# Checking $\delta$-Satisfiability of Reals with Integrals

CODY RIVERA, University of Illinois at Urbana-Champaign, USA
BISHNU BHUSAL, University of Missouri, USA
ROHIT CHADHA, University of Missouri, USA
A. PRASAD SISTLA, University of Illinois at Chicago, USA and Discovery Partners Institute, USA
MAHESH VISWANATHAN, University of Illinois at Urbana-Champaign, USA

Many synthesis and verification problems can be reduced to determining the truth of formulas over the real numbers. These formulas often involve constraints with integrals in them. To this end, we extend the framework of $\delta$-decision procedures with techniques for handling integrals of user-specified real functions. We implement this decision procedure in the tool $\int$dReal, which is built on top of dReal. We evaluate $\int$dReal on a suite of problems that include formulas verifying the fairness of algorithms and the privacy and the utility of privacy mechanisms and formulas that synthesize parameters for the desired utility of privacy mechanisms. The performance of the tool in these experiments demonstrates the effectiveness of $\int$dReal.

CCS Concepts: • **Theory of computation → Logic and verification**; • **Security and privacy → Logic and verification**; • **Mathematics of computing → Integral equations**; • **Software and its engineering → Model checking**.

Additional Key Words and Phrases: $\delta$-satisfiability, theory of reals, integrals, verification, synthesis.

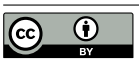
**ACM Reference Format:**
Cody Rivera, Bishnu Bhusal, Rohit Chadha, A. Prasad Sistla, and Mahesh Viswanathan. 2025. Checking $\delta$-Satisfiability of Reals with Integrals. *Proc. ACM Program. Lang.* 9, OOPSLA1, Article 105 (April 2025), 29 pages. https://doi.org/10.1145/3720446

## 1 Introduction

Verification of systems and synthesis of parameters/programs can often be reduced to the problem of determining the truth of first order logic sentences over fixed structures. For application domains like probabilistic programming, machine learning, cyberphysical systems, or security and privacy of data processing, the resulting formulas require reasoning about real numbers. Moreover, the formulas often involve exponentials, trignometric and hyperbolic functions, and integrals of probability density functions involving exponentials. Often expressions involving integrals, like the cumulative distribution function of a Gaussian/Normal distribution, do not even have closed form expressions.

The decidability of the first order theory of reals with functions such as exponentials is a long standing open problem at the interface of logic and transcendental number theory. However, it turns out that such sentences and those involving other more complicated functions and integrals can be "approximately" decided [11]. Such approximate decision procedures are called $\delta$-decision procedures and they have the following properties. Given a sentence $\varphi$ over the reals where all

Authors' Contact Information: Cody Rivera, University of Illinois at Urbana-Champaign, Urbana, USA, codyjr3@illinois.edu; Bishnu Bhusal, University of Missouri, Columbia, USA, bhusalb@mail.missouri.edu; Rohit Chadha, University of Missouri, Columbia, USA, chadhar@missouri.edu; A. Prasad Sistla, University of Illinois at Chicago, Chicago, USA and Discovery Partners Institute, Chicago, USA, sistla@uic.edu; Mahesh Viswanathan, University of Illinois at Urbana-Champaign, Urbana, USA, vmahesh@illinois.edu.




ACM 2475-1421/2025/4-ART105
https://doi.org/10.1145/3720446

quantified variables are restricted to take values in a closed interval, and a parameter $\delta > 0$, we construct another sentence called the "$\delta$-weakening" $\varphi^\delta$ of $\varphi$. Intuitively, $\varphi^\delta$ is a formula obtained by perturbing the constants in $\varphi$ by $\delta$. On input $\varphi$, a $\delta$-decision procedure either answers "unsat" to indicate that $\varphi$ is not true over the reals, or "$\delta$-sat" to indicate that $\varphi^\delta$ is true. It thus solves the decision problem approximately in the following sense — if $\varphi$ is true, then the procedure is guaranteed to answer "$\delta$-sat"; if both $\varphi$ and $\varphi^\delta$ are not true then the procedure is guaranteed to answer "unsat"; but when $\varphi$ is not true and $\varphi^\delta$ is true, the procedure may give either answer without violating its promise.

The framework of $\delta$-decision procedures has been incredibly effective in analyzing a variety of systems [4, 28]. dReal [12] is a robust implementation of these algorithms for special formulas that arise in the verification of cyberphysical systems. In cyberphysical systems, the system state is often described by a set of differential equations where the derivative is with respect to time, and verification involves reasoning about the state as it evolves over time. Hence, dReal assumes that all derivatives are with respect to a common variable (time) and there is no support for integrals over different variables (nested or otherwise). These restrictions pose significant roadblocks when analyzing probabilistic programs, privacy mechanisms, or machine learning algorithms.

### 1.1 Contributions

In this paper, we introduce a logic that we call $\mathcal{L}_{int}$ over reals in which formulas can be built using exponentials, logarithms, trigonometric and hyperbolic functions, and integrals. Integrals in the logic are treated as first class objects, and formulas can involve integrals of expressions over multiple variables. The logic allows terms involving nested integrals. Moreover, expressions involving integrals need not have closed-form expressions. We show that the $\delta$-satisfiability problem for such logic is decidable, and we present a $\delta$-decision procedure based on Interval Constraint Propagation (ICP) for $\mathcal{L}_{int}$.

We have implemented the algorithm in a tool called $\int$dReal (read "integral dee real") which extends dReal. The key ingredient in dReal is *Interval Constraint Propagation (ICP)* [5], which attempts to find a sufficiently small box — a tuple of intervals, one for each existentially-quantified variables — that satisfies a given set of atomic real constraints [5]. The algorithm works via a "branch-and-prune" procedure, where pruning tries to shrink this box according to the properties of the constraints themselves, and if pruning does not work, branching divides the box into two smaller subboxes, and generates a subproblem for each subbox.

Extending ICP to handle integration as implemented in $\int$dReal involves the following key innovations and engineering: (a) Developing forward and backward pruning algorithms for integral expressions, and (b) Integrating Arb [16], a library which provides arbitrary precision support for integration, into IBEX [15], the library dReal uses to implement interval constraint propagation. In developing pruning algorithms for integral expressions, we found that, although we can use a straightforward algorithm for forward pruning, the backward pruning algorithm required some engineering trade-offs. In particular, we do not perform backward pruning if the box to be pruned is too wide. We demonstrate that the resulting pruning procedure satisfies all the "good" properties required for the ICP algorithm.

We mention some salient points regarding the engineering challenges of extending dReal's ICP library, IBEX, to handle integrals using Arb, and our solutions to those. A core operation used in ICP is to compute an interval enclosure for a function $f$ over Reals containing all possible outputs, given intervals containing all possible inputs. IBEX implements this operation by first converting $f$ into an expression graph, and then traversing the graph and computing an interval enclosure for each subexpression. We add support for integral subexpressions by extending the expression graph data

structure to handle such subexpressions and use Arb to produce ball enclosures (sets represented by a midpoint and a radius, see Section 5.1 for more details) containing all of their possible outputs. When an integral is encountered, we first compute enclosures around the limits, and then invoke Arb to compute the integral. The auxiliary function provided allows Arb to compute the values of the integrand at particular points by evaluating an arbitrary IBEX expression graph. We implement this in a way that allows integrals to be nested arbitrarily.

We believe that the value in adding support for terms with integration is its application in a range of examples, to be described shortly. Additionally, a few of our examples contain ∃-∀ quantifier alternation, and we are unaware of any tool that checks the satisfiability of formulas with these features together. We leave addition of other operations such as differentiation to future work. But adding this support would require care: for example, $|x|$ is not differentiable at zero.

We demonstrate the effectiveness of our algorithm and tool by studying its performance on a number of examples drawn from probabilistic programming, security, and machine learning, as well as artificially created formulas. We compare its running time against the state of the art industrial tool Mathematica®, and show that on several examples we get at least a 2X speedup; we do not compare against dReal because it cannot handle the examples we study. Mathematica® and $\int$dReal is not a like-for-like comparison — the former attempts to solve the precise satisfiability problem while the latter is an approximate decision procedure. However, Mathematica® is the only tool (exact or otherwise) we are aware of that is capable of handling the formulas with integrals that we consider in our experiments. In our experiments, we found that, even though $\int$dReal implements a $\delta$-decision procedure, its answers are almost always correct with respect to the exact satisfiability problem. The speedup (with respect to Mathematica®) reported in our experiments should be interpreted as evidence that suggests that $\delta$-decision procedures can be useful as they show that even a prototype tool can be as effective as a mature, heavily engineered industry standard.

We highlight a few observations regarding the case studies used in our benchmarks. These case studies involve verifying fairness claims of decision trees, and privacy and accuracy claims of differential privacy. Our first observation is that the formulas generated demonstrate the need for nested integrals and integrals whose limits may be real expressions (and not just constants). Secondly, we show that our methods can be used to perform a more involved analysis. We highlight two such instances. We consider the fairness of a machine-learning model (See Section 2.1). This example is adapted from FairSquare [2]. FairSquare checks for the fairness of the model, assuming a fixed population model. We show that our tool allows us to verify fairness claims even if the parameters are not known to a certainty, but belong to a range. In another instance, we show how our methods can synthesize privacy algorithm parameters to obtain the desired accuracy/utility (See Section 2.2).

### 1.2 Outline of the Paper

The rest of the paper is organized as follows. We conclude this section with a short discussion of closely related work. In Section 2, we present some example verification and synthesis problems that motivate the need for tools that can check the satisfiability of first order formulas over reals that involve integrals. Section 3 formally introduces the problem of $\delta$-satisfiability and provides an overview of interval constraint propagation (ICP), the framework on which our algorithm is based. Our logic $\mathcal{L}_{int}$ is formally introduced in Section 4. Our $\delta$-decision procedure is also presented in Section 4. Section 5 discusses the implementation of our algorithm in the tool $\int$dReal. Some examples used in our experiments are presented in Section 6, and this is followed by our experimental results in Section 7.

### 1.3 Related Work

The verification of hybrid systems can naturally be reduced to the satisfiability question of first order logic over reals. Hybrid systems describe the discrete and continuous behavior where real-valued variables evolve continuously with time, and periodically, there are mode changes that influence the physical laws governing the evolution of the real-valued variables. The continuous dynamics of such systems are often described using ODEs, and checking the safety or reachability of such systems involves solving ODEs. Over the years, a number of tools and approaches have been proposed to handle the first order logic formulas that arise from the safety verification problem of hybrid systems. HSolver [22] checks the satisfiability of formulas arising from hybrid system verification using interval abstraction and progressive refinement. iSAT-ODE [10] checks formulas involving ODEs using an enclosure method for ODEs and monotonicity. Finally, dReach [19] is a tool that analyzes safety for hybrid systems by encoding the hybrid system reachability problem as a first order formula over reals and then calling dReal. Though all these tools present algorithms to check the satisfiability of formulas where some terms are integrals since they target hybrid systems, the integrations in the formulas are over only one variable (time) and, therefore, are not nested. The integrals also do not have expressions as bounds; all integral bounds are numbers. Further, HSolver and iSAT-ODE only handle polynomials and do not have any transcendental functions like exponentials or trigonometric functions. Because of these restrictions, none of these approaches can handle the class of formulas we consider in this paper.

## 2 Motivating Examples

Verification of systems and synthesis of parameters/programs can often be reduced to the problem of determining the truth of first-order logic sentences over fixed structures. In many applications, the fixed structure turns out to be real numbers. We present a couple of motivating examples that demonstrate the need for algorithms that can reason about formulas involving complicated integrations. Programs in our examples will sample from the Gaussian (or Normal) distribution, which we write as $\mathcal{N}(\mu,\ \sigma)$, where $\mu$ and $\sigma$ represent the mean and standard deviation respectively. Recall that the probability density function of the Gaussian distribution is given by the function

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}. \tag{1}$$

Before presenting the examples, we introduce another piece of notation that we use. For events $a$ and $b$, $\mathbb{P}[a]$ denotes the probability of the event $a$, and $\mathbb{P}[a|b]$ denotes the conditional probability of $a$ given $b$.

### 2.1 Verification of Fairness Properties

We present a simplified case study adapted from FairSquare [2]. Consider the program `dec` shown in Figure 1 on the right. `dec` is a program that makes a hiring decision based on a job applicant's college ranking (`colRank`) and years of experience (`yExp`). The goal of the program is to ensure fairness in hiring decisions. The core of the program's decision-making process is a decision tree, potentially derived through machine learning. An applicant is deemed suitable for hire if they either attended a top-5 college (`colRank <= 5`) or possess substantial experience relative to their college's ranking (`expRank > -5`). Importantly, the program `dec` does not factor ethnicity in its decision-making process.

A probabilistic model representing a basic population model is given as program `popModel` shown on the left in Figure 1. Each member of the population possesses three real-valued attributes: `ethnicity`, `colRank`, and `yExp`. The sensitive condition defines a member as part of a protected group if their `ethnicity` value exceeds 10 (`ethnicity > 10`) and increases the `colRank` by 5. The

```
1 define popModel(μc, σc)
2   ethnicity ~ N(0,10)
3   colRank ~ N(μc, σc)
4   yExp ~ N(10,5)
5   if (ethnicity > 10)
6     colRank ⟵ colRank + 5
7   return colRank, yExp
```

```
1 define dec(colRank, yExp)
2   expRank ⟵ 5*yExp - colRank
3   if (colRank <= 5)
4     hire ⟵ true
5   elif (expRank > -5)
6     hire ⟵ true
7   else
8     hire ⟵ false
9   return hire
```

Fig. 1. On the left, a program for generating a population model. On the right is a program for decision-making in hiring an employee.

population model intuitively offers a probabilistic representation of the population, serving as the source from which inputs for the program `dec` are drawn. In our illustrative scenario, `popModel` is parameterized by $\mu_c$ and $\sigma_c$ representing the mean and standard deviation, respectively, of the distribution for `colRank`.

