Smoothing a Program Soundly and Robustly*

Swarat Chaudhuri[1] and Armando Solar-Lezama[2]

[1] Rice University
[2] MIT

Abstract. We study the foundations of *smooth interpretation*, a recently-proposed program approximation scheme that facilitates the use of local numerical search techniques (e.g., gradient descent) in program analysis and synthesis. While the popular techniques for local optimization works well only on relatively smooth functions, functions encoded by real-world programs are infested with discontinuities and local irregular features. Smooth interpretation attenuates such features by taking the convolution of the program with a Gaussian function, effectively replacing discontinuous switches in the program by continuous transitions. In doing so, it extends to programs the notion of *Gaussian smoothing*, a popular signal-processing technique used to filter noise and discontinuities from signals.

Exact Gaussian smoothing of programs is undecidable, so algorithmic implementations of smooth interpretation must necessarily be approximate. In this paper, we characterize the approximations carried out by such algorithms. First, we identify three correctness properties—*soundness*, *robustness*, and *β-robustness*—that an approximate smooth interpreter should satisfy. In particular, a smooth interpreter is sound if it computes an abstraction of a program's "smoothed" semantics, and robust if it has arbitrary-order derivatives in the input variables at every point in its input space. Second, we describe the design of an approximate smooth interpreter that provably satisfies these properties. The interpreter combines program abstraction using a new domain with symbolic calculation of convolution.

1 Introduction

Smooth interpretation [5] is a recently-proposed program transformation permitting more effective use of numerical optimization in automated reasoning about programs. Many problems in program analysis and synthesis can be framed as optimization questions—examples include finding program parameters so that the resultant program behavior is *as close as possible* to a specification [5], or the generation of tests that maximize the number of times a certain operation is executed [2]. But rarely are such problems solvable using off-the-shelf numerical optimization engines. This is because search spaces arising in real-world programs are rife with discontinuities—on such spaces, local search algorithms like gradient descent or Newton iteration find themselves unable to converge on a good solution. It is this predicament that smooth interpretation tries to resolve.

* This work was supported by NSF CAREER Award #0953507 and the MIT CSAIL.

To aid numerical search over the input space of a program P, smooth interpretation transforms P into a smooth mathematical function. For example, if the semantics of P (viewed as a function $P(x_1, x_2)$ of inputs x_1 and x_2) is the discontinuous map in Fig. 1-(a), the "smoothed" version of P will typically have semantics as in Fig. 1-(b). More precisely, smooth interpretation extends to programs the notion of *Gaussian smoothing* [13], an elementary signal-processing technique for filtering out noise and discontinuities from ill-behaved real-world signals. To perform Gaussian smoothing of a signal, one takes the *convolution* of the signal with a Gaussian function. Likewise, to smooth a program, we take the convolution of the denotational semantics of P with a Gaussian function.

The smoothing transformation is parameterized by a value β which controls the degree of smoothing. Numerical search algorithms will converge faster and find better local minima when β is high and the function is very smooth, but a bigger β also introduces imprecision, as the minima that is found when using a big β may be far from the real minima. The numerical optimization algorithm from [5] addresses this by starting with a high value of β then reducing it every time a minima is found. When β is reduced, the location of the last minima is used as a starting point for a new round of numerical search.

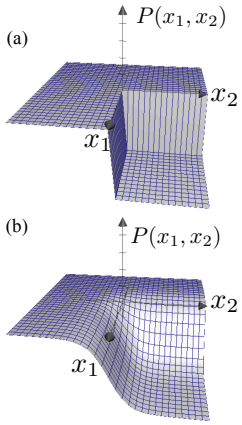

Fig. 1. (a) A discontinuous program. (b) After smoothing.

In our previous work, we showed the effectiveness of this approach for the problem of embedded controller synthesis. Specifically, we showed that the algorithm defined above could find optimal parameters for interesting controllers where simple numerical search would fail. But the benefits of the technique are not limited to parameter synthesis; smooth interpretation constitutes a wholly new form of program approximation, and is likely to have broad impact by opening the door to a wide array of applications of numerical optimization in program analysis.

Smooth interpretation exhibits many parallels with program abstraction, but it also introduces some new and important concerns. The goal of this paper is to understand these concerns by analyzing the foundations of smooth interpretation. In particular, we seek to characterize the approximations that must be made by algorithmic implementations of program smoothing given that computing the exact Gaussian convolution of an arbitrary program is undecidable.

Our concrete contributions are the following:

1. We identify three correctness properties that an algorithmic (and therefore approximate) implementation of smooth interpretation should ideally satisfy: *soundness*, *robustness*, and *β-robustness*.
   While the notion of soundness here is related to the corresponding notion in program abstraction, the two notions are semantically quite different: a sound smooth interpreter computes an abstraction of a "smoothed" semantics of programs. As for robustness, this property states that the function com-

puted by an approximate smooth interpreter has a linearly bounded derivative at every point in its input space—i.e., that even at the points in the input space where P is discontinuous or non-differentiable, the smoothed version of P is only as "steep" as a quadratic function. This property allows gradient-based optimization techniques to be applied to the program—indeed, many widely used algorithms for gradient-based nonlinear optimization [11, 8] are known to perform best under such a guarantee.

As for β-robustness, this property demands that the output of the smooth interpreter has a small partial derivative in β. The property is important to the success of the iterative algorithm described above, which relies on repeated numerical search with progressively smaller values of β. Without β-robustness, it would be possible for the approximation to change dramatically with small changes to β, making the algorithm impractical.

2. We give a concrete design for an approximate smooth interpreter (called $\mathit{Smooth}_P(\mathbf{x}, \beta)$) that satisfies the above properties. The framework combines symbolic computation of convolution with abstract interpretation: when asked to smooth a program P, $\mathit{Smooth}_P(\mathbf{x}, \beta)$ uses an abstract interpreter to approximate P by a pair of simpler programs P_{inf} and P_{sup}, then performs a symbolic computation on these approximations. The result of this computation is taken as the semantics of the smoothed version of P.

   The soundness of our method relies on the insight that Gaussian convolution is a monotone map on the pointwise partial order of vector functions. We establish robustness and β-robustness under a weak assumption about β, by bounding the derivative of a convolution. Thus, the techniques used to prove our analysis correct are very different from those in traditional program analysis. Also, so far as we know, the abstract domain used in our construction is new to the program analysis literature.

The paper is structured as follows. In Sec. 2, we recapitulate the elements of smooth interpretation and set up the programming language machinery needed for our subsequent development. In Sec. 3, we introduce our correctness requirements for smoothing; in Sec. 4, we present our framework for smooth interpretation. Sec. 5 studies the properties of interpreters derived from this framework. Our discussion of related work, as well as our conclusions, appear in Sec. 6.

2 Smooth interpretation

We begin by fixing, for the rest of the paper, a simple language of programs. Our programs are written in a flow-graph syntax [9], and maintain their state in k real-valued variables named $\mathtt{x_1}$ through $\mathtt{x_k}$.

