_1 M$ both stand for $M$. We often omit the index $n$ in a projection $\pi^n_i$, when inessential. The empty product type is denoted by 1. Given a type $A$, we write $A^n$ for the $n$-fold product $A \times \cdots \times A$.

The metavariables $\phi, \chi, \psi$ range over a set of function symbols, each coming with a type of the form $\mathsf{R}^k \to \mathsf{R}$, where $k$ is the *arity* of the symbol. We suppose that the set of function symbols contains at least all real numbers $r \in \mathbb{R}$ as nullary symbols, called *numerals*, as well as binary addition and multiplication, for which we use infix notation, *i.e.*, $M + N$ and $M \cdot N$ stand for $+(M, N)$ and $\cdot(M, N)$, respectively. We also write $n$-ary sums as syntactic sugar. Each function symbol $\phi : \mathsf{R}^k \to \mathsf{R}$ comes with a function $[\![\phi]\!] : \mathbb{R}^k \rightharpoonup \mathbb{R}$ and we assume that $[\![r]\!]$, $[\![+]\!]$ and $[\![\cdot]\!]$ are the corresponding numbers and operations on real numbers. The functions $[\![\phi]\!]$ for $\phi$ ranging over function symbols will be referred to as the *primitive functions* of $\mathsf{PCF}_\mathsf{R}$.

The notation $\mathsf{if}(P, M, N)$ is just a compact form of $\mathsf{if}\ P \leq 0\ \mathsf{then}\ M\ \mathsf{else}\ N$. We call $P$ the *guard* of the conditional.

The primitive functions one considers are usually very regular, typically analytic (*e.g.* exponential, logarithm, trigonometric functions, sigmoid maps…). Sect. 2.3 details the precise conditions that primitives must enjoy in order for our results to hold. The expressive power of $\mathsf{PCF}_\mathsf{R}$ considerably enlarges the set of definable functions, in particular introducing singularities. The following

[1] Proof of the equivalence: let $D := \{a \in A \mid f(a) \neq f(b)\}$. If $\lambda(D) = 0$, then we may take $A' := (\Downarrow f \cup \Downarrow g) \setminus D$ and $Z := D$. Conversely, given $A'$ and $Z$ with the required properties, notice that $D \subseteq \Downarrow f \cup \Downarrow g$, hence $D \subseteq A' \cup Z$. But observe that $D \cap A' = \emptyset$, so $D \subseteq Z$, which gives us $\lambda(D) = 0$.

$$A, B ::= \mathsf{R} \mid A \to B \mid A_1 \times \cdots \times A_n$$

(a) Types.

$$\frac{}{\Gamma, x^A \vdash x : A} \qquad \frac{\phi : \mathsf{R}^k \to \mathsf{R}, \quad \Gamma \vdash M_1 : \mathsf{R}, \quad \ldots, \quad \Gamma \vdash M_k : \mathsf{R}}{\Gamma \vdash \phi(M_1, \ldots, M_k) : \mathsf{R}}$$

$$\frac{\Gamma, x^A \vdash M : B}{\Gamma \vdash \lambda x^A.M : A \to B} \qquad \frac{\Gamma \vdash M : A \to B, \quad \Gamma \vdash N : A}{\Gamma \vdash MN : B}$$

$$\frac{\Gamma \vdash M_1 : A_1, \quad \ldots, \quad \Gamma \vdash M_k : A_k}{\Gamma \vdash \langle M_1, \ldots, M_k \rangle : A_1 \times \cdots \times A_k} \qquad \frac{\Gamma \vdash M : A_1 \times \cdots \times A_k}{\Gamma \vdash \pi_i^k M : A_i}$$

$$\frac{\Gamma \vdash P : \mathsf{R} \quad \Gamma \vdash M : A \quad \Gamma \vdash N : A}{\Gamma \vdash \mathsf{if}(P, M, N) : A} \qquad \frac{\Gamma, f : A \to B \vdash M : A \to B}{\Gamma \vdash \mathsf{fix} f^{A \to B}.M : A \to B}$$

(b) Terms and typing rules.

$$(\lambda x.M)N \to M\{N/x\} \qquad \pi_i^k \langle M_1, \ldots, M_k \rangle \to M_i \qquad \phi(r_1, \ldots, r_k) \to [\![\phi]\!](r_1, \ldots, r_k)$$

$$\mathsf{if}(r, M, N) \to \begin{cases} M & \text{if } r \leq 0 \\ N & \text{if } r > 0 \end{cases} \qquad \mathsf{fix} f.M \to M\{\lambda x.(\mathsf{fix} f.M)x/f\}$$

(c) Rewriting steps.

Fig. 1. The language $\mathsf{PCF}_\mathsf{R}$ over the ground type R of real numbers.

examples will be useful in the sequel:

$$\begin{array}{ll} \mathsf{ReLU} := \lambda x^\mathsf{R}.\mathsf{if}(x, 0, x) & \mathsf{Int} := \lambda x^\mathsf{R}.\lambda y^\mathsf{R}.\mathsf{if}(y - x, \mathsf{if}(y - x + 1, 1, 0), 1) \\ \mathsf{SillyId} := \lambda x^\mathsf{R}.\mathsf{if}(x, \mathsf{if}(-x, 0, x), x) & \mathsf{Floor} := \lambda x^\mathsf{R}.\left(\mathsf{fix} f.\lambda n^\mathsf{R}.\mathsf{if}(\mathsf{Int}\, x\, n, n, f(\mathsf{if}(x, n-1, n+1)))\right) 0 \end{array} \tag{1}$$

ReLU and SillyId are the $\mathsf{PCF}_\mathsf{R}$ terms corresponding to the namesake examples discussed in the Introduction. ReLU is a typical example of a continuous non-differentiable function, having a corner in 0. We know that AD fails on SillyId; Sect. 2.2 will elaborate on this point. The program Int takes two inputs $x$ and $y$ and gives 0 if $x \in [y, y+1[$, or 1 otherwise. It is auxiliary to the definition of the Floor function, mapping a real number to the greatest integer less than or equal to it. Floor is an example of how recursive definitions may yield maps with an infinite number of non-differentiable points, being discontinuous on the integers.

We denote by $M\{N/x\}$ the capture-free substitution of a term $N$ to the free occurrences of the variable $x$ in $M$.

Note that the presence of the additive structure on R turns the product $\mathsf{R}^n$ into a biproduct. Indeed, the injection $\iota_i^n$, for $i$ such that $1 \leq i \leq n$, may be defined as

$$\iota_i^n := \lambda x^\mathsf{R}.\, \langle 0, \ldots, 0, x, 0, \ldots, 0 \rangle\,, \tag{2}$$

where the variable $x$ is in the $i$-th position of the $n$-tuple. We omit the index $n$ when inessential or clear from the context.

For every type $A \to B$, we set $\Omega_{A\to B} := \mathsf{fix} f^{A\to B}.f$. We write just $\Omega$ when the type is irrelevant. Given a term $\Gamma, f : A \to B \vdash M : A \to B$ and $n \in \mathbb{N} \cup \{\infty\}$, we define $\mathsf{fix}_n f.M$ of type $A \to B$ as follows:

$$\mathsf{fix}_0 f.M := \Omega, \qquad \mathsf{fix}_{n+1} f.M := (\lambda f.M)(\lambda x.(\mathsf{fix}_n f.M)x), \qquad \mathsf{fix}_\infty f.M := \mathsf{fix} f.M. \tag{3}$$

Throughout the paper, we use boldface metavariables to denote sequences of metavariables, *i.e.*, $\mathbf{x} = x_1, \dots, x_n$ is a sequence of variables, $\mathbf{M} = M_1, \dots, M_n$ is a sequence of terms, etc. The length of the sequence is specified only when necessary.

Let us introduce two particularly important classes of terms:

Definition 1 (program, simple term). *A typing environment $\Gamma$ is* ground *whenever all of its variables have type* R. *A $\mathsf{PCF}_\mathsf{R}$ term $M$ is called a* program of arity $n$ and coarity $m$ *whenever* $x_1^\mathsf{R}, \dots, x_n^\mathsf{R} \vdash M : \mathsf{R}^m$.

*A term is called* simple *if it does not contain conditionals or fixpoints. Small Latin letters $t, u, v$ range over simple terms. Note that the subset of the simple terms of $\mathsf{PCF}_\mathsf{R}$ corresponds to the simply typed $\lambda$-calculus on the ground type* R *enriched with function symbols.*

A *context* is a term with a single occurrence of a special variable $\{\cdot\}$, called the *hole*. We use metavariables $\mathsf{C}, \mathsf{D}$ to range over contexts. Given a context $\mathsf{C}$ and a term $M$, we write $\mathsf{C}\{M\}$ for the term obtained by replacing the hole $\{\cdot\}$ of $\mathsf{C}$ with $M$, allowing the capture of the free variables in $M$ by the binders of $\mathsf{C}$.

The reduction relation $\to$ is defined by context closure of the rewriting rules in Fig. 1c:

Definition 2 (reduction). *Fig. 1c defines the set of* rewriting rules *of $\mathsf{PCF}_\mathsf{R}$, which are pairs $R \to P$ of terms, with $R$ called the* redex *and $P$ the* contractum *of the rule.*

*A* reduction step *$\sigma$ is a triple $(\mathsf{C}, R, P)$ such that $\mathsf{C}$ is a context, and $R \to P$ is a valid reduction rule. We also write $\sigma : \mathsf{C}\{R\} \to \mathsf{C}\{P\}$ or, when the context $\mathsf{C}$ is irrelevant, simply $\sigma : M \to N$, for $M = \mathsf{C}\{R\}$ and $N = \mathsf{C}\{P\}$, and say that $\sigma$* fires *the redex $R$ in $M$. A term $M$ without redexes, i.e. such that $M \neq \mathsf{C}\{R\}$ for any $\mathsf{C}$ and any redex $R$, is said to be* normal, *or a* normal form.

*Given another context $\mathsf{D}$, we denote by $\mathsf{D}\{\sigma\}$ the step $(\mathsf{D}\{\mathsf{C}\}, R, P)$. Similarly we write $\sigma\{N/x\}$ for the triple $(\mathsf{C}\{N/x\}, R\{N/x\}, P\{N/x\})$, which is still a valid reduction step.*

*A* reduction sequence *$\rho$ from a term $M$ to a term $N$, in symbols $\rho : M \to^* N$, is either empty, in which case $N = M$, or a sequence of reduction steps $(\mathsf{C}_i, R_i, P_i)_{1\le i\le n}$ with $n \ge 1$ such that $M = \mathsf{C}_1\{R_1\}$, $N = \mathsf{C}_n\{P_n\}$ and for all $i < n$, $\mathsf{C}_i\{P_i\} = \mathsf{C}_{i+1}\{R_{i+1}\}$. We call $M$ the* source *of $\rho$, $N$ its* target *and $n$ the* length *of $\rho$. We often identify a single reduction step and the corresponding reduction sequence of length 1. The notations $\mathsf{D}\{\rho\}$ and $\rho\{N/x\}$ are extended to reduction sequences in the obvious way.*

*Two reductions sequences $\rho : M \to^* M'$ and $\rho' : M' \to^* M''$ compose in the obvious way to yield a reduction sequence $\rho\rho' : M \to^* M''$. Empty reduction sequences are the identities of such an operation.*

*A sequence $\rho$ is* normalizing *if there is no reduction step $\sigma$ such that $\rho\sigma$ is a valid reduction sequence, or, equivalently, if the target of $\rho$ is a normal form. A term $M$ is called* normalizing *if there exists a normalizing reduction from $M$. Otherwise $M$ is said to be* diverging.

$\mathsf{PCF}_\mathsf{R}$ is Turing-complete: usual PCF [Plotkin 1977] is essentially the fragment obtained by restricting to integer (including negative) numerals and sum. We recall some results about reduction which are completely standard (see *e.g.* [Amadio and Curien 1998]).

Proposition 3 (confluence). *Whenever $M \to^* N_1$ and $M \to^* N_2$, there exist $N$ such that $N_1 \to^* N$ and $N_2 \to^* N$. In particular, if $N_1$ and $N_2$ are normal forms, then $N_1 = N_2$.*

Proposition 4 (subject reduction). *If $\Gamma \vdash M : A$ and $M \to M'$, then $\Gamma \vdash M' : A$.*

Proposition 5 (strong normalization for simple terms). *For every simple term $t$ there exists $n \in \mathbb{N}$ such that the length of every reduction sequence starting from $t$ is bounded by $n$.*

A *reduction strategy* $\mathcal{S}$ is a relation between terms $M$ of $\mathrm{PCF}_{\mathrm{R}}$ and occurrences of redexes in $M$. A strategy is called *deterministic* whenever it is a partial function. We denote by $\xrightarrow{\mathcal{S}}$ the reduction relation defined by reducing only the redexes fired by $\mathcal{S}$ and we write $\mathcal{S}$-nf for a normal form of $\xrightarrow{\mathcal{S}}$. We write $\beta$ for the maximal strategy, giving the reduction relation $\to$. Of course this strategy is not deterministic, however Proposition 3 assures that the normal form associated with a term is unique if it exists.

Reduction strategies are often defined by fixing a subset of redexes in Fig. 1c and a set of *evaluation contexts*. An example which will be useful in the sequel is *head reduction*, which is defined by taking all reduction rules but restricting their application to *head contexts*, generated by the following grammar:

$$\mathsf{H} ::= \{\cdot\} \mid \phi(M_1, \dots, \mathsf{H}, \dots, M_k) \mid \mathsf{H}N \mid \langle \mathsf{H}, N\rangle \mid \langle M, \mathsf{H}\rangle \mid \pi_i \mathsf{H} \mid \mathsf{if}(\mathsf{H}, M, N).$$

A *head reduction step* is of the form $\mathsf{H}\{R\} \to \mathsf{H}\{P\}$ with $R \to P$ a rewriting step of Fig. 1c and $\mathsf{H}$ a head context. A *head reduction sequence* is a reduction whose steps are all head reduction steps.

Observe that head reduction is not deterministic. Apart from the freedom in the order of the evaluation of the arguments of a function symbol, we allow to reduce within a pair as well as to project it: for instance, the term $\pi_1\langle(\lambda x.x)y, M\rangle$ may be decomposed either as $\mathsf{H}\{\pi_1\langle(\lambda x.x)y, M\rangle\}$ with the empty head context $\mathsf{H} = \{\cdot\}$, or as $\mathsf{H}'\{(\lambda x.x)y\}$ with the head context $\mathsf{H}' = \pi_1\langle\{\cdot\}, M\rangle$, and the two decompositions fire different redexes.

A classic result [Amadio and Curien 1998; Barendregt 1985] is that head reduction is a "winning strategy" for finding the $\beta$-normal form of a closed program:

Proposition 6. *Let $M$ be a normalizing closed program (i.e., of type $\mathsf{R}^n$) whose $\beta$-normal form is $N$. Then, there is a head reduction sequence $M \to^* N$.*

Let $\mathcal{S}$ be a reduction strategy, let $\Gamma = x_1^{\mathsf{R}}, \dots, x_n^{\mathsf{R}}$ and let $\Gamma \vdash M : \mathsf{R}^m$. We define the partial function $[\![M]\!]_\Gamma^{\mathcal{S}} : \mathbb{R}^n \rightharpoonup \mathbb{R}^m$ as follows:

$$[\![M]\!]_\Gamma^{\mathcal{S}}(r_1, \dots, r_n) = \begin{cases} \langle q_1, \dots, q_m\rangle & \text{if } M\{r_1/x_1\}\dots\{r_n/x_n\} \xrightarrow{\mathcal{S}}{}^* \langle q_1, \dots, q_m\rangle, \\ \bot & \text{otherwise,} \end{cases}$$

where $\bot$ means undefined. By Proposition 3, $[\![M]\!]_\Gamma^{\mathcal{S}}$ is a well-defined partial function, even in case $\mathcal{S}$ is not deterministic, in fact if $M\{r_1/x_1\}\dots\{r_n/x_n\} \xrightarrow{\mathcal{S}}{}^* \langle q_1, \dots, q_m\rangle$ and $M\{r_1/x_1\}\dots\{r_n/x_n\} \xrightarrow{\mathcal{S}}{}^* \langle q_1', \dots, q_m'\rangle$ we have that $q_i = q_i'$ for each $i$, as $\xrightarrow{\mathcal{S}} \subseteq \to$ and this latter is confluent.