Our goal is to check that `dec` does not discriminate against the protected group. Informally, this means that the probability of hiring someone from the protected group is not much lower than the probability of hiring someone not from the protected group. This is formally captured by the following condition.

$$\mathbb{P}[\texttt{hire}|\texttt{ethnicity} > 10] > (1-\epsilon)\mathbb{P}[\texttt{hire}|\texttt{ethnicity} \leq 10] \tag{2}$$

In this context, $\epsilon$ represents a small constant. The left and right-hand sides of (2) can be written as follows (here the random variables `ethnicity`, `yExp` and `colRank` correspond to the values of the variables `ethnicity`, `yExp` and `colRank` at end of line 4 in the population model given in Figure 1).

$$\mathbb{P}[\texttt{hire}|\texttt{ethnicity} > 10] = \mathbb{P}[\texttt{colRank} < 0] + \mathbb{P}[\texttt{yExp} > \frac{\texttt{colRank}}{5} \text{ and } \texttt{colRank} > 0] \tag{3}$$

$$\mathbb{P}[\texttt{hire}|\texttt{ethnicity} \leq 10] = \mathbb{P}[\texttt{colRank} < 5] + \mathbb{P}[\texttt{yExp} > \frac{(\texttt{colRank}-5)}{5} \text{ and } \texttt{colRank} > 5] \tag{4}$$

Thus, in the inequality (2), we replace the conditional probabilities with the sums of unconditional probabilities as given by equations (3) and (4). In the resulting inequality, we replace the unconditional probabilities with probability expressions involving integrals over the appropriate Gaussian density functions, given by (1), to obtain the following inequality.

$$\begin{aligned} &\frac{1}{\sigma_c\sqrt{2\pi}}\left[\int_{-\infty}^{0} e^{\frac{-(x-\mu_c)^2}{2{\sigma_c}^2}}dx + \int_{0}^{\infty}\int_{\frac{x}{5}}^{\infty}\frac{1}{5\sqrt{2\pi}}e^{\frac{-(y-10)^2}{50}}e^{\frac{-(x-\mu_c)^2}{2{\sigma_c}^2}}dydx\right] \\ &> (1-\epsilon)\frac{1}{\sigma_c\sqrt{2\pi}}\left[\int_{-\infty}^{5} e^{\frac{-(x-\mu_c)^2}{2{\sigma_c}^2}}dx + \int_{5}^{\infty}\int_{\frac{x-5}{5}}^{\infty}\frac{1}{5\sqrt{2\pi}}e^{\frac{-(y-10)^2}{50}}e^{\frac{-(x-\mu_c)^2}{2{\sigma_c}^2}}dydx\right] \end{aligned} \tag{5}$$

In the above inequality, the first and second summands on the left hand side, correspond to $\mathbb{P}[\texttt{colRank} < 0]$ and $\mathbb{P}[\texttt{yExp} > \frac{\texttt{colRank}}{5} \text{ and } \texttt{colRank} > 0]$, respectively; similarly the first and second summands on the right hand side, correspond to $\mathbb{P}[\texttt{colRank} < 5]$ and $\mathbb{P}[\texttt{yExp} > \frac{(\texttt{colRank}-5)}{5} \text{ and } \texttt{colRank} > 5]$, respectively. It is worth noting that the condition for checking fairness involves exponentials, integrals, nested integrals, and integral bounds that are not only from the extended reals but could also be expressions with variables.

$\delta$-decision procedures rely on variables and integration bounds being bounded. Thus, we would like to replace the $\infty$ and $-\infty$ in the upper and lower bounds in equation (5) with appropriately chosen finite values. Since integrals involving these bounds correspond to terms arising from the Gaussian distribution, we can use Gaussian tail bounds for this purpose. For example, we can use the concentration bound

$$\mathbb{P}[|X-\mu|\geq t] \leq 2e^{-\frac{t^2}{2\sigma^2}} \tag{6}$$

where $X$ is a Gaussian random variable with mean $\mu$ and standard deviation $\sigma$. `colRank` has mean and standard deviation $\mu_c$ and $\sigma_c$, respectively. When we replace the lower limit of $-\infty$ and upper limit of $\infty$, of the integral over this random variable, by $\mu_c - k$ and by $\mu_c + k$, respectively, we get a tail error bound $\xi$, given below.

$$\xi \sim 2e^{-\frac{k^2}{2{\sigma_c}^2}} \tag{7}$$

When we consider $k$ to be $4\sigma_c$, $\xi$ becomes approximately $\frac{7}{10000}$.

Next, observe that for the random variable `yExp`, the mean is 10 and standard deviation is 5. We also replace the lower limit of $-\infty$ and upper limit of $\infty$, of integrals over this random variable by $10-k$ and $10+k$. For $\sigma_c \geq 5$, the error bound for integrals over this random variable is no more than $\xi$.

Using these Gaussian tail bounds, we replace $\infty$ and $-\infty$ bounds in (5) by finite values. Our goal is to identify a new condition, which if it holds would imply the satisfaction of (5). To accomplish this, we ignore the error bounds that arise when the left hand side of the inequality is modified, and we simply add an appropriate error bound on the right hand side; in this case this error on the right side turns out to be at most $2\xi$. Based on these observations, we get the following new condition.

$$\begin{aligned}\phi ::= &\left[\frac{1}{\sigma_c\sqrt{2\pi}}\left[\int_{\mu_c-k}^{0} e^{\frac{-(x-\mu_c)^2}{2{\sigma_c}^2}}dx + \int_{0}^{\mu_c+k}\int_{\frac{x}{5}}^{10+k}\frac{1}{5\sqrt{2\pi}}e^{\frac{-(y-10)^2}{50}}e^{\frac{-(x-\mu_c)^2}{2{\sigma_c}^2}}dydx\right] > \right.\\ &\left.(1-\epsilon)\frac{1}{\sigma_c\sqrt{2\pi}}\left[\int_{\mu_c-k}^{5} e^{\frac{-(x-\mu_c)^2}{2{\sigma_c}^2}}dx + \int_{5}^{\mu_c+k}\int_{\frac{x-5}{5}}^{10+k}\frac{1}{5\sqrt{2\pi}}e^{\frac{-(y-10)^2}{50}}e^{\frac{-(x-\mu_c)^2}{2{\sigma_c}^2}}dydx + 2\xi\right]\right]\end{aligned} \tag{8}$$

It is easy to see that if (8) holds then (5) also holds. This is what we desire. Note that the previous calculations must be performed by the user and are not automatically performed by $\int$dReal.

Verifying the fairness of `dec` with respect to the population model `popModel` is done by running satisfiability queries based on $\phi$ in equation (8). One verification problem is to check if $\phi$ holds for specific values of $\mu_c$ and $\sigma_c$. The second verification problem is to check that $\phi$ holds for all values of $\mu_c$ and $\sigma_c$ that lie in a prescribed range, i.e., `dec` is unfair for all values of $\mu_c$ and $\sigma_c$ that lie in some range. Our $\delta$-decision procedure can handle such queries. In our experiments, to establish validity and thus fairness, we check that the negated $\phi$ is unsatisfiable. Specifically, we consider

(1) (eth_colrank_fair_00): For $\epsilon = 0.1$, $\mu_c = 25$, $\sigma_c = 10$, check if $\neg\phi$ is unsatisfiable.
(2) (eth_colrank_fair_01): For $\epsilon = 0.1$, check if $\exists\mu_c \in [20, 30]$. $\exists\sigma_c \in [10, 15]$. $\neg\phi$ is unsatisfiable. In the experiments, we replace $\mu_c - k$ in lower limits by its lowest possible value, $20 - 4 * 15$, and $\mu_c + k$ by its highest possible value $30 + 4 * 15$.

The verification problem described by (eth_colrank_fair_00) has been checked before in [2]. However, more complicated queries, like checking fairness for a range as expressed by (eth_colrank_fair_01), have not. Our tool is able to handle such formulas.

### 2.2 Synthesis of Parameters for Accuracy of a Privacy Mechanism

Given a threshold `threshold`, and a sequence of numbers, suppose we are interested in determining the first position in the sequence of numbers that exceeds `threshold`. One program accomplishing this task may output $\perp$ each time the number read is less than `threshold`, and output $\top$ and

halt when the first number exceeding `threshold` is encountered. Suppose we want to solve this problem in a manner that ensures the privacy of the numbers read, what should we do? Sparse Vector Technique (SVT) [3, 8, 21] is an algorithm that solves this problem while ensuring differential privacy. The idea behind this algorithm is simple. Instead of comparing the read input with the threshold, it compares a noisy version of the input against a noisy version of `threshold`. But how much noise should we add? Clearly, if we add a lot of noise, privacy can be easily guaranteed since the output of the algorithm may have little correlation with the true answer. The key is to add just enough noise so that privacy can be guaranteed, while at the same time answers can be "accurate". This turns out to be a subtle problem and there are many examples of proposed algorithms that have been shown to be incorrect [21].

The problem of identifying the right amount of noise in SVT is a synthesis problem and one can reduce it to the satisfaction of a first order formula. For simplicity, let us consider SVT for the case when the input sequence is of length 1. The algorithm adds Gaussian noise, controlled by privacy budget $\epsilon$ and the parameter $a$, to both the threshold and the input, compares them and outputs $\top$ is the noisy input is larger than the noisy threshold and outputs $\bot$ otherwise. This is shown on the left in Figure 2.

```
1 define SVT(input)
2   noisy_th ~ 𝒩(threshold, 1/(aϵ))
3   noisy_input ~ 𝒩(input, 1/((1−a)ϵ))
4   if (noisy_input ≥ noisy_th)
5     return ⊤
6   else
7     return ⊥
```

```
1 define detSVT(input)
2   if (input ≥ threshold)
3     return ⊤
4   else
5     return ⊥
```

Fig. 2. On the left Sparse Vector Technique (SVT). On the right deterministic variant of the SVT

In order to say if this algorithm outputs "useful", "correct" results, we need to compare its output with that of a program that solves the original problem without attempting to ensure privacy. In other words, compare it with the algorithm that does not add any noise during its execution. We call this later algorithm `detSVT` and it is shown on the right in Figure 2.

In this simple case of 1-input SVT, accuracy of SVT on input ($\mathcal{U}$) is measured by

$$\gamma(\mathcal{U}) ::= \mathbb{P}[\texttt{SVT}(\mathcal{U}) = \texttt{detSVT}(\mathcal{U})] \tag{9}$$

Suppose `threshold` is 0 and the `input` ($\mathcal{U}$) is 1, then `detSVT` outputs $\top$, i.e. `detSVT`(1) = $\top$. Hence $\gamma(1) = \mathbb{P}[\texttt{SVT}(1) = \top]$. Using standard probability analysis, it is easy to see that $\gamma(1)$ is given by the following equation.

$$\gamma(1) ::= \frac{a(1-a)\epsilon^2}{2\pi} \int_{-\infty}^{\infty} \int_{x}^{\infty} e^{-\frac{a^2\epsilon^2x^2}{2}} e^{-\frac{(1-a)^2\epsilon^2(y-1)^2}{2}} dydx \tag{10}$$

Now, we outline a simple method to check if $\gamma(1) > 0.5$ using $\int$dReal. To do this, we chose a constant $k > 0$ and replace limits $-\infty$ and $\infty$ of the integrals in the expression for $\gamma(1)$ by $-k$ and $k$, respectively. Let the resulting expression be denoted by $\gamma'(1)$, i.e.,

$$\gamma'(1) ::= \frac{a(1-a)\epsilon^2}{2\pi} \int_{-k}^{k} \int_{x}^{k} e^{-\frac{a^2\epsilon^2x^2}{2}} e^{-\frac{(1-a)^2\epsilon^2(y-1)^2}{2}} dydx \tag{11}$$

Clearly $\gamma'(1) < \gamma(1)$. Hence if $\gamma'(1) > 0.5$ then $\gamma(1) > 0.5$. We can check the former condition using $\int$dReal. Consider a small constant $\delta > 0$ and check if $\gamma'(1) - 0.5 - \delta > 0$ is $\delta-$satisfiable. We do this by passing $\delta$ as the parameter to $\int$dReal. If the tool says that it is $\delta$-satisfiable, then it means $-\delta < \gamma'(1) - 0.5 - \delta < \delta$. This implies that $\gamma'(1) > 0.5$. In this procedure, we fix the parameters $a, \epsilon$ and $k$.

Now, we outline a procedure for synthesizing a value for parameter $a$ so that $\gamma'(1) > 0.5$ (and hence $\gamma(1) > 0.5$), for all values of $\epsilon$ in a given range. For this, consider the formula $\phi(a, \epsilon, \delta, k)$ as given below.

$$\phi(a, \epsilon, \delta, k) ::= \frac{a(1-a)\epsilon^2}{2\pi} \int_{-k}^{k} \int_{x}^{k} e^{-\frac{a^2\epsilon^2x^2}{2}} e^{-\frac{(1-a)^2\epsilon^2(y-1)^2}{2}} dydx - 0.5 - \delta > 0 \tag{12}$$

Now, we construct a $\exists$-$\forall$ query where $\exists$ quantifies over the parameter $a$ and $\forall$ quantifies over the parameter $\epsilon$. The required query denoted as gauss_forall_00 is given below:

$$\exists a \in [0.25, 0.75], \forall \epsilon \in [0.5, 0.9], \phi(a, \epsilon, \delta, k).$$

The query is run in $\int$dReal, using a constant $\delta$ which is also passed as the parameter for $\delta$-satisfiability to the tool. We also use a constant $k$, which much be sufficiently large to ensure $\gamma'(1) > 0.5$; here we set $k = 20$. The tool answered saying the above formula is $\delta$-satisfiable; and returned a small interval strictly contained in the interval $[0.25, 0.75]$ centered around the point 0.5 indicating that for some value of $a$ within the small interval, $\gamma(1) > 0.5$ for all values of $\epsilon \in [0.5, 0.9]$. We then checked whether the formula $\exists \epsilon \in [0.5, 0.9], \neg\phi(a, \epsilon, \delta, k)$, with $a = 0.5$, is $\delta$-satisfiable using $\int$dReal. The tool answered that it was unsatisfiable. From this we concluded that for $a = 0.5$, $\gamma(1) > 0.5$ for all $\epsilon \in [0.5, 0.9]$. While we guessed the value of $a$ once we got that the formula is valid, one could also perform a binary search to narrow down the sub-interval containing $a$.

## 3 Background

The goal of this paper is to develop $\delta$-decision procedures for first order logic formulas over reals. In this section we formally introduce what we mean by determining the $\delta$-satisfiability of a formula (Section 3.1). After that we introduce the interval constraint propagation (ICP) algorithm (Section 3.2) that forms the backbone of a $\delta$-decision procedure.

### 3.1 $\delta$-Decision Procedures over the Real Numbers

We recall the theory of $\delta$-decision procedures outlined in [11]. We begin with a quick recap of recursive real analysis [18]. Let $\mathbb{N}, \mathbb{Z}, \mathbb{R}$ denote the sets of natural numbers, integers, and reals, respectively. Let $A^n$ denote the set of $n$-tuples over $A$. For $x = (x_1, \ldots, x_n) \in \mathbb{R}^n$, $||x|| = \max_{1 \le i \le n} |x_i|$.