Formally, let Re denote the set of linear arithmetic expressions over $\mathtt{x_1}, \ldots, \mathtt{x_k}$, encoding linear transformations of the type $\mathbb{R}^k \to \mathbb{R}^k$. Also, let Be the set of boolean expressions of the form $Q > 0$ or $Q \geq 0$, where $Q \in \mathbb{R}^k \to \mathbb{R}$. A program P in our language is a directed graph. Nodes of this graph can be of five types:

– An *entry node* has one outgoing edge and no incoming edge, and an *exit node* has one incoming edge and no outgoing edge. A program has a single entry node and a single exit node.
– An *assignment node* has a single incoming edge and single outgoing edge. Each assignment node u is labeled with an expression $E \in Re$. Intuitively, this expression is the r-value of the assignment.
– A *test node* u has one incoming edge and two outgoing edges (known as the true and false-edges), and is labeled by a boolean expression $Test(u) \in Be$. Intuitively, u is a conditional branch.
– A *junction node* has a single outgoing edge and two incoming edges.

For example, Fig. 2 depicts a simple program over a single variable.

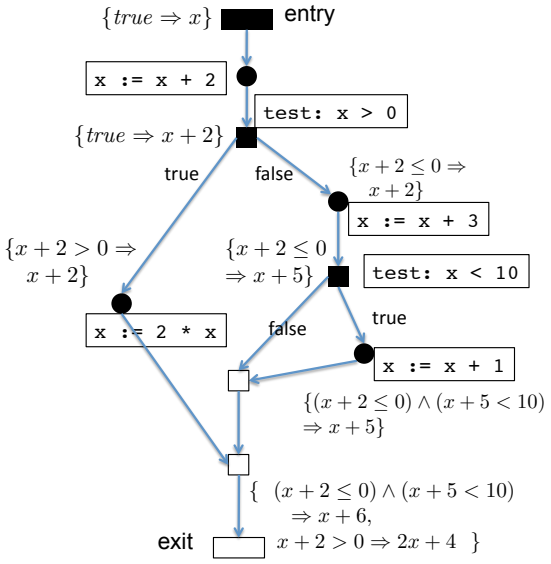

Fig. 2. A program and its collecting semantics

Semantics. The intuitive operational semantics of P is that it starts executing at its entry node, taking transitions along the edges, and terminates at the exit node. For our subsequent development, however, a denotational semantics as well as an abstract-interpretation-style collecting semantics are more appropriate than an operational one. Now we define these semantics.

Let a *state* of P be a vector $\mathbf{x} = \langle x_1, \ldots, x_k \rangle \in \mathbb{R}^k$, where each x_i captures the value of the variable $\mathtt{x_i}$. For each arithmetic expression $E \in Re$, the denotational semantics $[\![E]\!](\mathbf{x}) : \mathbb{R}^k \to \mathbb{R}^k$ produces the value of E at the state $\mathbf{x}$. For each $B \in Be$, the denotation function $[\![B]\!]$ produces $[\![B]\!](\mathbf{x}) = 0$ if B is false at the state $\mathbf{x}$, and 1 otherwise.

To define the semantics of P, we need some more machinery. Let a *guarded linear expression* be an expression of the form

$$\texttt{if } B \texttt{ then } F \texttt{ else } \mathbf{0},$$

where B is a conjunction of linear inequalities over the variables $x_1, \ldots, x_k$, and F is a linear arithmetic expression. We abbreviate the above expression by the notation $(B \Rightarrow F)$, and lift the semantic function $[\![\circ]\!]$ to such expressions:

$$[\![B \Rightarrow F]\!](\mathbf{x}) = \texttt{if } [\![B]\!](\mathbf{x}) \texttt{ then } [\![F]\!](\mathbf{x}) \texttt{ else } \mathbf{0} = [\![B]\!](\mathbf{x}) \cdot [\![F]\!](\mathbf{x}).$$

The *collecting semantics* of P is given by a map Ψ_P that associates with each node u of P a set $\Psi_P(u)$ of guarded linear expressions. Intuitively, each such

expression captures the computation carried out along a path in P ending at u. As the program P can have "loops," $\Psi_P(u)$ is potentially infinite.

In more detail, we define the sets $\Psi_P(u)$ so that they form the least fixpoint of a monotone map. For each node u of P, $\Psi_P(u)$ is the least set of guarded linear expressions satisfying the following conditions:

- If u is the entry node and its out-edge goes to v, then $\{(\mathit{true} \Rightarrow \mathbf{x})\} \subseteq \Psi_P(v)$.
- Suppose u is an assignment node labeled by E and its outgoing edge leads to v. Let $\circ$ be the usual composition operator over expressions. Then for all $(B \Rightarrow F) \in \Psi_P(u)$, the expression $B \Rightarrow (E \circ F)$ is in $\Psi_P(v)$.
- Suppose u is a branch node, and let v_t and v_f respectively be the targets of the true- and false-edges out of it. Let $Q_t = \mathit{Test}(u) \circ F$ and $Q_f = (\neg \mathit{Test}(u)) \circ F$; then, for all $(B \Rightarrow F) \in \Psi_P(u)$, we have

$$(B \wedge Q_t) \Rightarrow F \in \Psi_P(v_t) \qquad (B \wedge Q_f) \Rightarrow F \in \Psi_P(v_f).$$

- If u is a junction node and its outgoing edge leads to v, then $\Psi_P(u) \subseteq \Psi_P(v)$.

For example, consider the program in Fig. 2. Most nodes u of this program are labeled with $\Psi_P(u)$. We note that one of the three control flow paths to the exit node ex is infeasible; as a result $\Psi_P(ex)$ has two formulas rather than three.

The denotational semantics $[\![P]\!]$ of P is now defined using the above collecting semantics. Let ex be the exit node of P. We define:

$$[\![P]\!](\mathbf{x}) = \sum_{(B \Rightarrow F) \in \Psi_P(ex)} [\![B]\!](\mathbf{x}) \cdot [\![F]\!](\mathbf{x}).$$

Intuitively, for any $\mathbf{x} \in \mathbb{R}^k$, $[\![P]\!](\mathbf{x})$ is the output of P on input $\mathbf{x}$.

Smoothed semantics. Now we recall the definition [5] of the *smoothed semantics* of programs, which is the semantics that smooth interpretation seeks to compute. To avoid confusion, the previously defined semantics is from now on known as the *crisp semantics*.

Let $\beta > 0$ be a real-valued *smoothing parameter*, and let $\mathcal{N}(\mathbf{x}, \beta)$ be the joint density function of k independent normal variables, each with mean 0 and standard deviation β. In more detail, letting $\mathbf{x} = \langle x_1, \ldots, x_k \rangle$ as before, $\mathcal{N}(\mathbf{x}, \beta)$ is given by $\mathcal{N}(\mathbf{x}, \beta) = \frac{1}{(2\pi\beta^2)^{k/2}} e^{-\frac{\sum_{i=1}^{k} x_i^2}{2\beta^2}} = \prod_{i=0..k} \mathcal{N}(x_i)$.