We write $\Downarrow^{\mathcal{S}} M$ for $\Downarrow [\![M]\!]_\Gamma^{\mathcal{S}}$ and $\mathrm{d}^{\mathcal{S}}(M)$ for $\mathrm{d}([\![M]\!]_\Gamma^{\mathcal{S}})$. In case $\mathcal{S} = \beta$, we omit the indices from the above notations and write simply $[\![M]\!]_\Gamma$, $\Downarrow M$ and $\mathrm{d}(M)$. The first of the next two statements follows immediately from Proposition 6, the second from the definitions and the confluence property.

Proposition 7. *Let $M$, $N$ and $P$ be programs ($P$ of coarity 1), then the following relations hold:*

$$[\![\phi(M_1, \dots, M_k)]\!]_\Gamma = [\![\phi]\!]_\Gamma \circ \left\langle [\![M_1]\!]_\Gamma, \dots, [\![M_k]\!]_\Gamma \right\rangle, \qquad [\![\langle M_1, \dots, M_k\rangle]\!]_\Gamma = \left\langle [\![M_1]\!]_\Gamma, \dots, [\![M_k]\!]_\Gamma \right\rangle,$$

$$[\![\mathsf{if}(P, M, N)]\!]_\Gamma = \mathbf{r} \mapsto \begin{cases} [\![M]\!]_\Gamma(\mathbf{r}) & \text{if } [\![P]\!]_\Gamma(\mathbf{r}) \le 0, \\ [\![N]\!]_\Gamma(\mathbf{r}) & \text{if } [\![P]\!]_\Gamma(\mathbf{r}) > 0, \\ \bot & \text{if } [\![P]\!]_\Gamma(\mathbf{r}) = \bot. \end{cases}$$

$$\overrightarrow{\mathbf{D}}_n(\mathrm{R}) := \mathrm{R} \times \mathrm{R}^n \qquad \mathbf{D}_n(A \to B) := \mathbf{D}_n(A) \to \mathbf{D}_n(B)$$
$$\overleftarrow{\mathbf{D}}_n(\mathrm{R}) := \mathrm{R} \times \mathrm{R}^{\perp n} \qquad \mathbf{D}_n(A_1 \times \cdots \times A_k) := \mathbf{D}_n(A_1) \times \cdots \times \mathbf{D}_n(A_k)$$

(a) The action over types

$$\overrightarrow{\mathbf{D}}_n(\phi(\mathbf{M})) := \left(\lambda \mathbf{z}^{\mathrm{R}\times\mathrm{R}^n}.\left\langle \phi(\pi_1\mathbf{z})\,,\ \sum_{i=1}^{k} \partial_i\phi(\pi_1\mathbf{z}) \cdot \pi_2 z_i\,,\ldots,\ \sum_{i=1}^{k} \partial_i\phi(\pi_1\mathbf{z}) \cdot \pi_{n+1} z_i \right\rangle\right) \overrightarrow{\mathbf{D}}_n(\mathbf{M})$$
$$\overleftarrow{\mathbf{D}}_n(\phi(\mathbf{M})) := \left(\lambda \mathbf{z}^{\mathrm{R}\times\mathrm{R}^{\perp n}}.\left\langle \phi(\pi_1\mathbf{z})\,,\ \lambda a^{\mathrm{R}}.\sum_{i=1}^{k} \pi_2 z_i(\partial_i\phi(\pi_1\mathbf{z}) \cdot a)\right\rangle\right) \overleftarrow{\mathbf{D}}_n(\mathbf{M})$$

(b) The action over a function symbol $\phi$ of arity $k$. We suppose that, for every $1 \le i \le k$, there is an associated function symbol $\partial_i\phi$ of arity $k$ such that $\partial_i[\![\phi]\!](\mathbf{r}) = [\![\partial_i\phi]\!](\mathbf{r})$ for every $\mathbf{r} \in \mathbb{R}^k$ on which $\partial_i[\![\phi]\!]$ is defined. The writing $\phi(\mathbf{M})$ is a shortcut for $\phi(M_1,\ldots,M_k)$, and, similarly, $\lambda\mathbf{z}$ stands for the sequence of abstractions $\lambda z_1 \ldots \lambda z_k$ and $\mathbf{D}_n(\mathbf{M})$ for the sequence of applications to $\mathbf{D}_n(M_1)\ldots\mathbf{D}_n(M_k)$. These sequences are supposed empty if $k = 0$.

$$\mathbf{D}_n(x^A) := x^{\mathbf{D}_n(A)} \qquad \mathbf{D}_n(\lambda x^A.M) := \lambda x^{\mathbf{D}_n(A)}.\mathbf{D}_n(M) \qquad \mathbf{D}_n(MN) := \mathbf{D}_n(M)\mathbf{D}_n(N)$$
$$\mathbf{D}_n(\langle M_1,\ldots,M_k\rangle) := \langle \mathbf{D}_n(M_1),\ldots,\mathbf{D}_n(M_k)\rangle \qquad \mathbf{D}_n(\pi_i M) := \pi_i \mathbf{D}_n(M)$$
$$\mathbf{D}_n(\mathsf{if}(P,M,N)) := \mathsf{if}(\pi_1\mathbf{D}_n(P),\mathbf{D}_n(M),\mathbf{D}_n(N)) \qquad \mathbf{D}_n(\mathsf{fix} f^A.M) := \mathsf{fix} f^{\mathbf{D}_n(A)}.\mathbf{D}_n(M)$$

(c) The action over the other programming primitives.

Fig. 2. The forward and reverse AD transformations. The symbol $\mathbf{D}$ denotes either one of them. The index $n$ refers to the gradient dimension and acts only over ground type annotations. We will omit the index when inessential or clear from the context.

Proposition 8. *For every program $M$ and strategy $\mathcal{S}$, we have, for all $\mathbf{r} \in \Downarrow^{\mathcal{S}} M$, $[\![M]\!]^{\mathcal{S}}_{\Gamma}(\mathbf{r}) = [\![M]\!]_{\Gamma}(\mathbf{r})$. So, in particular, $\Downarrow^{\mathcal{S}} M \subseteq \Downarrow M$, $\mathrm{d}^{\mathcal{S}}(M) \subseteq \mathrm{d}(M)$ and, for every $\mathbf{r} \in \mathrm{d}^{\mathcal{S}}(M)$, $\mathcal{J}[\![M]\!]^{\mathcal{S}}_{\Gamma}(\mathbf{r}) = \mathcal{J}[\![M]\!]_{\Gamma}(\mathbf{r})$.*

## 2.2 Automatic Differentiation

As mentioned in the Introduction, the two modes of AD are performed by means of the program transformations $\overrightarrow{\mathbf{D}}$ (forward) and $\overleftarrow{\mathbf{D}}$ (reverse), outlined in full detail in Fig. 2. The cornerstone of these transformations is the chain rule, which describes the derivative of a composition of functions:

$$(f \circ g)'(x) = f'(g(x)) \cdot g'(x). \tag{4}$$

This says in particular that $(-)'$ is not functorial (*i.e.*, modular/compositional): the right-hand side of Equation (4) does not use only $g'(x)$ but also $g(x)$. Functoriality may be achieved by transforming a map $f : \mathbb{R} \to \mathbb{R}$ into a map $\overrightarrow{\mathbf{D}}(f) : \mathbb{R}^2 \to \mathbb{R}^2$ acting as follows:

$$\overrightarrow{\mathbf{D}}(f) : \langle z, \dot{z}\rangle \mapsto \langle f(z), f'(z) \cdot \dot{z}\rangle\,. \tag{5}$$

Referring to Equation (4), the input $z$, which is often called *primal* in the AD literature, corresponds to the output of $g$, whereas $\dot{z}$, called *tangent*, corresponds to the output of $g'$. The reader may easily check functoriality: $\overrightarrow{\mathbf{D}}(f \circ g) = \overrightarrow{\mathbf{D}}(f) \circ \overrightarrow{\mathbf{D}}(g)$. This generalizes to $n$-ary maps in the definition of

$\overrightarrow{\mathbf{D}}_n$ in Fig. 2b[2] and is the essential part of forward mode AD. The attribute *forward* refers to the fact that the computation of both the primal and the tangent follows the input-to-output flow, in particular the derivative $f'(z)$ is computed after having accumulated in $\dot{z}$ the derivative of its input function.

Notice that, if we denote by $|M|$ the size of a term $M$, we have that $|\overrightarrow{\mathbf{D}}_n(\phi(M))| = O(n)+|\overrightarrow{\mathbf{D}}_n(M)|$. So, supposing that $F$ consists only of function symbols and variables, evaluating both $F$ and $\overrightarrow{\mathbf{D}}_n(F)$ on a given input requires a number of operations roughly equal to their size. But $|\overrightarrow{\mathbf{D}}_n(F)| = O(n|F|)$, therefore the evaluation of $\overrightarrow{\mathbf{D}}_n(F)$ is blown up by a factor of $n$ with respect to the evaluation of $F$. In applications to deep learning, $F$ is a loss function and $n$ is the number of learning parameters, which may be huge (hundreds of millions).

Luckily, AD offers a more efficient method for applying the chain rule in these cases, called reverse mode AD, or *backpropagation*, because the idea is to accumulate the tangents in the reverse order with respect to the primals. More precisely, by taking the notation of (4), the backpropagation $\overleftarrow{\mathbf{D}}(f)$ of $f$ first computes $f'(g(x))$ and then waits for the derivative $g'(x)$ in order to perform the multiplication. As first observed by [Pearlmutter and Siskind 2008], this mode may be naturally expressed in a functional programming language by replacing the tangent variables $\dot{z}$ with *backpropagators* $z^*$ representing functions (in fact, special forms of continuations):

$$\overleftarrow{\mathbf{D}}(f) : \langle z, z^* \rangle \mapsto \left\langle f(z), \lambda a^{\mathrm{R}}.z^*(f'(z) \cdot a) \right\rangle \tag{6}$$

A backpropagator is a map $\mathbb{R} \to \mathbb{R}^n$ *waiting* for a real number (the derivative of the next function) in order to achieve the computation of the gradient of the whole function. In (6), the second component of the returned pair is the backpropagator associated with $f$, the variable $z^*$ being the awaited backpropagator associated with the input function of $f$ (called $g$ in (4)). This transformation generalizes to the definition of $\overleftarrow{\mathbf{D}}_n$ in Fig. 2b for $n$-ary maps.

We encourage the reader to check that $|\overleftarrow{\mathbf{D}}_n(\phi(M))| = O(1) + |\overleftarrow{\mathbf{D}}_n(M)|$, so if $F$ consists only of function symbols and variables, the evaluation of $\overleftarrow{\mathbf{D}}_n(F)$ is asymptotically as costly as the evaluation of $F$. However, in more complex cases, the sole transformation (6) is not enough to guarantee efficiency: if $F$ contains sharing, *e.g.* a subroutine $g$ called several times, the evaluation of $\overleftarrow{\mathbf{D}}(F)$ may duplicate uselessly the computation associated with the backpropagator of $g$, and this may result in an exponential blowup. This highlights a key difference between forward and reverse mode: if $F$ is a first order program (*i.e.*, every abstraction $\lambda x^A$ in $F$ is such that $A$ has no arrows), then $\overrightarrow{\mathbf{D}}(F)$ is also a first order program, whereas $\overleftarrow{\mathbf{D}}(F)$ is a higher order term. When backpropagation is expressed in an imperative language, as is usually the case, duplication is not a problem because efficiency is automatically achieved by accumulating the tangents (in reverse order) in memory. But for functional languages, subtle techniques have been introduced to avoid this problem, *e.g.* closure conversions [Pearlmutter and Siskind 2008] or memory references and delimited continuations [Wang et al. 2019].

In this paper we follow the approach of [Brunel et al. 2020], giving a purely functional solution based on linear logic types: backpropagators have type $\mathrm{R}^{\perp_n}$, which corresponds to the set of *linear* maps from $\mathbb{R}$ to $\mathbb{R}^n$. The efficiency of the transformation is then guaranteed by a *factoring rule* added to the operational semantics, which allows sharing the evaluation of different occurrences of a backpropagator $z^*$ in an expression:

$$z^*M + z^*N \quad \to \quad z^*(M + N). \tag{7}$$

[2]Modulo some syntactic bureaucracy, *i.e.*, the pair $\langle z, \dot{z} \rangle$ of (5) corresponds in Fig. 2b to a single variable $z$ of type $\mathrm{R} \times \mathrm{R}^n$, whose components are obtained by using the projections $\pi_i$

This rewriting rule is sound because backpropagators are linear maps, so they commute with sums. In particular, the normal form of a term obtained using (7) is the same one would have obtained, perhaps in more steps, without using it. As mentioned in the Introduction, in our present setting we are concerned only with soundness, not efficiency. For this reason, we adopt the definition

$$\mathsf{R}^{\perp_n} := \mathsf{R} \to \mathsf{R}^n$$

and do not consider the linear factoring rule (7). In fact, by the above remark, rule (7) only speeds up computation without introducing any error, so our almost-everywhere soundness result is transparent to its use, and enforcing linearity would only lead to unnecessary complications induced by a more sophisticated type system. We do retain the notation $(-)^\perp$ as a reminder that this is supposed to be a *linear* arrow (*i.e.*, $\mathsf{R}^{\perp_n}$ should really be $\mathsf{R} \multimap \mathsf{R}^n$), but the transformation of [Brunel et al. 2020] remains well typed with the above "non-linear" definition of negation. Indeed, apart from the addition of conditional and fixpoints, the transformation of Fig. 2 is exactly that of *loc. cit.* and it is efficient as long as it is executed according to the operational semantics enriched with (7), so nothing is lost with respect to our previous work.

So far we have explained AD transformations only in regard to primitive functions, which are the "elementary blocks" of straight-line programs mentioned in the Introduction. It is an observation first formalized in [Wang et al. 2019] that the transformations may be extended to arbitrary programs simply by applying the functoriality principle: $\overrightarrow{\mathbf{D}}$ and $\overleftarrow{\mathbf{D}}$ are defined to commute with the programming constructs of the language, resulting in Fig. 2c. As a result, the abstract syntax tree of an expression is basically preserved by the two transformations,[3] only the types of the variables are lifted so as to accommodate primals and tangents or backpropagators at the ground level. This behavior is often described by saying that AD is implemented via "operator overloading". We prefer the term "functoriality" because we find it technically more appropriate.

Notice that, in the definition of $\mathbf{D}(\mathsf{if}(P, M, N))$, the transformation is applied also to the guard $P$ in order to preserve typability, because $P$ may share free variables with $M$ and $N$. However, the computation of the gradient of $P$ is useless and therefore $\mathbf{D}(P)$ is projected to the first component. A possible optimization would be to define $\mathbf{D}(\mathsf{if}(P, M, N)) := \mathsf{if}(\mathbf{D}'(P), \mathbf{D}(M), \mathbf{D}(N))$ where $\mathbf{D}'$ is an auxiliary transformation such that $\mathbf{D}'(x^A) = \pi_1 x^{\mathbf{D}(A)}$ and which behaves homomorphically on every other term. We avoid introducing $\mathbf{D}'$ because it is not crucial for our results and because such an optimization is not so relevant at our level of abstraction (indeed, the evaluation of $\mathbf{D}(P)$ is linear in the evaluation of $P$, as proved in [Brunel et al. 2020], so asymptotically there is no gain).