Let $\mathbb{D} = \{\frac{m}{2^n} \mid m \in \mathbb{Z},\ n \in \mathbb{N}\}$ be the set of *dyadic rational numbers*. Real numbers shall be represented by sequences of dyadic rational numbers that converge to them. Such sequences are called *names* and are defined as follows.

*Definition 3.1.* A *name* of a real number $r \in \mathbb{R}$ is a function $\langle r \rangle : \mathbb{N} \to \mathbb{D}$ such that for all $i \in \mathbb{N}$, $|\langle r \rangle(i) - r| < 2^{-i}$. A real number $r$ is be said to be *computable* if it has a computable name $\langle r \rangle$, i.e., there is a Turing Machine $M_{\langle r \rangle}$ which on input $n$ outputs $\langle r \rangle(n)$.

Since real numbers are functions in this representation, a function on $\mathbb{R}$ is a *functional*, ie, a function that maps functions to functions. To define computable functions on $\mathbb{R}$, we recall the notion of a *recursive functional* [18]. "Computable functionals" are defined using a *function oracle Turing machine*. A function oracle Turing machine $M$ is a Turing machine with a special *query tape*, and special states $q_{\text{query}}$ and $q_{\text{ans}}$. When $M$ is run with the function $f$ as the oracle, if $M$ enters

state $q_{\text{query}}$ with $i$ on the query tape, then in one step, $M$ moves to state $q_{\text{ans}}$ and query tape is overwritten to have $f(i)$. As always, $M$ with the oracle $f$ is denoted as $M^f$.

*Definition 3.2 (Computable Functions).* A function $f{:}\subseteq \mathbb{R} \to \mathbb{R}$ is said to be computable if there is a function oracle Turing machine $M$ outputting dyadic rational numbers such that for any $r \in \mathbb{R} \cap \text{dom}(f)$, name $\langle r \rangle$, and $i \in \mathbb{N}$, $|M^{\langle r \rangle}(i) - f(r)| < 2^{-i}$. In other words, $M^{\langle r \rangle}(0), M^{\langle r \rangle}(1), \ldots$ is a name of $f(r)$.

Many natural functions are computable in this sense, including sum, product, exponentiation, and trigonometric functions. The composition of computable functions is also computable. Definitions of computable numbers and computable functions can be generalized to computable tuples in $\mathbb{R}^n$ and computable functions of the type $\mathbb{R}^n \to \mathbb{R}$, respectively.

Computable functions over $\mathbb{R}^n$ are continuous [18]. Uniformly continuous functions on a domain $C \subseteq \mathbb{R}^n$ are continuous functions such that for each $\epsilon > 0$ there exist a $\delta > 0$ such that for all $x, y \in C$, $|f(x) - f(y)| < \epsilon$ whenever $||x - y|| < \delta$. Any continuous function over a compact subset of $\mathbb{R}$ is uniformly continuous [25]. Given a uniformly continuous function $f$ on domain $C$, a uniform modulus of continuity for $f$ is a function $m : \mathbb{N} \to \mathbb{N}$ such that for any $x, y \in C$, and $i \in \mathbb{N}$, if $||x - y|| < 2^{-m(i)}$ then $|f(x) - f(y)| < 2^{-i}$. It turns out that computable functions have *computable* uniform modulus of continuity over any compact subset.

Theorem 1 ([18]). *For any computable $f{:}\subseteq \mathbb{R}^n \to \mathbb{R}$ and compact subset $C \subseteq \text{dom}(f)$, $f$ has a computable uniform modulus of continuity over $C$, ie, there is a Turing machine $M_{f,C}$ such that on input $i$, outputs $j$ such that if $x, y \in C$ and $||x - y|| < 2^{-j}$ then $|f(x) - f(y)| < 2^{-i}$.*

Let us now define a logic, $\mathcal{L}_{\mathcal{F}}$, built over such computable functions over tuples of real numbers. Let $\mathcal{F}$ be a set of computable functions over tuples of real numbers. A $\Sigma_1$ sentence is a sentence of the form $\exists x_1 \in I_1 \exists x_2 \in I_2 \cdots \exists x_k \in I_k \psi$ where $I_j$ is a closed interval, and $\psi$ is quantifier-free and is a Boolean combination of atomic formulas of the form $f(x_j) \bowtie 0$ with $\bowtie \in \{<, \leq, >, \geq\}$ and $f \in \mathcal{F}$. It is easy to show that any such sentence $\varphi$ can be converted to a normal form with the structure $\exists x_1 \in I_1 \cdots \exists x_\ell \in I_\ell \bigwedge_i \bigvee_j f_{ij} = 0$. For such a sentence $\varphi$, a $\delta$-weakening ($\delta \geq 0$) of $\varphi$ is the formula $\varphi^\delta = \exists x_1 \in I_1 \cdots \exists x_k \in I_k \bigwedge_i \bigvee_j |f_{ij}| < \delta$.

Theorem 2 ([11]). *Given sentence $\varphi$ of the form $\exists x_1 \in I_1 \cdots \exists x_\ell \in I_\ell \bigwedge_i \bigvee_j f_{ij} = 0$ where each $f_{ij} \in \mathcal{F}$, defined over $I_1 \times I_2 \cdots \times I_\ell$ and $\delta > 0$, there is an algorithm $\mathcal{A}$ that always terminates and outputs one of two answers — unsat or $\delta$-sat — with the following property. If $\mathcal{A}$ outputs unsat then $\varphi$ is not true over $\mathbb{R}$ and if $\mathcal{A}$ outputs $\delta$-sat then $\varphi^\delta$ is true over $\mathbb{R}$.*

The algorithm given by Theorem 2 solves the problem of checking if $\varphi$ is true approximately as follows. If $\varphi$ is true then $\mathcal{A}$ always answers $\delta$-sat. And if both $\varphi$ and $\varphi^\delta$ are not true then $\mathcal{A}$ always answers unsat. But if $\varphi$ is not true but $\varphi^\delta$ is true then $\mathcal{A}$ may respond with either unsat or $\delta$-sat. The proof of Theorem 2 crucially exploits the uniform continuity of computable functions. Theorem 2 can be generalized to give an approximate decision procedure for $\exists\forall$-sentences, rather than only existentially quantified sentences [20]. However, all quantified variables are still constrained to take values in a closed interval.

### 3.2 Interval Constraint Propagation (ICP) and dReal

The core engine of the $\delta$-decision procedure is *Interval Constraint Propagation (ICP)* [5], which through a combination of pruning and branching, attempts to determine if a $\delta$ weakening of $\bigwedge_i f_i = 0$ holds when the free variables are constrained to be within an initial box. Let us start with some useful definitions.

*Definition 3.3 (Intervals, Boxes, and Hulls [11]).* The set of real closed *intervals* with floating-point endpoints is defined as follows: $\mathbb{IF} = \{[a, b] \mid a, b \in \mathbb{F}, a \leq b\}$, where $\mathbb{F}$ is the finite set of floating point numbers augmented with $\infty$ and $-\infty$. The set of *boxes* containing these intervals is defined by $\mathbb{BF} = \bigcup_{i=1}^{\infty} \mathbb{IF}^i$. The *hull* of a given set $S \subseteq \mathbb{R}$ is defined by $\mathit{Hull}(S) = \bigcap\{I \in \mathbb{IF} \mid S \subseteq I\}$.

For the rest of the paper, we shall assume that there is a positive integer ulp such for any two floating points $a < b$ with $b - a > \frac{1}{2^{\text{ulp}}}$, there is a floating point $c$ such $a < c < b$.

```
Input: The constraints f_1 = 0, ..., f_m = 0
Input: The initial box B_0 = I_0^0 × ··· × I_n^0
Input: A positive real constant δ
Output: δ-sat or unsat

S ← Stack{B_0}

while S ≠ ∅:
  B ← S.pop()
  while ∃1 ≤ i ≤ m, B ≠ Prune(B, f_i):
    B ← Prune(B, f_i)
  if B ≠ ∅:
    if ∃1 ≤ i ≤ m, |f_i^#(B)| ≥ δ:
      {B_1, B_2} ← Branch(B)
      S.push({B_1, B_2})
    else:
      return δ-sat

return unsat
```

**Algorithm** 1. Interval Constraint Propagation (as presented in [13])

We continue with the notion of an interval extension of a function.

*Definition 3.4 (Interval Extension).* An *interval extension* of a function $f{:}\subseteq \mathbb{R}^n \to \mathbb{R}$ is a function $f^{\#}{:}\subseteq \mathbb{BF} \to \mathbb{IF}$ such that for all $B \in \mathbb{BF} \cap \mathrm{dom}(f^{\#})$, $\{f(\mathbf{y}) \mid y \in B\} \subseteq f^{\#}(B)$.

The ICP algorithm is shown as Algorithm 1. A crucial subroutine in the algorithm is *pruning*, written as $\mathrm{Prune}(B, f_i)$, whose output is a smaller box that satisfies the following properties.

*Definition 3.5 ([11]).* A *well-defined pruning operator* $\mathrm{Prune} : \mathbb{BF} \times \mathcal{F} \to \mathbb{BF}$, where $\mathcal{F}$ is a finite set of computable functions, has the following properties: (W1) $\mathrm{Prune}(B, f) \subseteq B$; (W2) If $\mathrm{Prune}(B, f) \neq \emptyset$, then $0 \in f^{\#}(\mathrm{Prune}(B, f))$; (W3) $B \cap Z_f \subseteq \mathrm{Prune}(B, f)$, where $Z_f = \{\mathbf{x} \in \mathbb{R}^n \mid f(x) = 0\}$.

Summarized, condition (W1) states that a pruning operator never adds points to the box, while condition (W2) states that if the pruned box is non-empty, then the interval extension of $f$ on that pruned box must contain zero, meaning that the pruned box must contain a potential root of $f$. Condition (W3) states that the pruning operator must not prune out actual roots of $f$ if they exist.

On the other hand, pruning is often insufficient in shrinking the box enough to find a solution. The fallback is to perform *branching*: splitting one of the intervals in the box, and dividing the

problem into subproblems. More formally, if the original box $B = I_1 \times \cdots \times I_k \times \cdots I_n$, then Branch$(B, k)$ yields the two boxes $B' = I_1 \times \cdots \times I'_k \times \cdots I_n$ and $B'' = I_1 \times \cdots \times I''_k \times \cdots I_n$, where $I'_k = [a, c], I''_k = [c, b]$, and $I_k = [a, b]$ for some $a < c < b$. In case there is no floating point between $a$ and $b$, Branch$(B, k) = B$. Further, we let Branch$(B)$ be Branch$(B, k)$ if $k$ is the smallest index $1 \leq k \leq n$ such that Branch$(B, k) \neq B$. If no such $k$ exists, then we let Branch$(B) = B$.

Let $\mathcal{F}$ be a finite set of computable functions on $\mathbb{R}^n$. Recall that each $f_i \in \mathcal{F}$ is uniformly continuous on each compact set $C$ on its domain. For each $\mathcal{F}, r \in \mathbb{R}$ and bounded box $B_0 \subseteq \bigcap_{f \in \mathcal{F}} \text{dom}(f)$, we let modulus$(\mathcal{F}, r, B_0)$ be the smallest integer $j > 0$ such that $|f_i(\mathbf{x} - \mathbf{y})| < r$ for each $f_i \in \mathcal{F}$ whenever $\mathbf{x}, \mathbf{y} \in B_0$ and $||\mathbf{x} - \mathbf{y}||_\infty < \frac{1}{2^j}$. . Let $\mathcal{F}^\# = \{f_1^\#, \ldots, f_m^\#\}$. We say that $\mathcal{F}^\#$ is *$\delta$-proximal*, where $\delta \in \mathbb{Q}$, if for all $B \in \mathbb{BF}$, and for all $f_i$, if $z \in f_i^\#(B)$, then there exists $\mathbf{x} \in B \cap \text{dom}(f)$ such that $|z - f_i(\mathbf{x})| < \delta/3$. We state the termination of Algorithm 1, given a reasonable assumption on interval extensions, as follows:

Theorem 3. *Let $\delta \in \mathbb{Q}$ and $\mathcal{F} = \{f_1, \ldots, f_m\}$ be arbitrary. Let $\mathcal{F}^\# = \{f_1^\#, \ldots, f_m^\#\}$. Algorithm 1 terminates if*

(1) Prune *is well-defined,*
(2) *$\mathcal{F}^\#$ is $\delta$-proximal, and*
(3) modulus$(\mathcal{F}, \frac{\delta}{4}, B_0) >$ ulp.

We prove this theorem in Appendix A of our auxiliary material.

Given termination, the correctness of Algorithm 1 is provided by the following theorem:

Theorem 4 ([11]). *Let $\delta \in \mathbb{Q}$. Algorithm 1 is a $\delta$-decision procedure if and only if* Prune *is a well-defined pruning operator.*

Algorithm 1 is combined with a SAT solver to provide a $\delta$-decision procedure for $\mathcal{L}_\mathcal{F}$, which is implemented in the tool dReal [12]. The functions $\mathcal{F}$ for the version of $\mathcal{L}_\mathcal{F}$ implemented in dReal do not contain integrals and this paper extends the approach to include integrals.

*∃-∀ Queries.* The ICP algorithm as presented works for queries with only existential quantification, however, the algorithm has been extended to support queries involving ∃-∀ formulas in a later work [20]. Formulas with such quantifier alternation are simplified to the following standard form: $\exists x_1 \in I_1 \cdots \exists x_\ell \in I_\ell \bigwedge_i \left(\forall y_1 \in I_1 \cdots \forall y_\ell \in I_\ell \bigvee_j f_{ij} \geq 0\right)$. The resulting solver is similar to Algorithm 1, but instead of dealing with a set of constraints of the form $f_{ij} = 0$, it deals with a set of constraints of the form $\forall y_1 \in I_1 \cdots \forall y_\ell \in I_\ell \bigvee_j f_{ij} \geq 0$. The Prune operator is changed to accommodate these kinds of constraints.

In order to prune the constraint, the theory solver first tries to find a satisfying assignment, using the existing machinery for checking $\delta$-satisfiability over existentially-quantified formulas, for the variables $y_1, \cdots, y_\ell$ such that $\bigwedge_j f_{ij} < 0$, the negation of $\bigvee_j f_{ij} \geq 0$, holds. In fact, in order to rule out spurious assignments, the formula is strengthened to $\bigwedge_j f_{ij} < \epsilon$, for appropriately chosen $\epsilon$. If such an assignment $a_1, \cdots, a_\ell$ exists, then the ordinary pruning algorithm is performed on each of the constraints $f_{ij} \geq 0$, for each $j$, with $a_1, \cdots, a_\ell$ replacing $y_1, \cdots, y_\ell$. The intersection of the results of pruning for each disjunct $j$ gives us a feasible pruned box for the constraint $\forall y_1 \in I_1 \cdots \forall y_\ell \in I_\ell \bigvee_j f_{ij} \geq 0$.

