Synthesizing Precise Static Analyzers for Automatic Differentiation

JACOB LAUREL, University of Illinois at Urbana-Champaign, USA
SIYUAN BRANT QIAN, University of Illinois at Urbana-Champaign, USA and Zhejiang University, China
GAGANDEEP SINGH, University of Illinois at Urbana-Champaign and VMware Research, USA
SASA MISAILOVIC, University of Illinois at Urbana-Champaign, USA

We present Pasado, a technique for synthesizing precise static analyzers for Automatic Differentiation. Our technique allows one to automatically construct a static analyzer specialized for the Chain Rule, Product Rule, and Quotient Rule computations for Automatic Differentiation in a way that abstracts all of the nonlinear operations of each respective rule simultaneously. By directly synthesizing an abstract transformer for the composite expressions of these 3 most common rules of AD, we are able to obtain significant precision improvement compared to prior works which compose standard abstract transformers together suboptimally. We prove our synthesized static analyzers sound and additionally demonstrate the generality of our approach by instantiating these AD static analyzers with different nonlinear functions, different abstract domains (both intervals and zonotopes) and both forward-mode and reverse-mode AD.

We evaluate Pasado on multiple case studies, namely computing certified bounds on a neural network's local Lipschitz constant, soundly bounding the sensitivities of financial models, certifying monotonicity, and lastly, bounding sensitivities of the solutions of differential equations from climate science and chemistry for verified ranges of initial conditions and parameters. The local Lipschitz constants computed by Pasado on our largest CNN are up to 2750× more precise compared to the existing state-of-the-art zonotope analysis. Additionally, the bounds obtained on the sensitivities of the climate, chemical, and financial differential equation solutions are between 1.31 − 2.81× more precise (on average) compared to a state-of-the-art zonotope analysis.

CCS Concepts: • **Mathematics of computing → Differential calculus**; **Automatic differentiation**; • **Theory of computation → Program analysis**; **Abstraction**.

Additional Key Words and Phrases: Differentiable Programming, Abstract Interpretation

ACM Reference Format:
Jacob Laurel, Siyuan Brant Qian, Gagandeep Singh, and Sasa Misailovic. 2023. Synthesizing Precise Static Analyzers for Automatic Differentiation. *Proc. ACM Program. Lang.* 7, OOPSLA2, Article 291 (October 2023), 29 pages. https://doi.org/10.1145/3622867

1 INTRODUCTION

Automatic Differentiation (AD) has served as the backbone for many applications across Computer Science. Indeed, AD has driven much of the modern deep learning revolution since derivative computations are essential both for training Deep Neural Networks (DNNs) and ensuring trustworthy Machine Learning (ML). In particular, guarantees on derivatives are necessary for establishing

Authors' addresses: Jacob Laurel, jlaurel2@illinois.edu, University of Illinois at Urbana-Champaign, USA; Siyuan Brant Qian, siyuanq4@illinois.edu, University of Illinois at Urbana-Champaign, USA and Zhejiang University, China; Gagandeep Singh, ggnds@illinois.edu, University of Illinois at Urbana-Champaign and VMware Research, USA; Sasa Misailovic, misailo@illinois.edu, University of Illinois at Urbana-Champaign, USA.

2475-1421/2023/10-ART291
https://doi.org/10.1145/3622867

many important properties in ML systems, e.g., certifying Lipschitz bounds of DNNs [Jordan and Dimakis 2020], demonstrating privacy of DNNs [Rosenberg et al. 2023], proving fairness [Gupta et al. 2021] or providing explanations of how DNNs make their decisions [Lerman et al. 2021]. AD is also used extensively in scientific computing for tasks as diverse as climate modeling [Mametjanov et al. 2012], analyzing differential equations [Bendtsen and Stauning 1996; Ma et al. 2021] and sensitivity analysis [Hovland et al. 2005]. Formalizations of derivatives computed using AD and their properties also have widespread use in Graphics [Bangaru et al. 2021; Yang et al. 2022].

Given the importance of formalizing properties over derivatives, formal verification techniques have recently been applied to AD [Hückelheim and Hascoët 2022; Hückelheim et al. 2018; Jordan and Dimakis 2021; Laurel et al. 2022a]. In particular, abstract interpretation [Cousot and Cousot 1977] has been used to statically analyze AD code [Jordan and Dimakis 2021; Laurel et al. 2022a; Vassiliadis et al. 2016] to ensure guaranteed bounds on derivatives.

Existing Abstract Interpretation work focuses primarily on developing precise abstract transformers for *linear* operations and assignments [Cousot and Halbwachs 1978; Singh et al. 2017]. However, AD computations are *highly non-linear*. Compared to just the original program (called the *primal*), the derivative program AD computes (called the *adjoint*) can have 2-5× more non-linear operations [Griewank and Walther 2008], e.g., for the most common operations:

- Every composition with a *non-linear function* in the primal requires a separate composition with that function's derivative in the adjoint and an additional multiplication, due to the chain rule.
- A single multiplication in the primal leads to 2 separate multiplications in the adjoint due to the product rule.
- A single division in the primal leads to 4 nonlinear operations in the adjoint due to quotient rule.

While one could reduce these groups of nonlinear operations to a series of basic (nonlinear) primitives, and then compose the corresponding primitives' abstract transformers, this construction loses precision. Even leveraging variable sharing in AD programs to reduce the number of applications of those basic abstract transformers, as in Laurel et al. [2022b] still loses precision as our evaluation will show. While one could try to design custom abstract transformers for groups of nonlinear operations, as in Singh et al. [2019a], this idea requires significant human expertise to handcraft the transformers and must be redone for each different function (e.g. $\tanh(x)$, $\sigma(x)$) [Fryazinov et al. 2010] and for each different abstract domain. Hence to date, there exists no general recipe for constructing AD static analyzers which can precisely abstract multiple non-linear operations for different functions, support multiple abstract domains and support both modes of AD.

Challenges. We focus on developing a general technique to synthesize precise static analyzers *tailored to the needs and structure of AD.* By abstracting multiple nonlinear computations all at once instead of suboptimally composing abstractions of each primitive operation, we seek to significantly improve the precision of the abstract interpretation of AD. Lastly, we seek to remove the burden on the abstraction designer from having to design transformers for each function and each abstract domain and each mode of AD. However, this approach must overcome two major challenges:

(1) Determine the right granularity for the abstraction. While grouping more operations together for the abstraction improves precision, the difficulty in synthesizing abstract transformers scales with the number of operations due to the need to solve multivariate optimization problems. Hence we must identify patterns to abstract that strike a balance between scalability and precision. The composite abstraction of a group of AD operations should be general enough to yield efficient and precise static analyzers for (1) multiple different functions (e.g. $\sigma(x), \exp(x), \sqrt{x}$) and (2) be instantiated with different abstract domains (e.g. interval, zonotope [Ghorbal et al. 2009], DeepPoly [Singh et al. 2019b]). Finally, we want to make sure the AD patterns we abstract are general enough to support both forward-mode *and* reverse-mode AD.

(2) Prove the synthesis procedure obtains sound abstract transformers for different functions and abstract domains. To synthesize an abstract transformer for a group of nonlinear operations over multiple variables, one must first solve a multi-dimensional, non-convex optimization problem to obtain soundness guarantees. However, we also want the proof techniques used to prove soundness to be general so that we can easily apply them to different nonlinear functions as well as have these proof technique support different abstract domains.

Our Work. To address these challenges, we propose Pasado. The main idea of Pasado is to soundly synthesize precise linear abstract transformers that are tailored to the Chain rule, Product Rule and Quotient Rule expressions of AD. Given their ubiquity in AD programs, we have identified that these patterns strike a desirable balance between efficiency, precision and tractability of the synthesized transformers. Pasado synthesizes sound abstract transformers for these *composite, multivariate, nonlinear* AD expressions directly instead of naively composing standard abstractions for each primitive nonlinear operation together. Additionally, Pasado supports both forward- and reverse-mode AD. Pasado's procedure is automated, hence one need not design hand-crafted abstract transformers as in Singh et al. [2018] and Du et al. [2021]. Furthermore, Pasado is general enough to support different function primitives and different abstract domains so that this idea can be fully leveraged to synthesize general static analyzers for differentiable programming languages. Lastly, Pasado is tractable, scalable and efficient enough to analyze large computations such as automatically differentiating through Convolutional Neural Networks (CNNs).

Pasado synthesizes these abstract transformers using a multi-step procedure. We first use linear regression to fit a hyperplane to the pattern we are trying to abstractly interpret. This step involves sub-sampling the AD expression at concrete points in a given input interval and using those evaluations for the linear regression. These input bounds may come from variables in the primal, hence Pasado simultaneously uses existing abstract interpretation methods to obtain bounds on the primal. Additionally, because the hyperplane is only a linear approximation it is unsound, hence we must also account for the deviation between this hyperplane and the original function to maintain soundness. Thus we solve for the maximum deviation between this hyperplane and the original AD expression we are abstracting and use this deviation to obtain sound linear bounds enclosing the hyperplane. A core part of our contribution is the mathematical proof of an efficient solution to this maximization problem. Specifically, we show how to provably rule out virtually all critical points, effectively reducing this optimization problem to examining boundary points.

We evaluate Pasado on multiple Case Studies including (1) robust sensitivity analysis of ODE solutions for both Neural ODEs and climate models, (2) provable bounds on sensitivities of financial models, (3) local Lipschitz robustness certification and (4) monotonicity certification. Due to Pasado's precision and scalability, the bounds obtained on the sensitivities of the climate, chemical, and financial differential equation solutions are between $1.31 - 2.81\times$ more precise (on average) compared to a state-of-the-art zonotope AD analysis and on our largest CNN, the local Lipschitz constants computed by Pasado are up to $2750\times$ more precise compared to the existing state-of-the-art zonotope AD analysis from Laurel et al. [2022b].

Contributions. In summary, this paper makes the following contributions:

(1) **Approach**. We present Pasado, a general formalism for synthesizing precise static analyzers for composite AD expressions. Due to Pasado's generality, this approach can be instantiated for a broad class of functions with both forward and reverse mode AD and with different abstract domains. (Sections 4.1-4.3)
(2) **Guarantees**. We formally prove the soundness of the abstract transformers synthesized by Pasado (Section 4.4) and theorems about their precision and generality (Sections 4.5-4.6).

(3) **Implementation**. We implement Pasado as a practical tool and instantiate it with both forward- and reverse-mode AD, as well as both the interval and zonotope domains and their reduced product. (Section 5.1). Pasado is available at https://github.com/uiuc-arc/Pasado.
(4) **Evaluation**. We experimentally show the benefit of Pasado on Lipschitz certification, monotonicity analysis and differential equation models from chemistry, finance, and climate science, and derive polyhedral bounds on sensitivities of these ODE solutions for the first time. (Sections 5.2-5.5)

2 EXAMPLE

We next proceed with a simple example illustrating our technique.

Running Example. A key application of *static analysis of Automatic Differentiation* that we target in this work is for robust sensitivity analysis of ordinary differential equation (ODE) solvers. We note that we are the first to consider such a static analysis. Our running example consists of automatically differentiating through the ODE solver (as in Ma et al. [2021]) for a simple ODE. Our ODE is a popular energy balance model used in climate science [Kaper and Engler 2013], where the global temperature T is a function of time t and Q, R, α, e are the global insolation, heat capacity, albedo and (scaled) emissivity parameters respectively. This ODE is given as:

$$\frac{dT}{dt} = \frac{Q \cdot (1-\alpha) - e \cdot T^4}{R} \tag{1}$$

The program that we will statically analyze solves this ODE numerically, thus obtaining a sequence $[T_0, ..., T_m]$ of temperatures for m time steps. To solve the ODE numerically, we will need the system dynamics function (RHS of Eq. 1) which as can be seen, is described by the function $f(T, t, Q, \alpha, e, R) = \frac{Q \cdot (1-\alpha) - e \cdot T^4}{R}$. The program for a simple Euler ODE solver applied to Eq. 1, which we will denote as $\texttt{ODESolve}_f$ is given below:

```
1 function ODESolve_f(Q, α, e, R, t_0, T_0, step_size, steps):
2     for (i=1; i < steps; i++) {
3         T_i = T_{i-1} + f(T_{i-1}, t_{i-1}, Q, α, e, R) · step_size   // Euler integrator
4         t_i = t_{i-1} + step_size }
5     return T, t
```

Thus by solving the ODE numerically (instead of symbolically) we can automatically differentiate through the numerical ODE solver itself, to get the derivative of the numerical solver's output with respect to the initial condition inputs, in this case T_0.

Applying AD to `ODESolve` is possible because `ODESolve` is *itself* a differentiable program since it performs only differentiable operations, which are just addition, multiplication (by the step size), as well as the multiplications, division, subtraction, and the (4^{th}) power function found inside of the dynamics f. Automatically differentiating through `ODESolve` will allow us to better understand how sensitive the numerical solution of ODEs are to different initial conditions or parameters (e.g. different values of e or R). Hence we will ultimately compute $\frac{\partial \texttt{ODESolve}_f(Q,\alpha,e,R,t_0,T_0,step\_size,steps)}{\partial T_0}$.

Abstract Sensitivity Analysis. However, our goal is to not just automatically differentiate through `ODESolve` to perform the sensitivity analysis at individual points (as done in Ma et al. [2021]), rather our goal is to *abstractly* compute these sensitivities for an *entire range* of initial conditions. Furthermore, since the physical parameters of this climate ODE can vary (e.g. e has dependence on the weather), we want this sensitivity analysis to capture an entire range of feasible parameter values. This abstract sensitivity analysis is performed by abstractly interpreting AD applied to the ODE solver. Hence we abstractly compute $\frac{\partial \texttt{ODESolve}_f(Q,\alpha,e,R,t_0,T_0,step\_size,steps)}{\partial T_0}$ for

```
1  T[0].real, T[0].dual = ... // These values are passed in as inputs
2  e.real, e.dual =  ...
3  R.real, R.dual = ...
4  step_size = ... // always treated as fixed scalar constant
5  Q, alpha = ...  // treated as fixed scalar constants for this example
6  C = Q*(1-alpha) // constant propagation for Q and alpha (both are constants)
7
8  i1.real = FourthPower(T[0].real) // Computes Line 3 of ODESolve_f
9  i1.dual = 4*Cube(T[0].real)*T[0].dual    // Chain Rule
10
11 i2.real = e.real*i1.real
12 i2.dual = (e.real*i1.dual)+(e.dual*i1.real)  // Product Rule
13
14 i3.real, i3.dual = C - i2.real, -i2.dual  // Linearity
15
16 i4.real = i3.real/R.real
17 i4.dual = ((i3.dual*R.real) - (i3.real*R.dual)) / Square(R.real)    // Quotient Rule
18
19 T[1].real, T[1].dual = (step_size*i4.real)+T[0].real, (step_size*i4.dual)+T[0].dual
```

Fig. 1. Unrolled AD source code for differentiating a single iteration of $\texttt{ODESolve}_f$

various ranges of T_0, e and R. This abstract sensitivity analysis can be viewed as a static analysis (using abstract interpretation) of a differentiable program, in this case the `ODESolve` program.