Some examples of the two modes of $\mathbf{D}$ are given in Fig. 3. In the case of subtraction (Fig. 3b), notice how the size of the term resulting from the forward transformation $\overrightarrow{\mathbf{D}}_n$ increases with the gradient dimension $n$, while it is constant in $\overleftarrow{\mathbf{D}}_n$, as expected from the above discussion.

Given a term $\Gamma \vdash M : A$ with $\Gamma = x_1^{A_1}, \dots, x_n^{A_n}$, one can check that $\mathbf{D}(\Gamma) \vdash \mathbf{D}(M) : \mathbf{D}(A)$, where $\mathbf{D}(\Gamma) = x_1^{\mathbf{D}(A_1)}, \dots, x_n^{\mathbf{D}(A_n)}$. In particular, if $M$ is a program, then

$$x_1^{\mathsf{R}\times\mathsf{R}^n}, \dots, x_n^{\mathsf{R}\times\mathsf{R}^n} \vdash \overrightarrow{\mathbf{D}}_n(M) : \mathsf{R} \times \mathsf{R}^n \qquad x_1^{\mathsf{R}\times\mathsf{R}^{\perp n}}, \dots, x_n^{\mathsf{R}\times\mathsf{R}^{\perp n}} \vdash \overleftarrow{\mathbf{D}}_n(M) : \mathsf{R} \times \mathsf{R}^{\perp n} \tag{8}$$

If, furthermore, $M$ is simple, then the computational behavior of the transformations $\overrightarrow{\mathbf{D}}$ and $\overleftarrow{\mathbf{D}}$ is given by the following result, which was proved in [Barthe et al. 2020; Brunel et al. 2020; Huot et al. 2020],[4] and in which $\iota_i^n$ are the injections of $\mathsf{R}$ into $\mathsf{R}^n$ as defined in Equation (2).

[3] This is the case for all constructs except the conditional, where a projection is added to the transformation of the guard. This minor technicality is due to the fact that we consider only the ground type of real numbers and not that of Booleans.
[4] In a personal communication, Mitchell Wand showed us that soundness of reverse mode AD may also be proved by means of "open" logical relations of the kind discussed in [Barthe et al. 2020] and used here for the unsoundness bound (Sect. 4).

$$\overrightarrow{\mathbf{D}}_1(\mathrm{ReLU}) = \lambda x^{\mathrm{R}\times\mathrm{R}}.\mathsf{if}(\pi_1 x, \langle 0, 0\rangle, x) \qquad \overleftarrow{\mathbf{D}}_1(\mathrm{ReLU}) = \lambda x^{\mathrm{R}\times\mathrm{R}^\perp}.\mathsf{if}(\pi_1 x, \langle 0, \lambda a.0\rangle, x)$$

(a) $\mathbf{D}_1$ of the rectified linear unit ReLU, defined in (1).

$$\overrightarrow{\mathbf{D}}_1(x^{\mathrm{R}} - y^{\mathrm{R}}) = \left(\lambda z_1^{\mathrm{R}\times\mathrm{R}} z_2^{\mathrm{R}\times\mathrm{R}}.\left\langle \pi_1^2(z_1) - \pi_1^2(z_2), \pi_2^2(z_1) - \pi_2^2(z_2)\right\rangle\right) x^{\mathrm{R}\times\mathrm{R}} y^{\mathrm{R}\times\mathrm{R}}$$
$$\overrightarrow{\mathbf{D}}_2(x^{\mathrm{R}} - y^{\mathrm{R}}) = \left(\lambda z_1^{\mathrm{R}\times\mathrm{R}^2} z_2^{\mathrm{R}\times\mathrm{R}^2}.\left\langle \pi_1^3(z_1) - \pi_1^3(z_2), \pi_2^3(z_1) - \pi_2^3(z_2), \pi_3^3(z_1) - \pi_3^3(z_2)\right\rangle\right) x^{\mathrm{R}\times\mathrm{R}^2} y^{\mathrm{R}\times\mathrm{R}^2}$$
$$\overleftarrow{\mathbf{D}}_n(x^{\mathrm{R}} - y^{\mathrm{R}}) = \left(\lambda z_1^{\mathrm{R}\times\mathrm{R}^{\perp n}} z_2^{\mathrm{R}\times\mathrm{R}^{\perp n}}.\left\langle \pi_1^2(z_1) - \pi_1^2(z_2), \lambda a^{\mathrm{R}}.(\pi_2^2(z_1) 1\cdot a + \pi_2^2(z_2)(-1)\cdot a)\right\rangle\right) x^{\mathrm{R}\times\mathrm{R}^{\perp n}} y^{\mathrm{R}\times\mathrm{R}^{\perp n}}$$

(b) $\mathbf{D}_n$ of the subtraction $x - y$, with $n = 1, 2$, where we suppose $\partial_1(x - y) := 1$ and $\partial_2(x - y) := -1$.

Fig. 3. Some examples of the $\mathbf{D}$ transformations. Notice that we take the liberty of using the same name for the ground variables $x^{\mathrm{R}}$ and $y^{\mathrm{R}}$ and their images $x^{\mathbf{D}(\mathrm{R})}$ and $y^{\mathbf{D}(\mathrm{R})}$ under $\mathbf{D}$.

Proposition 9 (soundness of AD for simple terms). *Let $\Gamma = x_1^{\mathrm{R}}, \dots, x_n^{\mathrm{R}}$ and let $\Gamma \vdash t : \mathrm{R}$ be a simple program. Then, for all $\mathbf{r} = (r_1, \dots, r_n) \in \mathrm{d}(t)$, we have*

$$\overrightarrow{\mathbf{D}}_n(t)\{\langle r_1, \iota_1^n 1\rangle/x_1\}\dots\{\langle r_n, \iota_n^n 1\rangle/x_n\} \to^* \langle [\![t]\!]_\Gamma(\mathbf{r}), \nabla[\![t]\!]_\Gamma(\mathbf{r})\rangle$$
$$\overleftarrow{\mathbf{D}}_n(t)\{\langle r_1, \iota_1^n\rangle/x_1\}\dots\{\langle r_n, \iota_n^n\rangle/x_n\}) \to^* \langle [\![t]\!]_\Gamma(\mathbf{r}), u\rangle$$

*such that $u1 \to^* \nabla[\![t]\!]_\Gamma(\mathbf{r})$.*

Looking at Proposition 9, if $M$ is an arbitrary program of arity $n$ and coarity 1, it is reasonable to believe that the following programs compute $\nabla[\![M]\!]_\Gamma$ in $\mathbf{r} = (r_1, \dots, r_n) \in \mathbb{R}^n$ whenever this is defined:

$$\overrightarrow{grad}_n(M)(\mathbf{r}) := \pi_2^2\overrightarrow{\mathbf{D}}_n(M)\{\langle r_1, \iota_1^n 1\rangle/x_1\}\dots\{\langle r_n, \iota_n^n 1\rangle/x_n\}, \tag{9}$$
$$\overleftarrow{grad}_n(M)(\mathbf{r}) := (\pi_2^2\overleftarrow{\mathbf{D}}_n(M)\{\langle r_1, \iota_1^n\rangle/x_1\}\dots\{\langle r_n, \iota_n^n\rangle/x_n\})1. \tag{10}$$

In the sequel, we will omit the index $n$ when inessential or clear from the context. We will also write $grad(M)$ for either one the above terms.[5]

Referring to Fig. 3, it is immediate to check that, regardless of the mode, $grad_1(\mathrm{ReLU}\, z^{\mathrm{R}})(r) \to^* 1$ if $r > 0$ and $grad_1(\mathrm{ReLU}\, z^{\mathrm{R}})(r) \to^* 0$ if $r \le 0$. This is the expected result except for $r = 0$, where the map $[\![\mathrm{ReLU}\, z]\!]_{z^{\mathrm{R}}}$ is not differentiable. As discussed in the Introduction, our soundness result concerns only the domain of differentiability $\mathrm{d}(M)$ of a map $[\![M]\!]$ represented by a program and nothing is stated about the value of $grad(M)$ outside $\mathrm{d}(M)$. This situation is quite common in AD frameworks, where it may even be desirable to control the behavior of the "non-existent derivative" on singularities. For example, the following term cReLU implements the rectified linear unit (*i.e.*, $[\![\mathrm{cReLU}\, z]\!]_{z^{\mathrm{R}}} = [\![\mathrm{ReLU}\, z]\!]_{z^{\mathrm{R}}}$) so that $grad$ returns some arbitrarily chosen $q \in \mathbb{R}$ on 0:

$$\mathrm{cReLU}_q := \lambda x^{\mathrm{R}}.\mathsf{if}(x, \mathsf{if}(-x, q\cdot x, 0), x), \qquad grad_1(\mathrm{cReLU}_q\, z^{\mathrm{R}})(r) \to^* \begin{cases} 1 & \text{if } r > 0, \\ q & \text{if } r = 0, \\ 0 & \text{if } r < 0. \end{cases} \tag{11}$$

We do not consider these computations as errors because $\nabla[\![\mathrm{ReLU}\, z]\!]_{z^{\mathrm{R}}}$ is undefined at 0.

[5]Notice that the above definition of $grad$ is slightly different from the informal one used in the Introduction: it applies to terms with free ground variables, whereas in the Introduction we abusively applied $grad$ to closed terms.

We already discussed SillyId (see (1)) as a first example of a mismatch between AD and the gradient in the domain of differentiability of a map. Let us consider here a refined example:

$$\mathsf{EqProj} := \lambda x^{\mathsf{R}}.\lambda y^{\mathsf{R}}.\mathsf{if}(x-y, \mathsf{if}(y-x, x, y), y). \tag{12}$$

This term is extensionally equivalent to the binary projection $\lambda x^{\mathsf{R}}.\lambda y^{\mathsf{R}}.y$ and therefore its gradient should be $\langle 0, 1\rangle$ on the whole domain $\mathbb{R}^2$. By contrast, the reader may check that:

$$\begin{aligned}\overrightarrow{grad}_2(\mathsf{EqProj}\, x_1^{\mathsf{R}} x_2^{\mathsf{R}})(r_1, r_2) &= \pi_2^2\left(\overrightarrow{\mathbf{D}}_2(\mathsf{EqProj})\left\langle r_1, \iota_1^2 1\right\rangle\left\langle r_2, \iota_2^2 1\right\rangle\right)\\ &\to^* \pi_2^2\left(\mathsf{if}(r_1-r_2, \mathsf{if}(r_2-r_1, \langle r_1, \langle 1,0\rangle\rangle, \langle r_2, \langle 0,1\rangle\rangle), \langle r_2, \langle 0,1\rangle\rangle)\right)\\ &\to^* \begin{cases}\langle 0,1\rangle & \text{if } r_1 \neq r_2,\\ \langle 1,0\rangle & \text{if } r_1 = r_2.\end{cases}\end{aligned}$$

The diagonal of $\mathbb{R}^2$ gives an uncountable set of errors (a similar computation yields the same result also for the reverse mode). However, this set is negligible, *i.e.*, of Lebesgue measure zero, in accordance with the claim (ii') stated in the Introduction.

Let us add a last comment on example (12). Consider the unary program $\mathsf{EqProj}\, x_1^{\mathsf{R}} x_1^{\mathsf{R}}$, which is extensionally equivalent to the identity. One can check that $\overrightarrow{grad}_1(\mathsf{EqProj}\, x_1 x_1)(r) \to^* 1$ for every $r \in \mathbb{R}$, so there is no error at all in this case. This is in sharp contrast with approaches based on partial conditionals, such as [Abadi and Plotkin 2020], in which conditionals diverge when the guard evaluates to 0: under such semantics, $\mathsf{EqProj}\, x_1 x_1$ diverges everywhere.

Different reduction strategies change the convergence and differentiability domain of a program, so a priori the soundness of AD depends on the strategy. In the above examples, we considered the maximal reduction strategy $\beta$. If we wish to specialize to a more restrictive reduction strategy $\mathcal{S}$, we should prove that $grad(M)$ evaluates, in accordance with $\mathcal{S}$, to the gradient of $[\![M]\!]_\Gamma^{\mathcal{S}}$ at almost every point where this is defined. In fact, if we succeed in proving this with respect to $\beta$, then it follows also for any other "reasonable" strategy $\mathcal{S}$:

Proposition 10. *Let $\Gamma \vdash M : \mathsf{R}$ be a program. If $[\![grad(M)]\!]_\Gamma|_{\mathrm{d}(M)} \sim \nabla([\![M]\!]_\Gamma)$, then for any reduction strategy $\mathcal{S}$ such that $\mathrm{d}^{\mathcal{S}}(M) \subseteq \Downarrow^{\mathcal{S}}(grad(M))$ we also have $[\![grad(M)]\!]_\Gamma^{\mathcal{S}}|_{\mathrm{d}^{\mathcal{S}}(M)} \sim \nabla([\![M]\!]_\Gamma^{\mathcal{S}})$.*

Proof. By hypothesis we have that $\mathrm{d}(M) = A \cup Z$ with $Z$ of measure zero and $[\![grad(M)]\!]_\Gamma|_A = \nabla([\![M]\!]_\Gamma)|_A$. Let $\mathcal{S}$ be a reduction strategy satisfying the hypothesis and let $B := A \cap \mathrm{d}^{\mathcal{S}}(M)$. We need to prove that $[\![grad(M)]\!]_\Gamma^{\mathcal{S}}|_B = \nabla([\![M]\!]_\Gamma^{\mathcal{S}})|_B$ and that $\mathrm{d}^{\mathcal{S}}(M) \setminus B$ is negligible.

Let us start with this latter point. Notice that, by Proposition 8, $\mathrm{d}^{\mathcal{S}}(M) \setminus B = \mathrm{d}^{\mathcal{S}}(M) \setminus (A \cap \mathrm{d}^{\mathcal{S}}(M)) = \mathrm{d}^{\mathcal{S}}(M) \setminus A \subseteq \mathrm{d}(M) \setminus A$, and the latter set is of measure zero by hypothesis.

Let us now take $\mathbf{r} \in B$. We have:

$$\begin{aligned}[\![grad(M)]\!]_\Gamma^{\mathcal{S}}(\mathbf{r}) &= [\![grad(M)]\!]_\Gamma(\mathbf{r}) && \text{by } B \subseteq \mathrm{d}^{\mathcal{S}}(M) \subseteq \Downarrow^{\mathcal{S}}(grad(M)) \text{ and Proposition 8}\\ &= \nabla([\![M]\!]_\Gamma)(\mathbf{r}) && \text{because } B \subseteq A\\ &= \nabla([\![M]\!]_\Gamma^{\mathcal{S}})(\mathbf{r}) && \text{by } B \subseteq \mathrm{d}^{\mathcal{S}}(M) \text{ and Proposition 8.}\end{aligned}$$

□

The condition $\mathrm{d}^{\mathcal{S}}(M) \subseteq \Downarrow^{\mathcal{S}}(grad(M))$ is reasonable, since $\mathrm{d}^{\mathcal{S}}(M) \subseteq \Downarrow^{\mathcal{S}} M$ by definition and it is very likely that $\Downarrow^{\mathcal{S}} M \subseteq \Downarrow^{\mathcal{S}} grad(M)$, because the convergence of $grad(M)$ coincides with that of $\mathbf{D}(M)$ and the latter essentially behaves like $M$, as we will prove in Sect. 3.2. Notice that, when $t$ is simple, $\Downarrow^{\mathcal{S}} t = \Downarrow^{\mathcal{S}} grad(t)$ is trivially true because of strong normalization (Proposition 5), so

Proposition 9 in fact holds for any reduction strategy. Common strategies such as call-by-value, call-by-name and call-by-need are easily seen to enjoy the condition of Proposition 10. See Remark 30 below for a proof sketch in the case of call-by-value.

