Proving Expected Sensitivity of Probabilistic Programs with Randomized Variable-Dependent Termination Time*

PEIXIN WANG, Shanghai Jiao Tong University, China

HONGFEI FU†, Shanghai Jiao Tong University, China

KRISHNENDU CHATTERJEE, IST Austria (Institute of Science and Technology Austria), Austria

YUXIN DENG, East China Normal University, China and Pengcheng Laboratory, China

MING XU, East China Normal University, China

The notion of program sensitivity (aka Lipschitz continuity) specifies that changes in the program input result in proportional changes to the program output. For probabilistic programs the notion is naturally extended to expected sensitivity. A previous approach develops a relational program logic framework for proving expected sensitivity of probabilistic while loops, where the number of iterations is fixed and bounded. In this work, we consider probabilistic while loops where the number of iterations is not fixed, but randomized and depends on the initial input values. We present a sound approach for proving expected sensitivity of such programs. Our sound approach is martingale-based and can be automated through existing martingale-synthesis algorithms. Furthermore, our approach is compositional for sequential composition of while loops under a mild side condition. We demonstrate the effectiveness of our approach on several classical examples from Gambler's Ruin, stochastic hybrid systems and stochastic gradient descent. We also present experimental results showing that our automated approach can handle various probabilistic programs in the literature.

CCS Concepts: • **Theory of computation → Logic and verification**; **Automated reasoning**; **Program verification**.

Additional Key Words and Phrases: Probabilistic Programs, Martingales, Expected Sensitivity

ACM Reference Format:

Peixin Wang, Hongfei Fu, Krishnendu Chatterjee, Yuxin Deng, and Ming Xu. 2020. Proving Expected Sensitivity of Probabilistic Programs with Randomized Variable-Dependent Termination Time. *Proc. ACM Program. Lang.* 4, POPL, Article 25 (January 2020), 30 pages. https://doi.org/10.1145/3371093

*A full version is available at [Wang et al. 2019a].

†Corresponding Author: Hongfei Fu, fuhf@cs.sjtu.edu.cn

Authors' addresses: Peixin Wang, Basics Lab, Shanghai Jiao Tong University, Shanghai, China, wangpeixin@sjtu.edu.cn; Hongfei Fu, John Hopcroft Center for Computer Science, Shanghai Jiao Tong University, Shanghai, China, fuhf@cs.sjtu.edu.cn; Krishnendu Chatterjee, IST Austria (Institute of Science and Technology Austria), Klosterneuburg, Austria, krishnendu.chatterjee@ist.ac.at; Yuxin Deng, Shanghai Key Laboratory of Trustworthy Computing, East China Normal University, China , Center for Quantum Computing, Pengcheng Laboratory, Shenzhen, China, yxdeng@sei.ecnu.edu.cn; Ming Xu, Shanghai Key Laboratory of Trustworthy Computing, East China Normal University, China, mxu@cs.ecnu.edu.cn.

2475-1421/2020/1-ART25

https://doi.org/10.1145/3371093

1 INTRODUCTION

Continuity properties of systems. Continuity property for systems requires that the change in the output is bounded by a monotone function of the change in the input. Analysis of continuity properties are of great interest in program and reactive system analysis, such as: (a) robustness of numerical computations; (b) analysis of sensitivity of numerical queries [Dwork and Roth 2014] in databases; (c) analysis of stability of learning algorithms [Bousquet and Elisseeff 2002]; and (d) robustness analysis of programs [Chaudhuri et al. 2010].

Probabilistic systems. Continuity analysis is relevant for probabilistic systems in a similar way, where the notion of continuity is extended with expectation to average over the probabilistic behaviours of the system. For example, statistical notions of differential privacy [Dwork et al. 2006]; robustness analysis of Markov chains, Markov decision processes, and stochastic games [Aldous 1983; Chatterjee 2012; Desharnais et al. 2004; Fu 2012; van Breugel and Worrell 2006]; stability analysis of randomized learning algorithms [Bousquet and Elisseeff 2002; Hardt et al. 2016]; all fall under the umbrella of continuity analysis of probabilistic systems.