Specification. By performing the AD-based sensitivity analysis abstractly for *ranges* of initial conditions and parameter values, we can ultimately prove properties like the *monotonicity* of the numerical ODE solution with respect to the initial values for *all* values in the given input ranges. Proving monotonicity of the final ODE solution with respect to the initial conditions has been shown to be important for understanding physical processes [Wang et al. 2022]. Proving monotonicity of the output of this climate model with respect to the initial temperature conditions for a range of values and atmospheric conditions, allows us to prove properties such as the future temperature (after N time steps) being a strictly increasing function of the current temperature (the initial condition), even under a range of different atmospheric conditions. Thus, our goal in this example is to prove monotonicity of the future temperature computed by $\texttt{ODESolve}_f$ with respect to the initial temperature, T_0 after $N = 12$ steps when both T_0 and the atmospheric conditions are known only up to some interval. We will prove this property by certifying that the derivative is always non-zero. While abstract interpretation of AD has been done before [Laurel et al. 2022a,b; Vassiliadis et al. 2016], we will soon see that those techniques are not precise enough to prove the monotonicity property that we are interested in.

Example Source Code. Since ODE solvers run for a fixed, finite number of iterations, conceptually they can be unrolled. Hence the first step is to unroll the ODE solver program into straight-line code that we can abstractly interpret. For illustration simplicity, Pasado's abstract AD applied to a single iteration of $\texttt{ODESolve}_f$ is shown in Fig. 1, however, Pasado can analyze any fixed, finite number of iterations, though more iterations will diminish the precision. Indeed, the full results of Pasado's analysis after 12 iterations are later shown in Fig. 3. Since we are performing AD, we must also keep track of each variable's derivative. For this example, we elect for forward mode AD, hence we store each variable's value in the `real` component and its derivative in the associated `dual` component, as we encode derivatives using the canonical *dual numbers* [Griewank and Walther 2008]. Hence, our concrete semantics are first-order version of [Laurel et al. 2022b]. As will be seen later, our approach is general enough to support both forward and reverse mode AD. Lines 1-5 correspond to program statements that read in and store the input values for the initial condition T_0, as well as the parameter values for Q, α, e and R.

Chain Rule. On line 9, we compute the derivative of composing the 4^{th} power function with the input variable T_0, stored in variable `i1`. Due to the chain rule, this step also requires multiplying

by `T[0].dual`. We compute the derivative of this function application using the chain rule, where in this case $\frac{d}{dT_0}T_0^4 = 4T_0^3$. Between the cube function (`Cube(T[0])`), and the multiplication by `T[0].dual`, there are 3 nonlinear operations (all multiplications) involved in the computation of `i1.dual`. These multiple nonlinear operations pose a direct challenge for the subsequent abstract interpretation, however, as we will see, Pasado is specifically designed to overcome this challenge.

Product Rule. To propagate derivatives through the computation, we now need to compute the derivative of the product of T^4 with e. This step involves computing the product rule for the program variables `i1` and `e`. However, as can be seen on line 12, the product rule involves 2 nonlinear multiplication operations. Upon computing both the product of T^4 with e (line 11) and its derivative (line 12), we next execute line 14 to compute the difference of this product with $Q \cdot (1-a)$, which has already been stored in variable `C` in line 6. Because of linearity, the derivative of this difference (line 14) is straightforward to compute. Thus upon completion of line 14, we finally have computed both the numerator of the dynamics function f, and its associated derivative.

Quotient Rule. Having computed the intermediate expression needed for the numerator, we next come to the division operation shown in line 16. To compute the derivative of this division, AD must compute the quotient rule, as shown in line 17. It is important to see that the quotient rule has 4 nonlinear operations, which as mentioned, pose challenges for precise abstract interpretation.

Upon computing the quotient and its derivative (with respect to T_0), we finally compute the value of the next time step, T_1 and its associated derivative in line 19. Thus we now have both the value of the temperature at the next time step as well as the derivative of the temperature at the next time step with respect to the initial condition, which is just $\frac{\partial \texttt{ODESolve}_f(Q,\alpha,e,R,t_0,T_0,step\_size,steps)}{\partial T_0}$. We will next see how to abstractly interpret this AD computation precisely using Pasado.

Precise Abstract Interpretation with Pasado. As observed in the preceding description, there are multiple nonlinear operations involved in computing the derivatives that are stored in the intermediate variables' `dual` components. These nonlinearities present a challenge for precise abstract interpretation, as precise numerical abstract domains like zonotopes and polyhedra were originally designed for analyzing linear functions and are thus not well suited for handling nonlinear operations [Adjé et al. 2010]. For instance, with the zonotope abstract domain, each nonlinear operation introduces a new noise symbol, thus the large number of nonlinear operations (like multiplications) in AD runs the risk of substantial over-approximation. Existing abstract interpretations for AD [Laurel et al. 2022a,b] suffer from these weaknesses and, as we will later see, produce bounds that are too loose to be useful for our analysis.

For this example, we will use the reduced product of the zonotope domain with intervals to represent our abstraction. Pasado's synthesis method produces precise abstractions for both domains improving the precision of the reduced product. As we will later see, our construction is general and thus could be instantiated with other abstract domains such as the DeepPoly domain or quadratic zonotopes. Since we are using the reduced product of zonotopes with intervals, the abstract program state is shown in purple where $\wr\ \text{Var} = a_0 + \sum_j a_j\epsilon_j, l_{Var} = c_l, u_{Var} = c_u\ \wr$ signifies that variable Var (which could be either a real or dual component) is abstractly represented with affine form $a_0 + \sum_j a_j\epsilon_j$, but also maintains, tighter, refined bounds $[c_l, c_u]$. Here $a_j, c_l, c_u \in \mathbb{R}$ and each $\epsilon_j \in [-1, 1]$ is a noise symbol, which intuitively is a term in a first-order symbolic polynomial.

Fig. 2 shows the abstract interpretation of the original source code. In this example we want to abstractly compute guaranteed bounds on $\frac{\partial \texttt{ODESolve}_f(Q,\alpha,e,R,t_0,T_0,step\_size,steps)}{\partial T_0}$ when $T_0 \in [275, 400]$, $R \in [2.65, 2.95]$, $e \in [0.6\sigma, 0.9\sigma]$, $\alpha = 0.3$, $Q = 342$, $step\_size = 0.025$ and $steps = 1$. For the scaled emmisivity e, the value of σ is the Stefan-Boltzmann constant $5.67 \cdot 10^{-8}$. Thus to specify these bounds over the input, we initialize the abstract program state as shown in lines 2-16.

Synthesized Abstraction for Chain Rule. The first computation we perform is the computation of the quartic power function on line 18 and its derivative on line 21. As our contribution is focused solely on constructing synthesized abstraction for just the derivative terms, we use the standard interval and zonotope multiplication abstract transformers for the quartic function on line 18. It is important to note that after abstractly executing line 18 but before executing line 21, the zonotope abstract state will have 8 noise symbols ϵ_{1-8}. To abstractly interpret line 21, we use Pasado's synthesized abstract transformer instead of the standard zonotope multiplication abstract transformer. The first step of Pasado's synthesis procedure is to solve a linear regression problem for $A, B, C \in \mathbb{R}$ such that within the box $[l_{T[0].real}, u_{T[0].real}] \times [l_{T[0].dual}, u_{T[0].dual}] \subset \mathbb{R}^2$:

$$A \cdot (T[0].real) + B \cdot (T[0].dual) + C \approx 4 \cdot (T[0].real)^3 \cdot (T[0].dual)$$

Pasado performs this step by sampling uniformly-spaced points in the box, which here is $[275, 400] \times [1, 1]$. For each sampled point x_i, y_i we evaluate $4x_i^3 \cdot y_i$ which is used as the "ground-truth" for the linear regression. For this example linear regression produces the result: $A = 1378673.47$, $B = 0$, $C = -304748724.48$. Hence the affine form will be $\widehat{i1.dual} = A\widehat{T[0].real} + B\widehat{T[0].dual} + C + D\epsilon_{new}$, where ϵ_{new} is a single new noise symbol and the $\widehat{hat}$ notation denotes the variable's affine form stored in the abstract state. The next step is computing D by soundly bounding the error between this linear approximation and the true function $4 \cdot (T[0].real)^3 \cdot (T[0].dual)$, as shown in Eq. 2

$$D = \max_{x \in [275,400], y \in [1,1]} |4x^3 \cdot y - (Ax + By + C)| \tag{2}$$

Noting that $[l_{T[0].real}, u_{T[0].real}] = [275, 400]$ and $[l_{T[0].dual}, u_{T[0].dual}] = [1, 1]$. While at first glance this may seem like a difficult nonconvex, multivariable optimization problem a core contribution of Pasado is proving that we can reduce this multivariable optimization problem to simpler 1D optimization subproblems as well as just checking the corner points. Hence we can actually find the *exact* value of D by solving for roots of $12(T[0].real)^2 \cdot l_{T[0].dual} - A = 0$ and $12(T[0].real)^2 \cdot u_{T[0].dual} - A = 0$, checking those roots and additionally checking the corner points $\{l_{T[0].real}, u_{T[0].real}\} \times \{l_{T[0].dual}, u_{T[0].dual}\}$. The coefficient D will be used as the new noise symbol's magnitude to ensure the linear approximation is still a sound zonotopic enclosure.

An important detail is that because of our technique, the abstract transformer for the composite expression `4*Cube(T[0].real)*T[0].dual` only introduces a *single* new noise symbol ($D\epsilon_{new}$), hence why there are only nine noise symbols (ϵ_{1-9}) as shown in line 22. This insight is critical, as evaluating the same expression with the standard zonotope multiplications would introduce three noise symbols instead of one. It is known that having fewer noise symbols can lead to computational savings [Turner 2020], and this reduction helps us offset some of the cost of synthesizing transformers. Furthermore, we can use the *same* technique to solve the simpler optimization problems $\min_{x \in [275,400], y \in [1,1]} 4x^3 \cdot y$ and $\max_{x \in [275,400], y \in [1,1]} 4x^3 \cdot y$, which can be used to give us refined interval lower and upper bounds, which in this case are $[8.318 \cdot 10^7, 2.56 \cdot 10^8]$. While these bounds appear large, they will eventually be scaled to a tight range upon multiplication with e since $e < 10^{-7}$. Further, while in this case these interval bounds are identical to those obtained via standard interval arithmetic, for other functions, our optimally solved bounds are often much more precise. We take the intersection of the bounding box of the affine form which is $[6.759 \cdot 10^7, 2.56 \cdot 10^8]$ with $[8.318 \cdot 10^7, 2.56 \cdot 10^8]$ to get the final refined lower and upper bounds. In this case the refined interval bounds obtained by solving the two simpler optimization problems are *strictly* tighter than the affine form's bounding box (hence the result is still $[8.318 \cdot 10^7, 2.56 \cdot 10^8]$), however, this need not always be true, hence why we take the intersection. A key benefit of our approach is that because our abstract transformers use interval bounds to solve their optimization problem they

```
1  T[0].real, T[0].dual = ...  // These values are passed in as inputs
2  ⟅ T[0].real = 337.5 + -62.5ϵ1, l_T[0].real = 275, u_T[0].real = 400 ⟆
3  ⟅ T[0].dual = 1, l_T[0].dual = 1, u_T[0].dual = 1 ⟆
4  e.real, e.dual = ...
5  ⟅ e.real = 4.25 · 10^-8 + -8.50 · 10^-9ϵ3, l_e.real = 3.40 · 10^-8, u_e.real = 5.1 · 10^-8 ⟆
6  ⟅ e.dual = 0, l_e.dual = 0, u_e.dual = 0 ⟆
7  R.real, R.dual = ...
8  ⟅ R.real = 2.8 + -0.15ϵ5, l_R.real = 2.65, u_R.real = 2.95 ⟆
9  ⟅ R.dual = 0, l_R.dual = 0, u_R.dual = 0 ⟆
10 step_size = ...  // always treated as fixed scalar constant
11 ⟅ step_size = 0.025, l_step_size = 0.025, u_step_size = 0.025 ⟆
12 Q, alpha = ...  // treated as fixed scalar constants for this example
13 ⟅ Q = 342, l_Q = 342, u_Q = 342 ⟆
14 ⟅ alpha = 0.3, l_alpha = 0.3, u_alpha = 0.3 ⟆
15 C = Q*(1-alpha)  // constant propagation for Q and alpha (both are constants)
16 ⟅ C = 239.4, l_C = 239.4, u_C = 239.4 ⟆
17
18 i1.real = FourthPower(T[0].real)  // Computes Line 3 of ODESolve_f
19 ⟅ i1.real = 12974633789.062 + -9610839843.75ϵ1 + 889892578.12ϵ7 + 2124633789.062ϵ8,
20    l_i1.real = 5719140625, u_i1.real = 25600000000 ⟆
21 i1.dual = 4*Cube(T[0].real)*T[0].dual  // Chain Rule
22 ⟅ i1.dual = 161798882.57 + -86167091.83ϵ1 + 8034025.58ϵ9, l_i1.dual = 83187500, u_i1.dual = 256000000 ⟆
23
24 i2.real = e.real*i1.real
25 ⟅ i2.real = 551.78 + -408.72ϵ1 + -110.35ϵ3 + 37.84ϵ7 + 90.35ϵ8 + 107.38ϵ10, l_i2.real = 194.57, u_i2.real = 1306.45 ⟆
26 i2.dual = (e.real*i1.dual)+(e.dual*i1.real)  // Product Rule
27 ⟅ i2.dual = 6.88 + -3.66ϵ1 + 2.94e-15ϵ7 + 7.03e-15ϵ8 + 0.34ϵ9 + 2.17ϵ11, l_i2.dual = 2.83, u_i2.dual = 13.06 ⟆
28
29 i3.real, i3.dual = C - i2.real, - i2.dual  // Linearity
30 ⟅ i3.real = -312.38 + 408.72ϵ1 + 110.35ϵ3 + -37.84ϵ7 + -90.35ϵ8 + -107.38ϵ10, l_i3.real = −1067.05, u_i3.real = 44.82 ⟆
31 ⟅ i3.dual = -6.88 + 3.66ϵ1 + -2.94e-15ϵ7 + -7.03e-15ϵ8 + -0.34ϵ9 + -2.17ϵ11, l_i3.dual = −13.06, u_i3.dual = −2.83 ⟆
32
33 i4.real = i3.real/R.real
34 ⟅ i4.real = -111.9 + 146.39ϵ1 + 39.52ϵ3 + -5.38ϵ5 + -13.55ϵ7 + -32.36ϵ8 + -38.46ϵ10 + -0.61ϵ12 + 14.48ϵ13,
35    l_i4.real = −402.66, u_i4.real = 16.91 ⟆
36 i4.dual = ((i3.dual*R.real)-(i3.real*R.dual))/Square(R.real) // Quotient Rule
37 ⟅ i4.dual = -2.47 + 1.31ϵ1 + -0.0009ϵ3 + -0.15ϵ5 + 0.0003ϵ7 + 0.0007ϵ8 + 0.0009ϵ10 + -0.12ϵ9 + -0.78ϵ11 + 0.11ϵ14,
38    l_i4.dual = -4.93, u_i4.dual = -0.96⟆
39
40 T[1].real, T[1].dual = (step_size*i4.real)+T[0].real, (step_size*i4.dual)+T[0].dual
41 ⟅ T[1].real = 334.7 + -58.84ϵ1 + 0.99ϵ3 + -0.13ϵ5 + -0.33ϵ7 + -0.81ϵ8 + -0.96ϵ10 + -0.015ϵ12 + 0.36ϵ13,
42    l_T[1].real = 272.25, u_T[1].real = 397.15⟆
43 ⟅ T[1].dual = 0.94 + 0.03ϵ1 + -2.01e-5ϵ3 + -0.004ϵ5 + 6.88e-6ϵ7 + 1.64e-5ϵ8 + -0.003ϵ9 + 1.95e-5ϵ10 + -0.02ϵ11 + 0.002ϵ14,
44    l_T[1].dual = 0.8767, u_T[1].real = 0.9760⟆
```

Fig. 2. Abstractly Interpreted AD source code for differentiating a single iteration of $\texttt{ODESolve}_f$

can *directly* benefit from having these refined bounds which directly leads to more precise affine forms. In contrast, as noted in [Turner 2020], the standard zonotope transformers for multiplication *cannot* take advantage of the refined bounds to improve precision of the resulting affine form.