### 2.3 Primitive Functions and Complete Quasicontinuity

The only assumption we made so far about the function symbols of $\mathsf{PCF}_{\mathsf{R}}$ is that for every $\phi$ of arity $k$ and every $1 \leq i \leq k$, there is another $k$-ary function symbol $\partial_i\phi$ corresponding to the partial derivative of $[\![\phi]\!]$ with respect to its $i$-th argument (Fig. 2b). In order to prove one of our main results (Theorem 42), we will need to make some further topological and measure-theoretic assumptions, which we proceed to spell out.

In what follows, we always consider $\mathbb{R}^n$ with its standard topology and the Lebesgue measure. We denote by $\mathrm{int}(X)$ the interior of a set $X$ (the largest open set contained in $X$) and by $\mathrm{bor}(X)$ its *border*, defined as $\mathrm{bor}(X) := X \setminus \mathrm{int}(X)$. Equivalently, $\mathrm{bor}(X) = \partial X \cap X$ where $\partial X$ is the boundary of $X$ (the closure of $X$ minus $\mathrm{int}(X)$). In case $X$ is closed, $\mathrm{bor}(X) = \partial X$.

We recall that a *clone* [Szendrei 1986] on a set $A$ is a collection $\mathbf{P}$ of functions $A^n \rightharpoonup A$ (for varying $n$) which is closed under composition[6] and contains all projections (in particular, the identity on $A$). Notice that clones are stable under arbitrary intersections, hence every set $\mathbf{F}$ of functions $A^n \rightharpoonup A$ (for possibly varying $n$) generates a clone $\langle \mathbf{F} \rangle$, the smallest clone containing $\mathbf{F}$.

Definition 11 (admissible primitive functions). *We say that a clone $\mathbf{P}$ on $\mathbb{R}$ is* admissible *if $f \in \mathbf{P}$ implies:*

(1) *$f$ is continuous on its domain;*
(2) *if $f : \mathbb{R}^n \rightharpoonup \mathbb{R}$ is not identically zero, then $f^{-1}(0)$ is of Lebesgue measure zero in $\mathbb{R}^n$.*

*Fix a set $\mathbf{F}$ of function symbols together with their semantics. We abusively denote by $\mathbf{F}$ also the set of all $[\![\phi]\!]$ with $\phi$ ranging over the chosen function symbols. We say that $\mathbf{F}$ forms an* admissible set of primitive functions *if $\langle \mathbf{F} \rangle$ is admissible.*

It is well known [Mityagin 2015] that an example of admissible clone is provided by the collection of all real functions which are defined and analytic on some open set $U \subseteq \mathbb{R}^n$, for varying $U$ and $n$. Notice that a subclone of an admissible clone is admissible. Therefore, a simple way of ensuring that the primitive functions of $\mathsf{PCF}_{\mathsf{R}}$ are admissible is to ask that they are all analytic where they are defined. This is of course true of our "mandatory" primitive functions (constants, addition and multiplication), as well as all functions usually taken as primitive, such as division, square root, exponential, logarithm, the trigonometric functions and their inverses, Gaussian functions, many sigmoid functions (*e.g.* the error function), etc. Other desirable functions which are not analytic (the step function, the floor function, the rectified linear unit…) are usually programmable in $\mathsf{PCF}_{\mathsf{R}}$ from these primitive functions.

The definitions that follow are parametric in a choice of admissible clone $\mathbf{B}$, so we should speak of $\mathbf{B}$-quasiopen set, $\mathbf{B}$-quasicontinuity, etc. However, for simplicity, we will omit the parameter $\mathbf{B}$, implicitly fixing once and for all an admissible set $\mathbf{F}$ of primitive functions and letting $\mathbf{B} := \langle \mathbf{F} \rangle$. Functions in $\mathbf{B}$ will be called *basic*.

Definition 12 (quasiopen set). *We define the class of* quasiopen *sets of $\mathbb{R}^n$ to be the smallest class of subsets of $\mathbb{R}^n$ which:*

(1) *contains every open set;*
(2) *contains the zero set of every basic function $\mathbb{R}^n \rightharpoonup \mathbb{R}$;*
(3) *is closed under countable unions and binary intersections.*

[6] We mean that if $f : A^k \rightharpoonup A$ and $g_1, \ldots, g_k : A^n \rightharpoonup A$ are in $\mathbf{P}$, then so is the function $\mathbf{a} \mapsto f(g_1(\mathbf{a}), \ldots, g_k(\mathbf{a}))$.

*Inductively, the quasiopen sets of $\mathbb{R}^n$ may be defined as follows:*

$$Q, Q' ::= U \mid h^{-1}(0) \mid \bigcup_{i\in I} Q_i \mid Q \cap Q',$$

*where $U$ ranges over the open sets of $\mathbb{R}^n$, $h : \mathbb{R}^n \rightharpoonup \mathbb{R}$ ranges over basic functions (which may further be supposed to be not identically zero) and $I$ is countable.*

Definition 13 (quasivariety). *A set $Z \subseteq \mathbb{R}^n$ is called a* quasivariety *if there exists a family $\{h_i\}_{i\in I}$ of basic functions $h_i : \mathbb{R}^n \to \mathbb{R}$ with $I$ countable and such that $Z \subseteq \bigcup_{i\in I} h^{-1}(\{0\})$. In other words, a quasivariety is an arbitrary subset of a countable union of zero sets of basic functions.*

The following result, which says that quasivarieties form a class of "negligible sets", will be frequently used in the sequel, without explicit mention:

Lemma 14. *Quasivarieties enjoy the following properties:*

(1) **measure zero:** *if $Z \subseteq \mathbb{R}^n$ is a quasivariety, then it a has Lebesgue measure zero in $\mathbb{R}^n$;*
(2) **stability under countable unions:** *if $\{Z_i\}_{i\in I}$ is a countable family of quasivarieties, then $\bigcup_{i\in I} Z_i$ is a quasivariety;*
(3) **stability under subsets:** *if $Z$ is a quasivariety and $Z' \subseteq Z$, then $Z'$ is a quasivariety.*

Proof. Immediate from the definition. □

Lemma 15. *Let $Q \subseteq \mathbb{R}^n$ be quasiopen. Then:*

(1) *there exists an open set $U$ and a quasivariety $Z$ such that $Q = U \cup Z$;*
(2) $\mathrm{bor}(Q)$ *is a quasivariety. Hence, in the above one may always take $U = \mathrm{int}(Q)$ and $Z = \mathrm{bor}(Q)$.*

The set of non-positive numbers $\mathbb{R}_{\leq 0}$ is an example of quasiopen subset of $\mathbb{R}$: to see why, simply notice that $\mathbb{R}_{\leq 0} = \mathbb{R}_{<0} \cup \{0\}$, the first being open and the second being the zero set of the identity, which is always a basic function. In a sense, the key property of the class of quasiopen sets is that it includes both $\mathbb{R}_{\leq 0}$ and $\mathbb{R}_{>0}$, a fact which will be used crucially in Lemma 38.

On the other hand, thick Cantor sets provide examples of non-quasiopen subsets of $\mathbb{R}$: such a set $K$ is closed, of positive measure and has empty interior, so $K = \mathrm{bor}(K)$, which would contradict Lemma 15.2 if $K$ were quasiopen.

In what follows, if $f : A \rightharpoonup B$ and $g : C \rightharpoonup D$ are partial functions between sets, we write $f \times g$ for the function of type $A \times C \rightharpoonup B \times D$ such that $(f \times g)(a, c) = (f(a), g(c))$ whenever $f(a)$ and $g(c)$ are defined, and is undefined otherwise.

Definition 16 ((complete) quasicontinuity). *A function $f : \mathbb{R}^n \rightharpoonup \mathbb{R}^m$ is* quasicontinuous[7] *if, for every quasiopen set $Q \subseteq \mathbb{R}^m$, $f^{-1}(Q)$ is quasiopen. We say that $f$ is* completely quasicontinuous *(cqc) if $\mathrm{id}_{\mathbb{R}^k} \times f$ is quasicontinuous, for all $k \in \mathbb{N}$.*

Complete quasicontinuity is needed in order to have Lemma 17.4 below. It is worth pointing out, however, that we have not been able to find an example of a quasicontinuous function which is not completely quasicontinuous. So, while we conjecture that complete quasicontinuity is strictly stronger than quasicontinuity, the two notions might coincide in reality.

Lemma 17. *We have the following properties:*

(1) *a function $f : \mathbb{R}^n \to \mathbb{R}^m$ is quasicontinuous iff for every $Q$ which is either open or the zero set of a basic function, $f^{-1}(Q)$ is quasiopen.*

[7] The terminology "quasicontinuous" already has a standard meaning, unrelated to the one defined here. On the other hand, "quasiopen" and "completely quasicontinuous", which are the fundamental notions used in this work, do not seem to have been used in the literature.

$$\frac{}{\mathrm{R} \sqsubset \mathrm{R}} \quad \frac{A' \sqsubset A \quad B' \sqsubset B}{A' \to B' \sqsubset A \to B} \quad \frac{A'_i \sqsubset A_i,\ \forall\, 1 \le i \le k}{A'_1 \times \cdots \times A'_k \sqsubset A_1 \times \cdots \times A_k} \quad \frac{A_i \sqsubset A,\ \forall\, 1 \le i \le n}{A_1 \times \cdots \times A_n \sqsubset A}$$

(a) The pre-trace relation on types.

$$\frac{}{\Xi \vdash \{\cdot\} \sqsubset \{\cdot\}} \qquad \frac{A_1 \times \cdots \times A_n \sqsubset A}{\Xi, p^{A_1 \times \cdots \times A_n} \sqsubset x^A \vdash \pi_i^n p \sqsubset x}\ i \in \{1, \dots, n\}$$

$$\frac{\Xi, p^{A'} \sqsubset x^A \vdash t \sqsubset M}{\Xi \vdash \lambda p^{A'}.t \sqsubset \lambda x.M} \qquad \frac{\Xi \vdash t \sqsubset M, \quad \Xi \vdash u_1 \sqsubset N \quad \dots \quad \Xi \vdash u_n \sqsubset N}{\Xi \vdash t\langle u_1, \dots, u_n\rangle \sqsubset MN}$$

$$\frac{\Xi \vdash t_1 \sqsubset M_1, \quad \dots, \quad \Xi \vdash t_k \sqsubset M_k}{\Xi \vdash \langle t_1, \dots, t_k\rangle \sqsubset \langle M_1, \dots, M_k\rangle} \qquad \frac{\Xi \vdash t \sqsubset M}{\Xi \vdash \pi_i^k t \sqsubset \pi_i^k M} \qquad \frac{\Xi \vdash t_1 \sqsubset M_1 \quad \dots \quad \Xi \vdash t_k \sqsubset M_k}{\Xi \vdash \phi(t_1, \dots, t_k) \sqsubset \phi(M_1, \dots, M_k)}$$

$$\frac{\Xi \vdash t_i \sqsubset M_i}{\Xi \vdash \pi_i\langle t_i, t_i\rangle \sqsubset \mathsf{if}(P, M_1, M_2)}\ i \in \{1, 2\} \qquad \frac{\Xi \vdash t \sqsubset \mathsf{fix}_n f.M}{\Xi \vdash t \sqsubset \mathsf{fix} f.M}\ n > 0$$

(b) The pre-trace relation on terms. In the variable rule, if $n = 1$, then $\pi_i$ is omitted. Recall the definition of $\mathsf{fix}_n f.M$ in (3).

Fig. 4. The pre-trace relation between simple and arbitrary $\mathrm{PCF}_{\mathrm{R}}$ terms.

(2) *Identities are cqc and cqc functions are stable under composition.*
(3) *Basic functions are cqc. In particular, projections are cqc.*
(4) *If $f : \mathbb{R}^k \rightharpoonup \mathbb{R}^m$ and $g : \mathbb{R}^k \rightharpoonup \mathbb{R}^n$ are cqc, then the function $\langle f, g\rangle : \mathbb{R}^k \to \mathbb{R}^{m+n}$ defined by $\langle f, g\rangle(z) := (f(z), g(z))$ if $z \in \Downarrow f \cap \Downarrow g$ and undefined otherwise, is also cqc.*

Contrarily to what the name might suggest, a continuous function is *not* in general quasicontinuous. In fact, it is well known that any closed subset of $\mathbb{R}$ may be the zero set of a map which is smooth everywhere (in particular, continuous). So let $\phi : \mathbb{R} \to \mathbb{R}$ be a smooth function whose zero set is a thick Cantor set $K$, which, as observed above, is not quasiopen. The set $\{0\}$ is quasiopen (it is the zero set of the identity, which is a basic map), and yet $\phi^{-1}(\{0\}) = K$, so $\phi$ is not quasicontinuous.

## 3 SOUNDNESS OF AD

We want to prove that $grad(M)$ computes the gradient of a program $M$ almost everywhere in $\mathrm{d}(M)$ (Theorem 42). The proof splits in two parts: Theorem 33 states that $grad(M)$ is sound for the set $\mathrm{S}(M)$ of stable points of $\mathrm{d}(M)$ (Definition 26) and Sect. 4 shows that $\mathrm{S}(M)$ is actually almost all of $\mathrm{d}(M) \subseteq \Downarrow M$, in the sense that $\Downarrow M \setminus \mathrm{S}(M)$ is of measure zero.

Intuitively, a point $\mathbf{r} \in \Downarrow M$ is stable whenever there exists a simple term $t$ that "traces" the evaluation of $M$ over an open ball $B_\varepsilon(\mathbf{r})$ of $\mathbf{r}$. Such a $t$ allows us to lift the soundness theorem for simple terms (Proposition 9) to $grad(M)$. This reasoning is based on the extrusion lemma (Lemma 31), which needs a notion of "trace" not only at level of terms (Definition 25), but also at the level of the reduction sequences (Definition 24).

### 3.1 Traces

The *pre-trace relation* is defined in Fig. 4. The judgments used in the definition are of the form $\Xi \vdash t \sqsubset M$, where $t$ is a simple term or simple context, $M$ is an arbitrary term or context of type, say, $\Gamma \vdash M : A$ and $\Xi$ is a function mapping any variable $x^A$ of $\Gamma$ to a fresh variable $p^{A'}$ with $A' \sqsubset A$.

We usually denote this map as a list $p_1^{A_1'} \sqsubset x_1^{A_1}, \dots, p_n^{A_n'} \sqsubset x_n^{A_n}$, supposing the $p_i$'s and $x_i$'s to be pairwise different.

For brevity, we omit to specify the types of the terms in the subjects of the judgments in Fig. 4b, but we encourage the reader to verify that if $p_1 \sqsubset x_1, \dots, p_n \sqsubset x_n \vdash t \sqsubset M$ and $x_1^{C_1}, \dots, x_n^{C_n} \vdash M : A$, then $p_1^{C_1'}, \dots p_n^{C_n'} \vdash t : A'$ with $A' \sqsubset A$ and, for all $1 \le i \le n$, $C_i' \sqsubset C_i$. In particular, $t$ is a simply-typed $\lambda$-term or context. We write $t \sqsubset M$ when the typing environment $\Xi$ is irrelevant.

Conditionals and fixpoints are the only additional features of $\mathsf{PCF}_{\mathsf{R}}$ with respect to the simply-typed $\lambda$-calculus, and the purpose of $\sqsubset$ is to "trace" them with simply-typed terms themselves. The last two rules of Fig. 4b "slice out" a conditional with the traces of its two branches and unfold a fixpoint into its finite approximations. In fact, the conditional rule is a bit more convoluted as it uses a dummy projection in order to encode the index of the chosen branch, a crucial information for the extrusion property (Lemma 31). For example, $u_1 := \lambda x^{\mathsf{R}}.\pi_1\langle 0, 0\rangle$ and $u_2 := \lambda x^{\mathsf{R}}.\pi_2\langle x, x\rangle$ are traces corresponding to the "then" and "else" branch, respectively, of ReLU defined in (1).