The *smoothed semantics* $\overline{[\![P]\!]}_\beta : \mathbb{R}^k \to \mathbb{R}^k$ of a program P with respect to β is obtained by taking the *convolution* of $\mathcal{N}$ and the crisp semantics of P as shown be the following equation.

$$\begin{aligned} \overline{[\![P]\!]}_\beta(\mathbf{x}) &= \textstyle\int_{\mathbf{r} \in \mathbb{R}^k} [\![P]\!](\mathbf{r})\ \mathcal{N}(\mathbf{x} - \mathbf{r}, \beta)\ d\mathbf{r} \\ &= \textstyle\sum_{(B \Rightarrow F) \in \Psi_P(ex)} \int_{\mathbf{r} \in \mathbb{R}^k} [\![B \Rightarrow F]\!](\mathbf{r})\ \mathcal{N}(\mathbf{x} - \mathbf{r}, \beta)\ d\mathbf{r}. \end{aligned} \tag{1}$$

As before, ex refers to the exit node of P. Note that because convolution is a commutative operator, we have the property

$$\overline{[\![P]\!]}_\beta(\mathbf{x}) = \int_{\mathbf{r} \in \mathbb{R}^k} [\![P]\!](\mathbf{x} - \mathbf{r})\ \mathcal{N}(\mathbf{r}, \beta)\ d\mathbf{r}.$$

When β is clear from context, we denote $\overline{[\![P]\!]}_\beta$ by $\overline{[\![P]\!]}$, and $\mathcal{N}(\mathbf{x}, \beta)$ by $\mathcal{N}(\mathbf{x})$.

One of the properties of smoothing is that even if $[\![P]\!]$ is highly discontinuous, $\overline{[\![P]\!]}$ is a smooth mathematical function that has all its derivatives defined at every point in $\mathbb{R}^k$. The smoothing parameter β can be used to control the extent to which $[\![P]\!]$ is smoothed by the above operation—the higher the value of β, the greater the extent of smoothing.

Example 1. Consider a program P over one variable x such that $[\![P]\!](x) = \texttt{if } x > a \texttt{ then } 1 \texttt{ else } 0$, where $a \in \mathbb{R}$. Let erf be the Gauss error function. We have

$$\begin{aligned}\overline{[\![P]\!]}(x) &= \textstyle\int_{-\infty}^{\infty} [\![P]\!](y)\mathcal{N}(x-y)\ dy = 0 + \int_{a}^{\infty} \mathcal{N}(x-y)\ dy \\ &= \textstyle\int_{0}^{\infty} \frac{1}{\sqrt{2\pi}\beta}\ e^{-(y-x+a)^2/2\beta^2} dy = \frac{1+\mathrm{erf}(\frac{x-a}{\sqrt{2}\beta})}{2}.\end{aligned}$$

Figure 3-(a) plots the crisp semantics $[\![P]\!]$ of P with $a = 2$, as well as $\overline{[\![P]\!]}$ for $\beta = 0.5$ and $\beta = 3$. While $[\![P]\!]$ has a discontinuous "step," $\overline{[\![P]\!]}$ is a smooth S-shaped curve, or a *sigmoid*. Note that as we decrease the "tuning knob" β, the sigmoid $\overline{[\![P]\!]}$ becomes steeper and steeper, and at the limit, approaches $[\![P]\!]$.

Now consider the program P' such that $[\![P']\!](x) = \texttt{if } a < x < c \texttt{ then } 1 \texttt{ else } 0$, where $a, c \in \mathbb{R}$ and $a < c$. The "bump-shaped" functions obtained by smoothing P' are plotted in Figure 3-(b) (here $a = -5$, $c = 5$, and β has two values 0.5 and 2). Note how the discontinuities are smoothed.

Now we consider an even more interesting program, one that is key to the main results of this paper. Consider $[\![P'']\!](x) = [\![B \Rightarrow F]\!](x)$, where B is the boolean expression $a < x < b$ for constants a, b, and $F(x) = \alpha \cdot x + \gamma$ for constants α, γ. In this case, the smoothed semantics of P'' can be evaluated symbolically as follows:

$$\begin{aligned}\overline{[\![P'']\!]}(x) &= \int_a^b (\alpha y + \gamma)\ \mathcal{N}(x-y)\ dy = \alpha \int_a^b y\ \mathcal{N}(x-y)\ dy + \gamma \int_a^b \mathcal{N}(x-y)\ dy \\ &= \frac{(\alpha\ x + \gamma) \cdot (\mathrm{erf}(\frac{b-x}{\sqrt{2}\beta}) - \mathrm{erf}(\frac{a-x}{\sqrt{2}\beta}))}{2} + \frac{\beta\ \alpha\ (e^{-(a-x)^2/2\beta^2} - e^{-(b-x)^2/2\beta^2})}{\sqrt{2\pi}}.\end{aligned}$$

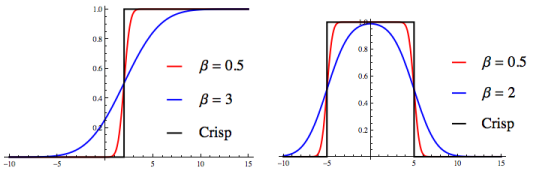

Fig. 3. (a) A sigmoid. (b) A bump.

Smooth interpretation. We use the phrase "*smooth interpreter* of a program P" to refer to an algorithm that can execute P according to the smoothed semantics $\overline{[\![\circ]\!]}_\beta$. The primary application of smooth interpretation is to enable the use of numerical search for parameter optimization problems. For example, suppose our goal is to compute an input $\mathbf{x}$ for which $[\![P]\!](\mathbf{x})$ is minimal. For a vast number of real-world programs, discontinuities in $[\![P]\!]$ would preclude the use of local numerical methods in this minimization problem. By eliminating such discontinuities, smooth interpretation can make numerical search practical.

In a way, smooth interpretation is to numerical optimization what abstraction is to model checking: a mechanism to approximate challenging local transitions in a program-derived search space. But the smoothed semantics that we use is very different from the collecting semantics used in abstraction. Rather, the smoothed semantics of P can be seen to be the expectation of a probabilistic semantics of P.

Consider an input $\mathbf{x} \in \mathbb{R}^k$ of P, and suppose that, before executing P, we randomly perturb $\mathbf{x}$ following a k-D normal distribution with independent components and standard deviation β. Thus, the input of P is now a random variable X following a normal distribution $\mathcal{N}$ with mean $\mathbf{x}$ and similar shape as the one used in the perturbation. Now let us execute P on X with crisp semantics. Consider the *expectation* of the output $[\![P]\!](\mathsf{X})$: $\mathbf{Exp}\,[[\![P]\!](\mathsf{X})] = \int_{-\infty}^{\infty} [\![P]\!](\mathbf{r})\, \mathcal{N}(\mathbf{x}-\mathbf{r})\, d\mathbf{r} = \overline{[\![P]\!]}(\mathbf{x})$. Here $\overline{[\![P]\!]}$ is computed using the Gaussian $\mathcal{N}$. In other words, the smoothed semantics of P is the expected crisp semantics of P under normally distributed perturbations to the program inputs.

Now that we have defined how a smooth interpreter must *behave*, consider the question of how to algorithmically implement such an interpreter. On any input $\mathbf{x}$, an idealized smooth interpreter for P must compute $\overline{[\![P]\!]}_\beta(\mathbf{x})$—in other words, integrate the semantics of P over a real space. This problem is of course undecidable in general; therefore, any algorithmic implementation of a smooth interpreter must necessarily be *approximate*. But when do we judge such an approximation to be "correct"? Now we proceed to answer this question.

3 Approximate smooth interpreters and their correctness

In this section, we define three correctness properties for algorithmic implementations of smooth interpretation: *soundness*, *robustness*, and *β-robustness*. While an algorithm for smooth interpretation of a program must necessarily be approximate, these desiderata impose limits on the approximations that it makes.