Synthesized Abstraction for Product Rule. Pasado's next step is to compute the product in line 24, as well as the derivative using the product rule in line 26. As mentioned, Pasado is focused solely on developing precise abstractions for the derivatives, hence the multiplication of the real parts in line 24 uses the standard interval and zonotope domain multiplications. Similarly to the chain rule, our synthesized abstract transformer will solve a linear regression problem for new $A, B, C, D, E \in \mathbb{R}$ such that in the region $[l_{e.real}, u_{real}] \times [l_{e.dual}, u_{e.dual}] \times [l_{i1.real}, u_{i1.real}] \times [l_{i1.dual}, u_{i1.dual}]$:

$$A \cdot (e.real) + B \cdot (e.dual) + C \cdot (i1.real) + D \cdot (i1.dual) + E \approx e.real \cdot i1.dual + e.dual \cdot i1.real$$

Hence the resulting affine form is $\widehat{i2.dual} = A(\widehat{e.real}) + B(\widehat{e.dual}) + C(\widehat{i1.real}) + D(\widehat{i1.dual}) + E + F\epsilon_{new}$ In this example, the linear regression finds the new coefficients should be $A = 0, B = -6 \cdot 10^{-24}$, $C = 3 \cdot 10^{-24}$, $D = 4.25 \cdot 10^{-8}$ and $E = -5 \cdot 10^{-14}$. Our abstraction must also solve for the maximum

error F. Among our key contributions is proving that for the product rule, this error can be obtained by just enumerating the 2^4 corner points. Hence we compute F as:

$$\max_{\substack{(x_1,y_1,x_2,y_2)\in \\ \{l_{e.real},u_{e.real}\}\times\{l_{e.dual},u_{e.dual}\}\times\{l_{i1.real},u_{i1.real}\}\times\{l_{i1.dual},u_{i1dual}\}}} |(x_1\cdot y_2)+(x_2\cdot y_1)-(Ax_1+By_1+Cx_2+Dy_2+E)|$$

In this example, $F = 2.17$. While our example has two multiplications on line 26, Pasado introduces only a single noise symbol ($F\epsilon_{new}$) instead of two as the standard zonotope multiplication would. Furthermore, we can solve a simpler version of the previous optimization problem for direct interval lower and upper bounds using the same approach (enumerating over the corners) and then take the intersection of those with the bounding box of the affine form. Performing this procedure ultimately gives us *i2.dual*'s refined bounds of $[2.83, 13.06]$ as shown on line 27.

Synthesized Abstraction for Quotient Rule. Upon computing the products, computing the affine transformations on line 29 is straightforward. Hence the abstract interpreter next proceeds to line 33 to compute the quotient. The derivative is computed via quotient rule on line 36. As before, Pasado uses linear regression to solve for coefficients $A, B, C, D, E \in \mathbb{R}$ such that

$$A\cdot(i3.real)+B\cdot(i3.dual)+C\cdot(R.real)+D\cdot(R.dual)+E \approx \frac{(R.real\cdot i3.dual)-(i3.real\cdot R.dual)}{R.real^2}$$

Here the values are $A = -4.2\cdot10^{-6}$, $B = 0.35$, $C = 1.01$, $D = 6.5\cdot10^{-6}$, and $E = -2.84$, where the resulting affine form is $\widehat{i4.dual} = A\widehat{(i3.real)} + B\widehat{(i3.dual)} + C\widehat{(R.real)} + D\widehat{(R.dual)} + E + F\epsilon_{new}$. Solving for the maximum error, F, is more involved, as we must solve:

$$\max_{\substack{x_1\in[l_{i3.real},u_{i3.real}],y_1\in[l_{i3.dual},u_{i3.dual}] \\ x_2\in[l_{R.real},u_{R.real}],y_2\in[l_{R.dual},u_{R.dual}]}} \left|\frac{(x_2\cdot y_1)-(x_1\cdot y_2)}{x_2^2} - (Ax_1+By_1+Cx_2+Dy_2+E)\right|$$

The solution to this optimization requires Pasado to solve for roots of a cubic equation, as we will see. However, Pasado can likewise use that same technique to directly solve for optimal interval bounds: $\min \frac{(x_2\cdot y_1)-(x_1\cdot y_2)}{x_2^2}$ and $\max \frac{(x_2\cdot y_1)-(x_1\cdot y_2)}{x_2^2}$ over the same 4D region as above. We ultimately obtain the affine form shown in line 43 and the refined bounds $[0.8767, 0.9760]$ in line 44.

Results. Abstractly interpreting the computation using Pasado's synthesized transformers offers notable precision gains over standard interval and zonotope abstract transformers. As mentioned, Pasado's bounds on the derivative after one iteration are: $[0.8767, 0.9760]$, whereas, the respective

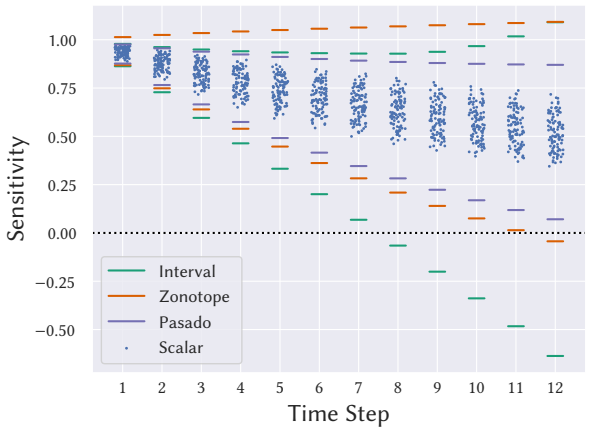


Fig. 3. Comparison of Pasado with interval and zonotope abstract AD for bounding derivative-based sensitivities with respect to T_0 for 12 full iterations of $\texttt{ODESolve}_f$. Each region between line segments of the same color represents the interval bound computed by that respective method's corresponding abstract AD. The dots represent the sensitivities evaluated concretely, using scalar points sampled from the input intervals.

bounds computed with intervals are $[0.862, 0.978]$ and with zonotopes are $[0.868, 1.013]$. While Figs. 1 and 2 show only a single iteration, the benefits of Pasado compound over multiple iterations.

Fig. 3 shows the bounds computed for up to $N = 12$ iterations by both Pasado and the interval and zonotope AD baselines. This plot shows how Pasado's improvement compounds, and how after 12 iterations (full code not shown), Pasado's derivative bounds stay provably positive, and thus can prove monotonicity with respect to T_0, whereas interval and zonotope abstract AD cannot.

3 PRELIMINARIES

We now detail the necessary preliminaries for describing both automatic differentiation as well as abstract interpretation. We also detail some of the key mathematical requirements for Pasado.

3.1 Automatic Differentiation Implementation

Forward-Mode AD. In forward-mode AD, one computes both the primal (the original program) as well as the tangent derivatives simultaneously in a single forward pass. For functions $f : \mathbb{R}^m \to \mathbb{R}^n$, computing the full Jacobian via forward mode requires m passes, hence forward mode is more efficient when $m < n$. Since the computation of both the original primal program and the derivatives are interwoven into a single forward pass and thus happens simultaneously, one must designate separate variables for the primal (the real part) and the associated derivatives (the dual part). This separation of variables into disjoint sets is canonically implemented with *dual numbers* [Griewank and Walther 2008], where the real component of the dual number encodes the variables of the primal, and the dual component of the dual number encodes the associated derivatives. The standard rules of calculus – chain rule, product rule and quotient rule can be encoded by overloading the respective arithmetic operation for dual numbers. For a given variable x, we will denote the real part as $x.real$ and the dual part storing the associated derivative as $x.dual$.

Reverse-Mode AD. In reverse mode AD, one first computes the primal and then back-propagates the derivatives from the output variable back to the input variables [Griewank and Walther 2008]. For functions $f : \mathbb{R}^m \to \mathbb{R}^n$, computing the full Jacobian via reverse mode requires n passes, hence reverse mode is more efficient when $m > n$. Unlike with forward-mode AD, in reverse-mode AD the original program (the primal) is computed in its entirety before even a single derivative is computed. However, one still needs a separate set of variables to store the derivatives. Following convention, we will simply let x denote a given variable in the primal and $\overline{x}$ denote the adjoint derivative for x that is computed in the backward pass.

3.2 Abstract Interpretation

We now describe the necessary preliminaries of Abstract Interpretation. Abstract Interpretation [Cousot and Cousot 1977] is a framework for soundly over-approximating the set of possible executions of a program. For Pasado's analysis, we require the following:

(1) A numerical abstract domain, $\mathbb{A}$ which may be either Intervals or any abstract domain that can represent linear transformations exactly (e.g. Zonotopes, Quadratic Zonotopes, Polyhedra). $\mathbb{A}$ can also be a reduced product of those aforementioned domains.
(2) A concretization function $\gamma : \mathbb{A} \to \mathcal{P}(\mathbb{R}^n)$ that maps an abstract element in the Abstract Domain to sets of AD program states (which are just tuples of n real numbers.).
(3) A bounding box function $Range : \mathbb{A} \to \mathbb{R}^n \times \mathbb{R}^n$ that takes an abstract element describing sets of AD program states of n variables and returns the bounds on each variable.
(4) Sound abstract transformers, $T_f^\sharp : \mathbb{A} \to \mathbb{A}$, for each univariate nonlinear function $f : \mathbb{R} \to \mathbb{R}$. The functions we consider are $\{\log(x), \exp(x), \sqrt{x}, x^2, x^3, x^4, \tanh(x), \sigma(x), NormalCDF(x)\}$.

(5) Sound abstract transformers, $T^{\sharp}_{op} : \mathbb{A} \to \mathbb{A}$, for each binary operation $op \in \{+, -, *, /\}$.

Additionally, unique to our approach is that we will also require a guaranteed root solver for the second derivative of each nonlinear function f listed in (4), so that we can solve for all $x^* \in [l, u]$ (or certify that none exist inside $[l, u]$) such that $f''(x^*) = C$ for any given C. For this root-solving, one may use a verified root finding technique like Bisection or a verified Newton's method, however for many of the functions such that f''^{-1} has an analytical formula, we can use the analytical formula to solve for x^* directly. For instance when $f(x) = \exp(x)$, we know $f''^{-1}(x) = \log(x)$.

For our purposes we will use the reduced product of the Zonotope domain with the Interval domain as this combination is essentially a (restricted) polyhedral domain that is more expressive than standard zonotopes, however as described in Theorem 4.5, our construction would equally work for other domains which can symbolically express linear relationships exactly such as quadratic zonotopes, or the DeepPoly domain.

For our purposes an abstract state $a \in \mathbb{A}$ will map each variable x_i to both an affine form, $a[x_i].\widehat{x_i} = x_{i0} + \sum_j c_j \epsilon_j$ and an interval $a[x_i].[l_{x_i}, u_{x_i}]$. The bounding box function returns the intersection of the associated interval with the bound on the affine form, hence $Range(a[x_i]) = [x_{i0} - \sum_j |c_j|, x_{i0} + \sum_j |c_j|] \sqcap a[x_i].[l_{x_i}, u_{x_i}]$. Since we take the reduced product, our concretization function uses both the standard interval and zonotope concretization functions $\gamma_{Int}, \gamma_{Zono}$ [Ghorbal et al. 2009] and is given as:

$$\gamma(a) = \left\{ (x_1, ..., x_n) \in \mathbb{R}^n : (x_1, ..., x_n) \in \gamma_{Zono}\left(\bigwedge_{j=1}^{n} a[x_j].\widehat{x_j}\right) \cap \gamma_{Int}\left(\bigwedge_{j=1}^{m} a[x_j].[l_{x_j}, u_{x_j}]\right) \right\}$$

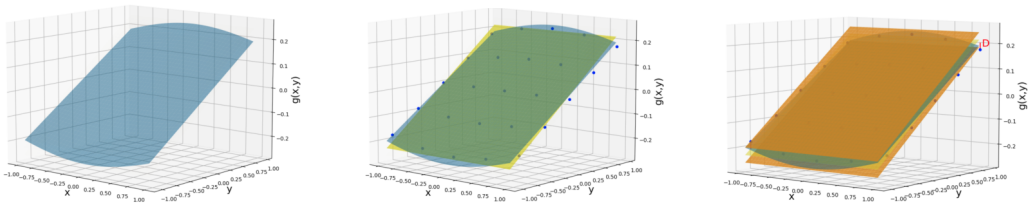
As mentioned in Section 2, for analyzing the real (non-derivative) part of the program we will use the given domain's standard abstract transformers required in (4) and (5) hence Pasado only focuses on synthesizing abstractions for the derivative terms to improve precision. Further, since addition can already be done exactly with zonotopes, Pasado will use the standard transformers for addition, even in the derivative computations.

4 SYNTHESIZING PRECISE STATIC ANALYZERS

We now present Pasado, our technique for synthesizing precise abstract transformers, specialized for AD. Pasado's technique allows us to synthesize precise abstract transformers for the Chain Rule, Product Rule and Quotient Rule of Calculus, which we will denote $T^{\sharp}_{C_f}$, $T^{\sharp}_{P}$, $T^{\sharp}_{Q}$: $\mathbb{A} \to \mathbb{A}$. Pasado will use standard abstract transformers (e.g., $T^{\sharp}_{f}$, $T^{\sharp}_{op}$ required in Section 3.2) to abstract the primal computation and $T^{\sharp}_{C_f}$, $T^{\sharp}_{P}$, $T^{\sharp}_{Q}$ to abstract the derivative computation.

Since both forward-mode AD and reverse-mode AD ultimately use these same rules, we can synthesize precise abstractions for each of the core operations for either mode of AD. The only difference between AD modes is the order of application, for instance in forward mode for $a \in \mathbb{A}$, the application order would be $T^{\sharp}_{P}(T^{\sharp}_{*}(T^{\sharp}_{C_f}(T^{\sharp}_{f}(a))))$ whereas for reverse mode the order would be $T^{\sharp}_{P}(T^{\sharp}_{C_f}(T^{\sharp}_{*}(T^{\sharp}_{f}(a))))$ as the entire primal must be abstracted before any derivatives can be. While there are other rules of calculus, like the generalized power rule, for which our techniques are inapplicable (due to differences in the Hessian behavior), these three rules comprise the majority of nonlinear AD operations and thus are the most important.

Pasado's abstract transformer synthesis involves a combination of linear regression at uniformly spaced points and solving a nonlinear optimization problem to ensure soundness. For tractability, we limit the number of sampled grid points used in the linear regression (which trades off precision).