The variable rule of Fig. 4 also deserves an explanation. Its non-trivial shape, which is due to higher order types, may be understood as follows. Let $T := \lambda f^{\mathsf{R}\to\mathsf{R}}.f(f0 + 1)$ be a term using its (higher order) argument twice and consider the program $T$ ReLU. Recall that ReLU contains a conditional controlled by its argument, and has two different traces $u_1$ and $u_2$ discussed above. However, the execution of $T$ ReLU, as sketched in Fig. 5, explores both branches of ReLU, so we allow to trace $T$ with $t := \lambda p.\pi_2 p((\pi_1 p)0 + 1)$ and $T$ ReLU with $t\langle u_1, u_2\rangle$. That is, we allow different instances of the same variable $f$ to be traced by different components of a tuple variable $p$, because, even if in the original program all occurrences of $f$ are replaced by copies of the same term ReLU, different copies of ReLU might be traced by different simple terms, so the occurrences of $f$ must be "separated" accordingly.

Lemma 18. *Let $M$ and $t$ be normal forms of type $\mathbf{D}_n(\mathsf{R})$ whose free variables have type belonging to $\{\mathbf{D}_n(\mathsf{R}), \mathsf{R}, \mathsf{R}^n, \mathsf{R}^{\perp_n}\}$. If $t \sqsubset M$ then $t = M$.*

Lemma 19. *If $\Xi \vdash t \sqsubset M$, then:*

(1) *$\mathbf{D}(\Xi) \vdash \mathbf{D}(t) \sqsubset \mathbf{D}(M)$, where $\mathbf{D}$ turns any assignment $p^{A'} \sqsubset x^A$ of $\Xi$ into $p^{\mathbf{D}(A')} \sqsubset x^{\mathbf{D}(A)}$.*
(2) *Let $\Xi = \Xi', x^A \sqsubset x^A$. For every closed simple term $u$ of type $A$, we have $\Xi' \vdash t\{u/x\} \sqsubset M\{u/x\}$.*

Lemma 20. *We have that $\Xi \vdash w \sqsubset M\{N/x\}$ is equivalent to*

- *$w = t\{u_1/x_1\} \dots \{u_n/x_n\}$, for some $n \in \mathbb{N}$ and terms $t, u_1, \dots, u_n$,*
- *such that $\Xi, p^{A_1\times\cdots\times A_n} \sqsubset x^A \vdash t\{\pi_1 p/x_1\} \dots \{\pi_n p/x_n\} \sqsubset M$, $p$ not free in $t$,*
- *and $\Xi \vdash u_i \sqsubset N$ for all $1 \le i \le n$.*

*In particular, $\Xi, p^{A_1\times\cdots\times A_n} \sqsubset x^A \vdash w \sqsubset M$ implies $w = t\{\pi_1 p/x_1\} \dots \{\pi_n p/x_n\}$ for some $t$ not containing $p$ free.*

Definition 21 (Tracing rewriting steps). *Let $\sigma : R \to P$ be a rewriting step of Fig. 1c, and $\upsilon$ be a reduction sequence between simple terms. We define $\upsilon \sqsubset \sigma$ depending on $R$.*

- *If $R = (\lambda x.M)N$: we ask that $\upsilon$ is any reduction of the form*

$$(\lambda p.t\{\boldsymbol{\pi} p/\mathbf{x}\})\langle \mathbf{u}\rangle \to t\{\boldsymbol{\pi}\langle \mathbf{u}\rangle/\mathbf{x}\} \to^* t\{\mathbf{u}/\mathbf{x}\}$$

  *where $p \sqsubset x \vdash t\{\boldsymbol{\pi} p/\mathbf{x}\} \sqsubset M$, with $\{\boldsymbol{\pi} p/\mathbf{x}\}$ denoting the substitutions $\{\pi_1 p/x_1\} \dots \{\pi_n p/x_n\}$ for some $n \in \mathbb{N}$, and where $\mathbf{u}$ is a sequence $u_1, \dots, u_n$ of simple terms such that $u_i \sqsubset N$ for all $i$, and in which the redexes $\pi_i\langle \mathbf{u}\rangle \to u_i$ are reduced in any order.*
- *If $R = \pi_i\langle M_1, \dots, M_k\rangle$: we ask that $\upsilon$ is any reduction of the form $\pi_i\langle t_1, \dots, t_k\rangle \to t_i$ for $t_1 \sqsubset M_1, \dots, t_k \sqsubset M_k$.*

$$\begin{array}{ccccccc} T\,\mathsf{ReLU} & & \longrightarrow & & \mathsf{ReLU}(\mathsf{ReLU}\,0+1) & \longrightarrow & \mathsf{if}(\mathsf{ReLU}\,0+1,0,\mathsf{ReLU}\,0+1) \\ \sqcup & & & & \sqcup & & \sqcup \\ t\langle u_1,u_2\rangle & \longrightarrow & \pi_2\langle u_1,u_2\rangle\,(\pi_1\langle u_1,u_2\rangle\,0+1) & \overset{*}{\longrightarrow} & u_2(u_1 0+1) & \longrightarrow & \pi_2\langle u_1 0+1,u_1 0+1\rangle \end{array}$$

$$\begin{array}{ccccccccccc} \mathsf{if}(\mathsf{ReLU}\,0+1,0,\mathsf{ReLU}\,0+1) & \overset{*}{\longrightarrow} & \mathsf{if}(1,0,\mathsf{ReLU}\,0+1) & \longrightarrow & \mathsf{ReLU}\,0+1 & \longrightarrow & \mathsf{if}(0,0,0)+1 & \to & 0+1 & \to & 1 \\ \sqcup & & \sqcup & & \sqcup & & \sqcup & & \sqcup & & \sqcup \\ \pi_2\langle u_1 0+1,u_1 0+1\rangle & = & \pi_2\langle u_1 0+1,u_1 0+1\rangle & \longrightarrow & u_1 0+1 & \longrightarrow & \pi_1\langle 0,0\rangle+1 & \to & 0+1 & \to & 1 \end{array}$$

Fig. 5. Tracing the head reduction $T\,\mathsf{ReLU} \to^* 1$. The term $T$ is $\lambda f.f(f0+1)$, of which $t := \lambda p.\pi_2 p((\pi_1 p)0+1)$ is a trace; ReLU is given in (1), with traces $u_1 := \lambda x.\pi_1\langle 0,0\rangle$, $u_2 := \lambda x.\pi_2\langle x,x\rangle$.

- *If $R = \phi(\mathbf{r})$: we ask that $v = \sigma$.*
- *If $R = \mathsf{if}(r, M_1, M_2)$: we ask that $v$ is any reduction of the form $\pi_i\langle t,t\rangle \to t$ with $t \sqsubset M_i$ and $i = 1$ if $r \le 0$, otherwise $i = 2$.*
- *If $R = \mathsf{fix} f.N$: we ask that $v \sqsubset \sigma'$ where $\sigma'$ is any reduction of the form*

$$(\lambda f.M)(\lambda x.(\mathsf{fix}_n f.M)x) \to M\{\lambda x.(\mathsf{fix}_n f.M)x/f\}$$

*for some $n \in \mathbb{N}$, as defined in the first case.*

Lemma 22 (pullback). *Let $\sigma : R \to P$ be a rewriting step. For any $w \sqsubset P$, there exist $t \sqsubset R$ and $\xi : t \to^* w$ such that $\xi \sqsubset \sigma$.*

We now extend Definition 21 to one-step head reductions (as defined in Sect. 2.1).

Definition 23 (Tracing head reduction steps). *Let $\sigma = (\mathsf{H}, R, P)$ be a reduction step with $\mathsf{H}$ a head context, and let $\sigma_0$ denote the rewriting step $R \to P$. If $\xi : t \to^* u$ is a reduction sequence on simple terms, we write $\xi \sqsubset \sigma$ whenever one of the following holds:*

- *there exists a simple context $\mathsf{h} \sqsubset \mathsf{H}$ such that $\xi = \mathsf{h}\{v\}$ with $v \sqsubset \sigma_0$ in the sense of Definition 21;*
- *the hole of $\mathsf{H}$ is in the guard of a conditional, $\xi$ is the empty sequence and $t = u \sqsubset \mathsf{H}\{R\}$.*

Finally, we extend the relation $\sqsubset$ to reduction sequences by reflexive-transitive closure.

Definition 24 (Tracing head reduction sequences). *Let $\rho$ be a head reduction sequence starting from a term $M$ and let $\xi$ be a reduction sequence starting from a simple term $t$. We write $\xi \sqsubset \rho$ if either $\rho$ and $\xi$ are empty and $t \sqsubset M$, or if $\rho = \sigma_1 \cdots \sigma_n$ is of length $n > 0$ and $\xi = \xi_1 \cdots \xi_n$ such that $\xi_i \sqsubset \sigma_i$ for every $1 \le i \le n$, according to Definition 23.*

Fig. 5 gives an example of tracing the head reduction of the term $T\,\mathsf{ReLU}$ discussed above. Notice that the reduct of $t\langle u_1,u_2\rangle$ is an intermediate term not corresponding to any trace. Notice also that all the reductions in the guard of a conditional (such as the first steps in the third line of Fig. 5) share the same traces. In general, $\xi \sqsubset \rho$ implies that $\rho$ is a head reduction but not necessary $\xi$, because the first case of Definition 21 requires $\xi$ to reduce projection redexes not in the "head position" of a simple term. In fact, the reduction of $t\langle u_1,u_2\rangle$ in Fig. 5 is not head.

Definition 25 (trace relation). *Let $M$ be a term and $t$ a simple term. We say that $t$* traces *$M$, in symbols $t \sqsubseteq\!\!\!\sim\, M$, whenever there exists a normalizing reduction $\xi$ starting from $t$ and a normalizing head reduction $\rho$ starting from $M$ such that $\xi \sqsubset \rho$.*

As an example, let us consider the term SillyId of (1). Let us define the simple terms:

$$t_1 := \lambda x.\pi_1\langle \pi_1\langle 0,0\rangle, \pi_1\langle 0,0\rangle\rangle, \qquad t_2 := \lambda x.\pi_1\langle \pi_2\langle x,x\rangle, \pi_2\langle x,x\rangle\rangle, \qquad t_3 := \lambda x.\pi_2\langle x,x\rangle. \tag{13}$$

$$\frac{}{\{\cdot\} \lhd \{\cdot\}} \qquad \frac{}{x^A \lhd x^{\mathbf{D}(A)}} \qquad \frac{M \lhd M'}{\lambda x.M \lhd \lambda x.M'} \qquad \frac{M \lhd M' \quad N \lhd N'}{MN \lhd M'N'} \qquad \frac{M \lhd M'}{\pi_i^k M \lhd \pi_i^k M'}$$

$$\frac{M_1 \lhd M_1' \quad \ldots, \quad M_k \lhd M_k'}{\langle M_1, \ldots, M_k \rangle \lhd \langle M_1', \ldots, M_k' \rangle} \qquad \frac{P \lhd P' \quad M \lhd M' \quad N \lhd N'}{\mathsf{if}(P, M, N) \lhd \mathsf{if}(\pi_1 P', M', N')} \qquad \frac{M \lhd M'}{\mathsf{fix} f.M \lhd \mathsf{fix} f.M'}$$

$$\frac{z_1^{\mathbf{D}(\mathrm{R})} \ldots z_k^{\mathbf{D}(\mathrm{R})} \vdash t : \mathrm{R}^{(\perp_n)} \text{ simple normal form}, \quad M_1 \lhd M_1', \quad \ldots, \quad M_k \lhd M_k'}{\phi(M_1, \ldots, M_k) \lhd (\lambda z_1^{\mathbf{D}(\mathrm{R})} \ldots \lambda z_k^{\mathbf{D}(\mathrm{R})}. \langle \phi(\pi_1 z_1, \ldots, \pi_1 z_k), t \rangle) M_1' \cdots M_k'}$$

Fig. 6. The expansion relation. In the rule on the third line, if $k = 0$ then the right-hand term in the conclusion is just $\langle \phi, t \rangle$.

Notice that, for any $i \in \{1, 2, 3\}$, we have $t_i \sqsubset \mathsf{SillyId}$ (see Fig. 4), therefore also $t_i r \sqsubset \mathsf{SillyId}\, r$ for any real number $r$. By contrast, we have that $t_1 r \sqsubsetneq \mathsf{SillyId}\, r$ iff $r = 0$, $t_2 r \sqsubsetneq \mathsf{SillyId}\, r$ iff $r < 0$ and, symmetrically, $t_3 r \sqsubsetneq \mathsf{SillyId}\, r$ iff $r > 0$. This highlights a sharp difference between the relations $\sqsubset$ and $\sqsubsetneq$: the former is static whereas the latter traces the execution of a term. Indeed, the projections of the $t_i$'s reflect the different choices in the conditionals of SillyId.

### 3.2 AD Is Sound on Stable Points

Consider a $\mathsf{PCF}_{\mathrm{R}}$ program $x_1^{\mathrm{R}}, \ldots, x_n^{\mathrm{R}} \vdash M : \mathrm{R}$. One can easily check that for every $\mathbf{r} \in \mathbb{R}^n$, $M\{\mathbf{r}/\mathbf{x}\}$ is normalizing if and only if there exists a simple term $t$ tracing $M\{\mathbf{r}/\mathbf{x}\}$. However, this term $t$ usually depends on the chosen $\mathbf{r}$. In the following definition of stable points, we consider a situation where $t$ can be "uniformly" chosen in an open ball around $\mathbf{r}$.

Definition 26. *We define the set of* stable points *of a program* $x_1^{\mathrm{R}}, \ldots, x_n^{\mathrm{R}} \vdash M : \mathrm{R}$ *as follows:*

$$\mathrm{S}(M) := \left\{ \begin{array}{c} \mathbf{r} \in \mathbb{R}^n \mid \exists\, \varepsilon > 0, \exists\, x_1^{\mathrm{R}}, \ldots, x_n^{\mathrm{R}} \vdash t : \mathrm{R} \text{ s.t. } t \sqsubset M \text{ and} \\ \forall\, \mathbf{r}' \in B_\varepsilon(\mathbf{r})\ t\{\mathbf{r}'/\mathbf{x}\} \sqsubsetneq M\{\mathbf{r}'/\mathbf{x}\} \end{array} \right\}.$$

Notice that we have restricted the tracing of $M$ to terms $t$ of the same type as $M$, in particular $t$ does not split different occurrences of a free variable $x_i$ of $M$. In fact, these variables are supposed to be replaced with numerals, which are not split by $\sqsubset$. Also observe that, by definition, we have $\mathrm{S}(M) \subseteq \Downarrow M$, as $t\{\mathbf{r}/\mathbf{x}\} \sqsubsetneq M\{\mathbf{r}/\mathbf{x}\}$ implies that $M\{\mathbf{r}/\mathbf{x}\}$ has a normal form. Moreover, $\mathrm{S}(M)$ is open: it is easy to check that $\mathrm{S}(M) = \bigcup_{\mathbf{r} \in \mathrm{S}(M)} B_{\varepsilon_{\mathbf{r}}}(\mathbf{r})$, where $\varepsilon_{\mathbf{r}}$ is the positive real whose existence is given by the very definition of stability of $\mathbf{r}$.