Program sensitivity. A particular interest among continuity is *program sensitivity* which specifies that the change in the output of a program is proportional to the change in the input. Formally, there is a constant L (the *Lipschitz constant*) such that if the input changes by an amount x, then the change in the ouput is at most $L \cdot x$. In this work we consider the expected sensitivity of probabilistic programs given as (sequential composition of) probabilistic while loops.

Previous results. The expected sensitivity analysis of probabilistic programs was first considered in [Bousquet and Elisseeff 2002; Hardt et al. 2016] for machine-learning algorithms such as stochastic gradient descent, through manual proofs. Then [Barthe et al. 2018] proposed an elegant method based on a relational program logic framework. The heart of the analysis technique is coupling-based methods, and the approach is shown to work effectively on several examples from machine learning to statistical physics. A recent result [Huang et al. 2018b] implemented a computer-algebra based tool that calculates tight sensitivity bounds for probabilistic programs. Although these previous approaches address the expected sensitivity analysis well, they work only on examples of probabilistic while loops whose number of iterations is fixed and bounded (i.e., the number of iterations is fixed to a given number T). In reality, many examples of probabilistic while loops do not have fixed number of iterations, rather the number of iterations is randomized and depends on the input values. Hence, such examples cannot be handled by the previous approaches. In this work, we focus on expected sensitivity analysis of such programs.

Our contributions. Our main contributions are as follows:

(1) We present a sound approach for proving expected sensitivity of probabilistic while loops whose number of iterations is randomized and depends on the initial input values.
(2) We show that our approach is compositional w.r.t sequential composition.
(3) In contrast to the previous coupling and computer-algebra based approaches, our approach relies on ranking supermartingales (RSMs) (see [Chakarov and Sankaranarayanan 2013; Chatterjee et al. 2018c]), a central notion in proving termination properties of probabilistic programs.
(4) Since RSM based approaches can be automated through constraint solving (see e.g. [Chakarov and Sankaranarayanan 2013; Chatterjee et al. 2018c]), the same results in conjunction with our sound approach present an automated approach for sensitivity analysis of probabilistic programs.

(5) We demonstrate the effectiveness of our approach through (i) a case study on stochastic gradient descent and (ii) experimental results on various probabilistic programs from the literature, including Gambler's Ruin, stochastic hybrid systems, random walks, etc.

Technical contribution. In terms of technical contribution there are key differences between our result and the previous results. The previous approaches are either coupling-based proof rules, or through computer-algebra tools, and all of them are restricted to loops with a fixed number of loop iterations. In contrast, our approach is based on RSMs and can handle loops whose number of iterations is randomized and depends on the input. Moreover, we prove the non-trivial fact that our approach is compositional under sequential composition. Furthermore, as RSM-synthesis algorithms have been well-established in the literature, our sound approach directly leads to automated algorithms for proving expected sensitivity of probabilistic programs.

Limitation. Our approach mainly focuses on (sequential composition of) probabilistic while loops where there is no conditional branch. Although the exclusion of conditional branches makes our contribution seemingly restrictive, we argue that typically inclusion of conditional branches will break sensitivity properties in general. Consider a loop of the form **while** Φ **do if** b **then** P **else** Q **od** where the programs P, Q perform completely different executions. Then two close-by program inputs $x_1, x_2 \models \Phi$ such that (**) $x_1 \models b$ but $x_2 \not\models b$ will lead to values that differ significantly after just one loop iteration. Thus, irrespective of analysis methods this type of programs has bad sensitivity property. Previous results also reflect the difficulty on handling conditional branches. For example, in previous approaches such as [Aguirre et al. 2019; Barthe et al. 2018], it must be manually ensured that the conditions of all conditional branches are either (i) both satisfied or (ii) both not satisfied by two close-by program valuations (i.e., the situation (**) above is not allowed) (see [Barthe et al. 2018, Figure 3] and [Aguirre et al. 2019, Figure 1]). Moreover, in all the experimental examples from [Huang et al. 2018b], conditional-branches within for-loops are either restricted to a finite set of values or directly transformed into probabilistic branches. For a possible extension to conditional branches see Remark 5.