(a) (b) (c)

Fig. 4. Visualization of Pasado's abstract transformer synthesis for the Chain rule pattern $g(x, y) = \sigma(x) \cdot (1 - \sigma(x)) \cdot y$ on $[-1, 1] \times [-1, 1]$. In (a), the blue surface represents $g(x, y)$. In (b), the blue dots on the blue surface represent evaluations of $g(x, y)$ at grid sampled points. The yellow hyperplane in (b) is computed by performing linear regression with these blue points and has equation $Ax + By + C$. In (c), the red lines show the difference between $g(x, y)$ and the plane and D represents the maximum such difference. The lower and upper orange planes in (c) are the enclosing linear bounds given by $Ax + By + C \pm D$. The enclosing bounds are parallel for the Zonotope domain and here the maximum difference D occurs at a corner point.

4.1 Chain Rule Synthesized Transformer

The first rule of Calculus for which we want to synthesize a precise abstraction is the Chain Rule. For functions $f, g : \mathbb{R} \to \mathbb{R}$, the chain rule is mathematically given as:

$$f(g(u))' = f'(g(u)) \cdot g'(u)$$

Forward-Mode Chain Rule. In forward-mode AD this rule is implemented via:

$$z.real = f(x.real);$$

$$z.dual = f'(x.real) \cdot x.dual;$$

where intuitively, $x.real = g(u)$, $x.dual = g'(u)$, $z.real = f(g(u))$, and $z.dual = f(g(u))'$.

Reverse-Mode Chain Rule. Likewise in reverse-mode AD, this rule is implemented as:

$$z = f(x); \quad ...$$

$$\overline{z} = ...$$

$$\overline{x} \mathrel{+}= f'(x) \cdot \overline{z};$$

where the "..." at the end of the first line represents the break between the end of the *primal* part of the differentiable program and the start of the *adjoint* part of the same differentiable program, which computes all the derivatives (e.g. $\overline{z}$, $\overline{x}$).

Chain Rule Abstraction Pattern. Based on these implementations, the main expression, present in both forward and reverse AD, for which we want to synthesize an abstract transformer, $T^{\sharp}_{C_f}$, is:

$$\boxed{g(x, y) = f'(x) \cdot y}$$

The benefit of synthesizing an abstraction for this chain rule pattern is that this pattern could have *multiple* nonlinear operations. For instance, if $f(x) = \sigma(x)$, then $f'(x) = \sigma(x) \cdot (1 - \sigma(x))$, which has a nonlinear multiplication, in addition to the nonlinear multiplication with y. Thus naively composing the abstract transformers for each nonlinear operation e.g., $T^{\sharp}_{*}(T^{\sharp}_{*}(T^{\sharp}_{-}(T^{\sharp}_{\sigma}(a))))$ as in Laurel et al. [2022b] can lead to imprecision. Particularly when using zonotopes, each of those nonlinear operations introduces a new noise symbol which adds additional over-approximation. In contrast, $T^{\sharp}_{C_f}$ introduces only a single noise symbol for the entire chain rule derivative expression.

Abstraction. We now present how to abstract the chain rule pattern. Algorithm 1 presents the abstract transformer $T^{\sharp}_{C_f}$ and Fig. 4 presents a geometric intuition. The core idea is to sample

Algorithm 1 Chain Rule Abstract Transformer $T^{\sharp}_{C_f}$

Input: Abstract state a where $\widehat{x} = a[x].\widehat{x}$ and $\widehat{y} = a[y].\widehat{y}$
$l_x, u_x \leftarrow Range(a[x])$
$l_y, u_y \leftarrow Range(a[y])$
$grid \leftarrow GridSample([l_x, u_x] \times [l_y, u_y])$
$pts \leftarrow \{f'(x) \cdot y : (x, y) \in grid\}$
$A, B, C \leftarrow LinearRegression(grid, pts)$
if $A = 0$ then $A \leftarrow A + \delta$
$D \leftarrow \max_{x \in [l_x, u_x], y \in [l_y, u_y]} |f'(x) \cdot y - (Ax + By + C)|$
$l, u \leftarrow \min_{x \in [l_x, u_x], y \in [l_y, u_y]} f'(x) \cdot y, \max_{x \in [l_x, u_x], y \in [l_y, u_y]} f'(x) \cdot y$
return $A\widehat{x} + B\widehat{y} + C + D\epsilon_{new}$, $[l, u]$

uniformly spaced points that lie within the range of the input intervals and then solve a linear regression problem to find the best linear approximation of $f'(x) \cdot y$ at those points. However, the most critical step for proving soundness is solving a challenging multidimensional, nonconvex optimization problem, to soundly enclose the linear approximation, which we now describe.

Optimization Problem. The core technical difficulty of the Chain Rule abstract transformer lies in solving the following equation for the maximum deviation between the linear approximation $(Ax + By + C)$ and the function $f'(x) \cdot y$ itself (example shown in Fig. 4). This maximum deviation is needed to obtain the tightest enclosure around the linear approximation $(Ax + By + C)$ such that this enclosure still provably contains the range of $f'(x) \cdot y$. This deviation D is computed as:

$$D = \max_{x \in [l_x, u_x], y \in [l_y, u_y]} |f'(x) \cdot y - (Ax + By + C)| \tag{3}$$

Pasado reduces this multivariate, non-convex optimization problem into two simpler univariate problems as well as simply checking the four corner points: $\{l_x, u_x\} \times \{l_y, u_y\}$ (we provide a full explanation in Section 4.4). For the correctness of our proof which is subsequently shown in Theorem 4.1, it is a technical requirement that $A \neq 0$. If linear regression obtains $A = 0$, we perturb A by a small quantity, $\delta < 10^{-9}$. To solve the two univariate optimization problems, we compute all $x^* \in [l_x, u_x]$ such that $f''(x^*) = \frac{A}{l_y}$ and all $x^{**} \in [l_x, u_x]$ such that $f''(x^{**}) = \frac{A}{u_y}$. Thus we must also examine the points (x^*, l_y) and (x^{**}, u_y). We can solve for all x^* and x^{**} using the guaranteed root solver that we required in Section 3.2. Hence the optimization problem ultimately reduces to:

$$D = \max_{(x,y) \in \left(\{l_{x_1}, u_{x_1}\} \times \{l_{y_1}, u_{y_1}\}\right) \cup \{(x^*, l_y), (x^{**}, u_y)\}} |f'(x) \cdot y - (Ax + By + C)|$$

While inspired by Ryou et al. [2021], our proof technique is more general as we can handle *any f satisfying the properties of Section 3.2*. The generality of our approach also stems from expanding this proof technique to other patterns arising from AD. We also show how to adapt this proof to obtain precise interval domain transformers. Indeed, the key benefit is that we can use virtually the same proof to get the *exact* lower and upper bounds of $f'(x) \cdot y$ for the given input intervals. Hence we can compute optimal lower and upper bounds, l and u, as follows:

$$l = \min_{(x,y) \in \left(\{l_{x_1}, u_{x_1}\} \times \{l_{y_1}, u_{y_1}\}\right) \cup \{(x^*, l_y), (x^{**}, u_y)\}} f'(x) \cdot y \tag{4}$$

$$u = \max_{(x,y) \in \left(\{l_{x_1}, u_{x_1}\} \times \{l_{y_1}, u_{y_1}\}\right) \cup \{(x^*, l_y), (x^{**}, u_y)\}} f'(x) \cdot y \tag{5}$$

The core benefit of having both zonotope affine forms and separately computed interval lower and upper bounds is that not only does the same proof strategy give us sound abstractions for both domains, but by taking their reduced product, we can always use the interval results to refine the zonotope, as in Singh et al. [2019c] to enhance precision.

4.2 Product Rule Synthesized Transformer

The next rule of calculus for which we wish to synthesize a precise abstract transformer, $T_P^\sharp$ is the Product Rule. For functions $f, g : \mathbb{R} \to \mathbb{R}$, the product rule is mathematically given as:

$$(f(u) \cdot g(u))' = f'(u) \cdot g(u) + f(u) \cdot g'(u)$$

Forward-Mode Product Rule. In forward-mode AD, the product rule is implemented via:

$$z.real = x.real \cdot y.real;$$

$$z.dual = (x.dual \cdot y.real) + (x.real \cdot y.dual);$$

where intuitively, $x.real = f(u)$, $y.real = g(u)$, $x.dual = f'(u)$, and $y.dual = g'(u)$. It is important to note that the computation of $z.dual$ involves 2 nonlinear multiplications.

Reverse-Mode Product Rule. Similarly, in reverse-mode AD, the product rule is encoded as:

$$z = x \cdot y; \quad ...$$
$$\overline{z} = ...$$
$$\overline{x} \mathrel{+}= y \cdot \overline{z};$$
$$\overline{y} \mathrel{+}= x \cdot \overline{z};$$

Product Rule Abstraction Pattern. Based on the product rule implementations shown above, the computational pattern for which we want to synthesize an abstraction is:

$$\boxed{g(x_1, y_1, x_2, y_2) = (x_1 \cdot y_2) + (x_2 \cdot y_1)}$$

In the event that some of the arguments are zero, this pattern could involve a single multiplication. Conversely, when all arguments are nonzero, this pattern entails two nonlinear multiplications. In particular, the instantiation of this abstraction for reverse-mode AD can be seen as a special case whereby we set the first argument x_1 to 0.

Abstraction. We now present Pasado's synthesis technique for the product rule abstract transformer, $T_P^\sharp$ in Algorithm 2. As with the chain rule pattern, we synthesize the coefficients by solving a linear regression, and a tractable, but nonconvex optimization problem.

There is another key benefit to Pasado's product rule abstract transformer. The standard zonotope multiplication operates directly on the coefficients of the input affine forms. As a result, the standard abstract transformer cannot leverage tighter ranges on the inputs, as these ranges are never taken into account. Thus, when performing the two multiplications of the product rule with regular zonotopes, the ability to leverage improved precision from a reduced product is limited. In contrast, since Pasado's synthesized abstraction *directly* incorporates the bounds on the input, Pasado can immediately benefit from the improved precision that a reduced product with intervals offers.

Optimization Problem. For the product rule abstraction, we must solve the following 4D nonlinear, nonconvex optimization problem:

$$F = \max_{\substack{x_1 \in [l_{x_1}, u_{x_1}], y_1 \in [l_{y_1}, u_{y_1}] \\ x_2 \in [l_{x_2}, u_{x_2}], y_2 \in [l_{y_2}, u_{y_2}]}} |(x_1 \cdot y_2) + (x_2 \cdot y_1) - (Ax_1 + By_1 + Cx_2 + Dy_2 + E)| \tag{6}$$

The benefit of this optimization problem is that we only need to check the 2^4 corner points, i.e., $\{l_{x_1}, u_{x_1}\} \times \{l_{y_1}, u_{y_1}\} \times \{l_{x_2}, u_{x_2}\} \times \{l_{y_2}, u_{y_2}\}$, hence the optimization problem of Eq. 6 reduces to:

Algorithm 2 Product Rule Abstract Transformer $T_P^\sharp$

Input: Abstract state a where $\widehat{x_1} = a[x_1].\widehat{x_1}$, $\widehat{y_1} = a[y_1].\widehat{y_1}$, $\widehat{x_2} = a[x_2].\widehat{x_2}$, and $\widehat{y_2} = a[y_2].\widehat{y_2}$

$l_{x_1}, u_{x_1} \leftarrow Range(a[x_1])$ and $l_{y_1}, u_{y_1} \leftarrow Range(a[y_1])$

$l_{x_2}, u_{x_2} \leftarrow Range(a[x_2])$ and $l_{y_2}, u_{y_2} \leftarrow Range(a[y_2])$

$grid \leftarrow GridSample([l_{x_1}, u_{x_1}] \times [l_{y_1}, u_{y_1}] \times [l_{x_2}, u_{x_2}] \times [l_{y_2}, u_{y_2}])$

$pts \leftarrow \{(x_1 \cdot y_2) + (x_2 \cdot y_1) : (x_1, y_1, x_2, y_2) \in grid\}$

$A, B, C, D, E \leftarrow LinearRegression(grid, pts)$

$$F \leftarrow \max_{\substack{x_1 \in [l_{x_1}, u_{x_1}], y_1 \in [l_{y_1}, u_{y_1}] \\ x_2 \in [l_{x_2}, u_{x_2}], y_2 \in [l_{y_2}, u_{y_2}]}} |(x_1 \cdot y_2) + (x_2 \cdot y_1) - (Ax_1 + By_1 + Cx_2 + Dy_2 + E)|$$

$$l, u \leftarrow \min_{\substack{x_1 \in [l_{x_1}, u_{x_1}], y_1 \in [l_{y_1}, u_{y_1}] \\ x_2 \in [l_{x_2}, u_{x_2}], y_2 \in [l_{y_2}, u_{y_2}]}} (x_1 \cdot y_2) + (x_2 \cdot y_1), \max_{\substack{x_1 \in [l_{x_1}, u_{x_1}], y_1 \in [l_{y_1}, u_{y_1}] \\ x_2 \in [l_{x_2}, u_{x_2}], y_2 \in [l_{y_2}, u_{y_2}]}} (x_1 \cdot y_2) + (x_2 \cdot y_1)$$

return $A\widehat{x_1} + B\widehat{y_1} + C\widehat{x_2} + D\widehat{y_2} + E + F\epsilon_{new}$, $[l, u]$

$$F = \max_{(x_1, y_1, x_2, y_2) \in \{l_{x_1}, u_{x_1}\} \times \{l_{y_1}, u_{y_1}\} \times \{l_{x_2}, u_{x_2}\} \times \{l_{y_2}, u_{y_2}\}} |(x_1 \cdot y_2) + (x_2 \cdot y_1) - (Ax_1 + By_1 + Cx_2 + Dy_2 + E)|$$

However, as with the chain rule, we will also solve for the lower and upper bounds, l and u, for $(x_1 \cdot y_2) + (x_2 \cdot y_1)$ exactly. Furthermore, we can use the same proof technique to obtain these bounds and thus merely enumerate over the 2^4 corners to evaluate the expression, hence:

$$l = \min_{(x_1, y_1, x_2, y_2) \in \{l_{x_1}, u_{x_1}\} \times \{l_{y_1}, u_{y_1}\} \times \{l_{x_2}, u_{x_2}\} \times \{l_{y_2}, u_{y_2}\}} (x_1 \cdot y_2) + (x_2 \cdot y_1) \tag{7}$$

$$u = \max_{(x_1, y_1, x_2, y_2) \in \{l_{x_1}, u_{x_1}\} \times \{l_{y_1}, u_{y_1}\} \times \{l_{x_2}, u_{x_2}\} \times \{l_{y_2}, u_{y_2}\}} (x_1 \cdot y_2) + (x_2 \cdot y_1) \tag{8}$$

4.3 Quotient Rule Synthesized Transformer

Synthesizing an abstract transformer for Quotient Rule is challenging due to the divisions and multiplications involved. In fact, many works [Shi et al. 2020; Stolfi and De Figueiredo 1997] decompose the abstraction of the division x/y into first abstracting the univariate function $1/y$, then abstracting the multiplication of that intermediate result with x. However, our goal is to *avoid* decomposing these expressions into basic primitives, as naively composing primitive abstract transformers leads to imprecision. For functions $f, g : \mathbb{R} \to \mathbb{R}$, the quotient rule is defined as:

$$\left(\frac{f(u)}{g(u)}\right)' = \frac{f'(u) \cdot g(u) - f(u) \cdot g'(u)}{g(u)^2}$$

Forward-Mode Quotient Rule. In forward-mode AD the quotient rule is implemented as:

$$z.real = \frac{x.real}{y.real};$$

$$z.dual = \frac{x.dual \cdot y.real - x.real \cdot y.dual}{(y.real)^2};$$

where intuitively $z.dual = (\frac{f(u)}{g(u)})'$, $x.dual = f'(u)$, and $y.dual = g'(u)$.