This technique for supporting quantifier alternation is implemented in the present version of dReal, dReal4 [1], and is able to handle our extensions to support integration with relatively minor modifications.

## 4 $\delta$-Decision Procedures in the Presence of Integrals

We present the main algorithmic results of this paper in this section. We begin by formally defining the logic $\mathcal{L}_{int}$, which consists of first-order formulas over reals that contain expressions involving exponentials, logarithms, trigonometric functions, and integrals of functions built through such expressions. After that, we present our $\delta$-decision procedure for $\mathcal{L}_{int}$ by providing algorithms for the key components in ICP to handle integrals.

### 4.1 Syntax of Terms

$\int$dReal implements a $\delta$-decision procedure for a logic we call $\mathcal{L}_{int}$. $\mathcal{L}_{int}$ is a version of $\mathcal{L}_{\mathcal{F}}$ (Section 3.1) where the following grammar gives the terms to build the elementary computable functions.

$$t ::= x \mid c \mid -t \mid 1/t \mid |t| \mid t + t \mid t \times t \mid \sin(t) \mid \cos(t) \mid e^t \mid \ln(|t|) \mid \int_t^t t dz \tag{13}$$

where $c$ is any rational number, $x$ is a variable, $-t, 1/t, |t|, t + t, t \times t, \sin(t), \cos(t), e^t, \ln(t)$ represent the standard negation, reciprocal, absolute value, addition, multiplication, trigonometric sine, trigonometric cosine, exponential and natural logarithm functions. These terms are already present in dReal. The term $\int_t^t t dz$, on the other hand, is an addition in our tool, $\int$dReal, and represents integration. Observe that $z$ is a bound variable in $\int_t^t t dz$, and the syntax allows for nesting of integrals. More formally, fvar($t$), the set of free variables of $t$ can be defined inductively as follows:

$$\text{fvar}(t) = \begin{cases} \{x\} & \text{if } t = x \\ \emptyset & \text{if } t = c \\ \text{fvar}(t_1) & \text{if } t \in \{-t_1, 1/t_1, |t_1|, \sin(t_1), \cos(t_1)\} \\ \text{fvar}(t_1) \cup \text{fvar}(t_2) & \text{if } t \in \{t_1 + t_2, t_1 \times t_2\} \\ \text{fvar}(t_1) \cup \text{fvar}(t_2) \cup (\text{fvar}(t_3) \setminus z) & \text{if } t = \int_{t_1}^{t_2} t_3 dz \end{cases}$$

Observe that a term $t$ with free variables $x_1, \ldots, x_m$ can be identified as a function $[\![t]\!] : \mathbb{R}^m \to \mathbb{R}$. The interpretation $[\![t]\!]$ is defined in the expected way, except for the terms of the form $\frac{1}{t}$ and $\ln(|t|)$. For $1/t$, we define $[\![1/t]\!] = \frac{1}{[\![t]\!]}$ for $|[\![t]\!]| > \delta$ and $\frac{1}{\delta}$ otherwise. For $\ln(|t|)$, we define $[\![\ln(|t|)]\!] = \ln(|[\![t]\!]|)$ for $|[\![t]\!]| > \delta$ and $\ln(\delta)$ otherwise (Note that $[\![t]\!]$ in these cases requires a constant $\delta$ — this should be set equal to the $\delta$ used in the decision procedure). Let $\mathcal{F}_{int}$ be the set of functions built using the term language according to equation 13, and the resulting logic be $\mathcal{L}_{int}$.

Finally, we observe that standard mathematical properties imply that the functions in $\mathcal{F}_{int}$ are *computable*. Thus, we can conclude from Theorem 2 that $\delta$-satisfiability of $\mathcal{L}_{int}$ is decidable.

Theorem 5 ($\delta$-satisfiability of $\mathcal{L}_{int}$). *For each term $t \in \mathcal{F}_{int}$, $[\![t]\!]$ is a computable function. Thus, there is a $\delta$-decision procedure for $\mathcal{L}_{int}$.*

We prove this theorem in Appendix B of our auxiliary material.

### 4.2 The Algorithm

We implement a decision procedure for $\delta$-satisfiability of $\mathcal{L}_{int}$ based on the ICP algorithm (See Algorithm 1). For this, one must come up with a concrete implementation of the Prune function and a means of calculating the interval extension of functions to check if a box might satisfy the constraints.

By default, dReal uses the HC4Revise algorithm [26] to implement the Prune operator. This algorithm decomposes each constraint in the input to the ICP algorithm into a tree form and attempts to shrink the input box by first computing the interval extension of the operation at each

node, given the intervals for the input variables, and then using the properties of the constraint to eliminate infeasible values from the input box.

The primitive operations underlying the HC4Revise algorithm include *forward propagation*: given input nodes with intervals $I_1, \cdots, I_n$, set the current node to $I_f = f^{\#}(I_1, \cdots, I_n)$, where $f^{\#}$ represents the interval extension of the function specified at the given node. The other operation is *backward propagation*: given an interval $I_f$ which represents the values that node can take on, calculate sub-intervals $I'_1, \cdots, I'_n$ such that $f^{\#}(I'_1, \cdots, I'_n) \subseteq I_f$. Thus, in order to incorporate integration, we have to implement both forward and backward propagation for constraints that include integral terms. We describe them next.

*Forward Propagation.* Forward propagation for terms with integrals depends on the ability to compute an interval enclosure of an integral. This is done using an external library, Arb, to numerically calculate an enclosure of an integral [16], and is discussed further in Section 5.

Input: A function $g : \mathbb{R}^{m+2} \to \mathbb{R}$
  such that $g(r_\ell, r_u, r_1, \ldots, r_m) = \int_{r_\ell}^{r_u} f(z, r_1, \ldots, r_m)dz$, where
    $f : \mathbb{R}^{m+1} \to \mathbb{R}$.
Input: An input box $B = [r_\ell, r_u, r_1, \ldots, r_n]$
Input: The interval size threshold $T$, and a small positive constant $\epsilon$.
Output: The output box $B' = [r_\ell, r_u, r_1, \ldots, r_n]$

$B' \leftarrow B$
**if** $\exists 1 \le i \le |B'|, |B'[i]| > T$:
  **return**
**for** $i \leftarrow 1$ to $|B'|$:
  $iv \leftarrow [\text{lb}(B'[i]), \text{lb}(B'[i]) + \epsilon]$
  $newiv \leftarrow \emptyset$
  **while** $\text{ub}(iv) \le \text{ub}(B'[i])$:
    $r \leftarrow \texttt{Eval}(g^{\#}, B'[i := iv])$
    **if** $r \cap \{0\} \neq \emptyset$:
      $newiv \leftarrow \texttt{Hull}(newiv \cup iv)$
    $iv \leftarrow iv + \epsilon$
  $B'[i] \leftarrow newiv$

**Algorithm** 2. Backwards Propagation of Intervals for Integration. For performance reasons, when an input interval is wider than a threshold $T$, backwards propagation does not happen, resulting in the ICP loop branching instead. In our experiments, this threshold $T = 0.1$.

*Backward Propagation.* Algorithm 2 is our technique for the backward propagation of integrals. As input, it takes a function $g(r_\ell, r_u, r_1, \ldots, r_m) = \int_{r_\ell}^{r_u} f(z, r_1, \ldots, r_m)dz$, where $f : \mathbb{R}^{m+1} \to \mathbb{R}$ is an expression in our term language, and a box consisting of "current" intervals for subexpressions corresponding to the limits of integration $(r_\ell, r_u)$ and free variables appearing in the integrand $(r_1, \cdots, r_n)$. In Algorithm 2, lb and ub represent the lower and upper bounds of an interval, $\epsilon$ is

a small non-negative number, the statement $S[i := e]$ represents a sequence $S$ with $i$th element replaced by $e$.

The algorithm iterates through all the input intervals in the given box $B$. At each iteration step $i$, the procedure breaks the $i$-th interval into subintervals of length $\epsilon$. Then, for each subinterval, the integral is evaluated after the current version of the $i$-th interval is replaced with it. If the new interval intersects the singleton set $\{0\}$, the $\epsilon$-wide subinterval is added to the new version of the $i$-th interval of the new box $B'$. Note that if $\epsilon$ is larger, the algorithm will run faster but perform a less precise pruning, while if $\epsilon$ is smaller, the algorithm will run slower but perform a more precise pruning.

The following result ensures that Algorithm 2 can be safely incorporated into the ICP algorithm.

Theorem 6. *Let* $g(r_\ell, r_u, r_1, \ldots, r_m) = \int_{r_\ell}^{r_u} f(z, r_1, \ldots, r_m)dz$ *be a computable function. Let* $\text{Prune}(B, g)$ *be the box output by Algorithm 2 (where* $B$ *is a box enclosing the variables* $r_\ell, r_u, r_1, \ldots r_m$*, such that if* $B \neq \emptyset$*, then* $0 \in g^\#(B)$*). Then* $\text{Prune}(B, g)$ *is a well-defined pruning operator.*

Proof. We proceed by proving the three properties in Definition 3.5. Briefly note that if the width of an interval in the box $B$ exceeds the threshold $T$, then $\text{Prune}(B, g) = B$, and the three properties are trivially satisfied. If this is not the case, we proceed as follows, noting that $B' = \text{Prune}(B, g)$ as in Algorithm 2.

(W1) ($B' \subseteq B$) : It suffices to show that $B' \subseteq B$ is maintained across the outer loop from lines 10-18. Since only the element $B'[i]$ is mutated in the outer loop body, given some $i \in 1, \ldots, |B'|$, it suffices to show that $B'[i] \subseteq B[i]$ is maintained. This is implied by the following inductive invariant on the inner loop from lines 13-18: $newiv \subseteq B'[i] \wedge \text{lb}(iv) \geq \text{lb}(B'[i])$. The invariant clearly holds on entry. To show preservation across the loop body, we proceed as follows. We first prove that the conjunct $newiv \subseteq B'[i]$ is maintained. Note that either $newiv$ remains unchanged or it is altered by line 16. In the first case, the conjunct is clearly maintained. But in the second case, note that $newiv \subseteq B'[i]$ by the invariant, and $iv \subseteq B'[i]$ by both the invariant and the truth of the guard $\text{ub}(iv) \leq \text{ub}(B'[i])$ on line 13. Recalling that both $newiv$ and $iv$ are intervals, it follows that $\min(\text{lb}(newiv), \text{lb}(iv))$ and $\max(\text{ub}(newiv), \text{ub}(iv))$ are in the interval $B'[i]$. Therefore, $\text{Hull}(newiv \cup iv) \subseteq B'[i]$, and the conjunct is maintained. Additionally, it is clear that the conjunct $\text{lb}(iv) \geq \text{lb}(B'[i])$ is maintained by line 17, since it is required that $\epsilon > 0$.

(W2) (If $B' \neq \emptyset$, then $0 \in g^\#(B')$): Note that if $B' \neq \emptyset$, then for all indices $i \in 1, \ldots, |B'|$, $B'[i] \neq \emptyset$. Therefore, at iteration $j = |B'|$ of the outer loop from lines 10-18, there must have been at least one point where the guard on line 15, $r \cap \{0\}$, was true. Therefore, there exists a value of $B'$ before iteration $j$, $B''$, and an interval $iv$ such that $0 \in g^\#(B''[j := iv])$. Let $B'''$ be the value of $B'$ at the end of the algorithm after iteration $j$. Note that since the interval $iv \subseteq B'''[j]$ (easy to show by analyzing the inner loop on lines 13-17), $B'''[i] = B''[i]$ for all other $i$, and $g^\#$ is convex, it follows that $0 \in g^\#(B''')$.

(W3) ($B \cap Z_g \subseteq B'$, where $Z_g = \{x \in \mathbb{R}^n \mid g(x) = 0\}$) : Let $R = B \cap Z_g$. It suffices to show that $R \subseteq B'$ is maintained across the outer loop from lines 10-18. Since only the element $B'[i]$ is mutated in the outer loop body, given some $i \in 1, \ldots, |B'|$, it suffices to show that $R[i] \subseteq B'[i]$ is maintained. Let $r \in R$ be fixed. Further note that $r[i] \in B'[i]$ initially. Thus, there exists a value $iv \subseteq B'[i]$ reached by the inner loop from lines 13-18 such that $r[i] \in iv$. Since $r \in B'[i := iv]$, and $g(r) = 0$, it follows that $0 \in g^\#(B'[i := iv])$. Therefore, the guard on line 15, $r \cap \{0\} = \emptyset$, will be true, and it follows that $iv \subseteq newiv$, the new value of $B'[i]$. Thus, $r[i] \in newiv$, and it follows that $R[i] \subseteq B'[i]$ is maintained. □

Note that, for the sake of performance, we implement an optimized version of the algorithm that, instead of performing a linear scan from $\mathrm{lb}(B'[i])$ to $\mathrm{ub}(B'[i])$ to determine the upper and lower bounds of *newiv*, performs a binary search to find those bounds.

## 5 Implementation

$\int$dReal implements a $\delta$-decision procedure for satisfiability of formulas with integrals over the reals. We outline the main features of our tool design.

### 5.1 Interval Computations

The present version of dReal, dReal4 [1], implements interval constraint propagation with the support of interval arithmetic and constraint programming library IBEX [15]. This library provides a means for constructing and computing the interval enclosure for arithmetic expressions involving a range of linear and non-linear real functions. Additionally, it provides functionality for contracting a box by pruning, by propagating interval arithmetic backward according to a constraint using algorithms such as HC4Revise [26]. However, IBEX does not presently support integral expressions.

To support integration, we incorporate an external library for computing *ball* enclosures of arithmetic expressions, Arb [16]. Unlike an interval, a ball is specified by a midpoint $m$ and a radius $r$, and represents the interval $[m - r, m + r]$ (these values are implemented as arbitrary precision numbers, hence the name Arb). Notably for our purposes, the library supports computing an interval enclosure of the integral for an arbitrary function; a programmer needs only to write an auxiliary function to compute the integrand at a particular point.