Formally, we let an *approximate smooth interpreter* $Smooth_P(\mathbf{x}, \beta)$ for P be an algorithm with two inputs: an input $\mathbf{x} \in \mathbb{R}^k$ and a smoothing parameter $\beta \in \mathbb{R}^+$. Given these, $Smooth_P$ returns a symbolic representation of a set $\mathbf{Y} \subseteq \mathbb{R}^k$. To avoid notation-heavy analytic machinery, we restrict the sets returned by $Smooth_P$ to be *intervals* in $\mathbb{R}^k$. Recall that such an interval is a Cartesian product $\langle [l_1, u_1], \ldots, [l_k, u_k] \rangle$ of intervals over $\mathbb{R}$; the interval can also be represented more conveniently as a pair of vectors $[\langle l_1, \ldots, l_k \rangle, \langle u_1, \ldots, u_k \rangle]$; from now on, we denote the set of all such intervals by $\mathbf{I}$.

Soundness. Just as a traditional static analysis of P is sound if it computes an abstraction of the crisp semantics of P, an approximate smooth interpreter is sound if it abstracts the smoothed semantics of P. In other words, the interval returned by $Smooth_P$ on any input $\mathbf{x}$ bounds the output of the idealized smoothed version of P on $\mathbf{x}$. We define:

Definition 1 (Soundness). *An approximate smooth interpreter* $Smooth_P : \mathbb{R}^k \times \mathbb{R}^+ \to \mathbf{I}$ *is* sound *iff for all* $\mathbf{x} \in \mathbb{R}^k$, $\overline{[\![P]\!]}_\beta(\mathbf{x}) \in Smooth_P(\mathbf{x}, \beta)$.

Robustness. A second requirement, critical for smooth interpreters but less relevant in abstract interpretation, is *robustness*. This property asserts that for all $\mathbf{x}$ and β, $Smooth_P(\mathbf{x}, \beta)$ has derivatives of all orders, and that further, its partial derivative with respect to the component scalars of $\mathbf{x}$ is small—i.e., bounded by a function linear in $\mathbf{x}$ and β.

The first of the two requirements above simply asserts that $Smooth_P(\mathbf{x}, \beta)$ computes a smooth function, in spite of all the approximations carried out for computability. The rationale behind the second requirement is apparent when we consider smooth interpretation in the broader context of local numerical optimization of programs. A large number of numerical search routines work by sampling the input space under the expectation that the derivative around each sample point is well defined. Our requirement guarantees this.

In fact, by putting a linear bound on the derivative of $Smooth_P$, we give a stronger guarantee: the absence of regions of extremely steep descent that can lead to numerical instability. Indeed, robustness implies that even at the points in the input space where P is discontinuous, the gradient of $Smooth_P$ is Lipschitz-continuous—i.e., $Smooth_P$ is only as "steep" as a quadratic function. Many algorithms for nonlinear optimization [11, 8] demand a guarantee of this sort for best performance. As we will see later, we can implement a smooth interpreter that is robust (under a weak additional assumption) even in our strong sense. Let us now define:

Definition 2 (Robustness). *$Smooth_P(\mathbf{x}, \beta)$ is* robust *if $\frac{\partial}{\partial x_i} Smooth_P(\mathbf{x}, \beta)$ exists at all $\mathbf{x} = \langle x_1, \ldots, x_k \rangle$ and for all i, and there exists a* linear *function $K(\mathbf{x}, \beta)$ in $\mathbf{x}$ that satisfies $\| \frac{\partial}{\partial x_i} Smooth_P(\mathbf{x}, \beta) \| \leq K(\mathbf{x}, \beta)$ for all $\mathbf{x}, i, \beta$.*

The definition above abuses notation somewhat because, as you may recall, $Smooth_P(\mathbf{x}, \beta)$ actually produces a Cartesian product of intervals, as opposed to a real number; so the derivative $\frac{\partial}{\partial x_i} Smooth_P(\mathbf{x}, \beta)$ is actually a pair of vectors $[\langle \frac{\partial}{\partial x_i} l_1, \ldots, \frac{\partial}{\partial x_i} l_k \rangle, \langle \frac{\partial}{\partial x_i} u_1, \ldots, \frac{\partial}{\partial x_i} u_k \rangle]$. The measure for such a pair of vectors is a simple Euclidian measure that adds the squares of each of the components.

β-robustness. Another correctness property for an approximate smooth interpreter is that it produces functions that have small partial derivatives with respect to the smoothing parameter β. In more detail, the derivative $\frac{\partial Smooth_P(\mathbf{x}, \beta)}{\partial \beta}$ must be bounded by a function linear in $\mathbf{x}$ and β. We consider this property important because of the way our numerical search algorithm from [5] uses smoothing: starting with a large β and progressively reducing it, improving the quality of the approximation in a way akin to the abstraction-refinement loop in program verification. The property of β-robustness guarantees that the functions optimized in two successive iterations of this process are not wildly different. In other words, the optima of one iteration of the process do not become entirely suboptimal in the next iteration.

Formally, we define the property of β-robustness as follows:

Definition 3 (Robustness in β). *$Smooth_P(\mathbf{x}, \beta)$ is* robust in β *if the partial derivative $\frac{\partial}{\partial \beta} Smooth_P(\mathbf{x}, \beta)$ exists for all $\beta > 0$, and there is a linear function $K(\mathbf{x}, \beta)$ such that for all $\mathbf{x}$ and β, $\| \frac{\partial}{\partial \beta} Smooth_P(\mathbf{x}, \beta) \| \leq K(\mathbf{x}, \beta)$.*

1. Given a program P for which an approximate smooth interpreter is to be constructed, use abstract interpretation to obtain programs P_{sup} and P_{inf} such that:
   - P_{sup} and P_{inf} are interval-guarded linear programs.
   - $[\![P_{inf}]\!] \preceq [\![P]\!] \preceq [\![P_{sup}]\!]$ (here $\preceq$ is the pointwise ordering over functions).
2. Construct symbolic representations of $\overline{[\![P_{sup}]\!]}_\beta$ and $\overline{[\![P_{inf}]\!]}_\beta$.
3. Let the approximate smooth interpreter for P be a function that on any $\mathbf{x}$ and β, returns the Cartesian interval $[\overline{[\![P_{inf}]\!]}_\beta(\mathbf{x}), \overline{[\![P_{sup}]\!]}_\beta(\mathbf{x})]$.

Fig. 4. The approximate smooth interpretation algorithm.

4 Designing a smooth interpreter

Now we present a framework for approximate smooth interpretation that satisfies the correctness properties defined in the previous section. We exploit the fact that under certain restrictions on a program P, it is possible to build an *exact* smooth interpreter for P—i.e., an algorithm that computes the smoothed semantics $\overline{[\![P]\!]}$ exactly. The idea behind our construction is to approximate P by programs for which exact smooth interpreters can be constructed.

Let us consider guarded linear expressions (defined in Sec. 2); recall that for nodes u of P, $\Psi_P(u)$ is a possibly-infinite set of guarded linear expressions $(B \Rightarrow F)$. By Eqn. (1), computing the exact smoothed semantics of P amounts to computing a sum of Gaussian convolutions of guarded linear expressions.

Unfortunately, the convolution integral of a general guarded linear expression does not have a clean symbolic solution. We overcome this problem by abstracting each such expression using an *interval-guarded linear expression* $B_{int} \Rightarrow F$, where B_{int} is an interval in $\mathbb{R}^k$ obtained through a Cartesian abstraction of B. From an argument as in Example 1, if an expression is interval-guarded and linear, then its exact smoothed semantics can be computed in closed-form.