Reverse-Mode Quotient Rule. Likewise in reverse-mode AD the quotient rule is encoded as:

$$z = \frac{x}{y};\ \ ...$$
$$\overline{z} = ...$$
$$\overline{x}\ \mathrel{+}= \frac{1}{y} \cdot \overline{z};$$
$$\overline{y}\ \mathrel{+}= \frac{-x}{y^2} \cdot \overline{z};$$

If one were to take the computational pattern of $z.dual$, but set $x.dual = 0$, the structure of the computation would be similar to the computation of $\overline{y}$. The abstraction for $\overline{x}$ can be handled separately using the chain rule abstraction $T^\sharp_{C_f}$, as the chain rule abstraction for $\log'(y) \cdot z$ is $\frac{z}{y}$.

Quotient Rule Abstraction Pattern. Based on the quotient rule implementations shown above, the computational pattern for which we want to synthesize an abstract transformer is:

$$\boxed{g(x_1, y_1, x_2, y_2) = \frac{(x_2 \cdot y_1) - (x_1 \cdot y_2)}{x_2^2}}$$

This pattern is a desirable choice for having a single abstraction, $T^\sharp_Q$ because there are 3 nonlinear multiplications and a nonlinear division, hence the pattern would otherwise be abstracted with the composition $T^\sharp_*(T^\sharp_/(T^\sharp_-(T^\sharp_*(T^\sharp_*(a)))))$. Furthermore there is a high degree of correlation between the numerator and denominator since they both contain x_2, however, due to the imprecision most abstract interpreters face in the presence of multiple nonlinearities, it is hard to precisely capture this dependency. Further, as mentioned, the computation of $\overline{y}$ in the reverse mode is a special instance of this abstraction pattern when the second argument $y_1 = 0$.

Abstraction. We now present the synthesis of the abstract transformer for the Quotient Rule, $T^\sharp_Q$, which Algorithm 3 presents. As before, Pasado must solve an optimization problem to obtain sound bounds around the synthesized linear approximation (obtained via regression). To ensure that there are not additional interior critical points to consider, we require some constraints on the synthesized coefficients A, B, C, D, E given in terms of the input bounds $\kappa_{x_i} \in \{l_{x_i}, u_{x_i}\}$, $\kappa_{y_i} \in \{l_{y_i}, u_{y_i}\}$. Further it is simple to enforce these constraints - if the synthesized coefficients do not satisfy the following constraints, we can perturb the coefficients by some small δ, which will not affect the abstraction's precision, but will ensure soundness. The conditions we require are:

$$\kappa_{x_1} \neq \frac{-D}{A^2} \wedge \sqrt[\frac{2}{3}]{\frac{-\kappa_{y_1}}{AD}} \neq \frac{-3\kappa_{y_1}}{2C} \wedge \frac{1}{B} \neq \kappa_{x_2} \wedge \frac{A}{B^2} \neq \kappa_{y_2}$$

The *Adjust* function checks that the coefficients obtained by the linear regression satisfy these constraints, and if not, it will add small perturbations ($\delta < 10^{-9}$) until the conditions are satisfied.

Optimization Problem. To obtain a sound enclosure around the linear approximation for the Quotient rule, we must solve the following nonlinear, nonconvex 4D optimization problem:

$$\max_{\substack{x_1 \in [l_{x_1}, u_{x_1}], y_1 \in [l_{y_1}, u_{y_1}] \\ x_2 \in [l_{x_2}, u_{x_2}], y_2 \in [l_{y_2}, u_{y_2}]}} \left| \frac{(x_2 \cdot y_1) - (x_1 \cdot y_2)}{x_2^2} - (Ax_1 + By_1 + Cx_2 + Dy_2 + E) \right| \tag{9}$$

This optimization problem is more difficult than the one needed for the product rule, however, much of the core idea is the same and *it is still tractable to solve automatically*. We will still need to check all 2^4 corner points $\{l_{x_1}, u_{x_1}\} \times \{l_{y_1}, u_{y_1}\} \times \{l_{x_2}, u_{x_2}\} \times \{l_{y_2}, u_{y_2}\}$, however, we will also need to check for potential critical points where the gradient could be zero and that cannot be

Algorithm 3 Quotient Rule Abstract Transformer $T_Q^\sharp$

Input: Abstract state a where $\widehat{x_1} = a[x_1].\widehat{x_1}$, $\widehat{y_1} = a[y_1].\widehat{y_1}$, $\widehat{x_2} = a[x_2].\widehat{x_2}$, and $\widehat{y_2} = a[y_2].\widehat{y_2}$

$l_{x_1}, u_{x_1} \leftarrow Range(a[x_1])$ and $l_{y_1}, u_{y_1} \leftarrow Range(a[y_1])$

$l_{x_2}, u_{x_2} \leftarrow Range(a[x_2])$ and $l_{y_2}, u_{y_2} \leftarrow Range(a[y_2])$

$grid \leftarrow GridSample([l_{x_1}, u_{x_1}] \times [l_{y_1}, u_{y_1}] \times [l_{x_2}, u_{x_2}] \times [l_{y_2}, u_{y_2}])$

$pts \leftarrow \left\{ \frac{(x_2 \cdot y_1) - (x_1 \cdot y_2)}{x_2^2} : (x_1, y_1, x_2, y_2) \in grid \right\}$

$A, B, C, D, E \leftarrow LinearRegression(grid, pts)$

$Adjust(A, B, C, D, E)$

$$F \leftarrow \max_{\substack{x_1 \in [l_{x_1}, u_{x_1}], y_1 \in [l_{y_1}, u_{y_1}] \\ x_2 \in [l_{x_2}, u_{x_2}], y_2 \in [l_{y_2}, u_{y_2}]}} \left| \frac{(x_2 \cdot y_1) - (x_1 \cdot y_2)}{x_2^2} - (Ax_1 + By_1 + Cx_2 + Dy_2 + E) \right|$$

$$l, u \leftarrow \min_{\substack{x_1 \in [l_{x_1}, u_{x_1}], y_1 \in [l_{y_1}, u_{y_1}] \\ x_2 \in [l_{x_2}, u_{x_2}], y_2 \in [l_{y_2}, u_{y_2}]}} \frac{(x_2 \cdot y_1) - (x_1 \cdot y_2)}{x_2^2}, \max_{\substack{x_1 \in [l_{x_1}, u_{x_1}], y_1 \in [l_{y_1}, u_{y_1}] \\ x_2 \in [l_{x_2}, u_{x_2}], y_2 \in [l_{y_2}, u_{y_2}]}} \frac{(x_2 \cdot y_1) - (x_1 \cdot y_2)}{x_2^2}$$

return $A\widehat{x_1} + B\widehat{y_1} + C\widehat{x_2} + D\widehat{y_2} + E + F\epsilon_{new}$, $[l, u]$

immediately ruled out by the Hessian test. Thankfully, solving for these critical points involves only *univariate* cubic polynomial equations, as we will be able to provably rule out any interior critical point where x_1, y_1, and y_2 are *not* fixed to their respective lower or upper bounds. Thus the only potential critical points are along the edges of the 4D hypercube where x_1, y_1, and y_2 are fixed. To find these potential critical points, we only have to solve a cubic polynomial in x_2. Thus this set will still be finite as each cubic polynomial has at most 3 real roots. We now define this set of potential critical points, S as solutions to 8 possible cubic equations.

$$S = \{(x_1, y_1, x_2, y_2) : x_1 \in \{l_{x_1}, u_{x_1}\}, y_1 \in \{l_{y_1}, u_{y_1}\}, y_2 \in \{l_{y_2}, u_{y_2}\}, Cx_2^3 + y_1 x_2 - 2x_1 y_2 = 0, x_2 \in [l_{x_2}, u_{x_2}]\}$$

Thus we can solve the maximization problem of Eq. 9 by enumerating over the corner points and all points in S, reducing the problem to:

$$\max_{\substack{(x_1, y_1, x_2, y_2) \in \{l_{x_1}, u_{x_1}\} \times \{l_{y_1}, u_{y_1}\} \times \{l_{x_2}, u_{x_2}\} \times \{l_{y_2}, u_{y_2}\} \\ \cup S}} \left| \frac{(x_2 \cdot y_1) - (x_1 \cdot y_2)}{x_2^2} - (Ax_1 + By_1 + Cx_2 + Dy_2 + E) \right|$$

This strategy of solving for roots of a cubic equation to check as possible extrema can also be used to solve for the optimal interval lower and upper bounds. Hence we compute the following:

$$l = \min_{\substack{x_1 \in [l_{x_1}, u_{x_1}], y_1 \in [l_{y_1}, u_{y_1}] \\ x_2 \in [l_{x_2}, u_{x_2}], y_2 \in [l_{y_2}, u_{y_2}]}} \frac{(x_2 \cdot y_1) - (x_1 \cdot y_2)}{x_2^2} = \min_{\substack{\{l_{x_1}, u_{x_1}\} \times \{l_{y_1}, u_{y_1}\} \times \{l_{x_2}, u_{x_2}\} \times \{l_{y_2}, u_{y_2}\} \\ \cup S}} \frac{(x_2 \cdot y_1) - (x_1 \cdot y_2)}{x_2^2} \tag{10}$$

$$u = \max_{\substack{x_1 \in [l_{x_1}, u_{x_1}], y_1 \in [l_{y_1}, u_{y_1}] \\ x_2 \in [l_{x_2}, u_{x_2}], y_2 \in [l_{y_2}, u_{y_2}]}} \frac{(x_2 \cdot y_1) - (x_1 \cdot y_2)}{x_2^2} = \max_{\substack{\{l_{x_1}, u_{x_1}\} \times \{l_{y_1}, u_{y_1}\} \times \{l_{x_2}, u_{x_2}\} \times \{l_{y_2}, u_{y_2}\} \\ \cup S}} \frac{(x_2 \cdot y_1) - (x_1 \cdot y_2)}{x_2^2} \tag{11}$$

where the only difference is that the set S in Equations 10 and 11 is now given by $S = \{(x_1, y_1, x_2, y_2) : x_1 \in \{l_{x_1}, u_{x_1}\}, y_1 \in \{l_{y_1}, u_{y_1}\}, y_2 \in \{l_{y_2}, u_{y_2}\}, x_2 = \frac{2x_1 y_2}{y_1}, x_2 \in [l_{x_2}, u_{x_2}]\}$.

4.4 Soundness

In this section we prove the soundness of Pasado's synthesized abstract transformers. At a high level, our proof relies on the fact that at interior critical points the Hessian will be indeterminant (ensuring they are saddle points). Intuitively, this insight allows for ruling out the (multi-dimensional) interior, and thus checking only (lower dimensional) boundaries to solve the optimization problem.

Theorem 4.1. *The Chain Rule Transformers $T^{\sharp}_{C_f}$, synthesized by Pasado for any $f : \mathbb{R} \to \mathbb{R}$ obeying the properties of Sec. 3.2 are sound. Equivalently for $a \in \mathbb{A}$, $\{f'(x) \cdot y : (x, y) \in \gamma(a)\} \subseteq \gamma(T^{\sharp}_{C_f}(a))$*

Proof. Let $f(x) : \mathbb{R} \to \mathbb{R}$ and $A, B, C \in \mathbb{R}$ be the coefficients inferred by the linear regression. To ensure soundness of $T^{\sharp}_{C_f}$, our goal reduces to solving the following optimization problem:

$$max_{x \in [l_x, u_x], y \in [l_y, u_y]} |f'(x) \cdot y - (Ax + By + C)|$$

Instead of looking for interior critical points with the first derivative test, we first examine the Hessian, H, of $f'(x) \cdot y - (Ax + By + C)$ which is given as:

$$H = \begin{bmatrix} f'''(x) \cdot y & f''(x) \\ f''(x) & 0 \end{bmatrix}$$

The Hessian determinant is $-(f''(x)^2)$ which is always ≤ 0. For any x such that $f''(x) \neq 0$, then $-(f''(x)^2) < 0$, ruling out any interior critical point since the Hessian determinant is negative. Further, any x such that $f''(x) = 0$ cannot be a critical point, since from the first derivative test $f''(x) - A = 0$ is necessary for a critical point, however, if $f''(x) = 0$ then $f''(x) - A = -A \neq 0$, since we required $A \neq 0$. Thus the optimal value will occur along the 4 boundary lines of the square. The first two boundary lines ($x = l_x$ or $x = u_x$) are simple, since the optimal value of a linear function will occur at the end points, thus it suffices to check the 4 corners $\{l_x, u_x\} \times \{l_y, u_y\}$. For the boundary lines where we fix $y = l_y$ or $y = u_y$, we now only have to solve univariate optimization problems. Applying the first derivative test to $f'(x)l_y$ and $f'(x)u_y$ we must solve for all $x^* \in [l_x, u_x]$ such that $f''(x^*)l_y - A = 0$ as well as all $x^{**} \in [l_x, u_x]$ such that $f''(x^{**})u_y - A = 0$. However, these equations can be solved as we already required that one has a verified root solver for f''. Hence we just check (x^*, l_y) and (x^{**}, u_y) for all x^*, $x^{**} \in [l_x, u_x]$ returned by the root solver. Furthermore, the Hessian determinant of $-(f'(x) \cdot y - (Ax + By + C))$ will be identical, and any critical point of $f'(x) \cdot y - (Ax + By + C)$ will be a critical point of $-(f'(x) \cdot y - (Ax + By + C))$, hence we need not worry about the absolute value in the maximization problem. The full details can be found in the appendix [Laurel et al. 2023a].

□

Theorem 4.2. *The Product Rule Transformer, $T^{\sharp}_{P}$, synthesized by Pasado is sound. Equivalently for $a \in \mathbb{A}$, $\{x_1 \cdot y_2 + x_2 \cdot y_1 : (x_1, y_1, x_2, y_2) \in \gamma(a)\} \subseteq \gamma(T^{\sharp}_{P}(a))$*

Proof. (sketch) The Hessian for $(x_1 \cdot y_2) + (x_2 \cdot y_1) - (Ax_1 + By_1 + Cx_2 + Dy_2 + E)$ at any point is the matrix $\begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$ which has both positive and negative eigenvalues, meaning the Hessian is everywhere indeterminant. Hence any potential critical point is necessarily a saddle point, thus the extrema must occur along the boundaries. We repeat this procedure for each of the $\binom{4}{3}$ 3D boundary optimization problems where in each case 1 of the 4 dimensions is fixed to a lower or

upper boundary. The (unique) 3D sub-problem Hessians are $\begin{bmatrix}0&0&0\\0&0&1\\0&1&0\end{bmatrix}$, $\begin{bmatrix}0&0&1\\0&0&0\\1&0&0\end{bmatrix}$, and $\begin{bmatrix}0&1&0\\1&0&0\\0&0&0\end{bmatrix}$ all of which have both positive and negative eigenvalues meaning any interior point along the 3D boundary of the 4D cube is necessarily a saddle point. Repeating this same procedure for the 2D subproblems we again find that all 2D Hessians are either indeterminant which implies any interior critical point is a saddle (thus the extrema occurs on the corners) or the 2D subproblem is such that the function is linear in which case the extrema will also occur only on the corners. Furthermore, by lemma A.1 these same properties hold for $-((x_1 \cdot y_2) + (x_2 \cdot y_1) - (Ax_1 + By_1 + Cx_2 + Dy_2 + E))$ thus handling the absolute value in the maximization problem. The full proof is found in the appendix. □

Theorem 4.3. *The Quotient Rule Transformer, $T_{\tilde{Q}}^{\sharp}$ synthesized by Pasado is sound. Equivalently for $a \in \mathbb{A}$, $\{\frac{x_2 \cdot y_1 - x_1 \cdot y_2}{x_2^2} : (x_1, y_1, x_2, y_2) \in \gamma(a)\} \subseteq \gamma(T_{\tilde{Q}}^{\sharp}(a))$*

Proof. (sketch) The 4D Hessian is $\begin{bmatrix} 0 & 0 & \frac{2d}{x_2^3} & \frac{-1}{x_2^2} \\ 0 & 0 & \frac{-1}{x_2^2} & 0 \\ \frac{2y_2}{c^3} & \frac{-1}{x_2^2} & \frac{6(y_1x_2 - x_1y_2)}{x_2^4} - \frac{4y_1}{x_2^3} & \frac{2x_1}{x_2^3} \\ \frac{-1}{x_2^2} & 0 & \frac{2x_1}{x_2^3} & 0 \end{bmatrix}$ and its determinant is $\frac{1}{x_2^8}$, meaning that the the Hessian has no zero eigenvalues since $x_2 \neq 0$. Furthermore, by Sylvester's criteria the Hessian is neither positive definite nor negative definite, hence it is indeterminant which implies any interior critical point is necessarily a saddle point, thus the extrema will occur along a boundary of the 4D cube. The cases for optimizing along these 3D boundaries, and their respective 2D boundaries, and their respective 1D boundaries are all shown in the Appendix. □

Given the similarity in all three proofs, the same ideas could theoretically be used to support other nonlinear expressions, provided their respective Hessians also rule out interior critical points.