We already argued in the Introduction that $\mathrm{S}(\mathsf{ReLU}\, x^{\mathrm{R}}) = \mathbb{R} \setminus \{0\}$. By recalling the above discussion about the simple terms $t_1$, $t_2$ and $t_3$ in (13), we may infer that also $\mathrm{S}(\mathsf{SillyId}\, x^{\mathrm{R}}) = \mathbb{R} \setminus \{0\}$, in fact 0 is the border where one has to swap between $t_1 r$ and either $t_2 r$ or $t_3 r$ in tracing $\mathsf{SillyId}\, r$. Similarly, but with more involved simple terms, one can check that $\mathrm{S}(\mathsf{Floor}\, x^{\mathrm{R}}) = \mathbb{R} \setminus \mathbb{Z}$ with Floor given in (1). As a last example, let us consider the term EqProj defined in (12) and the simple terms

$$t_1 := \lambda x.\lambda y.\pi_1 \langle \pi_1 \langle x, x \rangle, \pi_1 \langle x, x \rangle \rangle, \quad t_2 := \lambda x.\lambda y.\pi_1 \langle \pi_2 \langle y, y \rangle, \pi_2 \langle y, y \rangle \rangle, \quad t_3 := \lambda x.\lambda y.\pi_2 \langle y, y \rangle.$$

These are all such that $t_i \sqsubset \mathsf{EqProj}$. However, $t_1 rq \sqsubsetneq \mathsf{EqProj}\, rq$ iff $r = q$, $t_2 rq \sqsubsetneq \mathsf{EqProj}\, rq$ iff $r < q$ and $t_3 rq \sqsubsetneq \mathsf{EqProj}\, rq$ iff $r > q$. So the diagonal splits the plane $\mathbb{R}^2$ in two open sets where either $t_2$ or $t_3$ uniformly traces the execution of EqProj, whereas the execution on the diagonal is traced by $t_1$. But the diagonal contains no open set, therefore $\mathrm{S}(\mathsf{EqProj}\, x^{\mathrm{R}} y^{\mathrm{R}}) = \mathbb{R}^2 \setminus \{(r, r)\ ;\ r \in \mathbb{R}\}$. Notice, finally, that $\mathsf{EqProj}\, x^{\mathrm{R}} x^{\mathrm{R}}$ may be traced everywhere by $t_1 x^{\mathrm{R}} x^{\mathrm{R}}$, hence $\mathrm{S}(\mathsf{EqProj}\, x^{\mathrm{R}} x^{\mathrm{R}}) = \mathbb{R}$.

The tracing relation $\underset{\sim}{\sqsubseteq}$ is defined over programs. However, in order to prove soundness (Theorem 33), we must move from a program $M$ to its transformation $\mathbf{D}(M)$, this latter having a more complex type than $M$. The difficulty behind the proof of Theorem 33 is then to deduce from $t \underset{\sim}{\sqsubseteq} M$ a link between $\mathbf{D}(t)$ and $\mathbf{D}(M)$. In order to do that, we define a further relation $\lhd$ (Fig. 6) catching an invariant between $M$ and $\mathbf{D}(M)$ (as well as between $t$ and $\mathbf{D}(t)$) stable under the evaluation of the two terms. Then the extrusion lemma (Lemma 31 and its iterated version Lemma 32) will use $\lhd$ for deducing the needed link between $\mathbf{D}(t)$ and $\mathbf{D}(M)$ from the hypothesis $t \underset{\sim}{\sqsubseteq} M$.

Definition 27 (expansion). *The relation $\lhd$ on terms and contexts of $\mathsf{PCF}_{\mathsf{R}}$ is defined in Fig. 6.*

As mentioned above, expansion is used to establish that $M$ and $\mathbf{D}(M)$ "behave similarly". Such a link emerges from two of the main properties of $\lhd$, namely that $M \lhd \mathbf{D}(M)$ holds for every $M$ (Lemma 28), and that it is a simulation (if $M \lhd M'$, then $M'$ simulates $M$, Lemma 29). These justify the definition in the case $M = \phi(M_1, \dots, M_k)$: it mimics the definition of $\mathbf{D}(M)$ in the first component of the product, whereas the second component is chosen as an arbitrary closed simple normal form. The first gives us stability under reduction, in particular in the case of a conditional redex, while the second component cannot be asked to be linked with the partial derivatives of $\phi$, because $\phi(M_1, \dots, M_k)$ eventually reduces to a numeral, which has zero derivative.

Lemma 28. *For every program $x_1^{\mathsf{R}}, \dots, x_n^{\mathsf{R}} \vdash M : \mathsf{R}$, $\mathbf{r} \in \mathbb{R}^n$ and sequence $\mathbf{u} = u_1, \dots, u_n$ of simple closed normal forms of suitable type, we have $M\{\mathbf{r}/\mathbf{x}\} \lhd \mathbf{D}(M)\{\langle \mathbf{r}, \mathbf{u}\rangle/\mathbf{x}\}$, where by $\{\langle \mathbf{r}, \mathbf{u}\rangle/\mathbf{x}\}$ we mean $\{\langle r_1, u_1\rangle/x_1\} \cdots \{\langle r_n, u_n\rangle/x_n\}$.*

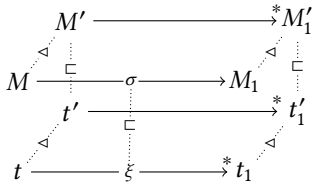
Lemma 29. *Let $M \lhd M'$ and $M \to N$, then there exists $N'$ such that $M' \to^* N'$ and $N \lhd N'$.*

Proof Sketch. Let $(\mathsf{C}, R, P)$ be the reduction step $M \to N$. The proof is an induction on C. The only non-trivial part is the base of the induction, *i.e.* $\mathsf{C} = \{\cdot\}$, in which the reasoning splits following Fig. 1c: the case of a $\beta$-reduction is a consequence of a substitution lemma on $\lhd$. If $R = \phi(\mathbf{r})$, then $M' \to^* \langle \phi(\pi_1 \mathbf{L}'), t\{\mathbf{L}'/\mathbf{z}\}\rangle$ for some $\mathbf{L}'$ of the same length as $\mathbf{r}$ such that $r_i \lhd L_i'$ for all $i$. Notice that $r_i \lhd L_i'$ implies that $\pi_1 L_i' \to r_i$ as well as $L_i'$ is a simple closed normal form, so in particular $t\{\mathbf{L}'/\mathbf{z}\}$ is normalizable by Proposition 5. We therefore have: $\langle \phi(\pi_1 \mathbf{L}'), t\{\mathbf{L}'/\mathbf{z}\}\rangle \to^* \left\langle [\![\phi]\!](\mathbf{r}), t'\right\rangle$ with $t'$ a simple closed normal form, and we conclude by taking $N' := \left\langle [\![\phi]\!](\mathbf{r}), t'\right\rangle$.

The cases of the other redexes (conditional, products and fixpoints) are immediate. □

Remark 30. *A variant of Lemma 29 can be used to prove that $\Downarrow^{\mathcal{S}} M \subseteq \Downarrow^{\mathcal{S}} grad(M)$, for some reduction strategy $\mathcal{S}$ (see discussion after Proposition 10). For example, if $\mathcal{S}$ is the call-by-value strategy, then one can prove the statement of Lemma 29 by replacing $\to$ and $\to^*$ with $\xrightarrow{\mathcal{S}}$ and $\xrightarrow{\mathcal{S}}{}^*$ (in the case of a $R = \phi(\mathbf{r})$ redex, one should notice that the terms $\mathbf{L}'$ are all values, and one should replace "normalizable" with "$\mathcal{S}$-normalizable" and "simple closed normal form with "simple closed $\mathcal{S}$-nf"). Then, consider a program $M$ of arity 1 and suppose $\mathbf{r} \in \Downarrow^{\mathcal{S}} M$, i.e. $M\{\mathbf{r}/\mathbf{x}\} \xrightarrow{\mathcal{S}}{}^* q$, for $q$ a numeral. We have by Lemma 28 and this variant of Lemma 29 that $\mathbf{D}(M)\{\langle \mathbf{r}, \mathbf{u}\rangle/\mathbf{x}\} \xrightarrow{\mathcal{S}}{}^* N$ with $q \lhd N$ and each $u_i$ either the normal form of $\iota_i^n 1$ or $\iota_i^n$, depending whether $\mathbf{D}$ is $\overrightarrow{\mathbf{D}}$ or $\overleftarrow{\mathbf{D}}$ (recall Equations (9) and (10)). By inspecting Fig. 6, we can deduce $N = \langle q, t\rangle$ for some closed simple $\mathcal{S}$-nf $t$. This means $\overleftarrow{grad}(M) \xrightarrow{\mathcal{S}}{}^* (\pi_2 \langle q, t\rangle)1 \xrightarrow{\mathcal{S}} t1$, this latter evaluating to a normal form because it is a simple term (Proposition 5). We conclude $\mathbf{r} \in \Downarrow^{\mathcal{S}} \overleftarrow{grad}(M)$. The case of $\overrightarrow{grad}(M)$ is simpler.*

Lemma 31 (extrusion). *Let $M \lhd M'$, $t \lhd t'$, $t' \sqsubset M'$. Let $\sigma : M \to M_1$ be a head reduction step and moreover let $\xi : t \to^* t_1$ be such that $\xi \sqsubset \sigma$ (so in particular $t \sqsubset M$ and $t_1 \sqsubset M_1$). Then there exist $M_1', t_1'$ such that the following relations hold:*




Proof Sketch. By Definition 23, $\sigma = \mathsf{H}\{\sigma_0\}$ for some head context $\mathsf{H}$ and reduction step $\sigma_0 : R \longrightarrow P$. The proof is by induction on $\mathsf{H}$.

The case $\mathsf{H} = \{\cdot\}$ splits following Fig. 1c. For example, let $\sigma$ be $M = (\lambda x.L)N \longrightarrow L\{N/x\} = M_1$, so that $\xi$ is the reduction $t = (\lambda p.w\{\boldsymbol{\pi} p/\mathbf{x}\})\,\langle \mathbf{u}\rangle \longrightarrow^* w\{\mathbf{u}/\mathbf{x}\} = t_1$. Then $M' = (\lambda x.L')N'$ with $L \lhd L'$ and $N \lhd N'$, and $t' = (\lambda p.\overline{w}')\,\langle \mathbf{u}'\rangle$ with $w\{\boldsymbol{\pi} p/\mathbf{x}\} \lhd \overline{w}'$ and $\langle \mathbf{u}\rangle \lhd \langle \mathbf{u}'\rangle$. Moreover, since $t' \sqsubset M'$, by Lemma 20, we have $\overline{w}' = w'\{\boldsymbol{\pi} p/\mathbf{x}\}$, with $w'\{\boldsymbol{\pi} p/\mathbf{x}\} \sqsubset L'$, $p$ not free in $w'$, and $u_i' \sqsubset N'$. Moreover, by induction on $w$, one can infer from $w\{\boldsymbol{\pi} p/\mathbf{x}\} \lhd \overline{w}'$ that actually $w \lhd w'$.

Of the induction cases, the only subtle one is $\mathsf{H} = \mathsf{if}(\overline{\mathsf{H}}, N_1, N_2)$. Under this hypothesis, the reduction $\xi$ is empty and $t_1 = t = \pi_i\langle u, u\rangle$ with $u \sqsubset N_i$ for some $i \in \{1, 2\}$, as well as $M' = \mathsf{if}(\pi_1\overline{M}', N_1', N_2')$ with $\overline{\mathsf{H}}\{R\} \lhd \overline{M}'$, $N_1 \lhd N_1'$, $N_2 \lhd N_2'$ and $t' = \pi_i\langle u', u'\rangle$ with $u \lhd u'$ and $u' \sqsubset N_i'$ (notice that the index $i$ of the projection is the same in $t$ and $t'$ because $t \lhd t'$). By Lemma 29, $\overline{M}' \longrightarrow^* L$ such that $\overline{\mathsf{H}}\{P\} \lhd L$. We can then conclude by setting $M_1' = \mathsf{if}(\pi_1 L, N_1', N_2')$ and $t_1' = t'$.

All of the remaining cases follow the same pattern. □

Lemma 32 (extrusion to normal form). *Let $M \lhd M'$, $t \lhd t'$, $t \precsim M$ and $t' \sqsubset M'$, for $t$ and $M$ closed terms both of type $\mathsf{R}^n$, for some $n \geq 0$. Then, $t' \longrightarrow^* t''$ and $M' \longrightarrow^* M''$ with $t''$ and $M''$ normal such that $t'' \sqsubset M''$.*

Proof Sketch. By Definition 25, there exist a normalizing reduction $\xi$ starting from $t$ and a normalizing reduction $\rho$ starting from $M$, such that $\xi \sqsubset \rho$. The proof is by induction on $\rho$. □

Theorem 33 (Soundness). *For every program $\Gamma \vdash M : \mathsf{R}$ and every $\mathbf{r} \in \mathrm{S}(M) \cap \mathrm{d}(M)$, we have:*

$$grad(M)(\mathbf{r}) \longrightarrow^* \nabla(\llbracket M \rrbracket_\Gamma)(\mathbf{r}).$$

Proof. The assumption $\mathbf{r} \in \mathrm{d}(M)$ tells us that there is an open ball $B_{\varepsilon_0}(\mathbf{r})$ where $\nabla\llbracket M \rrbracket_\Gamma$ exists. By Definition 26, $\mathbf{r} \in \mathrm{S}(M)$ means that there is a simple program $t \sqsubset M$ uniformly tracing $M$ in an open ball of $\mathbf{r}$, *i.e.*, there exists $\varepsilon > 0$ such that for all $\mathbf{r}' \in B_\varepsilon(\mathbf{r})$ we have $t\{\mathbf{r}'/\mathbf{x}\} \precsim M\{\mathbf{r}'/\mathbf{x}\}$, and of course we may take $\varepsilon \leq \varepsilon_0$. This implies in particular that $\llbracket t \rrbracket_\Gamma$ and $\llbracket M \rrbracket_\Gamma$ coincide on $B_\varepsilon(\mathbf{r})$ and, therefore, we have $\mathbf{r} \in \mathrm{d}(t)$ as well.