The above strategy alone is not enough to achieve convergence, given that $\Psi_P(u)$ can be infinite. Hence we use *pairs* of interval-guarded linear expressions of the form $\langle (B_{int} \Rightarrow F_{sup}), (B_{int} \Rightarrow F_{inf}) \rangle$ to abstract unbounded sets of linear expressions guarded by subintervals of B_{int}. Such a tuple is known as an *interval-guarded bounding expression*, and abbreviated by the notation $\langle B_{int}, F_{sup}, F_{inf} \rangle$.

To see what such an abstraction means, let us define the *pointwise ordering relation* $\preceq$ among functions of type $\mathbb{R}^k \to \mathbb{R}^k$ as follows: $F_1 \preceq F_2$ iff for all $\mathbf{x} \in \mathbb{R}^k$, we have $F_1(\mathbf{x}) \leq F_2(\mathbf{x})$. We lift this function to arithmetic expressions E, letting $E_1 \preceq E_2$ iff $[\![E_1]\!] \preceq [\![E_2]\!]$. We guarantee that if $\langle B_{int}, F_{sup}, F_{inf} \rangle$ abstracts a set S of interval-guarded linear expressions, then for all $(B \Rightarrow F) \in S$, we have

$$(B_{int} \Rightarrow F_{inf}) \preceq (B \Rightarrow F) \preceq (B_{int} \Rightarrow F_{sup}).$$

In fact, rather than tracking just a single interval-guarded bounding expression, an abstract state in our framework tracks bounded sets of such expressions. Using this abstraction, it is possible to approximate the semantics $[\![P]\!]$ of P by two programs P_{sup} and P_{inf}, whose semantics can be represented as a sum of a

bounded number of terms, each term being an interval-guarded linear expression. (We call such programs *interval-guarded linear programs*.)

The smoothed semantics of P_{sup} and P_{inf} can be computed in closed form, leading to an approximate smooth interpreter that is sketched in Fig. 4. In the rest of this section, we complete the above algorithm by describing the abstract interpreter in Step (1) and the analytic calculation in Step (2).

Step 1: Abstraction using interval-guarded bounding expressions

Abstract domain. An *abstract state* σ in our semantics is either a bounded-sized set of interval-guarded bounding expressions (we let N be a bound on the number of elements in such a set), or the special symbol $\top$. The set of all abstract states is denoted by $\mathcal{A}$.

As usual, we define a partial order over the domain $\mathcal{A}$. Before defining this order, let us define a partial order $\trianglelefteq$ over the universe IGB that consists of all interval-guarded bounding expressions, as well as $\top$. For all σ, we have $\sigma \trianglelefteq \top$. We also have:

$$\begin{aligned}\langle B, F_{sup}, F_{inf}\rangle \trianglelefteq \langle B', F'_{sup}, F'_{inf}\rangle \quad \text{iff} \quad & B \Rightarrow B' \text{ and} \\ & (B' \wedge B) \Rightarrow (F_{sup} \preceq F'_{sup}) \wedge (F'_{inf} \preceq F_{inf}) \text{ and} \\ & (B' \wedge \neg B) \Rightarrow (\mathbf{0} \preceq F'_{sup}) \wedge (F'_{inf} \preceq \mathbf{0}).\end{aligned}$$

Intuitively, in the above, B is a subinterval of B', and the expressions $(B' \Rightarrow F'_{inf})$ and $(B' \Rightarrow F'_{sup})$ define more relaxed bounds than $(B \Rightarrow F_{inf})$ and $(B \Rightarrow F_{sup})$.

Note that $\trianglelefteq$ is *not* a lattice relation—e.g., the interval-guarded expressions $(1 < x < 2) \Rightarrow 1$ and $(3 < x < 4) \Rightarrow 5$ do not have a unique least upper bound. However, it is easy to give an algorithm $\sqcup_{IGB}$ that, given $H_1, H_2 \in IGB$, returns a *minimal*, albeit nondeterministic, upper bound H of H_1 and H_2 (i.e., $H_1 \trianglelefteq H$, $H_2 \trianglelefteq H$, and there is no $H' \neq H$ such that $H' \trianglelefteq H$, $H_1 \trianglelefteq H'$, and $H_2 \trianglelefteq H'$).

Now can we define the partial order $\sqsubseteq$ over $\mathcal{A}$ that we use for abstract interpretation. For $\sigma_1, \sigma_2 \in \mathcal{A}$, we have $\sigma_1 \sqsubseteq \sigma_2$ iff either $\sigma_2 = \top$, or if for all $H \in \sigma_1$, there exists $H' \in \sigma_2$ such that $H \trianglelefteq H'$.

Once again, we can construct an algorithm $\sqcup_{\mathcal{A}}$ that, given $\sigma_1, \sigma_2 \in \mathcal{A}$, returns a minimal upper bound σ for σ_1 and σ_2. If σ_1 or σ_2 equals $\top$, the algorithm simply returns $\top$. Otherwise, it executes the following program:

1. Let $\sigma' := \sigma_1 \cup \sigma_2$.
2. While $|\sigma'| > N$, repeatedly: (a) Nondeterministically select two elements $H_1, H_2 \in \sigma'$; (b) assign $\sigma' := \sigma' \setminus \{H_1, H_2\} \cup (H_1 \sqcup_{IGB} H_2)$;
3. Return σ'.

Abstraction. The *abstract semantics* of the program P is given by a map $\Psi^{\#}_P$ that associates an abstract state $\Psi^{\#}_P(u)$ with each node u of P. To define this semantics, we need some more machinery. First, we need a way to capture the effect of an assignment node labeled by an expression E, on an abstract state σ. To this end, we define a notation $(E \circ \sigma)$ that denotes the composition of

E and σ. We have $E \circ \top = \top$ for all E. If $\sigma \neq \top$, then we have $(E \circ \sigma) = \{\langle B, (E \circ F_{sup}), (E \circ F_{inf})\rangle : \langle B, F_{sup}, F_{inf}\rangle \in \sigma\}$. Applied to the abstract state σ, the assignment produces the abstract state $(E \circ \sigma)$.

Second, we must be able to propagate an abstract state σ through a test node labeled by the boolean expression C. To achieve this, we define, for each abstract state σ and boolean expression C, the composition $(C \circ \sigma)$ of C and σ. In fact, we begin by defining the composition of $(C \circ H)$, where $H = \langle B, F_{sup}, F_{inf}\rangle$ is an interval-guarded boolean expression.

The idea here is that if a test node is labeled by C and H reaches the node, then $(C \circ H)$ is propagated along the true-branch. For simplicity, let us start with the case $B = \mathit{true}$. Clearly, $(C \circ H)$ should be of the form $\langle B', F_{sup}, F_{inf}\rangle$ for some B'. To see what B' should be, consider the scenario in Fig. 5, which shows the points $F_{sup}(\mathbf{x})$ and $F_{inf}(\mathbf{x})$ for a fixed $\mathbf{x} \in \mathbb{R}^2$. For all F such that $F_{inf} \preceq F \preceq F_{sup}$, the point $F(\mathbf{x})$ must lie within the dashed box. So long as an "extreme point" of this box satisfies the constraint C (the region to the left of the inclined line), $\mathbf{x}$ should satisfy B'.