4.5 Precision

Having now defined how to solve the optimization problems needed for Pasado's abstract transformers and their soundness, we can now state the following theorem about their precision:

Theorem 4.4. *The lower and upper interval bounds computed by Pasado's synthesized abstract transformers in Equations 4, 5, 7, 8, 10, and 11 are optimal for the interval domain.*

Proof. (sketch) Since Pasado solves these optimization problems exactly, instead of upper bounding the maximum or lower bounding the minimum, the bounds cannot be any tighter. □

Hence for these AD patterns, a standard interval arithmetic where one composes abstractions of each primitive function or operation ($T_f^{\sharp}$, $T_{op}^{\sharp}$) can never be more precise than Pasado. Further, because Pasado uses the standard abstract transformers in the real (primal) part of the program, those bounds are at least as precise as standard interval arithmetic, thus Pasado's derivative bounds are never less precise than abstractly interpreting AD with the interval domain.

4.6 Generality

While we focus on the zonotope and interval abstract domains, Pasado is applicable to other abstract domains that can represent linear relationships symbolically. We now state how to use Pasado to obtain sound transformers for quadratic and polynomial zonotopes and the DeepPoly domains.

Theorem 4.5. *The abstract transformers synthesized by Pasado are also sound transformers for quadratic and polynomial zonotopes and for the DeepPoly domain.*

Proof. As we only synthesize linear transformations of the input affine forms, if these affine forms were replaced with quadratic or polynomial forms, a linear transformation of them would still be a valid quadratic or polynomial term expressible in those domains. For adapting the chain rule to the DeepPoly domain we set the lower linear bound to be $a^{\geq} = Ax_1 + By_1 + (C - D)$ and the upper linear bound to be $a^{\leq} = Ax_1 + By_1 + (C + D)$ instead of returning an affine form and still use l, u for the interval bounds. Likewise for adapting the product and quotient rule transformers to the DeepPoly domain we set the lower linear bound as $a^{\geq} = Ax_1 + By_1 + Cx_2 + Dy_2 + (E - F)$ and the upper linear bound as $a^{\leq} = Ax_1 + By_1 + Cx_2 + Dy_2 + (E + F)$ and use l, u for the interval bounds. □

Thus, Pasado's combined support for multiple AD computational patterns, function primitives, and abstract domains, provides the necessary generality for synthesizing static analyzers for AD.

5 CASE STUDIES

We now present multiple Case Studies demonstrating the benefits of synthesizing precise abstract transformers tailored to the structure of both forward-mode AD (Sections 5.2 and 5.4) and reverse-mode AD (Sections 5.3 and 5.5).

5.1 Methodology

We describe our experimental setup. We ran the experiments in Sections 5.2 and 5.3 on a 10-core Apple M1 Pro SoC with 16 GB of unified memory. We ran the experiments in Sections 5.4 and 5.5 on a 32-core AMD Ryzen Threadripper PRO 3975WX CPU and an NVIDIA RTX A5000 GPU with 512 GB RAM. In all experiments, as baselines, we use AD with the interval domain [Laurel et al. 2022a] and AD with the zonotope domain [Laurel et al. 2022b], which collectively comprise the state of the art for abstracting AD.

Implementation. We implement Pasado in Python, using a combination of the `affapy` library [Helaire et al 2021], the `micrograd` library [Karpathy et al. 2020] and PyTorch [Paszke et al. 2019]. Pasado's implementation assumes ideal real arithmetic and is thus not floating-point sound (though floating-point sound versions of the operations exist [Miné 2004]). While the part of the implementation using `affapy` supports all cases, the part written in PyTorch leverages the specific structure of DNNs and is thus only applicable to DNN benchmarks. For the experiments in Sections 5.2, 5.3, and 5.5 we implemented the baselines ourselves, as there was no existing code-base to use, however for Section 5.4 we used the implementation of [Laurel et al. 2022b] directly.

5.2 Robust Sensitivity Analysis of Ordinary Differential Equations

This first case study involves performing provable sensitivity analysis on the solutions of ODEs. As mentioned in the example section, to perform sensitivity analysis on the numerical solution of any ODE, one must automatically differentiate through the ODE solver. For these experiments we instantiate our technique with forward-mode AD. In our experiments we use a 4th-order Runge-Kutta numerical solver which is more complicated than the Euler method, but a more commonly used and accurate solver in practice. Hence for this evaluation `ODESolve` does not use Euler integration (unlike the example in Section 2). We now examine the following ODEs:

Chemistry ODE. This ODE is taken from Kitchin [2018]; Saltelli et al. [2005] and models the concentration of chemical species C_A as a function of time t. The ODE is parameterized by rate constants k_1 and k_{-1}.

$$\frac{dC_A}{dt} = -k_1 \cdot C_A + k_{-1} \cdot (C_0 - C_A) \tag{12}$$

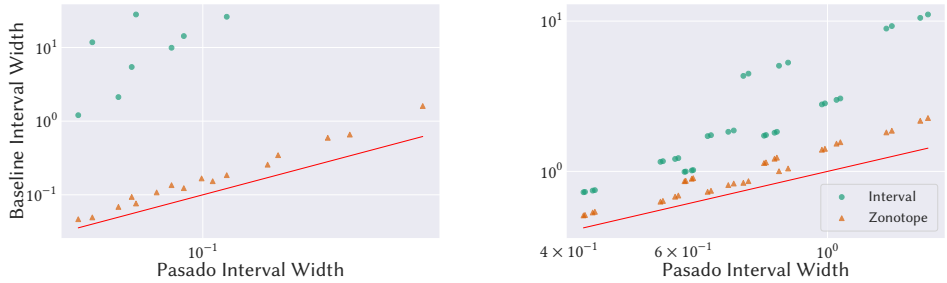


Fig. 5. Scatter plots in *logarithmic* scales comparing the interval widths of the bounds on the derivatives of (a) $\texttt{ODESolve}_{NN}$ with respect to k_1 and (b) $\texttt{ODESolve}_f$ with respect to T_0.

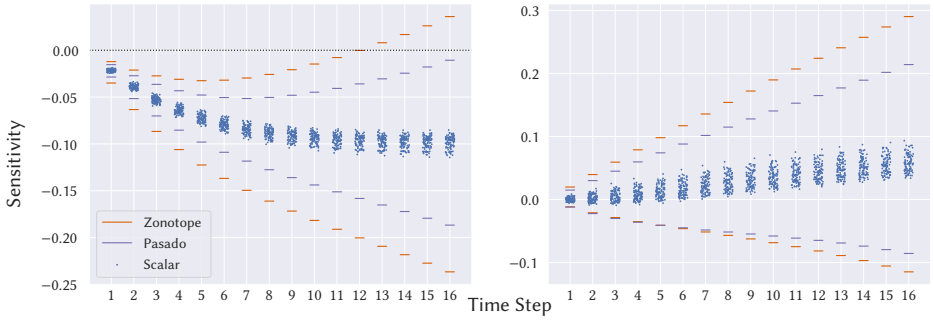


Fig. 6. Bounds on the sensitivities of $\texttt{ODESolve}_{NN}$ with respect to k_1 (left) and k_{-1} (right). Each region between line segments of the same color represents the interval bound computed by that respective method's abstract AD. The dots represent the sensitivities evaluated at points sampled from the input intervals.

However, instead of numerically solving the ODE given in Eq. 12, we train a neural network, NN, to learn the dynamics such that $NN(k_1, k_{-1}, C_0, C_A) \approx -k_1 \cdot C_A + k_{-1} \cdot (C_0 - C_A)$, hence we will actually numerically solve the following *Neural* ODE:

$$\frac{dC_A}{dt} = NN(k_1, k_{-1}, C_0, C_A) \tag{13}$$

The neural ODE approximation produces nearly identical results, hence it serves as a useful surrogate model and also as a representative workload for ODEs where the underlying dynamics are some learned model. In Kitchin [2018]; Saltelli et al. [2005], the authors perform sensitivity analysis on this chemistry model using AD. However, those works computed sensitivities at scalar points only, hence we are the first to abstractly compute this derivative-based sensitivity analysis for *sets* of points. A key reason for performing the sensitivity analysis is to understand how sensitive the final concentration is to the rate constant parameters k_1 and k_{-1}. For this evaluation, we parameterize the ODE solver by the neural network dynamics function NN, thus $\texttt{ODESolve}_{NN}$ denotes a 4^{th} order Runge-Kutta solver for Eq. 13. Thus we abstractly interpret AD to compute precise bounds on both $\frac{\partial}{\partial k_1}\texttt{ODESolve}_{NN}(k_1, k_{-1}, t_0, C_0, C_{A0}, h, n)$ and $\frac{\partial}{\partial k_{-1}}\texttt{ODESolve}_{NN}(k_1, k_{-1}, t_0, C_0, C_A, h, n)$, where h is the step size and n is the number of time steps.

In Fig. 5a, we plot the point $(u_{ours} - l_{ours}, u_{other} - l_{other})$, where the bounds l_{ours} and u_{ours} are with respect to k_1 and obtained from Pasado and l_{other} and u_{other} are the respective lower and upper bounds computed by the other method (regular intervals, regular zonotopes), for each input configuration and for each method. The red line denotes the identity function $y = x$, hence any plotted point that lies *above* the red line signifies that Pasado's bounds were tighter, as a point

will lie above the line if and only if $\frac{u_{other}-l_{other}}{u_{ours}-l_{ours}} > 1$. Furthermore, both the x- and y-axes use logarithmic scales, hence even if points visually appear close together, the difference in precision may be substantial. The input ranges we use to generate this plot are all the 512 combinations of $k_1 = 3 \pm \delta_{k_1}$, $k_{-1} = 3 \pm \delta_{k_{-1}}$, $C_0 = 1 \pm \delta_{C_0}$, $C_A = 1 \pm \delta_{C_A}$, $t_0 = 0$, $n = \{8, 10, 12, 16\}$ and $h \in \{0.025, 0.1\}$, where $\delta_{k_1} \in \{0.05, 0.1, 0.15, 0.2\}$, $\delta_{k_{-1}} \in \{0.1, 0.2, 0.25, 0.3\}$, $\delta_{C_0} \in \{0.1, 0.2\}$, $\delta_{C_A} \in \{0.1, 0.2\}$, which are based on the ranges considered in Saltelli et al. [2005].

Fig. 5a shows that in all input configurations, all the points for the baseline approaches lie above the red line, meaning that Pasado produces the most precise results. In all 512 cases the derivative bound computed with Pasado is strictly contained inside the bound computed via interval AD, and likewise in 497/512 cases the bound computed via Pasado is contained strictly inside the bound computed via zonotope AD (the bounds are incomparable in 15/512 cases). Only 20/512 green points are shown since in the other cases, the interval analysis generates results that are too over-approximate ($> 10^{20}$) to be meaningful. The *geometric mean* of precision improvement over all green points (comparing Pasado to interval AD) is 60.89 times, and the geometric mean of precision improvement over all orange points (comparing Pasado to zonotope AD) is 1.65 times.

We next focus on a specific input configuration for finer granularity. For our specific configuration we use the following ranges to perform the sensitivity analysis of the Chemical ODE: $k_1 \in [2.95, 3.05]$, $k_{-1} \in [2.8, 3.2]$, $t_0 = 0$, $C_0 \in [0.9, 1.1]$, $C_A \in [0.8, 1.2]$, $h = 0.025$, and $n = 16$, again based on the ranges considered in Saltelli et al. [2005]. Fig. 6 illustrates the bounds on these sensitivities for the numerical solution of Eq. 12 with respect to k_1 and k_{-1}, with the x-axis representing the time steps and the y-axis representing the sensitivities. For each color, the upper and lower line segments represent the upper and lower bounds obtained by that method, respectively. We can see from both figures that Pasado always produce narrower bounds compared to the standard zonotope analysis. Additionally, within all the 16 time steps, as is shown in the left subfigure of Fig. 6, Pasado can formally prove the monotonicity of $\texttt{ODESolve}_{NN}$ with respect to k_1 in this configuration (monotonically decreasing since the sensitivity is strictly less than 0), while the zonotope analysis cannot prove the monotonicity after the 12th step.

We measure the performance of the three methods by calculating the average runtimes for one specific input configuration over 16 time steps, since varying input bounds has negligible effects on runtimes. The average runtimes for the interval AD and zonotope AD are 0.12 and 81 seconds, respectively. Pasado takes 47 seconds on average, which is faster than zonotope AD - this improvement directly results from Pasado storing fewer noise symbols, which we found to be the biggest computational bottleneck of the `affapy` library.

Climate ODE. The Climate ODE is the same as in the example section and is taken from Kaper and Engler [2013] and models the global mean temperature through an energy balance model. In Fig. 5b, we plot the points representing $(u_{ours} - l_{ours}, u_{other} - l_{other})$ for the derivative of the numerical solution of Eq. 1 with respect to the initial condition T_0 following the same format as Fig. 5a. The different input configurations we use to generate this plot are all the 32 combinations of $T_0 \in \{337.5 \pm 37.5, 337.5 \pm 62.5\}$, $R = [2.65, 2.95]$, $Q \in \{342, 360 \pm 90\}$, $\alpha \in \{0.35, 0.325 \pm 0.025\}$, $e = [3.402224652 \times 10^{-8}, 5.103336978 \times 10^{-8}]$, $n \in \{8, 12\}$ and $h \in \{0.025, 0.05\}$, which are based on realistic ranges discussed in Walsh [2015].