To incorporate Arb's integration procedure into IBEX, we add support for integral expressions to the expression graph data structure it provides. When evaluating an integral, intervals around the limits are first evaluated and then converted to balls, and then Arb's integration procedure is given an auxiliary function that computes values of the integrand. This function is given extra data: the integrand, which is another expression graph, and all the values of variables that occur free with respect to the integrand. When called upon to provide the value of the integrand at a particular point, the function evaluates the integrand with the value of the variable of integration and the other values passed to it. Nested integrals are fully supported in our implementation of integration: when one is encountered in an integral body, a nested invocation of Arb's integration procedure and our support code is performed to compute its value. Once Arb returns from the integration procedure, the ball is converted into an interval in our tool, and passed onwards.

Due to the design of the integration algorithm, Arb initially treats the integrand as a complex-valued function on complex variables. If it is piecewise-holomorphic (complex-differentiable in a neighborhood of every point in a given domain), then the integration converges very quickly [17, 27]. In order to support the use of this algorithm, a complex ball enclosure must be computed around the integrand. To facilitate this, we implement a new expression evaluator which works on the same expression graph implemented in IBEX, but which uses complex balls instead of real intervals as the datatype it manipulates. This new evaluator also checks conservatively whether the integrand is holomorphic with respect to the variable of integration on the given input or not, signaling this fact to the integration procedure if the integrand may not be holomorphic.

### 5.2 Tool Engineering

Additional code written in our extension $\int$dReal of dReal [1] is in C++. An overview of $\int$dReal's architecture is shown in Fig. 3. The modifications made are highlighted in red, and are as follows. We amend the dReal driver to support new syntax for integration. In addition, we extend the expression evaluator in IBEX to compute interval enclosures of expressions to support integration, by linking to the aforementioned Arb library. Specifically, we add support for forward and backward

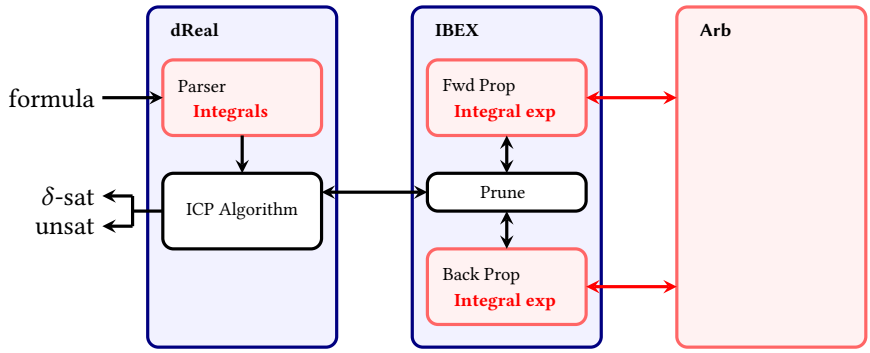



Fig. 3. ∫dReal Architecture. Extensions to dReal shown in **red**.

computation involving integrals. This is again supported by linking IBEX to Arb. A number of minor modifications were made to support queries with quantifier alternation, but the core implementation remains the same.

## 6 Case Studies

In order to provide insight on the kinds of queries ∫dReal can handle, we present the following case studies in addition to the motivating examples in Section 2.

### 6.1 Income Fairness by Gender

For this case study, we adapt another example related to algorithmic fairness from FairSquare [2]. This example is derived from their decision tree $DT_4$, which determines the salary of an employee given an independent population model. The procedure determines whether to pay a higher salary to someone based on a number of factors unrelated to gender. Intuitively, a fair algorithm should decide the higher salary with nearly equal probability depending on the employee's gender.

From this procedure, we derive a query to determine whether the given algorithm is fair based on the employee's gender. More specifically, we derive the following formula:

$$\phi(\epsilon,\mu,\sigma) ::= \frac{1}{\sqrt{2\pi\sigma}}\int_{7073.5}^{4\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}\,dx > (1-\epsilon)\left(\frac{1}{\sqrt{2\pi(\sigma+434.557)}}\int_{7073.5}^{4(\sigma+434.557)} e^{-\frac{(x-(\mu+31.5895))^2}{2(\sigma+434.557)^2}}\,dx + \frac{14}{100000}\right),$$

where $\epsilon$ is a small constant that determines the maximum extent of unfairness, while $\mu$ and $\sigma$ are the mean and standard deviation of the normal distribution of earnings.

Two queries in the benchmark suite are based on this formula. Here, we try to prove that a negated query is unsat, showing fairness of the algorithm.

(1) (high_inc_gd_00): $\epsilon = 0.15$, $\mu = 568.4105$, $\sigma = 24248365.5428$. $\neg\phi(\epsilon,\mu,\sigma)$ given to solver.
(2) (high_inc_gd_01): $\epsilon = 0.15$, The following formula is given to the solver:

$$\exists\mu \in [548.4105, 588.4105], \sigma \in [548.4105, 588.4105], \neg\phi(\epsilon,\mu,\sigma)$$

The latter formula allows for variation in the population model by permitting some inaccuracies in the measurement of $\mu$ and $\sigma$, and shows that the procedure remains fair even in the face of this variation.

We also test a version of this formula that reflects a situation where fairness cannot be proved:

$$\phi(\epsilon,\mu,\sigma) ::= \frac{1}{\sqrt{2\pi\sigma}}\int_{7073.5}^{4\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}\,dx + \frac{14}{100000} > (1-\epsilon)\left(\frac{1}{\sqrt{2\pi(\sigma+45079127)}}\int_{7073.5}^{4(\sigma+45079127)} e^{-\frac{(x-(\mu+760.9595))^2}{2(\sigma+45079127)^2}}\,dx\right).$$

The resulting formula is the result of modifying the population model of the two genders represented in the model by significantly skewing the mean and standard deviation between them. This induces a substantial disparity in income based on gender. Two queries in the benchmark suite are based on this formula. We try to prove that a query is unsat, which shows that the algorithm is unfair.

(1) (high_inc_gd_unfair_00): $\epsilon = 0.15$, $\mu = 568.4105$, $\sigma = 24248365.5428$. $\phi(\epsilon, \mu, \sigma)$ given to solver.
(2) (high_inc_gd_unfair_01): $\epsilon = 0.15$, The following formula is given to the solver:

$$\exists \mu \in [548.4105, 588.4105], \sigma \in [548.4105, 588.4105], \phi(\epsilon, \mu, \sigma)$$

### 6.2 Privacy of the Gaussian Mechanism

The next case study concerns the privacy of the Gaussian mechanism in differential privacy [7]. Here specifically, we take the mechanism which maps $(u, v)$ to $(u + \mathcal{N}(0, \sigma^2), v + \mathcal{N}(0, \sigma^2))$, where $\mathcal{N}$ is the normal distribution. We consider two points $(u_1, v_1)$ and $(u_2, v_2)$ neighbors if their Euclidean distance is less than 1, i.e., $\sqrt{(u_1 - u_2)^2 + (v_1 - v_2)^2} \leq 1$. The formula pertaining to determining that the mechanism is private can be seen below:

$$\phi(\epsilon) ::= P_{\text{gauss}}(u_1, v_1, a, b, c, d) \leq e^{\epsilon} P_{\text{gauss}}(u_2, v_2, a, b, c, d) + \delta,$$

where

$$P_{\text{gauss}}(u, v, a, b, c, d) = \frac{1}{2\pi\sigma^2} \int_a^b \int_c^d e^{-\frac{(x-u)^2}{2\sigma^2}} e^{-\frac{(y-v)^2}{2\sigma^2}} dxdy,$$

$\delta = \frac{1}{8}$, $\sigma = \frac{2.2}{\epsilon}$ and $a = b = c = d = 10$.

Here, $\epsilon$ is the privacy budget and $P_{\text{gauss}}(u, v, a, b, c, d)$ represents the probability that the Gaussian mechanism outputs a point within the rectangular region $[a, b] \times [c, d]$. Thus, this formula represents a special case of differential privacy.

Also note that $(u1, v1) = (0, 0)$ and $(u2, v2) = (1, 0)$. The formula here comprises the following query, which is expected to return unsat:

(1) (gauss_mech_00): $\exists \epsilon \in (0.1, 1), \neg\phi(\epsilon)$ sent to solver.

There is additionally a version of the query with the double integral converted to a product of integrals, where $P_{\text{gauss}}$ is defined as follows:

$$P_{\text{gauss}}(u, v, a, b, c, d) = \frac{1}{2\pi\sigma^2} \left( \int_a^b e^{-\frac{(x-u)^2}{2\sigma^2}} dx \right) \left( \int_c^d e^{-\frac{(y-v)^2}{2\sigma^2}} dy \right).$$

The query is as follows:

(1) (gauss_mech_00_alt1): $\exists \epsilon \in (0.1, 1), \neg\phi(\epsilon)$ sent to solver.

### 6.3 Privacy of the Laplace Mechanism

The next case study concerns the privacy of the Laplace mechanism in differential privacy [6]. The formula pertaining to the first (private) mechanism can be seen here:

$$\phi_1(\epsilon) ::= \int_a^b e^{-\epsilon|x|} dx \leq e^{\epsilon} \int_a^b e^{-\epsilon|x+1|} dx.$$

Here, $\phi_1(\epsilon)$ is the inequality that verifies the differential privacy of the Laplace mechanism when applied to adjacent inputs 0 and -1, with respect to producing an output in the interval $[a, b]$. Throughout the queries, $a = 1$ and $b = 2$. We try to prove the privacy of the mechanism for all $\epsilon$ in a given range by negating $\phi_1$ and checking for unsat, as well as try to disprove the privacy for all $\epsilon$ in the same range by checking $\phi_1$ for unsat. The queries are below:

(1) (lap_mech_00): $\exists \epsilon \in [0.1, 1], \neg\phi_1(\epsilon)$ given to solver.

(2) (lap_mech_00_not_pri): $\exists \epsilon \in [0.1, 1], \phi_1(\epsilon)$ given to solver.

The formula pertaining to the second (non-private) mechanism can be seen here:

$$\phi_2(\epsilon) ::= \int_a^b e^{-\epsilon|x|} dx \leq e^{\frac{\epsilon}{2}} \int_a^b e^{-\epsilon|x+1|} dx.$$

Throughout the queries, $a = 1$ and $b = 2$. We try to prove the privacy of the mechanism for all $\epsilon$ in a given range by negating $\phi_2$ and checking for unsat, as well as try to disprove the privacy for all $\epsilon$ in the same range by checking $\phi_2$ for unsat. The queries are below:

(1) (lap_mech_sat_01): $\exists \epsilon \in [0.1, 1], \neg\phi_2(\epsilon)$ given to solver.
(2) (lap_mech_sat_01_not_pri): $\exists \epsilon \in [0.1, 1], \phi_2(\epsilon)$ given to solver.

### 6.4 Sparse Vector Technique with Two Inputs

This final case study arises from differential privacy verification for the Sparse Vector Technique algorithm built on top of the Gaussian mechanism in differential privacy, with input length 2 [8, 21]. The following formula pertains to the privacy of this mechanism with respect to outputting $(\perp, \perp)$ on two inputs $(u_1, v_1)$ and $(u_2, v_2)$, each of which is an array of two queries, having a Manhattan distance of 1 from each other.

$$\phi(\epsilon) ::= P^{\text{SVT}}_{(u_1,v_1),LHS} \leq e^{\epsilon} P^{\text{SVT}}_{(u_2,v_2),RHS} + \delta,$$

where

$$P^{\text{SVT}}_{(u,v),LHS} = \frac{1}{2\pi\sigma^3\sqrt{2\pi}} \int_{-k}^{+k} \int_{v-k}^{z} \int_{u-k}^{z} e^{-\frac{(x-u)^2}{2\sigma^2}} e^{-\frac{(y-v)^2}{2\sigma^2}} e^{-\frac{z^2}{2\sigma^2}} dxdydz + \frac{3}{100},$$
$$P^{\text{SVT}}_{(u,v),RHS} = \frac{1}{2\pi\sigma^3\sqrt{2\pi}} \int_{-k}^{k} \int_{v-k}^{z} \int_{u-k}^{z} e^{-\frac{(x-u)^2}{2\sigma^2}} e^{-\frac{(y-v)^2}{2\sigma^2}} e^{-\frac{z^2}{2\sigma^2}} dxdydz.$$

Here, $\delta = \frac{1}{8}$. In the outer integral of the original formulas (not shown here), we have replaced the lower limit $-\infty$ with $-k$ and the upper limit $\infty$ with $k$. For the inner integrals of the original formulas, we have replaced the lower limit $-\infty$ with $v - k$ and $u - k$, respectively. Furthermore, an error tail bound of $\frac{3}{100}$ has been added to the left-hand side of the inequality. In addition to the first version of this formula with triple integrals, we can also write a version of $\phi$, $\phi'$ using a double integral over a product of integrals as follows (only redefining $P^{\text{SVT}}_{(u_1,v_1),LHS}$ and $P^{\text{SVT}}_{(u_1,v_1),RHS}$):

$$P^{\text{SVT}}_{(u,v),LHS} = \frac{1}{2\pi\sigma^3\sqrt{2\pi}} \int_{-k}^{+k} \left( \int_{u-k}^{z} e^{-\frac{(x-u)^2}{2\sigma^2}} dx \right) \left( \int_{v-k}^{z} e^{-\frac{(y-v)^2}{2\sigma^2}} dy \right) e^{-\frac{z^2}{2\sigma^2}} dz + \frac{3}{100}$$
$$P^{\text{SVT}}_{(u,v),RHS} = \frac{1}{2\pi\sigma^3\sqrt{2\pi}} \int_{-k}^{+k} \left( \int_{u-k}^{z} e^{-\frac{(x-u)^2}{2\sigma^2}} dx \right) \left( \int_{v-k}^{z} e^{-\frac{(y-v)^2}{2\sigma^2}} dy \right) e^{-\frac{z^2}{2\sigma^2}} dz.$$

We define the following queries based on these formulas, attempting to prove privacy by showing the following to be unsat.

(1) (svt_gauss_00): $\exists \epsilon \in [0.1, 0.9], \neg\phi(\epsilon)$ given to solver.
(2) (svt_gauss_prd_00): $\exists \epsilon \in [0.1, 0.9], \neg\phi'(\epsilon)$ given to solver.

We fix $(u_1, v_1) = (1, 0)$ and $(u_2, v_2) = (0, 0)$. We set $\sigma = \sqrt{4/\epsilon^2}$, and $k$ should be an upper bound on $\sigma$. In practice, we vary $k$ to evaluate the scalability of $\int$dReal as described in Section 7.3; we evaluate $k \in [1, 2, 3, 4]$ for svt_gauss_00, and $k \in [4, 8, 12, \ldots, 48]$ for svt_gauss_prd_00.