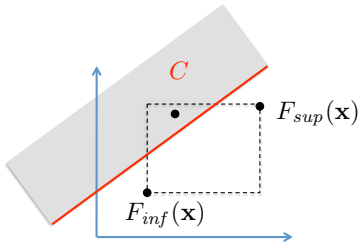

Fig. 5. Propagation through test C.

We need some more notation. For each function $F : \mathbb{R}^k \to \mathbb{R}^k$, let us define a collection of "components" $F^1, \ldots, F^k$ such that for all i, $F^i(\mathbf{x}) = (F(\mathbf{x}))(i)$. A collection of component functions $F_1, \ldots, F_k : \mathbb{R}^k \to \mathbb{R}$ can be "combined" into a function $F = \langle F_1, \ldots, F_k\rangle : \mathbb{R}^k \to \mathbb{R}^k$, where for all $\mathbf{x}$, $F(\mathbf{x}) = \langle F_1(\mathbf{x}), \ldots, F_k(\mathbf{x})\rangle$. The extreme points of our dashed box can now be seen to be obtained by taking all possible combinations of components from F_{sup} and F_{inf} and combining those functions. Then the property that some extreme point of the dashed box of Fig. 5 satisfies C is captured by the formula

$$(C \circ \langle F^1_{sup}, F^2_{sup}\rangle) \vee (C \circ \langle F^1_{sup}, F^2_{inf}\rangle) \vee (C \circ \langle F^1_{inf}, F^2_{sup}\rangle) \vee (C \circ \langle F^1_{inf}, F^2_{inf}\rangle).$$

The above can now be generalized to k-dimensions and the case $B \neq \mathit{true}$. The composition of C with an interval-guarded boolean expression $\langle B, F_{sup}, F_{inf}\rangle$ is defined to be $C \circ \langle B, F_{sup}, F_{inf}\rangle = \langle B', F_{sup}, F_{inf}\rangle$, where

$$B' = B \wedge \left(\bigvee_{G^i \in \{F^i_{sup}, F^i_{inf}\}} (C \circ \langle G^1, \ldots, G^k\rangle) \right).$$

Let us now lift this composition operator to abstract states. We define $C \circ \top = \top$ for all C. For all $\sigma \neq \top$, we have $(C \circ \sigma) = \{C \circ \langle B, F_{sup}, F_{inf}\rangle : \langle B, F_{sup}, F_{inf}\rangle \in \sigma\}$. Finally, for any boolean expression C, let us define $C^{\#}$ to be an interval that overapproximates C.

The abstract semantics $\Psi^{\#}_P$ of P is now defined using the algorithm in Figure 6. We note that $\Psi^{\#}_P$ can have different values depending on the sequence of nondeterministic choices made by our upper-bound operators. However, *every* resolution of such choices leads to an abstract semantics that can support a

1. If u is the entry node of P and its outgoing edge leads to v, then assign $\Psi_P^\#(v) := \{\langle true, \mathbf{x}, \mathbf{x}\rangle\}$. Assign $\Psi_P^\#(v') = \emptyset$ for every other node v'.
2. Until fixpoint, repeat:
   (a) If u is an assignment node labeled by E and its outgoing edge leads to v, then assign $\Psi_P^\#(v) := \Psi_P^\#(v) \sqcup (E \circ \Psi_P^\#(u))$.
   (b) Suppose u is a branch node; then let v_t and v_f respectively be the targets of the true- and false-edges out of it, and let $Q_t = Test(u)$ and $Q_f = \neg Test(u)$

$$\Psi_P^\#(v_t) := \Psi_P^\#(v_t) \sqcup \{Q_t \circ \langle B, F_{sup}, F_{inf}\rangle : \langle B, F_{sup}, F_{inf}\rangle \in \Psi_P^\#(u)\}$$
$$\Psi_P^\#(v_f) := \Psi_P^\#(v_f) \sqcup \{Q_f \circ \langle B, F_{sup}, F_{inf}\rangle : \langle B, F_{sup}, F_{inf}\rangle \in \Psi_P^\#(u)\}$$

   (c) If u is a junction node with an out-edge to v, then $\Psi_P^\#(v) := \Psi_P^\#(u) \sqcup \Psi_P^\#(v)$.

Fig. 6. Algorithm to compute $\Psi_P^\#$

.

sound and robust approximate smooth interpreter. Consequently, from now on, we will ignore the fact that $\Psi_P^\#$ actually represents a family of maps, and instead view it as a function of P.

Widening. As our abstract domain is infinite, our fixpoint computation does not guarantee termination. For termination of abstract interpretation, our domain needs a widening operator [9]. For example, one such operator $\bigtriangledown$ can be defined as follows.

First we define $\bigtriangledown$ on interval-guarded bounding expressions. Let us suppose $\langle B_w, F''_{sup}, F''_{inf}\rangle = \langle B, F_{sup}, F_{inf}\rangle \bigtriangledown \langle B', F'_{sup}, F'_{inf}\rangle$. Then we have:

- $B_w = B \bigtriangledown_{int} B'$, where $\bigtriangledown_{int}$ is the standard widening operator for the interval domain [9].
- F''_{sup} is a minimal function in the pointwise order $\preceq$ such that for all $\mathbf{x} \in B_w$, we have $F''_{sup}(\mathbf{x}) \geq (B \Rightarrow F_{sup})(\mathbf{x})$ and $F''_{sup}(\mathbf{x}) \geq (B' \Rightarrow F'_{sup})(\mathbf{x})$.
- F''_{inf} is a maximal function such that for all $\mathbf{x} \in B_w$, we have $F''_{sup}(\mathbf{x}) \leq (B \Rightarrow F_{inf})(\mathbf{x})$ and $F''_{inf}(\mathbf{x}) \leq (B' \Rightarrow F'_{inf})(\mathbf{x})$.

This operator is now lifted to abstract states in the natural way.

Computing P_{sup} and P_{inf}. Now we can compute the interval-guarded linear programs P_{sup} and P_{inf} that bound P. Let ex be the exit node of P, and let $\Psi_P^\#(ex) = \{\langle B^1, F^1_{sup}, F^1_{inf}\rangle, \ldots, \langle B^n, F^n_{sup}, F^n_{inf}\rangle\}$ for some $n \leq N$. Then, the symbolic representation of the semantics $[\![P_{sup}]\!]$ and $[\![P_{inf}]\!]$ of P_{sup} and P_{inf} is as follows:

$$[\![P_{sup}]\!] = \textstyle\sum_i^N [\![B^i \Rightarrow F^i_{sup}]\!] \qquad\qquad [\![P_{inf}]\!] = \textstyle\sum_i^N [\![B^i \Rightarrow F^i_{inf}]\!].$$

Step 2: Symbolic convolution of interval-guarded linear programs

Now we give a closed-form expression for the smoothed semantics of P_{sup} (the case of $[\![P_{inf}]\!]$ is symmetric). We have:

$$\overline{[\![P_{sup}]\!]}(\mathbf{x}) = \int_{\mathbf{r}\in\mathbb{R}^k} [\![P_{sup}]\!](\mathbf{r})\, \mathcal{N}(\mathbf{x}-\mathbf{r})\, d\mathbf{r} = \sum_i \int_{\mathbf{r}\in\mathbb{R}^k} [\![B^i \Rightarrow F^i_{sup}]\!](\mathbf{r})\, \mathcal{N}(\mathbf{x}-\mathbf{r})\, d\mathbf{r}.$$