All of the points for the baseline approaches in Fig. 5b lie above the red line (Pasado), indicating that Pasado yields more accurate results in all cases, and additionally for all 32 of the configurations, the derivative bounds computed by Pasado are strictly contained within the bounds computed via interval AD and zonotope AD. The geometric mean of the precision improvement over all green points (comparing Pasado to interval AD) is approximately 2.85 times, and the geometric mean of the precision improvement over all orange points (comparing Pasado to zonotope AD) is

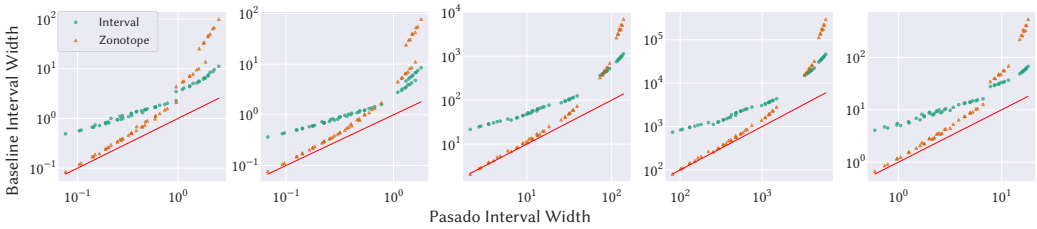


Fig. 7. Scatter plots in *logarithmic* scales comparing the interval widths of the output bounds on the reverse-mode derivatives of the Black-Scholes solution with respect to K, S, σ, τ, and r, from left to right, respectively.

approximately 1.31 times. Additionally, we observe that the benefit gained from Pasado becomes more pronounced for larger input intervals, as demonstrated by the greater vertical distance between the data points and the red line as we move further right along the x-axis.

Fig. 3 in Section 2 represents a plot of a specific input configuration and shows the bounds on the sensitivities of numerical solution of Eq. 1 with respect to the initial condition input T_0, formatted identically to each subfigure of Fig. 6. The specific input configuration used for the plot in Fig. 3 is $T_0 \in [275, 400]$, $R \in [2.65, 2.95]$, $Q = 342$, $\alpha = 0.3$, $e \in [3.402224652 \times 10^{-8}, 5.103336978 \times 10^{-8}]$, $h = 0.025$, and $n = 12$. Again, Pasado produces the most precise bounds among the three methods. Within all the 12 iterations, Pasado can show the monotonicity of the numerical solution with respect to the initial condition T_0.

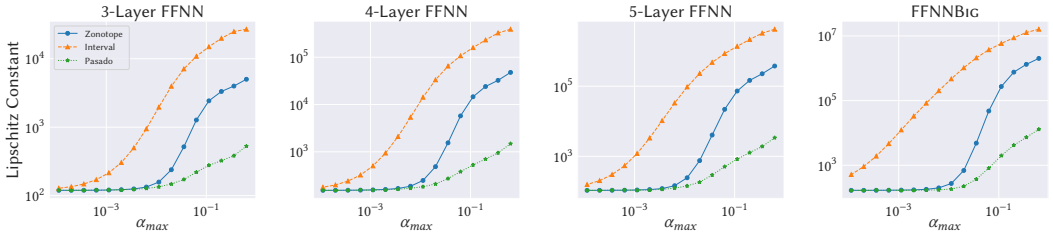
We measure the performance of the three methods by calculating the average runtimes for one specific input configuration over 12 time steps, since varying input bounds has negligible effects on runtimes. The average runtimes for the interval AD, zonotope AD, and Pasado are 0.0044, 0.068, and 0.25 seconds, respectively.

5.3 Black Scholes

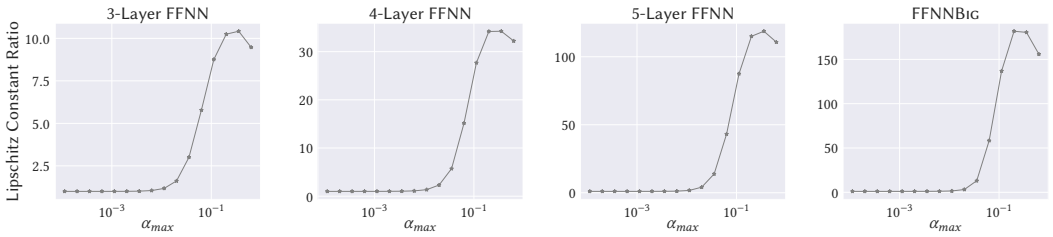
In the next case study, we use our synthesized AD abstractions to compute bounds on the derivatives of the Black-Scholes solution with respect to the different parameters. The Black-Scholes model is a solution to a Partial Differential Equation (PDE) modeling financial option values. The parameters of the Black-Scholes model are volatility σ, time τ, strike price K, spot price S and interest rate r. The derivatives of the Black-Scholes solution with respect to these parameters are commonly known as the "greeks," and while prior work has computed bounds on the greeks [Vassiliadis et al. 2016], only the interval domain was used, hence this experiment demonstrates the precision improvement obtained from using Pasado's synthesized abstract transformers. Since there are multiple inputs but only a single output, we elect for (abstract) reverse-mode AD.

For this experiment we compare our approach against both interval and zonotope abstract AD. We perform the evaluation for different ranges of the parameter values to study how varying the size of the input intervals affects the precision. The input ranges we use are all the 54 combinations of $K = 100 \pm \delta_K$, $S = 105 \pm \delta_S$, $\sigma = 5 \pm \delta_\sigma$, $\tau = 0.08219 \pm \delta_\tau$, $r = 0.0125 \pm \delta_r$, where $\delta_K \in \{1, 5, 10\}$, $\delta_\tau \in \{1, 5, 10\}$, $\delta_\sigma \in \{0.5, 1, 2\}$, $\delta_\tau \in \{0.001, 0.01\}$, $\delta_r = 0.001$. Furthermore, there are 5 greeks and 54 different configurations, hence there are 270 total derivative bounds computed.

We plot the points $(u_{ours} - l_{ours}, u_{other} - l_{other})$ in Fig. 7 for all the derivatives of the Black-Scholes solution with respect to the five "greeks," with the same layout as Fig. 5a. As can be seen, in all cases all points lie above the red line, hence demonstrating that Pasado produces the most precise results. The bounds computed with Pasado are strictly contained inside the bounds computed with zonotope AD in 263/270 cases (rest are incomparable). The geometric mean of the precision improvement over all green points (comparing Pasado to interval AD) is approximately 4.03 times, and the geometric mean of the precision improvement over all orange points (comparing Pasado to



(a) Average upper bounds on the local Lipschitz constants for the interval AD, zonotope AD, and Pasado.



(b) Increase in precisions of Pasado over zonotope AD.

Fig. 8. Lipschitz robustness of FFNNs with 3 (far left), 4 (left-center), or 5 (right-center) affine layers and FFNNBIG (far right) against the haze perturbation on 1000 correctly classified test images. The top row (Fig. 8a) shows the average upper bounds on the local Lipschitz constant with respect to different α_{max} for the interval AD, zonotope AD, and Pasado. The bottom row (Fig. 8b) shows the increase in precision of Pasado, computed as the ratio of the zonotope-bounded Lipschitz constant over the Pasado-bounded Lipschitz constant.

zonotope AD) is approximately 2.81 times. Furthermore, for larger input intervals (further right along the x-axis) the improvement obtained from Pasado becomes more substantial, as evidenced by the higher vertical distance of the points above the line.

We measure the runtime performance of each method by calculating the average runtimes across all input configurations. The average runtimes (per configuration) for the interval AD, zonotope AD, and Pasado are 0.0015, 0.01, and 0.05 seconds, respectively.

5.4 Lipschitz Robustness of Neural Networks

In this case study, we consider the task of computing the local Lipschitz constant of a neural network's output with respect to an adversarial perturbation parameter as in [Laurel et al. 2022a,b]. We study DNNs trained on the MNIST dataset and consider the Haze perturbation given as $p_\alpha(x_i; \alpha) = (1 - \alpha)x_i + \alpha$, where α represents the amount of haze effects and x_i corresponds to the i^{th} input pixel. For the sake of direct comparison, we evaluate on the same range of values for the perturbation parameter α as in Laurel et al. [2022b].

Fig. 8 shows the results of Lipschitz certification for feed-forward neural networks (FFNNs) with 3, 4, and 5 affine layers. The test accuracies for these three FFNNs are 92.7%, 89.2%, and 86.7%, respectively. Fig. 8a illustrates the bounds on the Lipschitz constants, where the x-axis represents the values of α_{max} and the y-axis represents the average upper bounds on the corresponding Lipschitz constants (smaller being preferable) for interval AD, zonotope AD, and Pasado. Fig. 8b exhibits the increase in precision of Pasado over zonotope AD, with the x-axis showing the values of α_{max} and the y-axis showing the *ratio* of the zonotope-bounded Lipschitz constants over the Pasado-bounded Lipschitz constants (larger being preferable). Given that the results produced by zonotope AD and Pasado are orders of magnitude smaller than those from interval AD for $\alpha_{max} > 10^{-2}$, we employ logarithmic scales for both the x- and the y-axes to enable a more visually clear separation between the curves representing zonotope AD and Pasado.

As shown in Fig. 8, there is a significant improvement in precision when using Pasado in comparison to using the interval AD or zonotope AD. In particular, for sufficiently large values of α_{max}, specifically where $\alpha_{max} > 10^{-2}$, the bounds on the Lipschitz constants for 5-layer network that we compute are between $4 - 100\times$ smaller than the bounds computed by the zonotope AD. In contrast, for smaller values of α_{max}, particularly when $\alpha_{max} < 10^{-3}$, Pasado and zonotope AD produce Lipschitz constants of nearly identical magnitudes.

We measure the performance of the three methods by calculating the average runtime across all input configurations used to generate the plots. For the 3-layer network, the average runtimes for the interval AD, zonotope AD, and Pasado are 0.00338, 0.00210, and 0.0636 seconds, respectively. For the 4-layer network, the average runtimes are 0.00383, 0.00352, and 0.0969 seconds, respectively. For the 5-layer network, the average runtimes are 0.00469, 0.00533, and 0.135 seconds, respectively.

The far right subfigure in Fig. 8 shows that Pasado is scalable to a larger FFNN, specifically the FFNNBig architecture from Singh et al. [2019b]. The test accuracy for our FFNNBig instance is 95.8%. As in previous plots, both the x- and the y-axes use logarithmic scales. Compared to the zonotope analysis, Pasado can be up to 182× more precise and the local Lipschitz constants can be up to 2.01×10^6 smaller. The average runtimes for interval AD, Zonotope AD and Pasado on the FFNNBig network are 0.0542, 0.361, and 1.49 seconds, respectively.

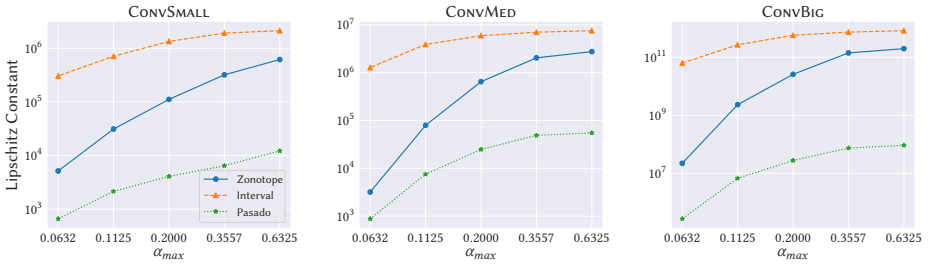
To further illustrate the scalability and versatility of Pasado, we also evaluate it on three convolutional neural networks (CNNs), namely ConvSmall, ConvMed, and ConvBig, as defined in Singh et al. [2018], which are state-of-the-art CNN benchmarks for verification. When the convolutional layers of these networks are unrolled into an equivalent affine layer, the corresponding number of intermediate neurons is more than 25,000, which means Pasado's chain rule abstract transformer will be called that number of times, hence these benchmarks highlight Pasado's scalability.

Fig. 9 presents the results of Lipschitz robustness analysis for ConvSmall, ConvMed, and ConvBig. The test accuracies for these three CNNs are 98.5%, 99.2%, and 98.9%, respectively. As in previous plots, we use logarithmic scales for clarity. Across all three CNNs, Pasado generates much more precise bounds. Notably, in the ConvBig network, Pasado can offer up to 2750× greater precision, and the local Lipschitz constants can be up to 1.99×10^{11} smaller. For ConvSmall, the average runtimes for the interval analysis, zonotope analysis, and Pasado are 3.38, 3.64, and 4.70 seconds, respectively. For ConvMed, the average runtimes are 5.11, 5.70, and 7.27 seconds, respectively. For ConvBig, the average runtimes are 115, 212, and 191 seconds, respectively.

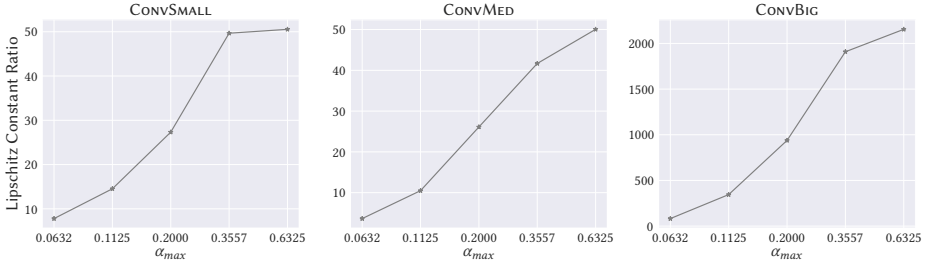
In summary, we observe that as the network architectures become larger, the precision improvements offered by Pasado become more pronounced. Furthermore, in the largest convolutional benchmark, ConvBig, the runtime of Pasado was actually *faster* than the standard zonotope AD analysis baseline. This behavior is the consequence of Pasado generating fewer noise symbols compared to standard zonotope transformers, similar to the observation in the ODE benchmarks of Section 5.2. Hence, for sufficiently large benchmarks (like CNNs), the computational savings obtained by propagating fewer noise symbols (as Pasado does) outweigh the costs (e.g. due to the linear regression) associated with Pasado's more precise abstract transformers.

5.5 Monotonicity Analysis of an Adult Income Network

In this case study, we conduct a monotonicity analysis on a multilayer perceptron (MLP) trained on the Adult dataset [Becker and Kohavi 1996]. Monotonocity has been shown to be highly desirable in order to ensure fairness [Shi et al. 2022; Sivaraman et al. 2020]. Our MLP takes 87 input features (where 81 of the 87 features result from one-hot encodings of the original dataset's categorical variables), passes these features through two hidden layers (each containing 10 neurons and applying tanh activation), and outputs a single binary classification score predicting the income level. Our goal is to verify the monotonicity (both increasing and decreasing) of



(a) Average upper bounds on the local Lipschitz constants for the interval AD, zonotope AD, and Pasado.



(b) Increase in precisions of Pasado over zonotope AD.