## 7 Experiments

Our benchmark suite consists of examples from verifying privacy and accuracy properties of security algorithms, checking for fairness in machine learning models, and synthesis problems. They also include artificial examples involving double integrals and trignometric functions. Several of our examples feature quantifier alternation. For comparison, all examples are also implemented in Mathematica® and tested on version 13.3.1. All experiments were run on an Ubuntu 24.04 computer equipped with a 3.8 GHz AMD® Ryzen 7 Pro 7840HS processor and 32 GB of RAM. Benchmark examples were chosen to provide variety in the number of input variables, terms with integral expressions, and nesting depth of integrals, and include formulas with integral terms with constants for limits of integration and formulas with non-constants as limits of integration. For all the experiments, we run $\int$dReal with the parameter $\delta = 0.001$.

We organize the experimental results by dividing benchmarks into four groups: single integral, double integral, triple integral and quantifier alternation examples in Table 3. We refer to Table 1 for the explanation of the column names. Additionally, we refer to Table 2 to correlate motivating examples and case studies with our experimental queries.

Table 1. Legend for the column names used in Table 3.

| Column Name | Description |
|---|---|
| *fv* | is the number of free variables in the integrals. |
| *size* | is the maximum interval size of the free variables. |
| *n* | is the number of terms with integrals in the formula. |
| *b* | ✓ iff any integral in the formula contains a non-constant as an upper or lower limit. |
| time | The time represents the total time taken by $\int$dReal or Mathematica®, averaged over three executions, and is measured in seconds. |
| result | The result indicates the output from $\int$dReal or Mathematica®. |
| speed factor | The speed factor is the ratio of Mathematica® time over $\int$dReal time. |
| $k$ | serves as the upper bound for the integrals. |
| $-k$ | serves as the lower bound for the integrals. |
| $\epsilon$-width | represents the width of the range of $\epsilon$ |
| ind | denotes that Mathematica® couldn't resolve to true or false. |
| TO | denotes $\int$dReal timeout (we have a 10-minute threshold for timeouts). |
| tt | denotes true. |
| ff | denotes false. |

*Note on Mathematica®:* As discussed earlier, we choose Mathematica® since to the extent of our knowledge, it is the only tool that can handle the range of benchmark formulas we evaluate. We encode the examples as follows. For four examples containing no quantifiers, we use numerical integration to implement them in Mathematica®. For the other examples (43 in total) where we have quantification, we use Mathematica®'s symbolic integration in conjunction with the Resolve (https://reference.wolfram.com/language/ref/Resolve.html) and FindInstance (https://reference.wolfram.com/language/ref/FindInstance.html) functions in Mathematica®. Note that Mathematica® is a proprietary software and few details exist about how these functions are implemented internally.

*Note on specialized tools:* Since several of the examples are drawn from probabilistic programs, one might consider the use of tools such as PSI [14], and in the case of the fairness examples,

Table 2. Legend correlating case studies with queries.

| Case Study | Query |
|---|---|
| Motivating Example 2.1 | eth_colrank_fair_00, eth_colrank_fair_01 |
| Motivating Example 2.2 | gauss_forall_00 |
| Case Study 6.1 | high_inc_gd_00, high_inc_gd_01, high_inc_gd_unfair_00, high_inc_gd_unfair_01 |
| Case Study 6.2 | gauss_mech_00, gauss_mech_00_alt1 |
| Case Study 6.3 | lap_mech_00, lap_mech_00_not_pri, lap_mech_sat_01, lap_mech_sat_01_not_pri |
| Case Study 6.4 | svt_gauss_00, svt_gauss_sat_00, svt_gauss_prd_00 |

FairSquare [2]. PSI is inappropriate since to the extent of our knowledge, it computes a symbolic expression for the posterior probability density function, and does not help in checking the (un)satisfiability of formulas. FairSquare is only somewhat appropriate since it can only handle a subset of fairness problems. For example, FairSquare determines whether a hiring algorithm is fair using a fixed population model, while several of our queries determine whether that algorithm is fair across a range of (Gaussian) population models, where $\mu$ and $\sigma$ are varied. We do include performance results for FairSquare on determining fairness of programs with a fixed population model which correlate to our experiments in Table 4, although we acknowledge that this is not a perfect comparison, as FairSquare must also compute the appropriate formulas from the input program. It is important to note that for the fair programs in Table 4, an unsatisfiability result implies fairness, while on unfair programs, an unsatisfiability result implies unfairness.

### 7.1 Benchmarks

The first portion of Table 3 presents results for single integral examples. These include queries for group fairness [2] for women in salary allocation by employers, as shown in Case Study 6.1, as well as privacy and accuracy of the Laplace mechanism (privacy was shown in Case Study 6.3) [9]. The second portion of Table 3 presents the results for double integral examples. These include fairness of a hiring algorithm, as shown in Motivating Example 2.1 [2], privacy of the Gaussian mechanism, as in Case Study 6.2, and synthetic queries involving volume comparisons across different regions.

The third portion of Table 3 presents the results for triple integral examples. These include two versions of privacy queries for the Sparse Vector Technique (SVT) based on the Gaussian mechanism, adapted from the first query from Case Study 6.4: one is unsat, and the other is $\delta$-sat. The fourth and final portion of Table 3 presents the results for quantifier alternation examples, more specifically $\exists$-$\forall$ queries. These include synthesizing a parameter of a privacy mechanism in order to obtain the desired utility, as seen in Motivating Example 2.2. We also have different variants of volume comparison examples. All formulas being checked are of the form $\exists a.\forall \epsilon.\psi(a, \epsilon)$, contain double integrals, and we try to find the value of $a$ such that the formula holds true for a range of $\epsilon$.

### 7.2 Discussion

Some salient insights from our experiments are as follows.

- Whenever both $\int$dReal and Mathematica® give answers, their answers are almost always consistent: on all but one example, if $\int$dReal outputs "unsat" on an example, then Mathematica® outputs "false" on the corresponding example (similarly for the outputs "$\delta$-sat" and "true").

Table 3. Results for $\int$dReal and Mathematica® for benchmarks, split in sections from top to bottom: single integrals, double integrals, triple integrals and ∃-∀ queries.

| Benchmarks | | | | | $\int$dReal | | Mathematica® | | **Speed** |
|---|---|---|---|---|---|---|---|---|---|
| Example | *fv* | *size* | *n* | *b* | time (s) | result | time (s) | result | **factor** |
| high_inc_gd_00 | 0 | | 2 | × | 0.005 | unsat | 0.650 | ff | 130 |
| high_inc_gd_unfair_00 | 0 | | 2 | × | 0.005 | unsat | 0.644 | ff | 129 |
| lap_mech_00 | 1 | 0.9 | 2 | × | 0.007 | $\delta$-sat | 0.762 | ff | 109 |
| lap_mech_00_not_pri | 1 | 0.9 | 2 | × | 0.007 | $\delta$-sat | 0.843 | tt | 120 |
| lap_mech_sat_01 | 1 | 0.9 | 2 | × | 0.006 | $\delta$-sat | 0.822 | tt | 137 |
| lap_mech_sat_01_not_pri | 1 | 0.9 | 2 | × | 0.006 | unsat | 0.872 | ff | 145 |
| gauss_mech_00_alt1 | 1 | 0.9 | 4 | × | 0.015 | unsat | 1.601 | ff | 107 |
| lap_acc_00_alt2 | 2 | 0.999 | 1 | ✓ | 0.075 | $\delta$-sat | 18.531 | ind | 247 |
| lap_acc_01 | 2 | 0.999 | 1 | ✓ | 0.088 | $\delta$-sat | 14.366 | ind | 163 |
| lap_acc_02 | 2 | 0.999 | 1 | ✓ | 0.095 | $\delta$-sat | 14.557 | ind | 153 |
| lap_acc_00_alt1 | 2 | 0.999 | 2 | ✓ | 0.142 | $\delta$-sat | 25.894 | ind | 182 |
| lap_acc_01_alt1 | 2 | 0.999 | 3 | ✓ | 0.246 | $\delta$-sat | 35.266 | ind | 143 |
| lap_acc_02_alt1 | 2 | 0.999 | 4 | ✓ | 0.377 | $\delta$-sat | 45.006 | ind | 119 |
| high_inc_gd_01 | 2 | 40.0 | 2 | × | 0.004 | unsat | 4.122 | ind | 1030 |
| high_inc_gd_unfair_01 | 2 | 40.0 | 2 | × | 0.004 | unsat | 4.104 | ind | 1026 |
| eth_colrank_fair_00 | 0 | | 4 | ✓ | 0.020 | unsat | 0.767 | ff | 38 |
| eth_colrank_unfair_00 | 0 | | 4 | ✓ | 0.025 | unsat | 0.766 | ff | 31 |
| vol_cmp_00 | 1 | 0.1 | 2 | ✓ | 0.005 | unsat | 0.951 | ff | 190 |
| vol_cmp_01 | 1 | 0.1 | 2 | ✓ | 0.007 | $\delta$-sat | 0.943 | tt | 135 |
| vol_cmp_02 | 1 | 0.1 | 2 | ✓ | 0.005 | unsat | 1.712 | ff | 342 |
| vol_cmp_03 | 1 | 0.1 | 2 | ✓ | 0.007 | $\delta$-sat | 1.710 | tt | 244 |
| vol_cmp_04 | 1 | 0.1 | 2 | ✓ | 0.004 | unsat | 1.571 | ff | 393 |
| vol_cmp_05 | 1 | 0.1 | 2 | ✓ | 0.008 | $\delta$-sat | 1.553 | tt | 194 |
| vol_cmp_06 | 1 | 0.1 | 2 | ✓ | 0.045 | $\delta$-sat | 2.164 | tt | 48 |
| vol_cmp_07 | 1 | 0.1 | 2 | ✓ | 0.021 | $\delta$-sat | 2.163 | tt | 103 |
| vol_cmp_08 | 1 | 0.1 | 2 | ✓ | 0.005 | unsat | 2.021 | ind | 404 |
| vol_cmp_09 | 1 | 0.1 | 2 | ✓ | 0.007 | $\delta$-sat | 2.121 | ind | 303 |
| vol_cmp_10 | 1 | 0.1 | 2 | ✓ | 0.004 | unsat | 1.302 | ff | 326 |
| vol_cmp_11 | 1 | 0.1 | 2 | ✓ | 0.008 | $\delta$-sat | 1.263 | tt | 158 |
| gauss_mech_00 | 1 | 0.9 | 2 | × | 53.082 | unsat | 2.571 | ff | <1 |
| eth_colrank_fair_01 | 2 | 10 | 4 | ✓ | 233.267 | unsat | 159.626 | ind | <1 |
| eth_colrank_unfair_01 | 2 | 10 | 4 | ✓ | 15.450 | unsat | 156.861 | ind | 10 |
| svt_gauss_00 | 1 | 0.4 | 2 | ✓ | 3.972 | unsat | 1.445 | ff | <1 |
| svt_gauss_sat_00 | 1 | 0.4 | 2 | ✓ | 274.049 | $\delta$-sat | 1.458 | tt | <1 |
| gauss_forall_00 | 1 | 0.5 | 1 | ✓ | 255.687 | $\delta$-sat | 126.654 | ind | <1 |
| vol_cmp_00_forall | 1 | 1 | 2 | ✓ | 0.006 | $\delta$-sat | 1.081 | $a \leftarrow \frac{43}{34}$ | 180 |
| vol_cmp_02_forall | 1 | 1 | 2 | ✓ | 0.005 | $\delta$-sat | 1.509 | ind | 302 |
| vol_cmp_04_forall | 1 | 1 | 2 | ✓ | 0.006 | $\delta$-sat | 1.373 | ind | 229 |
| vol_cmp_06_forall | 1 | 1 | 2 | ✓ | 0.026 | $\delta$-sat | 1.586 | ind | 61 |
| vol_cmp_08_forall | 1 | 1 | 2 | ✓ | 0.006 | $\delta$-sat | 1.153 | ind | 192 |
| vol_cmp_10_forall | 1 | 1 | 2 | ✓ | 0.006 | $\delta$-sat | 1.190 | ind | 198 |

Table 4. Experimental results comparing the time taken by FairSquare to check programs for fairness versus the time taken by ∫dReal to solve the corresponding fairness/unfairness query. For queries given to ∫dReal, on fair programs, unsatisfiability implies fairness, and on unfair programs, unsatisfiability implies unfairness.

| Benchmarks | FairSquare | | ∫dReal | |
|---|---|---|---|---|
| Example | time | result | time | result |
| high_inc_gd_00 | 1.014 | fair | 0.004 | unsat |
| high_inc_gd_unfair_00 | 1.031 | unfair | 0.004 | unsat |
| eth_colrank_fair_00 | 5.189 | fair | 0.02 | unsat |
| eth_colrank_unfair_00 | 2.51 | unfair | 0.025 | unsat |

- On the one example where the output from ∫dReal and Mathematica® does not correspond (lap_mech_00), ∫dReal outputs "$\delta$-sat" but Mathematica® outputs "false". In this example, we check whether one expression is strictly less than the other. These two expressions are actually equal, but ∫dReal is unable to conclude the falsehood of this formula. Nevertheless, the answer returned by ∫dReal is sound.
- The performance of ∫dReal on single and double integral examples is significantly faster compared to Mathematica® in most examples. Also, for all these examples, ∫dReal can give an answer, while Mathematica® sometimes cannot resolve the formula to a boolean value.
- In the case of triple integrals, we were able to solve some examples by setting the integral interval small, i.e., from -4 to 4, and using a smaller range of $\epsilon$. However, Mathematica® outperformed ∫dReal in all triple integral examples.
- In the case of quantifier alternation, our tool successfully solves examples featuring nested integrals and functions such as exponentiation and trigonometric sine. Conversely, Mathematica® demonstrates limited success, solving only one example while requiring a significantly longer computation time for that example compared to our tool.

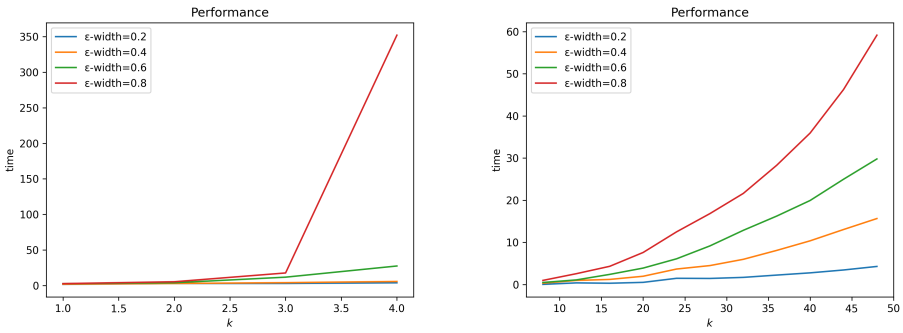
### 7.3 Scaling Behavior Analysis

We additionally evaluate ∫dReal's scalability by executing families of queries in which a particular attribute is varied. We evaluate scalability by varying integral width, interval width of variables, number of variables, and number of integral subterms.