By Lemma 28, $t\{\mathbf{r}/\mathbf{x}\} \lhd \mathbf{D}(t)\{\langle \mathbf{r}, \mathbf{u}\rangle/\mathbf{x}\}$ and $M\{\mathbf{r}/\mathbf{x}\} \lhd \mathbf{D}(M)\{\langle \mathbf{r}, \mathbf{u}\rangle/\mathbf{x}\}$, where $u_i$ is either the normal form of $\iota_i^n 1$ or $\iota_i^n$, depending whether $\mathbf{D}$ is $\overrightarrow{\mathbf{D}}$ or $\overleftarrow{\mathbf{D}}$ (recall Equations (9) and (10)). Moreover, by Lemma 19 we have that $t \sqsubset M$ gives $\mathbf{D}(t)\{\langle \mathbf{r}, \mathbf{u}\rangle/\mathbf{x}\} \sqsubset \mathbf{D}(M)\{\langle \mathbf{r}, \mathbf{u}\rangle/\mathbf{x}\}$. We are then in position of applying Lemma 32 to $M\{\mathbf{r}/\mathbf{x}\}$, $t\{\mathbf{r}/\mathbf{x}\}$ (closed terms of ground type) and $\mathbf{D}(t)\{\langle \mathbf{r}, \mathbf{u}\rangle/\mathbf{x}\}$, $\mathbf{D}(M)\{\langle \mathbf{r}, \mathbf{u}\rangle/\mathbf{x}\}$. This gives us a normal form $t'$ of $\mathbf{D}(t)\{\langle \mathbf{r}, \mathbf{u}\rangle/\mathbf{x}\}$ and $M'$ of $\mathbf{D}(M)\{\langle \mathbf{r}, \mathbf{u}\rangle/\mathbf{x}\}$ such that $t' \sqsubset M'$. By subject reduction (Proposition 4), $t', M'$ are closed normal forms of type $\mathbf{D}(\mathsf{R})$, so Lemma 18 gives us $t' = M'$. Since $\mathbf{r} \in \mathrm{d}(t)$, we conclude by Proposition 9. □

## 4 UNSOUNDNESS OF AD

Definition 34 (unstable point). *The set of* unstable points *of a program $M$, denoted by $\mathrm{U}(M)$, is the complement of $\mathrm{S}(M)$ (Definition 26) in $\Downarrow\!M$, i.e., $\mathrm{U}(M) := \Downarrow\!M \setminus \mathrm{S}(M)$.*

We know from the remark after Definition 26 that $\mathrm{U}(M)$ is closed in $\Downarrow\!M$ (with respect to to the subspace topology). The goal of this section is to prove that it is a quasivariety, hence of

$$\begin{aligned}
\mathrm{P}_\Gamma(\mathsf{R}) &:= \{\Gamma \vdash M : \mathsf{R} \mid [\![M]\!]_\Gamma \text{ is cqc and } \mathrm{U}(M) \text{ is a quasivariety}\}\\
\mathrm{P}_\Gamma(A \to B) &:= \{\Gamma \vdash M : A \to B \mid \forall N \in \mathrm{P}_\Gamma(A), MN \in \mathrm{P}_\Gamma(B)\}\\
\mathrm{P}_\Gamma(A_1 \times \cdots \times A_k) &:= \{\Gamma \vdash M : A_1 \times \cdots \times A_k \mid \pi_i M \in \mathrm{P}_\Gamma(A_i), \forall i \le k\}
\end{aligned}$$

Fig. 7. The definition of the logical predicate $\mathrm{P}_\Gamma(A)$, with $\Gamma$ a ground context.

measure zero. The main tool is the logical predicate defined in Fig. 7 and its adequacy (Lemma 41). The structure of the proof is standard, but some new notions are needed. First, our programs are first-order functions, *i.e.*, terms with some free variables of ground type, so the logical predicate is indexed by a typing environment. Second, Lemma 35 states some properties of the notion of stability necessary to achieve the standard auxiliary lemmas of a logical predicate, such as closure under expansion (Lemma 37) or (a syntactic variant of) Scott-continuity (Lemma 39 and Lemma 40). Third, and more important, the standard lemma of logical predicates for the conditional (Lemma 38) is particularly subtle. This should not come as a surprise: as discussed above, the possibility of unsoundness of AD is due to conditionals. In particular, let us underline that Lemma 38 uses the notion of completely quasicontinuous map introduced in Sect. 2.3.

The logical predicate on which the proof is based is defined in Fig. 7. In the rest of the section, unless otherwise stated, $\Gamma := x_1^{\mathsf{R}}, \dots, x_n^{\mathsf{R}}$ is a ground context. Moreover, if $M : A_1 \to \cdots \to A_p \to B$ and $\mathbf{L} = L_1, \dots, L_p$ such that $L_i : A_i$ for all $1 \le i \le p$, then the notation $M\mathbf{L}$ stands for $ML_1 \cdots L_p$, which is of course a term of type $B$.

Lemma 35. *We have the following inclusions, where the terms appearing in the statements are supposed to be typed under a ground context* $\Gamma$.

(1) *Let* $\phi$ *be a function symbol of arity* $k$ *and let* $M_1, \dots, M_k$ *be programs, then* $\bigcap_i \mathrm{S}(M_i) \subseteq \mathrm{S}(\phi(M_1, \dots, M_k))$.
(2) *Let* $R \to P$ *be one of the rewriting rules in Fig. 1c, with* $R, P$ *of type* $B_1 \to \cdots \to B_p \to \mathsf{R}^m$. *For all* $1 \le i \le p$, *let* $\Gamma \vdash L_i : B_i$. *Then, for all* $1 \le j \le m$, $\mathrm{S}(\pi_j(P\mathbf{L})) \subseteq \mathrm{S}(\pi_j(R\mathbf{L}))$.
(3) *Let* $P : \mathsf{R}$ *and* $M_1, M_2$ *be of type* $B_1 \to \cdots \to B_p \to \mathsf{R}^m$. *For all* $1 \le i \le p$, *let* $\Gamma \vdash L_i : B_i$. *Let* $X_1 := [\![P]\!]_\Gamma^{-1}(\mathbb{R}_{\le 0})$ *and* $X_2 := [\![P]\!]_\Gamma^{-1}(\mathbb{R}_{>0})$. *Then, for all* $1 \le j \le m$ *and all* $l \in \{1, 2\}$, *we have that* $\mathrm{S}(\pi_j(M_l\mathbf{L})) \cap \mathrm{int}(X_l) \subseteq \mathrm{S}(\pi_j(\mathrm{if}(P, M_1, M_2)\mathbf{L}))$.
(4) *Let* $B = B_1 \to \cdots \to B_p \to \mathsf{R}^m$, *let* $\Gamma \vdash L_0 : A$ *and* $\Gamma \vdash L_i : B_i$ *for all* $1 \le i \le p$. *For all* $k \in \mathbb{N}$ *and* $1 \le j \le m$, $\mathrm{S}(\pi_j((\mathsf{fix}_k f^{A\to B}.M)\mathbf{L})) \subseteq \mathrm{S}(\pi_j((\mathsf{fix} f^{A\to B}.M)\mathbf{L}))$, *where* $\mathbf{L} := L_0, L_1, \dots, L_p$.

Proof Sketch. We only detail the proof of the branching case, the other cases are easy variants.

By taking the notations of point (3), let $l \in \{1, 2\}$ and let $\mathbf{r} \in \mathrm{S}(\pi_j(M_l\mathbf{L})) \cap \mathrm{int}(X_l)$. By definition, there exist $\varepsilon > 0$, $u \sqsubset \pi_j(M_l\mathbf{L})$ such that, for all $\mathbf{r}' \in B_\varepsilon(\mathbf{r})$, $u\{\mathbf{r}'/\mathbf{x}\} \precsim \pi_j(M_l\mathbf{L})\{\mathbf{r}'/\mathbf{x}\}$ and $\mathbf{r}' \in \mathrm{int}(X_l)$. Notice that $u = \pi_j(u'\mathbf{u}'')$, with $u' \sqsubset M_l$ and $\mathbf{u}'' = \langle \mathbf{u}_1'' \rangle \cdots \langle \mathbf{u}_p \rangle$ such that, for every $1 \le i \le p$ and every element $u_{i,h}''$ of $\mathbf{u}_i''$, we have $u_{i,h}'' \sqsubset L_i$. Let $t := \pi_j((\pi_\ell \langle u', u' \rangle)\mathbf{u}'')$ and notice that $t \sqsubset \pi_j(\mathrm{if}(P, M_1, M_2)\mathbf{L})$. Let us prove that $t\{\mathbf{r}'/\mathbf{x}\} \precsim \pi_j(\mathrm{if}(P, M_1, M_2)\mathbf{L})\{\mathbf{r}'/\mathbf{x}\}$.

Since $\mathbf{r}' \in \mathrm{int}(X_l) \subseteq \Downarrow P$, we have that $P\{\mathbf{r}'/\mathbf{x}\}$ is normalizing. Since $P$ is ground, by Proposition 6 there is a head reduction sequence $\rho : P\{\mathbf{r}'/\mathbf{x}\} \to^* q$ such that $q$ is a numeral. Let now $\mathsf{H} = \pi_j(\mathrm{if}(\{\cdot\}, M_1, M_2)\mathbf{u}'')\{\mathbf{r}'/\mathbf{x}\}$. Notice that $\nu \sqsubset \mathsf{H}\{\rho\}$ for $\nu$ the empty reduction sequence of $t\{\mathbf{r}'/\mathbf{x}\}$. Moreover, by hypothesis we have normalizing reduction sequences $\nu' \sqsubset \rho'$ from $u\{\mathbf{r}'/\mathbf{x}\} = \pi_j(u'\mathbf{u}'')\{\mathbf{r}'/\mathbf{x}\}$ and $\pi_j(M_l\mathbf{L})\{\mathbf{r}'/\mathbf{x}\}$, respectively. Furthermore, we have the head reduction steps:

$$\nu_0 : \pi_j(\pi_\ell \langle u', u' \rangle \mathbf{u}'')\{\mathbf{r}'/\mathbf{x}\} \to \pi_j(u'\mathbf{u}'')\{\mathbf{r}'/\mathbf{x}\}, \quad \rho_0 : \pi_j(\mathrm{if}(q, M_1, M_2)\mathbf{L})\{\mathbf{r}'/\mathbf{x}\} \to \pi_j(M_l\mathbf{L})\{\mathbf{r}'/\mathbf{x}\}$$

such that $\nu_0 \sqsubset \rho_0$. We then have $\nu\nu_0\nu' \sqsubset \mathsf{H}\{\rho\}\rho_0\rho'$, which allows us to conclude. □

Lemma 36 (function symbols). *Let $\phi$ be a function symbol of arity $k$ and, for each $1 \leq i \leq k$, $M_i \in \mathrm{P}_\Gamma(\mathrm{R})$. If $[\![\phi]\!]$ is cqc, then $\phi(M_1, \dots, M_k) \in \mathrm{P}_\Gamma(\mathrm{R})$.*

Lemma 37 (closure under expansion). *Let $R \to P$ be one of the rewriting rules in Figure 1c. If $P \in \mathrm{P}_\Gamma(A)$, then $R \in \mathrm{P}_\Gamma(A)$.*

Lemma 38 (conditional). *If $P \in \mathrm{P}_\Gamma(\mathrm{R})$ and $M_1, M_2 \in \mathrm{P}_\Gamma(A)$, then $\mathsf{if}(P, M_1, M_2) \in \mathrm{P}_\Gamma(A)$.*

Proof. Let $A = A_1 \to \cdots \to A_p \to \mathrm{R}^m$. It is enough to prove that, given $\mathbf{L} = L_1, \dots, L_p$ such that $L_i \in \mathrm{P}_\Gamma(A_i)$ for every $1 \leq i \leq p$, we have $\pi_j(\mathsf{if}(P, M_1, M_2)\mathbf{L}) \in \mathrm{P}_\Gamma(\mathrm{R})$ for every $1 \leq j \leq m$.

Let $N := \pi_j(\mathsf{if}(P, M_1, M_2)\mathbf{L})$, $Q_1 := [\![P]\!]_\Gamma^{-1}(\mathbb{R}_{\leq 0})$ and $Q_2 := [\![P]\!]_\Gamma^{-1}(\mathbb{R}_{>0})$. Observe that both $Q_1$ and $Q_2$ are quasiopen, because $\mathbb{R}_{\leq 0}$ and $\mathbb{R}_{>0}$ are quasiopen and $[\![P]\!]_\Gamma$ is cqc by hypothesis. Furthermore, $\mathbb{R}^l \times Q_i$ is quasiopen for every $l \geq 0$ and $i \in \{1, 2\}$, because it is the inverse image of $Q_i$ via a projection $\mathbb{R}^l \times \mathbb{R}^p \to \mathbb{R}^p$, and these are cqc (Lemma 17.3).

To prove that $[\![N]\!]_\Gamma$ is cqc we need to show that, for every $l \geq 0$ and every quasiopen set $Q \subseteq \mathbb{R}^{l+1}$, the set $(\mathrm{id}_{\mathbb{R}^l} \times [\![N]\!]_\Gamma)^{-1}(Q)$ is quasiopen. But Proposition 7 tells us that $(\mathrm{id}_{\mathbb{R}^l} \times [\![N]\!]_\Gamma)^{-1}(Q)$ is equal to $\big((\mathbb{R}^l \times Q_1) \cap (\mathrm{id}_{\mathbb{R}^l} \times [\![\pi_j(M_1\mathbf{L})]\!]_\Gamma)^{-1}(Q)\big) \cup \big((\mathbb{R}^l \times Q_2) \cap (\mathrm{id}_{\mathbb{R}^l} \times [\![\pi_j(M_2\mathbf{L})]\!]_\Gamma)^{-1}(Q)\big)$, which is quasiopen because $[\![\pi_j(M_1\mathbf{L})]\!]_\Gamma$ and $[\![\pi_j(M_1\mathbf{L})]\!]_\Gamma$ are cqc by hypothesis.

For what concerns the unstable points, we first observe that, by Proposition 7, $\Downarrow N = \Downarrow P \cap ((Q_1 \cap \Downarrow M_1) \cup (Q_2 \cap \Downarrow M_2)) = (Q_1 \cap \Downarrow M_1) \cup (Q_2 \cap \Downarrow M_2)$, the second equality holding because $Q_1, Q_2 \subseteq \Downarrow P$. We may therefore write

$$\mathrm{U}(N) = \left( \bigcup_{i \in \{1,2\}} Q_i \cap \Downarrow M_i \right) \setminus \mathrm{S}(N) = \bigcup_{i \in \{1,2\}} (Q_i \cap \Downarrow M_i) \setminus \mathrm{S}(N) \subseteq \bigcup_{i \in \{1,2\}} (Q_i \cap \Downarrow M_i) \setminus (\mathrm{int}(Q_i) \cap \mathrm{S}(M_i))$$

where the inclusion is by Lemma 35.3. Now, for all $i \in \{1, 2\}$, let us write $A_i := (Q_i \cap \Downarrow M_i) \setminus (\mathrm{int}(Q_i) \cap \mathrm{S}(M_i))$. Notice that, by Lemma 15, $Q_i = \mathrm{int}(Q_i) \cup Z_i$ with $Z_i$ a quasivariety, hence

$$A_i = ((Q_i \cap \Downarrow M_i) \setminus \mathrm{int}(Q_i)) \cup ((Q_i \cap \Downarrow M_i) \setminus \mathrm{S}(M_i)) \subseteq (Q_i \setminus \mathrm{int}(Q_i)) \cup (\Downarrow M_i \setminus \mathrm{S}(M_i)) = Z_i \cup \mathrm{U}(M_i),$$

so $A_i$ is a quasivariety, because $Z_i$ and $\mathrm{U}(M_i)$ are. Since $\mathrm{U}(N) \subseteq A_1 \cup A_2$, we are done. □

Lemma 39 (divergence). *For every type $A \to B$, $\Omega_{A \to B} \in \mathrm{P}_\Gamma(A \to B)$.*

Lemma 40 (fixpoints). *If $\forall k \in \mathbb{N}, \mathsf{fix}_k f.M \in \mathrm{P}_\Gamma(A \to B)$, then $\mathsf{fix} f.M \in \mathrm{P}_\Gamma(A \to B)$.*

Lemma 41 (adequacy). *Suppose that the primitive functions of $\mathsf{PCF}_\mathrm{R}$ are admissible. Let $\Gamma := x_1^\mathrm{R}, \dots, x_n^\mathrm{R}$, $\Delta := y_1^{A_1}, \dots, y_m^{A_m}$, let $\Gamma, \Delta \vdash M : A$ and let $\Gamma \vdash N_i \in \mathrm{P}_\Gamma(A_i)$ for all $1 \leq i \leq m$. Then,*

$$M\{N_1/y_1\} \cdots \{N_m/y_m\} \in \mathrm{P}_\Gamma(A).$$

Theorem 42. *Assuming that the primitive functions of $\mathsf{PCF}_\mathrm{R}$ are admissible, for every program $\Gamma \vdash M : \mathrm{R}$ the set*

$$\mathrm{Fail}(M) := \{\mathbf{r} \in \mathrm{d}(M) \ ; \ grad(M)(\mathbf{r}) \not\to^* \nabla([\![M]\!]_\Gamma)(\mathbf{r})\}$$

*is a quasivariety, hence of measure zero.*

Proof. By Theorem 33, we know that $\mathrm{Fail}(M) \subseteq \mathrm{U}(M)$, which is a quasivariety because $M \in \mathrm{P}_\Gamma(\mathrm{R})$ by Lemma 41. □

## 5 DISCUSSION AND PERSPECTIVES

*On the significance of the measure zero bound.* Since the set of real numbers representable on an actual computer is of measure zero in $\mathbb{R}$ (it is finite!), one may feel skeptical about the significance of Theorem 42. This issue was already raised by Speelpenning [Speelpenning 1980] while commenting on Joss's theorem [Joss 1976]. He defines a program similar to the following:

$$\mathsf{SlowId} := \lambda x^{\mathsf{R}}.\left(\mathsf{fix} f^{\mathsf{R}\to\mathsf{R}}.\lambda y^{\mathsf{R}}.\mathsf{if}(x-y, \mathsf{if}(y-x, y, f\ \mathsf{Next}), f\ \mathsf{Next})\right) 0,$$

where $\vdash \mathsf{Next} : \mathsf{R}$ is a primitive which cycles through machine-representable real numbers, based on some internal state (Speelpenning uses a random number generator in his example). So, given $r \in \mathbb{R}$, $\mathsf{SlowId}(r)$ will eventually output $r$ if this is machine-representable, and diverge otherwise. When executed on an actual computer, every $r$ is necessarily representable and SlowId behaves like the identity. And yet, $grad(\mathsf{SlowId}\, x)(r) \to^* 0$ for every machine-representable $r$, because Next is treated as a constant by AD transformations (there is no sensible alternative). Speelpenning concludes that, although SlowId is not a counterexample to Joss's theorem (or to ours), from the practical viewpoint AD fails *everywhere* on it.