Fig. 9. Lipschitz robustness of ConvSmall (left), ConvMed (center), and ConvBig (right) against the haze perturbation on 30 correctly classified test-set images. The top row (Fig. 9a) presents the average upper bounds on the local Lipschitz constant with respect to different α_{max} for the interval AD, zonotope AD, and Pasado. The bottom row (Fig. 9b) presents the increase in precision of the Pasado domain, computed as the ratio of the zonotope-bounded Lipschitz constant over the Pasado-bounded Lipschitz constant.

the MLP's output with respect to 5 continuous input features which are: *Age*, *Education-Num*, *Capital Gain*, *Capital Loss*, and *Hours per week*. Whereas prior work [Shi et al. 2022] varied one feature at a time while holding the value of all other features as fixed, our experiments allow all 5 of the aforementioned continuous features to simultaneously vary within interval bounds. Hence we abstractly interpret the continuous features with a 5D L_∞-ball, with a radius $\epsilon \in [0, 1]$, while holding all the remaining features as fixed. Since training data is normalized to have zero mean and unit variance, passing a 5D L_∞-ball with $\epsilon = 0.4$ through the MLP is equivalent to exploring an infinite set of inputs that satisfy *Age* $\in [33.2, 44.1]$, *Education-Num* $\in [9.05, 11.1]$, *Capital Gain* $\in [-1900, 4060]$, *Capital Loss* $\in [-73.7, 249]$, and *Hours per week* $\in [35.5, 45.4]$. For this analysis, we used Pasado's reverse-mode AD abstract transformers. Hence in a single (abstract) pass, Pasado computes bounds on the partial derivatives of the output with respect to each of the five input features.

For each L_∞-ball radius ϵ, Pasado abstractly computes bounds on the five partial derivatives when the original input is perturbed by the L_∞-ball for 100 different inputs, computing 500 partial derivative bounds in total. In addition, we compare Pasado against interval AD and zonotope AD. For a given input L_∞-ball, to verify monotonicity with respect to a chosen feature, the partial derivative bound with respect to that feature should provably exclude 0, meaning the interval should be strictly positive (monotonically increasing) or strictly negative (monotonically decreasing). This condition ensures that the MLP is monotonic with respect to that feature for *all*

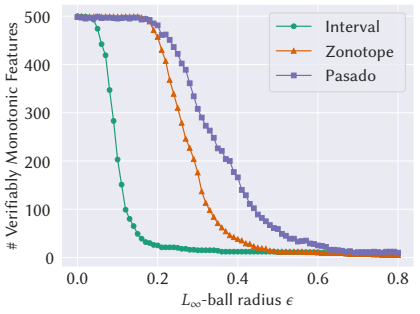


Fig. 10. Counts of verifiably monotonic features of Adult MLP over 100 test-set inputs.

input points in the given L_∞-ball. Hence in Fig. 10, we show the total number of partial derivative bounds that exclude 0 over 100 test inputs, for different-sized L_∞-balls.

Fig. 10 shows that the ability of interval AD to prove monotonicity sharply decreases for $\epsilon \geq 0.05$ due to the inherent imprecision of the interval domain. For small ϵ such as $0 \leq \epsilon \leq 0.2$, zonotope AD and Pasado produce similar counts, meaning both can prove monotonicity. However, their respective performances diverge as ϵ increases. When $0.2 \leq \epsilon \leq 0.6$, the counts for zonotope AD decline rapidly to nearly zero, whereas the counts for Pasado remain high. Hence, in these cases Pasado can prove monotonicity in significantly more instances. For $\epsilon > 0.6$, all three analyses struggle to prove monotonicity for most continuous input features. The average runtimes for the interval AD, zonotope AD, and Pasado are 0.079, 12, and 39 seconds, respectively. In summary, Pasado is able to prove the most monotonicity specifications across all inputs.

6 RELATED WORK

Composite and Synthesized Abstractions. The idea of abstracting a composite numeric expression all at once (as Pasado does) to obtain better precision than sub-optimally composing individual abstract transformers for nonlinear primitives has emerged in the literature. While beneficial for improving precision, abstracting composite expressions is challenging because soundly bounding a composite expression typically involves solving a nonconvex, multivariable optimization problem. As discussed in Fryazinov et al. [2010], these problems typically cannot be solved by hand due to the need to consider interior critical points (as Pasado does). Hence recent works [Ko et al. 2019; Kochdumper et al. 2022; Paulsen and Wang 2022a,b; Ryou et al. 2021; Singh et al. 2019a] have tried to *synthesize* precise composite abstractions by solving an optimization problem (using gradient methods or SMT solvers). However, none of these works target AD, thus none of these techniques can leverage the structure of an AD computation as Pasado does. Further, unlike [Paulsen and Wang 2022a,b] we do not use an SMT solver, as we can solve our optimization problems directly.

Abstract Interpretation of AD. Abstract interpretation has also been applied to AD [Jordan and Dimakis 2021; Laurel et al. 2022a,b; Misra et al. 2023; Vassiliadis et al. 2016], however, all of these works use either the standard interval domain or standard zonotope domain abstract transformers, and compose them naively instead of jointly abstracting multiple AD operations as we do. Furthermore, [Jordan and Dimakis 2021; Laurel et al. 2022a; Vassiliadis et al. 2016] are only formalized for a single abstract domain and thus cannot immediately produce general static analyzers for a more general set of abstract domains as Pasado can. While Laurel et al. [2022b] can be instantiated with different abstract domains, that work uses standard abstract transformers and thus cannot dynamically synthesize new abstractions and only supports forward mode AD.

7 CONCLUSION

We present Pasado, the first technique for synthesizing precise static analyzers tailored specifically to Automatic Differentiation. We show the generality of Pasado by instantiating it for the Product Rule, Quotient Rule and Chain Rule patterns, with the latter supporting multiple different non-linear functions. Pasado's generality also extends to multiple different abstract domains and both forward-mode and reverse-mode AD. Our evaluation on multiple challenging scenarios from machine learning and scientific computing shows that Pasado significantly improves precision compared to prior techniques while simultaneously offering scalability to large computations including CNNs.

ACKNOWLEDGEMENTS

We thank the anonymous reviewers for their comments. This research was supported in part by NSF Grants No. CCF-1846354, CCF-1956374, CCF-2008883, CNS-2148583, CCF-2217144, and CCF-2238079, a gift from Meta, Google research scholar award, and a Sloan UCEM Graduate Scholarship.

DATA-AVAILABILITY STATEMENT

In addition to the github repo https://github.com/uiuc-arc/Pasado, our software and documentation are also available on Zenodo [Laurel et al. 2023b].

REFERENCES

Assalé Adjé, Stéphane Gaubert, and Eric Goubault. 2010. Coupling Policy Iteration with Semi-definite Relaxation to Compute Accurate Numerical Invariants in Static Analysis. In *European Symposium on Programming*.

Sai Praveen Bangaru, Jesse Michel, Kevin Mu, Gilbert Bernstein, Tzu-Mao Li, and Jonathan Ragan-Kelley. 2021. Systematically differentiating parametric discontinuities. *ACM Transactions on Graphics (TOG)* 40, 4 (2021).

Barry Becker and Ronny Kohavi. 1996. Adult. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5XW20.

Claus Bendtsen and Ole Stauning. 1996. FADBAD, a flexible C++ package for automatic differentiation. (1996).

Patrick Cousot and Radhia Cousot. 1977. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*.

Patrick Cousot and Nicolas Halbwachs. 1978. Automatic discovery of linear restraints among variables of a program. In *Proceedings of the 5th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*.

Tianyu Du, Shouling Ji, Lujia Shen, Yao Zhang, Jinfeng Li, Jie Shi, Chengfang Fang, Jianwei Yin, Raheem Beyah, and Ting Wang. 2021. Cert-RNN: Towards Certifying the Robustness of Recurrent Neural Networks.. In *CCS*.

Oleg Fryazinov, Alexander Pasko, and Peter Comninos. 2010. Fast reliable interrogation of procedurally defined implicit surfaces using extended revised affine arithmetic. *Computers & Graphics* 34, 6 (2010).

Khalil Ghorbal, Eric Goubault, and Sylvie Putot. 2009. The zonotope abstract domain taylor1+. In *International Conference on Computer Aided Verification*. 627–633.

Andreas Griewank and Andrea Walther. 2008. *Evaluating derivatives: principles and techniques of algorithmic differentiation*. SIAM.

Akhil Gupta, Lavanya Marla, Ruoyu Sun, Naman Shukla, and Arinbjörn Kolbeinsson. 2021. Pender: Incorporating shape constraints via penalized derivatives. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 35.

Thibault Helaire et al. 2021. affapy library. (2021).

Paul D Hovland, Boyana Norris, Michelle Mills Strout, Sanjukta Bhowmick, and Jean Utke. 2005. Sensitivity analysis and design optimization through automatic differentiation. In *Journal of Physics: Conference Series*.

Jan Hückelheim and Laurent Hascoët. 2022. Automatic differentiation of parallel loops with formal methods. In *Proceedings of the 51st International Conference on Parallel Processing*.

Jan Hückelheim, Ziqing Luo, Sri Hari Krishna Narayanan, Stephen Siegel, and Paul D Hovland. 2018. Verifying Properties of Differentiable Programs. In *International Static Analysis Symposium*. 205–222.

Matt Jordan and Alex Dimakis. 2021. Provable Lipschitz certification for generative models. In *International Conference on Machine Learning*. PMLR, 5118–5126.

Matt Jordan and Alexandros G Dimakis. 2020. Exactly computing the local lipschitz constant of relu networks. *Advances in Neural Information Processing Systems* (2020).

Hans Kaper and Hans Engler. 2013. *Mathematics and climate*. SIAM.

Andrej Karpathy et al. 2020. micrograd library. (2020).

John Kitchin. 2018. A differentiable ODE integrator for sensitivity analysis. (2018).

Ching-Yun Ko, Zhaoyang Lyu, Lily Weng, Luca Daniel, Ngai Wong, and Dahua Lin. 2019. POPQORN: Quantifying robustness of recurrent neural networks. In *International Conference on Machine Learning*. PMLR, 3468–3477.

Niklas Kochdumper, Christian Schilling, Matthias Althoff, and Stanley Bak. 2022. Open-and closed-loop neural network verification using polynomial zonotopes. *arXiv preprint arXiv:2207.02715* (2022).

Jacob Laurel, Siyuan Brant Qian, Gagandeep Singh, and Sasa Misailovic. 2023a. Appendix for Synthesizing Precise Static Analyzers for Automatic Differentiation. (2023). https://jsl1994.github.io/papers/OOPSLA2023_appendix.pdf

Jacob Laurel, Siyuan Brant Qian, Gagandeep Singh, and Sasa Misailovic. 2023b. Artifact for Synthesizing Precise Static Analyzers for Automatic Differentiation. (2023). https://doi.org/10.5281/zenodo.8332724

Jacob Laurel, Rem Yang, Gagandeep Singh, and Sasa Misailovic. 2022a. A Dual Number Abstraction for Static Analysis of Clarke Jacobians. *Proceedings of the ACM on Programming Languages* POPL (2022), 1–30.

Jacob Laurel, Rem Yang, Shubham Ugare, Robert Nagel, Gagandeep Singh, and Sasa Misailovic. 2022b. A general construction for abstract interpretation of higher-order automatic differentiation. *Proceedings of the ACM on Programming Languages* 6, OOPSLA2 (2022), 1007–1035.

Samuel Lerman, Charles Venuto, Henry Kautz, and Chenliang Xu. 2021. Explaining Local, Global, And Higher-Order Interactions In Deep Learning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 1224–1233.

Yingbo Ma, Vaibhav Dixit, Michael J Innes, Xingjian Guo, and Chris Rackauckas. 2021. A comparison of automatic differentiation and continuous sensitivity analysis for derivatives of differential equation solutions. In *2021 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, 1–9.

Azamat Mametjanov, Boyana Norris, Xiaoyan Zeng, Beth Drewniak, Jean Utke, Mihai Anitescu, and Paul Hovland. 2012. Applying automatic differentiation to the Community Land Model. In *Recent Advances in Algorithmic Differentiation*.

Antoine Miné. 2004. Relational abstract domains for the detection of floating-point run-time errors. In *European Symposium on Programming*. 3–17.

Ashitabh Misra, Jacob Laurel, and Sasa Misailovic. 2023. ViX: Analysis-driven Compiler for Efficient Low-Precision Variational Inference. In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems* 32 (2019).

Brandon Paulsen and Chao Wang. 2022a. Example Guided Synthesis of Linear Approximations for Neural Network Verification. In *International Conference on Computer Aided Verification*.

Brandon Paulsen and Chao Wang. 2022b. LinSyn: Synthesizing Tight Linear Bounds for Arbitrary Neural Network Activation Functions. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. 357–376.

Harrison Rosenberg, Brian Tang, Kassem Fawaz, and Somesh Jha. 2023. Fairness properties of face recognition and obfuscation systems. In *32nd USENIX Security Symposium (USENIX Security 23)*.

Wonryong Ryou, Jiayu Chen, Mislav Balunovic, Gagandeep Singh, Andrei Dan, and Martin Vechev. 2021. Scalable polyhedral verification of recurrent neural networks. In *International Conference on Computer Aided Verification*. 225–248.

Andrea Saltelli, Marco Ratto, Stefano Tarantola, and Francesca Campolongo. 2005. Sensitivity analysis for chemical models. *Chemical reviews* 105, 7 (2005), 2811–2828.

Zhouxing Shi, Yihan Wang, Huan Zhang, Zico Kolter, and Cho-Jui Hsieh. 2022. Efficiently Computing Local Lipschitz Constants of Neural Networks via Bound Propagation. In *Advances in Neural Information Processing Systems*.

Zhouxing Shi, Huan Zhang, Kai-Wei Chang, Minlie Huang, and Cho-Jui Hsieh. 2020. Robustness Verification for Transformers. In *ICLR*.

Gagandeep Singh, Rupanshu Ganvir, Markus Püschel, and Martin Vechev. 2019a. Beyond the single neuron convex barrier for neural network certification. *Advances in Neural Information Processing Systems* 32 (2019).

Gagandeep Singh, Timon Gehr, Matthew Mirman, Markus Püschel, and Martin T Vechev. 2018. Fast and Effective Robustness Certification. *NeurIPS* 1, 4 (2018), 6.

Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin Vechev. 2019b. An abstract domain for certifying neural networks. *Proceedings of the ACM on Programming Languages* 3, POPL (2019), 1–30.

Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin Vechev. 2019c. Boosting robustness certification of neural networks. In *International conference on learning representations*.

Gagandeep Singh, Markus Püschel, and Martin Vechev. 2017. Fast polyhedra abstract domain. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages*.

Aishwarya Sivaraman, Golnoosh Farnadi, Todd Millstein, and Guy Van den Broeck. 2020. Counterexample-guided learning of monotonic neural networks. *Neural Information Processing Systems* (2020).

Jorge Stolfi and Luiz Henrique De Figueiredo. 1997. Self-validated numerical methods and applications. In *Monograph for 21st Brazilian Mathematics Colloquium, IMPA, Rio de Janeiro. Citeseer*, Vol. 5. Citeseer.

James Paul Turner. 2020. *Analysing and Bounding Numerical Error in Spiking Neural Network Simulations*. Ph. D. Dissertation. University of Sussex.

Vassilis Vassiliadis, Jan Riehme, Jens Deussen, Konstantinos Parasyris, Christos D Antonopoulos, Nikolaos Bellas, Spyros Lalis, and Uwe Naumann. 2016. Towards automatic significance analysis for approximate computing. In *2016 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*. IEEE.

James Walsh. 2015. Climate modeling in differential equations. *The UMAP Journal* 36, 4 (2015), 325–363.

Yu Wang, Qitong Gao, and Miroslav Pajic. 2022. Learning Monotone Dynamics by Neural Networks. In *2022 American Control Conference (ACC)*. IEEE, 1485–1490.

Yuting Yang, Connelly Barnes, Andrew Adams, and Adam Finkelstein. 2022. A δ: autodiff for discontinuous programs-applied to shaders. *ACM Transactions on Graphics (TOG)* 41, 4 (2022).