To study scalability of ∫dReal with respect to integral width and interval width of variables, we turn to the query for verifying privacy of the Sparse Vector Technique (SVT) with triple integrals and double integrals, as discussed in Case Study 6.4. We first examine scalability with respect to integral width: the largest integral in these examples ranges from $-k$ to $k$. With triple integrals, our tool could not handle integrals with limits greater than $k = 4$ without timing out after 10 minutes. However, by redefining the query to an equivalent one with double integrals, we were able to handle integrals with limits up to $k = 48$, resulting in both better scalability and performance. To examine scalability with regard to the interval width of the existentially-quantified $\epsilon$, note that in the case of double integrals, as the width increases, time increases quadratically, and in triple integrals, time increases more rapidly. These results are illustrated by Figure 4.

To examine how ∫dReal scales with respect to the number of variables, we construct the following family of queries $\psi_1(n)$, in which the number of existentially-quantified variables is parameterized by $n$:

$$\psi_1(n) = \exists d_1 \in [0,1], \ldots, d_n \in [0,1], \neg\phi(n, d_1, \ldots, d_n),$$



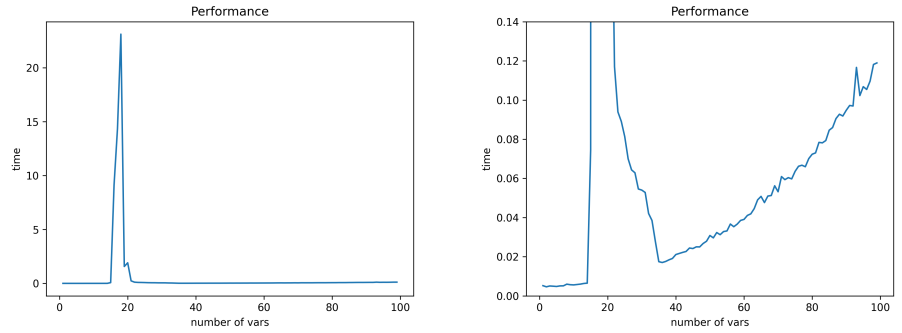

(a) svt_gauss_00 (b) svt_gauss_prd_00

Fig. 4. Figure 4a and 4b depict the scaling behavior and performance comparison of svt_gauss_00 and svt_gauss_prd_00, varying the width of the integral $k$, and the width of the interval for the existentially quantified variable $\epsilon$.

where $\phi$ is defined as follows, with $\epsilon = k = 1$:

$$\phi(n, d_1, \ldots, d_n) = \frac{1}{2\pi\sqrt{\frac{4}{\epsilon^2}}^3\sqrt{2\pi}}\left(\int_{-k}^{k} e^{-\left(\frac{(y-1)(y-1)}{8\epsilon^2}\right)}dy + \sum_{i=1}^{n}\left(\int_{-k}^{d_i} e^{-\left(\frac{(y-0)(y-0)\epsilon^2}{8}\right)}dy\right)\right) \leq \frac{1}{4}.$$

We run $\psi_1(n)$ for $n \in 1, \ldots, 99$, with performance results shown in Figure 5. Note that for all $n \leq 16$, $\int$dReal returns unsat, while for all $n \geq 17$, $\int$dReal returns $\delta$-sat — the intuition behind this being that if one adds enough integrals on the left-hand side, it will be bigger than any constant on the right-hand side. This becomes relevant for performance since for this family of queries, as the number of existentially-quantified variables increases, the performance increases linearly except for the transition between unsat and $\delta$-sat, where run time peaks. For the sake of illustrating the performance here, we present the results both in full, in Figure 5a, and by not displaying peak values, in Figure 5b.




(a) Performance vs. Num. Variables (b) Performance vs. Num. Variables (no peak)

Fig. 5. Figure 5a and 5b depict the performance behavior of $\psi_1(n)$ as the number of existentially quantified variables $n$ is varied from 1 to 99. In 5b, the maximum value of the y-axis is 0.14 seconds to show how performance scales without the impact of the peak around $n = 19$.

To examine how ∫dReal scales with respect to the number of integral subterms, we construct the following two families of queries $\psi_2$ and $\psi_3$, in which the number of existentially-quantified variables is parameterized by $n$ (note that $m = 2, 3$). $\psi_2$ represents a sum of $n$ integrals, while $\psi_3$ represents a product of $n$ integrals nested inside an outer integral:

$$\psi_m(n) = \exists \epsilon \in [0.5, 0.6], \neg \phi_m(n, \epsilon),$$

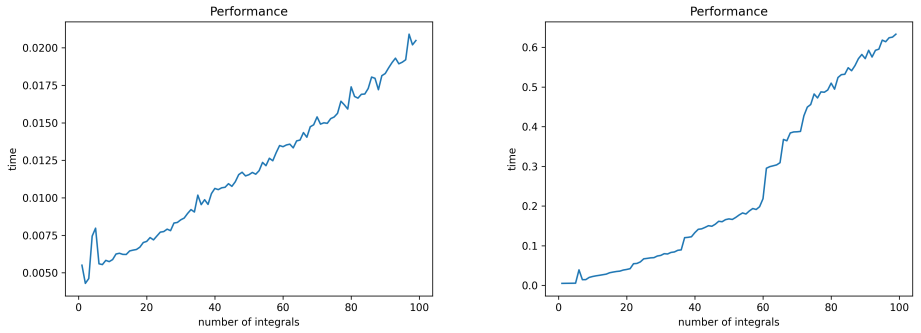
where $\phi_2$ is defined as follows, with $k = 1$:

$$\phi_2(n,\epsilon) = \frac{1}{2\pi\sqrt{\frac{4}{\epsilon^2}}^3\sqrt{2\pi}}\left(\int_{-k}^{k} e^{-\left(\frac{(y-1)(y-1)}{8\epsilon^2}\right)}dy + \sum_{i=2}^{n}\left(\int_{-k}^{k} e^{-\left(\frac{(y-0)(y-0)i\epsilon^2}{8}\right)}dy\right)\right) \leq \frac{1}{100},$$

and $\phi_3$ is defined as follows, with $k = 1$:

$$\phi_3(n,\epsilon) = \frac{1}{2\pi\sqrt{\frac{4}{\epsilon^2}}^3\sqrt{2\pi}}\left(\int_{-k}^{k}\left(\int_{-k}^{k} e^{-\left(\frac{(y-1)(y-1)}{8\epsilon^2}\right)}dy\right)\left(\prod_{i=2}^{n}\int_{-k}^{z} e^{-\left(\frac{(y-0)(y-0)\epsilon^2}{8}\right)}dy\right)\left(e^{-\left(\frac{z^2\epsilon^2}{8}\right)}\right)dz\right) \leq \frac{1}{100},$$

We run $\psi_2(n)$ and $\psi_3(n)$ for $n \in 1, \ldots, 99$, with performance results shown in Figure 6. For $\psi_2$, we have that for all $n \leq 3$, ∫dReal returns unsat, while for all $n \geq 4$, ∫dReal returns $\delta$-sat, and for $\psi_3$, we have that for all $n \leq 5$, ∫dReal returns unsat, while for all $n \geq 6$, ∫dReal returns $\delta$-sat. One can observe that performance increases roughly linearly with the number of integral terms — one also observes that this performance is not spoiled by one layer of nested integration.




(a) Performance of $\psi_2$ vs. Num. Integrals

(b) Performance of $\psi_3$ vs. Num. Integrals

Fig. 6. Figure 6a and 6b depict the performance behavior of $\psi_2(n)$ and $\psi_3(n)$ as $n$, the number of integral expressions and inner integral expressions respectively, is varied from 1 to 99.

## 8 Conclusions and Future Work

The framework of $\delta$-decision procedures, with its support for a wide variety of non-linear real functions, shows much promise in solving a range of verification and synthesis problems. To this end, we extend an existing tool, dReal, with support for integrals to create ∫dReal. We believe that the value in adding support for terms with integration is its application in a range of examples, some of which are described in this paper. We evaluate ∫dReal on a variety of queries taken from domains including checking the fairness of machine learning models, and the privacy and the accuracy of differential privacy mechanisms. The performance of ∫dReal on many of these benchmarks over symbolic tools such as Mathematica® shows great promise for this approach's use on a broader

range of verification and synthesis problems. As part of future work, we plan to extend $\int$dReal with additional operations such as volume integration and partial derivatives.

## Data Availability Statement

There is a publicly available artifact providing both the implementation of $\int$dReal as well as our input files [24]. It also provides the equivalent Wolfram Mathematica implementations of the relevant files, as well as FairSquare programs corresponding to the relevant fairness queries. An earlier version was submitted for artifact evaluation [23].

## Acknowledgements

This work was partially supported by the National Science Foundation: Bishnu Bhusal and Rohit Chadha were partially supported by grant CCF 1900924, A. Prasad Sistla was partially supported by grant CCF 1901069, and Mahesh Viswanathan was partially supported by grants CCF 1901069 and CCF 2007428.

## References

[1] Accessed 2023. DReal4. https://github.com/dreal/dreal4.

[2] Aws Albarghouthi, Loris D'Antoni, Samuel Drews, and Aditya V. Nori. 2017. FairSquare: Probabilistic Verification of Program Fairness. *Proc. ACM Program. Lang.* 1, OOPSLA, Article 80 (oct 2017), 30 pages. https://doi.org/10.1145/3133904

[3] Gilles Barthe, Rohit Chadha, Paul Krogmeier, A Prasad Sistla, and Mahesh Viswanathan. 2021. Deciding accuracy of differential privacy schemes. *Proceedings of the ACM on Programming Languages* 5, POPL (2021), 1–30. https://doi.org/10.1145/3434289

[4] Fabian Bauer-Marquart, Stefan Leue, and Christian Schilling. 2023. symQV: Automated Symbolic Verification of Quantum Programs. In *Formal Methods*, Marsha Chechik, Joost-Pieter Katoen, and Martin Leucker (Eds.). Springer International Publishing, Cham, 181–198. https://doi.org/10.1007/978-3-031-27481-7_12

[5] Frédéric Benhamou and Laurent Granvilliers. 2006. Chapter 16 - Continuous and Interval Constraints. In *Handbook of Constraint Programming*, Francesca Rossi, Peter van Beek, and Toby Walsh (Eds.). Foundations of Artificial Intelligence, Vol. 2. Elsevier, 571–603. https://doi.org/10.1016/S1574-6526(06)80020-9

[6] Rohit Chadha, A. Prasad Sistla, Mahesh Viswanathan, and Bishnu Bhusal. 2023. Deciding Differential Privacy of Online Algorithms with Multiple Variables *(CCS '23)*. Association for Computing Machinery, New York, NY, USA, 1761–1775. https://doi.org/10.1145/3576915.3623170

[7] Cynthia Dwork. 2006. Differential Privacy. In *Automata, Languages and Programming*, Michele Bugliesi, Bart Preneel, Vladimiro Sassone, and Ingo Wegener (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 1–12. https://doi.org/10.1007/11787006_1

[8] Cynthia Dwork, Moni Naor, Omer Reingold, Guy N Rothblum, and Salil Vadhan. 2009. On the complexity of differentially private data release: efficient algorithms and hardness results. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*. 381–390. https://doi.org/10.1145/1536414.1536467

[9] Cynthia Dwork and Aaron Roth. 2014. The Algorithmic Foundations of Differential Privacy. *Found. Trends Theor. Comput. Sci.* 9, 3–4 (aug 2014), 211–407. https://doi.org/10.1561/0400000042

[10] A. Eggers, N. Ramdani, N. Nedialkov, and M. Fränzle. 2011. Improving SAT Modulo ODE for Hybrid Systems Analysis by Combining Different Enclosure Methods. In *Proceedings of the International Conference on Software Engineering and Formal Methods*. 172–187. https://doi.org/10.1007/978-3-642-24690-6_13

[11] Sicun Gao, Jeremy Avigad, and Edmund M. Clarke. 2012. $\delta$-Complete Decision Procedures for Satisfiability over the Reals. In *Automated Reasoning*, Bernhard Gramlich, Dale Miller, and Uli Sattler (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 286–300. https://doi.org/10.1007/978-3-642-31365-3_23

[12] Sicun Gao, Soonho Kong, and Edmund M. Clarke. 2013. dReal: An SMT Solver for Nonlinear Theories over the Reals. In *Automated Deduction – CADE-24*, Maria Paola Bonacina (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 208–214. https://doi.org/10.1007/978-3-642-38574-2_14

[13] Sicun Gao, Soonho Kong, and Edmund M. Clarke. 2013. Satisfiability modulo ODEs. In *2013 Formal Methods in Computer-Aided Design*. 105–112. https://doi.org/10.1109/FMCAD.2013.6679398

[14] Timon Gehr, Sasa Misailovic, and Martin Vechev. 2016. PSI: Exact Symbolic Inference for Probabilistic Programs. In *Computer Aided Verification*, Swarat Chaudhuri and Azadeh Farzan (Eds.). Springer International Publishing, Cham, 62–83. https://doi.org/10.1007/978-3-319-41528-4_4

[15] Ibex Team. Accessed 2023. Ibex Library. https://ibex-team.github.io/ibex-lib/.

[16] Fredrik Johansson. 2014. Arb: A C Library for Ball Arithmetic. *ACM Commun. Comput. Algebra* 47, 3/4 (jan 2014), 166–169. https://doi.org/10.1145/2576802.2576828

[17] Fredrik Johansson. 2017. New rigorous numerical integration in Arb. https://fredrikj.net/blog/2017/11/new-rigorous-numerical-integration-in-arb/.