To solve this integral, we first observe that by our assumptions, $\mathcal{N}$ is the joint distribution of k univariate *independent* Gaussians with the same standard deviation β. Therefore, we can split $\mathcal{N}$ into a product of univariate Gaussians $\mathcal{N}_1, \ldots, \mathcal{N}_k$, where the Gaussian $\mathcal{N}_j$-th ranges over x_j. Letting r_j be the j-th component of $\mathbf{r}$, we have:

$$\int_{\mathbf{r}\in\mathbb{R}^k} [\![B^i \Rightarrow F^i_{sup}]\!](\mathbf{r})\ \mathcal{N}(\mathbf{x}-\mathbf{r})\ d\mathbf{r} = \int_{-\infty}^{\infty} \ldots (\int_{-\infty}^{\infty} [\![B^i \Rightarrow F^i_{sup}]\!](\mathbf{r})\ \mathcal{N}_1(x_1 - r_1)\ dr_1) \ldots\ \mathcal{N}_k(x_k - r_k) dr_k.$$

Thus, due to independence of the x_j's, it is possible to reduce the vector integral involved in convolution to a sequence of integrals over one variable. Each of these integrals will be of the form $\int_{-\infty}^{\infty} [\![B^i \Rightarrow F^i_{sup}]\!](\mathbf{r})\ \mathcal{N}_j(x_j - r_j)\ dr_j$. Projecting the interval B^i over the axis for x_j leaves us with an integral like the one we solved analytically in Example 1. This allows us to represent the result of smooth interpretation as a finite sum of closed form expressions.

5 Properties of the interpreter

Now we prove that the algorithm as defined above actually satisfies the properties claimed in Sec. 3. We first establish that the approximate smooth interpreter presented in this paper is sound. Next we show that under a weak assumption about β, it is robust and β-robust as well.

Theorem 1. *The approximate smooth interpreter of Figure 4 is sound.*

Proof: In order to prove soundness, we need to prove that $\overline{[\![P]\!]}_\beta(\mathbf{x}) \in \mathit{Smooth}_P(\mathbf{x}, \beta)$. This follows directly from the soundness of abstract interpretation thanks to a property of Gaussian convolution: that it is a monotone map on the pointwise partial order $\preceq$ of functions between $\mathbb{R}^k$.

First, we have seen how to use abstract interpretation to produce two programs P_{inf} and P_{sup} such that $[\![P_{inf}]\!] \preceq [\![P]\!] \preceq [\![P_{sup}]\!]$.

Now, we defined $\mathit{Smooth}_P(\mathbf{x}, \beta)$ as the interval $[\overline{[\![P_{inf}]\!]}(\mathbf{x}), \overline{[\![P_{sup}]\!]}(\mathbf{x})]$; therefore, all we have to do to prove soundness is to show that $\overline{[\![P_{inf}]\!]} \preceq \overline{[\![P]\!]}_\beta \preceq \overline{[\![P_{sup}]\!]}$. In other words, we need to show that Gaussian smoothing preserves the ordering among functions.

This follows directly from a property of convolution. Let $F \preceq G$ for functions $F, G : \mathbb{R}^k \to \mathbb{R}^k$, and $H : \mathbb{R}^k \to \mathbb{R}^k$ be any function satisfying $H(\mathbf{x}) > 0$ for all $\mathbf{x}$ (note that the Gaussian function satisfies this property). Also, let F_H and G_H be respectively the convolutions of F and H, and G and H. Then we have $F' \preceq H'$.

Theorem 2. *For every constant $\epsilon > 0$, the approximate smooth interpreter of Figure 4 is robust in the region $\beta > \epsilon$.*

Proof: To prove robustness of $Smooth_P$, it suffices to show that both $\overline{[\![P_{inf}]\!]}$ and $\overline{[\![P_{sup}]\!]}$ satisfy the robustness condition. We focus on proving $\overline{[\![P_{sup}]\!]}$ robust, since the proof for $\overline{[\![P_{inf}]\!]}$ is symmetric. Now, we know that $[\![P_{sup}]\!]$ has the form $[\![P_{sup}]\!](\mathbf{x}) = \sum_{i=0}^{N} B_i(\mathbf{x})\, F_i(\mathbf{x})$, where $B_i(\mathbf{x})$ is an interval in $\mathbb{R}^k$ and $F_i(\mathbf{x})$ is a linear function. Hence we have:

$$\overline{[\![P_{sup}]\!]}(\mathbf{x}) = \int_{\mathbf{r}\in\mathbb{R}^k} (\sum_{i=0}^{N} B_i(\mathbf{r}) \cdot F_i(\mathbf{r})) \cdot \mathcal{N}(\mathbf{r}-\mathbf{x}, \beta)\ d\mathbf{r}.$$

where $\mathcal{N}$ is the joint density function of k independent 1-D Gaussians. It is easy to see that this function is differentiable arbitrarily many times in x_j. We have:

$$\frac{\partial \overline{[\![P_{sup}]\!]}(\mathbf{x})}{\partial x_j} = \int_{\mathbf{r}\in\mathbb{R}^k} (\sum_{i=0}^{N} B_i(\mathbf{r}) \cdot F_i(\mathbf{r})) \cdot (\frac{\partial \mathcal{N}(\mathbf{r}-\mathbf{x}, \beta)}{\partial x_j})\, d\mathbf{r}$$

Now note that $\frac{\partial \mathcal{N}(\mathbf{r}-\mathbf{x},\beta)}{\partial x_j} = \frac{(r_j - x_j)}{\beta^2} \cdot \mathcal{N}(\mathbf{r}-\mathbf{x}, \beta)$. Hence,

$$\begin{aligned}\frac{\partial \overline{[\![P_{sup}]\!]}(\mathbf{x})}{\partial x_j} &= \int_{\mathbf{r}\in\mathbb{R}^k} (\frac{1}{\beta^2} \sum_{i=0}^{N} B_i(\mathbf{r}) \cdot F_i(\mathbf{r}) \cdot (r_j - x_j)) \cdot \mathcal{N}(\mathbf{r}-\mathbf{x}, \beta)\ d\mathbf{r} \\ &= (\frac{1}{\beta^2} \sum_{i=0}^{N} \int_{\mathbf{r}\in B_i} F_i(\mathbf{r}) \cdot (r_j - x_j) \cdot \mathcal{N}(\mathbf{r}-\mathbf{x}, \beta)\ d\mathbf{r} \qquad (2)\end{aligned}$$

Each F_i is a linear function of type $\mathbb{R}^k \to \mathbb{R}^k$, so for each $n < k$, we have $(F_i(\mathbf{r}))(n) = (\sum_{l=0}^{k-1} \alpha_{i,l,n} \cdot r_l) + \gamma_{i,n}$ for constants $\alpha_{i,j,n}$ and $\gamma_{i,n}$. Substituting in Eqn. 2, we find that the n-th coordinate of the vector-valued expression $\frac{\partial \overline{[\![P_{sup}]\!]}(\mathbf{x})}{\partial x_j}$ can be expanded into a linear sum of terms as below:

$$\frac{1}{\beta^2} \int_{r_j=a}^{b} \mathcal{N}(r_j - x_j, \beta)\ dr_j \qquad (3)$$

$$\frac{1}{\beta^2} \int_{r_j=a}^{b} (r_j - x_j) \cdot \mathcal{N}(r_j - x_j, \beta)\ dr_j \qquad (4)$$

$$\frac{1}{\beta^2} \int_{r_j=a}^{b} r_j \cdot \mathcal{N}(r_j - x_j, \beta)\ dr_j \qquad (5)$$

$$\frac{1}{\beta^2} \int_{r_j=a}^{b} r_j \cdot (r_j - x_j) \cdot \mathcal{N}(r_j - x_j, \beta)\ dr_j \qquad (6)$$

In order to show that $\left|\frac{\partial \overline{[\![P_{sup}]\!]}(\mathbf{x})}{\partial x_j}\right|$ is bounded by a linear function, we first observe that for all $\beta > \epsilon$, the multiplier $\frac{1}{\beta^2}$ is bounded by a constant. Now we show that the each integrals in the above is bounded by a function that is linear in x_j, even when $a = -\infty$ or $b = \infty$.