We believe that Speelpenning's example is misleading. The reason why SlowId is not a counterexample to Theorem 42 is not that the set where AD fails on SlowId is of measure zero because it coincides with the set of machine-representable reals; it is because $[\![\mathsf{SlowId}(x)]\!]_{x^{\mathsf{R}}}$ is nowhere differentiable! That is, in the notations of Theorem 42, we actually have $\mathrm{Fail}(\mathsf{SlowId}(x)) = \emptyset$ because $\mathrm{d}(\mathsf{SlowId}(x)) = \emptyset$, and this is because $[\![\mathsf{SlowId}(x)]\!]_{x^{\mathsf{R}}}$ is defined only on a discrete set.

Anyway, if the set $R := \{r_1, \ldots, r_c\}$ of machine-representable reals is finite, there are impractically large but straightforward programs achieving the intended behavior of Speelpenning's example. For instance, with some syntactic sugar, define

$$\mathsf{SlowId}' \quad := \quad \lambda x^{\mathsf{R}}.\mathsf{if}\ x = r_1\ \mathsf{then}\ r_1\ \mathsf{else}\ (\ldots \mathsf{if}\ x = r_c\ \mathsf{then}\ r_c\ \mathsf{else}\ x \ldots).$$

We have that $[\![\mathsf{SlowId}'(x)]\!]_{x^{\mathsf{R}}}$ is actually the identity function, so $\mathrm{Fail}(\mathsf{SlowId}'(x)) = R$ and we may legitimately say that AD is wrong "everywhere". But this is just a giant-sized version of the program SillyId of the Introduction, and speaks more of the contrivance of toying with $\mathrm{PCF}_{\mathsf{R}}$ as a machine-executable language (which it is not) than of the value of our result. In general, questioning the significance of Theorem 42 on the grounds that computers are finite is like questioning Turing machines because of their infinite tape, or objecting to the whole idea of studying the asymptotic complexity of programs because in practice we only implement finite functions, whose asymptotic complexity is $O(1)$. In our opinion, there is little point in discussing this standpoint further.

More constructively, we may argue that the significance of Theorem 42 lies in the fact that it gives a finer bound than just measure zero. Let us call $\mathrm{PCF}_{\mathsf{R}}$ with only the "mandatory" primitive functions (constants, addition, multiplication) *minimal* $\mathrm{PCF}_{\mathsf{R}}$. This is already enough to express all differentiable programming architectures based on neural networks with rectified linear unit activation. Moreover, by using Taylor series, minimal $\mathrm{PCF}_{\mathsf{R}}$ may also approximate every analytic function with arbitrary precision, so it has a wide range of potential applications. Theorem 42 tells us that, if $f : \mathbb{R}^n \rightharpoonup \mathbb{R}$ is a function definable in minimal $\mathrm{PCF}_{\mathsf{R}}$, then the set of points on which $\nabla f$ exists but AD methods fail to compute it is contained in a countable union of algebraic varieties (*i.e.*, zeros of polynomials). In particular, when $n = 1$, this set is countable.

Zero sets of polynomial equations have been studied for literally millennia as part of the vast field known as algebraic geometry. Albeit extremely complex in general, many results exist on their structure, which may be described or approximated very accurately in several cases. It is not excluded that, in the future, these results may be leveraged to develop static analysis techniques (*e.g.* type systems) for establishing the absence of errors in differentiable programs.

*From Gradients to Jacobians.* We limited our attention to programs implementing functions $\mathbb{R}^n \multimap \mathbb{R}^m$ with $m = 1$. The case $m > 1$, in which one would speak of Jacobians rather than gradients, is conceptually identical. First of all, observe that a function $f : \mathbb{R}^n \multimap \mathbb{R}^m$ may always be decomposed into $m$ functions $f_i := \pi_i f : \mathbb{R}^n \multimap \mathbb{R}$, so restricting primitives to one output causes no loss of generality and Fig. 2 needs no modification. When $\Gamma \vdash M : \mathsf{R}^m$ with $\Gamma$ containing $n$ variables and $m > 1$, what needs to be modified are the Equations (9) and (10), which must yield an $m \times n$ matrix whose lines are built out of $m$ expressions of the form $grad(\pi_i M)$. Theorem 33 and Theorem 42 lift to this setting because the Jacobian is just the collection of the $m$ gradients.

*Internalizing AD.* The transformations of Fig. 2, and thus the definition of $grad(M)$ for a program $M$ (Equations (9) and (10)) are external to $\mathsf{PCF}_\mathsf{R}$: a programmer may apply them for instance via a compiler, but the transformations are not accessible from within the program itself. For practical purposes, it would be interesting to have a programming language in which $\overrightarrow{grad}$ and $\overleftarrow{grad}$ are syntactic constructs, typed $x_1^\mathsf{R}, \dots, x_n^\mathsf{R} \vdash grad(M) : \mathsf{R}^n$ whenever $x_1^\mathsf{R}, \dots, x_n^\mathsf{R} \vdash M : \mathsf{R}$, and with $grad(M)$ being executed in such a way as to reflect the application of AD to $M$. A naive way of achieving this would be to turn the definition of Fig. 2 into rewriting rules; a more sophisticated approach was provided by Pearlmutter and Siskind for Stalin$\nabla$ [Pearlmutter and Siskind 2008].

The errors introduced by AD have the important consequence that *such an internalization is impossible without breaking the expected extensional semantics of programs.* This is because, as any denotational semantics, the standard semantics defined in Sect. 2.1 is contextual, in the sense that $[\![M]\!] = [\![N]\!]$ implies $[\![\mathsf{C}\{M\}]\!] = [\![\mathsf{C}\{N\}]\!]$ for any context C. Now, referring to (1), we have $[\![\mathsf{SillyId}(x)]\!]_{x^\mathsf{R}} = [\![x]\!]_{x^\mathsf{R}}$ and yet we know that $[\![grad(\mathsf{SillyId}(x))]\!]_{x^\mathsf{R}} \neq [\![grad(x)]\!]_{x^\mathsf{R}}$, because the latter two functions differ at 0. So any denotational semantics of $\mathsf{PCF}_\mathsf{R}$ with "internal AD" needs to interpret SillyId and $\mathsf{Id} := \lambda x^\mathsf{R}.x$ differently. "Resource-sensitive" semantics coming from linear logic do distinguish them, but it is easy to find other examples on which these semantics too fail. An example of denotational semantics which consistently works is the one introduced by Abadi and Plotkin [Abadi and Plotkin 2020], whose first order language does have internal AD. In that semantics, Id is the identity whereas SillyId is a "partial identity", undefined at 0. It is not clear whether this extends to higher order (and thus to $\mathsf{PCF}_\mathsf{R}$, similarly to [Di Gianantonio and Edalat 2013]), but assuming it does, the meaning it gives to programs is somewhat unusual: for example, using the definition given in (1), $\mathsf{Floor}(r)$ would diverge whenever $r$ is an integer. This is a further drawback of partial conditional semantics, in addition to the one pointed out in Sect. 2.2 (concerning example (12)).

Another loosely related remark worth making at this point is that, seen as a functional on the Scott domain $\mathbb{R}_\perp \to \mathbb{R}_\perp$, where $\mathbb{R}_\perp$ is the "flat" Scott domain typically used for interpreting R as a ground type with total conditionals, the derivative operator $\partial$ is *not* Scott continuous.[8] Although the technical consequences of this observation are not entirely clear, from an intuitive point of view it means that in the naive flat semantics the derivative operator is "not computable", and therefore no recursive procedure (like those given by AD transformations) will be error-free.

## ACKNOWLEDGMENTS

We would like to thank A. Brunel, T. Ehrhard and B.A. Pearlmutter for useful comments and discussions. This work was partially supported by ANR PRC project PPS (ANR-19-CE48-0014).

[8] Given $A \subseteq \mathbb{R}$, say that $\chi$ is the *indicator function* of $A$ if $\chi(x) = 0$ when $x \in A$ and $\chi(x) = \perp$ otherwise. For $n > 0$, let $I_n := ]-\infty, 0] \cup ]\frac{1}{n}, +\infty[$ and let $\varphi_n$ be the indicator function of $I_n$. Notice that each $\varphi_n$ is differentiable on $J_n := I_n \setminus \{0\}$ and $\partial\varphi_n$ is the indicator function of $J_n$. As elements of the Scott domain $\mathbb{R}_\perp \to \mathbb{R}_\perp$, the functions $(\varphi_n)_{n>0}$ and $(\partial\varphi_n)_{n>0}$ form two directed chains whose suprema are the identically zero function and the indicator function of $\mathbb{R} \setminus \{0\}$, respectively. In particular, $\partial\big(\sup_{n>0} \varphi_n\big) \neq \sup_{n>0} \partial\varphi_n$, so $\partial$ is not Scott continuous.

## REFERENCES


Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek Gordon Murray, Benoit Steiner, Paul A. Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2016. TensorFlow: A System for Large-Scale Machine Learning. In *Proceedings of OSDI*. USENIX Association, 265–283.

Martín Abadi and Gordon D. Plotkin. 2020. A simple differentiable programming language. *Proc. ACM Program. Lang.* 4, POPL (2020), 38:1–38:28.

Roberto Amadio and Pierre-Louis Curien. 1998. *Domains and Lambda-Calculi*. Vol. 46. Cambridge University Press.

Henk P. Barendregt. 1985. *The Lambda Calculus, Its Syntax and Semantics*. North Holland.

Gilles Barthe, Raphaëlle Crubillé, Ugo Dal Lago, and Francesco Gavazzo. 2020. On the Versatility of Open Logical Relations - Continuity, Automatic Differentiation, and a Containment Theorem. In *Proceedings of ESOP*. 56–83.

Atilim Baydin, Barak Pearlmutter, Alexey Radul, and Jeffrey Siskind. 2018. Automatic differentiation in machine learning: A survey. *Journal of Machine Learning Research* 18 (2018), 1–43.

Thomas Beck and Herbert Fischer. 1994. The if-problem in automatic differentiation. *J. Comput. Appl. Math.* 50, 1 (1994), 119–131.

Aloïs Brunel, Damiano Mazza, and Michele Pagani. 2020. Backpropagation in the simply typed lambda-calculus with linear negation. *PACMPL* 4, POPL (2020), 64:1–64:27.

Pietro Di Gianantonio and Abbas Edalat. 2013. A Language for Differentiable Functions. In *Proceedings of FOSSACS*. 337–352.

Thomas Ehrhard and Laurent Regnier. 2006. Böhm Trees, Krivine's Machine and the Taylor Expansion of Lambda-Terms. In *Proceedings of CiE*. 186–197.

Thomas Ehrhard and Laurent Regnier. 2008. Uniformity and the Taylor expansion of ordinary lambda-terms. *Theor. Comput. Sci.* 403, 2-3 (2008), 347–372.

Conal Elliott. 2018. The simple essence of automatic differentiation. *PACMPL* 2, ICFP (2018), 70:1–70:29.

Martín Hötzel Escardó. 1996. PCF Extended with Real Numbers. *Theor. Comput. Sci.* 162, 1 (1996), 79–115.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press. http://www.deeplearningbook.org

Andreas Griewank and Andrea Walther. 2008. *Evaluating derivatives - principles and techniques of algorithmic differentiation, Second Edition*. SIAM.

Mathieu Huot, Sam Staton, and Matthijs Vákár. 2020. Correctness of Automatic Differentiation via Diffeologies and Categorical Gluing. In *Proceedings of FOSSACS*. 319–338.

Johan Joss. 1976. *Algorthmisches Differenzieren*. Ph.D. Dissertation. ETH Zurich.

Yann LeCun. 2018. Deep Learning est mort. Vive Differentiable Programming! (2018). https://www.facebook.com/yann.lecun/posts/10155003011462143

Wonyeol Lee, Hangyeol Yu, Xavier Rival, and Hongseok Yang. 2020. On Correctness of Automatic Differentiation for Non-Differentiable Functions. *CoRR* abs/2006.06903 (2020).

Carol Mak, C.-H. Luke Ong, Hugo Paquet, and Dominik Wagner. 2020. Densities of almost-surely terminating probabilistic programs are differentiable almost everywhere. *CoRR* abs/2004.03924 (2020).

Damiano Mazza. 2017. *Polyadic Approximations in Logic and Computation. Habilitation* thesis. Université Paris 13.

Damiano Mazza, Luc Pellissier, and Pierre Vial. 2018. Polyadic approximations, fibrations and intersection types. *Proc. ACM Program. Lang.* 2, POPL (2018), 6:1–6:28.

Boris Mityagin. 2015. The Zero Set of a Real Analytic Function. *arXiv:1512.07276 [math.CA]* (2015).

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in PyTorch. (2017).

Barak A. Pearlmutter and Jeffrey Mark Siskind. 2008. Reverse-mode AD in a Functional Framework: Lambda the Ultimate Backpropagator. *ACM Trans. Program. Lang. Syst.* 30, 2 (2008), 7:1–7:36.

Gordon Plotkin. 1977. LCF Considered as a Programming Language. *Theoretical Computer Science* 5, 3 (1977), 223–255.

Amir Shaikhha, Andrew Fitzgibbon, Dimitrios Vytiniotis, and Simon Peyton Jones. 2019. Efficient differentiable programming in a functional array-processing language. *PACMPL* 3, ICFP (2019), 97:1–97:30.

Bert Speelpenning. 1980. *Compiling Fast Partial Derivatives of Functions Given by Algorithms*. Ph.D. Dissertation. University of Illinois at Urbana-Champaign.

Ágnes Szendrei. 1986. *Clones in Universal Algebra*. Presses de l'Université de Montréal.

Fei Wang, Daniel Zheng, James M. Decker, Xilun Wu, Grégory M. Essertel, and Tiark Rompf. 2019. Demystifying differentiable programming: shift/reset the penultimate backpropagator. *PACMPL* 3, ICFP (2019), 96:1–96:31.

Yuan Zhou, Bradley J. Gram-Hansen, Tobias Kohn, Tom Rainforth, Hongseok Yang, and Frank Wood. 2019. LF-PPL: A Low-Level First Order Probabilistic Programming Language for Non-Differentiable Models. In *Proceedings of AISTATS*. 148–157.