[18] K.-I. Ko. 1991. *Complexity Theory of Real Functions*. Birkhauser. https://doi.org/10.1007/978-1-4684-6802-1

[19] Soonho Kong, Sicun Gao, Wei Chen, and Edmund M. Clarke. 2015. dReach: $\delta$-Reachability Analysis for Hybrid Systems. In *Tools and Algorithms for the Construction and Analysis of Systems - 21st International Conference, TACAS 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015. Proceedings*. 200–205. https://doi.org/10.1007/978-3-662-46681-0_15

[20] Soonho Kong, Armando Solar-Lezama, and Sicun Gao. 2018. Delta-Decision Procedures for Exists-Forall Problems over the Reals. In *Computer Aided Verification*, Hana Chockler and Georg Weissenbacher (Eds.). Springer International Publishing, Cham, 219–235. https://doi.org/10.1007/978-3-319-96142-2_15

[21] Min Lyu, Dong Su, and Ninghui Li. 2017. Understanding the Sparse Vector Technique for Differential Privacy. *Proc. VLDB Endow.* 10, 6 (feb 2017), 637–648. https://doi.org/10.14778/3055330.3055331

[22] Stefan Ratschan and Zhikun She. 2007. Safety Verification of Hybrid Systems by Constraint Propagation Based Abstraction Refinement. *ACM Transactions in Embedded Computing Systems* 6, 1 (2007). https://doi.org/10.1007/978-3-540-31954-2_37

[23] Cody Rivera, Bishnu Bhusal, Rohit Chadha, Aravinda Prasad Sistla, and Mahesh Viswanathan. 2025. *Artifact for Paper Submission "Checking $\delta$-Satisfiability of Reals with Integrals (v1)"*. https://doi.org/10.5281/zenodo.14593603

[24] Cody Rivera, Bishnu Bhusal, Rohit Chadha, Aravinda Prasad Sistla, and Mahesh Viswanathan. 2025. *Artifact for Paper Submission "Checking $\delta$-Satisfiability of Reals with Integrals (v2)"*. https://doi.org/10.5281/zenodo.14948095

[25] Halsey L. Royden. 1988. *Real analysis* (3rd ed.). Macmillan, New York.

[26] Danny De Schreye. 1999. Revising hull and box consistency. In *Logic Programming: Proceedings of the 1999 International Conference on Logic Programming*. 230–244. https://doi.org/10.7551/mitpress/4304.003.0024

[27] Lloyd N. Trefethen. 2008. Is Gauss Quadrature Better than Clenshaw–Curtis? *SIAM Rev.* 50, 1 (2008), 67–87. https://doi.org/10.1137/060659831

[28] Qinsi Wang, Paolo Zuliani, Soonho Kong, Sicun Gao, and Edmund M. Clarke. 2015. SReach: A Probabilistic Bounded Delta-Reachability Analyzer for Stochastic Hybrid Systems. In *Computational Methods in Systems Biology*, Olivier Roux and Jérémie Bourdon (Eds.). Springer International Publishing, Cham, 15–27. https://doi.org/10.1007/978-3-319-23401-4_3

## A Proof of Theorem 3

Proof. Since the set of $n$-dimensional boxes is finite, and a well-defined Prune always returns a smaller box (W1), the loop in lines 10–11 always terminates. We need to demonstrate that the loop in lines 8–17 always terminates.

Consider the subsets of $\mathbb{R}^n$ that can be formed using the union of a collection of $n$-dimensional boxes in $\mathbb{BF}$. Formally,

$$\mathcal{H}^n = \{\cup_{i\in J} B_i \mid \{B_i \mid i \in J\} \subseteq \mathbb{F}^n\}.$$

Clearly, $\mathcal{H}^n$ is finite. A sequence of sets $C_1 \supseteq C_2 \supseteq \cdots \supseteq C_n \supseteq \cdots$ is said to be *eventually strictly decreasing* if that for each $i$,

(1) $C_i \in \mathcal{H}^n$, and
(2) if $C_i \neq \emptyset$ then there exists $k_i > i$. $C_i \supsetneq C_{k_i}$.

It is easy to see that an eventually strictly decreasing sequence is either finite or there is an $\ell$ such that $C_j = \emptyset$ for all $j \geq \ell$.

Let $S_i$ denote the stack at the beginning of the $i$ iteration of the loop in lines 8–17, let $C_i$ denote the set formed by the union of all the boxes in $S_i$, and let $S_i.top$ denotes the topmost box in $S_i$.

From the Branch operator and well-definedness of Prune, it is easily seen that $C_i \supseteq C_{i+1}$ for each $i$. Furthermore, one of the following must be true for each $i$:

(1) $S_i.top$ is removed in iteration $i$ from the stack, and no element is pushed onto $S_i$. In this case, it is easy to see that $C_i \supsetneq C_{i+1}$.
(2) $S_i.top$ is removed from the stack in iteration $i$ and replaced by one or two intervals such $S_i.top \supsetneq S_{i+1}.top$.
(3) $S_i.top$ is a box $B = I_1 \times \ldots \times I_n$ such that for each $k$, $I_k = [a_k, b_k]$ with $b_k - a_k \leq 2^{-\text{modulus}(\mathcal{F}, \frac{\delta}{4}, B_0)}$. In this case, observe that well-definedness of Prune (W1) means that for each $j$, either $\text{Prune}(B, f_j^\#) = \emptyset$ or $\text{Prune}(B, f_j^\#) = B$. Thus, either $S_i.top$ will be removed (in which case we get $C_{i+1} \supsetneq C_i$) or $\text{Prune}(B, f_j^\#) = B$ for each $j$.
Assume that $\text{Prune}(B, f_j^\#) = B$ for each $j$. By well-definedness of Prune (W2), this implies that $0 \in f_j^\#(B)$ for each $j$. $\delta$-proximality implies that for each $j$, there exists an $\mathbf{x}_j \in B$ such that $|0 - f_j(\mathbf{x}_j)| < \frac{\delta}{3}$, ie, $|f_j(\mathbf{x}_j)| < \frac{\delta}{3}$. We also have by $\delta$-proximality that for each $z \in f_j^\#(B)$ there is a $\mathbf{x}_{j,z} \in B$ such that $|z - f(\mathbf{x}_{j,z}))| < \frac{\delta}{3}$. Thus, we get by the triangle inequality, for each $z \in f_j^\#(B)$,

$$\begin{aligned} |z| &< |z - f(\mathbf{x}_{j,z})| + |f(\mathbf{x}_{j,z}) - f_j(\mathbf{x}_j)| + |f_j(\mathbf{x}_j)| \\ &< \frac{2\delta}{3} + |f(\mathbf{x}_{j,z}) - f_j(\mathbf{x}_j)| \end{aligned}$$

From the fact that $b_k - a_k < 2^{-\text{modulus}(\mathcal{F}, \frac{\delta}{4}, B_0)}$, we get that $|f(\mathbf{x}_{j,z}) - f_j(\mathbf{x}_j)| < \frac{\delta}{4}$. Thus for each $z \in f_j^\#(B)$, $|z| \leq \frac{11\delta}{12}$. Thus, $|f^\#(B_j)| < \delta$ and in this case, the algorithm must return sat.

Since the number of $n$-dimensional boxes is finite, it is easy to see that the above observations imply that the sequence $C_1 \supseteq C_2 \supseteq \cdots$ is eventually strictly decreasing. Thus, there is an $\ell$ such that either $S_\ell$ is empty or the loop terminates in iteration $\ell$ returning sat. In the former case, the algorithm terminates and returns unsat. □

## B Proof of Theorem 5

Proof. The proof proceeds by induction on the term language in Equation 13. As the composition of computable functions is computable, and the addition, multiplication, subtraction, exponentiation, division, trigonometric functions, and logarithms are computable, it suffices to show that for any

computable function $f : \mathbb{R}^{m+1} \to R$, and computable numbers $r_\ell, r_u$, the function $g : \mathbb{R}^{m+2} \to \mathbb{R}$ defined as

$$g(r_\ell, r_u, r_1, \ldots, r_m) = \int_{r_\ell}^{r_u} f(z, r_1, \ldots, r_m)dz$$

is a computable function.

In order to show this, we outline how $M_g^{\langle r_\ell\rangle,\langle r_u\rangle,\langle r_1\rangle,\ldots,\langle r_m\rangle}$, the function oracle Turing Machine for $g$ with oracles for$\langle r_\ell\rangle, \langle r_u\rangle, \langle r_1\rangle, \ldots, \langle r_m\rangle$ can be constructed. For brevity, we shall use $M_g$ to denote $M_g^{\langle r_\ell\rangle,\langle r_u\rangle,\langle r_1\rangle,\ldots,\langle r_m\rangle}$.

On input $i$, $M_g$ is supposed to output a value $v$ such that $|g(r_\ell, r_u, r_1, \ldots, r_m) - v| \leq \frac{1}{2^i}$ When $M_g$ starts on input $i$, it computes using its oracles, closed intervals $I_0, I_1, \ldots, I_m$ such that $r_\ell, r_u \in I_0$ and $r_k \subseteq I_k$ for $1 \leq k \leq m$. $M_g$ also computes a bound $L \geq 1$ such that $|r_u - r_\ell| \leq L$. Further, by simulating $f$, it then computes a bound $B \geq 1$ such that $|f(r_u, r_1, \ldots r_m)|, |f(r_\ell, r_1, \ldots r_m)| \leq B$.

Let $C = I_0 \times I_1 \times \cdots \times I_m$. Observe that $C$ is compact. Since $f$ is computable, $M_g$ then computes a $j$ such $|f(x) - f(y)| \leq \frac{1}{2^{i+2}L}$ for each $x, y \in C$ such that $||x - y|| \leq \frac{1}{2^j}$. Without loss of generality, we can assume $j \geq i + 3$.

$M_g$ computes, using the oracles of $r_\ell$ and $r_u$, intervals $[a_0, a_1]$ and $[b_0, b_1]$ such that $r_\ell \in [a_0, a_1], r_u \in [b_0, b_1], a_1 - a_0 \leq \frac{1}{2^j B}$ and $b_1 - b_0 \leq \frac{1}{2^j B}$. We consider three mutually exclusive and exhaustive cases:

(1) Case 1: $(a_1 < b_0)$. Observe that we have

$$\int_{r_\ell}^{r_u} f(z, r_1, \ldots, r_m)dz = \int_{r_\ell}^{a_1} f(z, r_1, \ldots, r_m)dz + \int_{a_1}^{b_0} f(z, r_1, \ldots, r_m)dz + \int_{b_0}^{r_u} f(z, r_1, \ldots, r_m)dz.$$

Now, observe that by construction of $j$, $|f(z, r_1, \ldots, r_m) - f(r_\ell, r_1, \ldots, r_m)| \leq \frac{1}{2^{i+2}L}$ for each $z \in [r_\ell, a_1]$. Thus, $|f(z, r_1, \ldots, r_m)| \leq |f(r_\ell, r_1, \ldots, r_m)| + \frac{1}{2^{i+2}L} \leq B + \frac{1}{2^{i+2}L} \leq 2B$. Hence, we have that

$$|\int_{r_\ell}^{a_1} f(z, r_1, \ldots, r_m)dz| \leq 2B(a_1 - r_\ell) \leq 2B\frac{1}{2^j B} \leq \frac{2}{2^j} \leq \frac{1}{2^{i+2}}$$

where the last inequality follows from the assumption that $j \geq i + 3$. Similarly,

$$|\int_{b_0}^{r_u} f(z, r_1, \ldots, r_m)dz| \leq \frac{1}{2^{i+2}}.$$

Thus, if $M_g$ can compute $\int_{a_1}^{b_0} f(z, r_1, \ldots, r_m)dz$ up to the precision of $\frac{1}{2^{i+1}}$, then that value is the value of $g(r_u, r_\ell, r_1, \ldots, r_m)$ up to a precision of $\frac{1}{2^i}$. Now, in order to compute $\int_{a_1}^{b_0} f(z, r_1, \ldots, r_m)dz$, $M_g$ constructs rational numbers $a_1 = c_0 < c_1 < \cdots < c_n = b_1$ such that $c_{k+1} - c_k < \frac{1}{2^j}$ for each $0 \leq k < n$. We have by construction of $j$, for each $0 \leq k < n$ and each $z \in [c_k, c_{k+1}]$

$$f(c_k, r_1, \ldots, r_m) - \frac{1}{2^{i+2}L} \leq f(z, r_1, \ldots, r_m) \leq f(c_k, r_1, \ldots, r_m) + \frac{1}{2^{i+2}}.$$

From this, it is easy to see that

$$|\int_{a_1}^{b_0} f(z, r_1, \ldots, r_m)dz - \sum_{k=0}^{n-1} f(c_k, r_1, \ldots, r_m)(c_{k+1} - c_k)| \leq \frac{b_0 - a_1}{2^{i+2}L} \leq \frac{1}{2^{i+2}L}$$

where the last equality follows from the fact that $|r_u - r_\ell| \leq L$.
Now, thanks to the computability of $f$, $M_g$ computes the Reimann sum

$$\sum_{k=0}^{n-1} f(c_k, r_1, \ldots, r_m)(c_{k+1} - c_k)$$

up to a precision of $\frac{1}{2^{i+2}}$. If $v$ is the sum computed, it can be easily seen that

$$|\int_{a_1}^{b_0} f(z, r_1, \ldots, r_m)dz - v| \leq \frac{1}{2^{i+1}}.$$

Thus, $v$ is the required value and $M_g$ outputs $v$.

(2) Case 2: ($b_1 \leq a_0$). In this case $M_g$ computes $\int_{b_1}^{a_0} f(z, r_1, \ldots, r_m)dz$ up to a precision of $\frac{1}{2^{i+1}}$ similar to the computation of $\int_{a_1}^{b_0} f(z, r_1, \ldots, r_m)dz$ in the above case. If $v$ is the value computed, $M_g$ outputs $-v$. That the output value is the desired value can be seen via an argument similar to the first Case.

(3) Case 3: ($(a_0, a_1) \cap (b_0, b_1) \neq \emptyset$). Note that this implies that $|r_u - r_\ell| \leq \frac{2}{2^j B}$. Assume that $r_\ell \leq r_u$. The case when $r_u \leq r_\ell$ is similar. Now, for any $z \in [r_\ell, r_u]$, we have either $|f(z, r_1, \ldots r_k) - f(r_\ell, r_1, \ldots r_k)| \leq \frac{1}{2^{i+2}L}$ or $|f(z, r_1, \ldots r_k) - f(r_u, r_1, \ldots r_k)| \leq \frac{1}{2^{i+2}L}$. Thus, for any $z \in [r_\ell, r_u]$, we have that $|f(z, r_1, \ldots r_k)| \leq 2B$. Thus,

$$|\int_{r_\ell}^{r_u} f(z, r_1, \ldots, r_m)dz| \leq 2B(r_u - r_\ell) \leq 2B\frac{2}{2^j B} \leq \frac{1}{2^{i+1}}.$$

Thus, 0 is an approximation of $\int_{r_\ell}^{r_u} f(z, r_1, \ldots, r_m)dz$ up to a precision of $\frac{1}{2^{i+1}}$ and $M_g$ can output 0. □

Received 2024-10-16; accepted 2025-02-18