It is easy to see that of these integrals, the first two are bounded in value for any value of x_j; this follows from the fact that $\mathcal{N}$ decays exponentially as its first

argument goes to infinity. The last integral will also be bounded as a function of x_j, while the second to last function can grow linearly with x_j. It follows that $\left|\frac{\partial[\![\overline{P_{sup}}]\!](\mathbf{x})}{\partial x_j}\right|$ can grow only linearly with $\mathbf{x}$ and β. Hence $\overline{[\![P_{sup}]\!]}$ is robust.

Theorem 3. *For every constant $\epsilon > 0$, the approximate smooth interpreter from Figure 4 is β-robust in the region $\beta > \epsilon$.*

Proof: As in the proof of Theorem 2, it suffices to only consider the derivative of P_{sup} (w.r.t. β this time)—the case for P_{inf} is symmetric. From the definition of smoothing, we have:

$$\begin{aligned}
\frac{\partial[\![\overline{P_{sup}}]\!](\mathbf{x})}{\partial\beta} &= \int_{\mathbf{r}\in\mathbb{R}^k} (\sum_{i=0}^{N} B_i(\mathbf{r}) \cdot F_i(\mathbf{r})) \cdot \frac{\partial \mathcal{N}(\mathbf{r}-\mathbf{x},\beta)}{\partial\beta}\ d\mathbf{r} \\
&= \int_{\mathbf{r}\in\mathbb{R}^k} (\sum_{i=0}^{N} B_i(\mathbf{r}) \cdot F_i(\mathbf{r})) \cdot (\frac{\|\mathbf{r}-\mathbf{x}\|^2}{\beta^3} - \frac{k}{\beta})) \cdot \mathcal{N}(\mathbf{r}-\mathbf{x},\beta)\ d\mathbf{r} \\
&= \sum_{i=0}^{N} \int_{\mathbf{r}\in B_i} F_i(\mathbf{r}) \cdot (\frac{\|\mathbf{r}-\mathbf{x}\|^2}{\beta^3} - \frac{k}{\beta}) \cdot \mathcal{N}(\mathbf{r}-\mathbf{x},\beta)\ d\mathbf{r}. \qquad (7)
\end{aligned}$$

Just like in the proof of robustness, we can decompose the integral into a linear sum of terms of one of the following forms:

$$\frac{1}{\beta^3}\int_{r_j=a}^{b} (r_j - x_j)^2 \cdot \mathcal{N}(r_j - x_j,\beta)\ dr_j \qquad (8)$$

$$\frac{1}{\beta^3}\int_{r_j=a}^{b} r_j \cdot (r_j - x_j)^2 \cdot \mathcal{N}(r_j - x_j,\beta)\ dr_j \qquad (9)$$

$$\frac{1}{\beta}\int_{r_j=a}^{b} r_j \cdot \mathcal{N}(r_j - x_j,\beta)\ dr_j \qquad (10)$$

$$\frac{1}{\beta}\int_{r_j=a}^{b} \mathcal{N}(r_j - x_j,\beta)\ dr_j \qquad (11)$$

The first and last terms are clearly bounded by constants. The third term is similar to the term we saw in the proof of robustness, and is bounded by a linear function; in fact, when a and b go to infinity, the term corresponds to the mean of a Gaussian centered at x. As for the second term, it is bounded when a and b are bounded, but will grow as $O(x_j)$ when a or b are $-\infty$ or ∞ respectively. Putting the facts together, we note that $\left|\frac{\partial[\![\overline{P_{sup}}]\!](\mathbf{x})}{\partial\beta}\right|$ is bounded by a linear function of $\mathbf{x}$ and β, which completes the proof of β-robustness.

6 Related work and conclusion

In a recent paper [5], we introduced smooth interpretation as a program approximation that facilitates more effective use of numerical optimization in reasoning

about programs. While the paper showed the effectiveness of the method, it left open a lot of theoretical questions. For example, while we implemented and empirically evaluated a smooth interpreter, could we also formally characterize smooth interpretation? How did program smoothing relate to program abstraction? In the present paper, we have answered the above questions.

Regarding related work, Gaussian smoothing is ubiquitous in signal and image processing [13]. Also, the idea of using smooth, robust approximations to enable optimization of non-smooth functions has been previously studied in the optimization community [1, 12]. However, these approaches are technically very different from ours, and we were the first to propose and find an application for Gaussian smoothing of programs written in a general-purpose programming language. As for work in the software engineering community, aside from a cursory note by DeMillo and Lipton [10], there does not seem to be any prior proposal here for the use of "smooth" models of programs (although some recent work in program verification studies continuity properties [3, 4] of programs).

The abstract interpretation used in our smoothing framework is closely related to a large body of prior work on static analysis, in particular analysis using intervals [9], interval equalities [7], and interval polyhedra [6]. However, so far as we know, there is no existing abstract domain that can conditionally bound the denotational semantics of a program from above and below.

References

1. D.P. Bertsekas. Nondifferentiable optimization via approximation. *Nondifferentiable Optimization*, pages 1–25, 1975.
2. J. Burnim, S. Juvekar, and K. Sen. Wise: Automated test generation for worst-case complexity. In *ICSE*, pages 463–473, 2009.
3. S. Chaudhuri, S. Gulwani, and R. Lublinerman. Continuity analysis of programs. In *POPL*, 2010.
4. S. Chaudhuri, S. Gulwani, R. Lublinerman, and S. Navidpour. Proving programs robust, 2011.
5. S. Chaudhuri and A. Solar-Lezama. Smooth interpretation. In *PLDI*, 2010.
6. L. Chen, A. Miné, J. Wang, and P. Cousot. Interval polyhedra: An abstract domain to infer interval linear relationships. In *SAS*, pages 309–325, 2009.
7. L. Chen, A. Miné, J. Wang, and P. Cousot. An abstract domain to discover interval linear equalities. In *VMCAI*, pages 112–128, 2010.
8. X. Chen. Convergence of the BFGS method for LC convex constrained optimization. *SIAM J Control and Optim*, 14:2063, 1996.
9. P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *POPL*, 1977.
10. R. DeMillo and R. Lipton. Defining software by continuous, smooth functions. *IEEE Transactions on Software Engineering*, 17(4):383–384, 1991.
11. W.W. Hager and H. Zhang. A survey of nonlinear conjugate gradient methods. *Pacific journal of Optimization*, 2(1):35–58, 2006.
12. Y. Nesterov. Smooth minimization of non-smooth functions. *Mathematical Programming*, 103(1):127–152, 2005.
13. J. Russ. *The image processing handbook*. CRC Press, 2007.