2 PROBABILISTIC PROGRAMS

We first present the syntax and semantics of our probabilistic programming language, then define the syntactical subclass of *simple* while loops to which our approach applies. Throughout the paper, we denote by $\mathbb{N}$, $\mathbb{Z}$, and $\mathbb{R}$ the sets of all natural numbers, integers, and real numbers, respectively.

The Syntax. Our probabilistic programming language is imperative and consists of statements. We present a succinct description below (see [Wang et al. 2019a, Appendix A] for the detailed syntax).

- *Variables.* Expressions $\langle pvar \rangle$ (resp. $\langle rvar \rangle$) range over program (resp. sampling) variables, respectively. Program variables are normal variables that control the flow of the program, while each sampling variable is a special variable whose value is sampled from a fixed predefined probability distribution each time the variable is accessed in the program.
- *Constants.* Expressions $\langle const \rangle$ range over decimals.
- *Arithmetic Expressions.* Expressions $\langle expr \rangle$ (resp. $\langle pexpr \rangle$) range over arithmetic expressions over both program and sampling variables (resp. program variables only). For example, if x, y are program variables and r is a sampling variable, then $x + 3 \cdot y$ is an instance of $\langle pexpr \rangle$ and $x - y + 2 \cdot r$ is an instance of $\langle expr \rangle$. In this paper, we consider a general setting of arithmetic expressions and do not fix a detailed syntax for $\langle expr \rangle$ and $\langle pexpr \rangle$.
- *Boolean Expressions.* Expressions $\langle bexpr \rangle$ are boolean expressions over program variables, for which atomic propositions are comparisons between expressions from $\langle pexpr \rangle$ and general expressions are built from atomic propositions and propositional operators.

- *Statements* $\langle stmt \rangle$. Assignment statements are indicated by ':='; '**skip**' is the statement that does nothing; Standard conditional branches are indicated by the keyword '**if**' with its **then**- and **else**-branches, and a boolean expression that serves as the condition for the conditional branch. Probabilistic choices are modelled as probabilistic branches with the key word "**if prob**(p)" that lead to the **then**-branch with probability p and to the **else**-branch with probability $1-p$. While-loops are indicated by the keyword '**while**' with a boolean expression as the loop guard. Finally, sequential compositions are indicated by semicolons.

Note that probabilistic branches can be implemented as a sampling of Bernoulli distribution followed by a conditional branch, but for algorithmic purpose we consider probabilistic branches directly. In this work, we consider probabilistic programs without non-determinism.

The Semantics. To define the semantics, we first recall several standard notions from probability theory as follows (see e.g. standard textbooks [Billingsley 1995; Williams 1991] for details).

Probability Spaces. A *probability space* is a triple $(\Omega, \mathcal{F}, \mathbb{P})$, where Ω is a nonempty set (so-called *sample space*), $\mathcal{F}$ is a *sigma-algebra* over Ω (i.e., a collection of subsets of Ω that contains the empty set $\emptyset$ and is closed under complementation and countable union), and $\mathbb{P}$ is a *probability measure* on $\mathcal{F}$, i.e., a function $\mathbb{P}\colon \mathcal{F} \to [0,1]$ such that (i) $\mathbb{P}(\Omega) = 1$ and (ii) for all set-sequences $A_1, A_2, \cdots \in \mathcal{F}$ that are pairwise-disjoint (i.e., $A_i \cap A_j = \emptyset$ whenever $i \neq j$) it holds that $\sum_{i=1}^{\infty} \mathbb{P}(A_i) = \mathbb{P}\left(\bigcup_{i=1}^{\infty} A_i\right)$. Elements in $\mathcal{F}$ are called *events*. An event $A \in \mathcal{F}$ is said to hold *almost surely* (a.s.) if $\mathbb{P}(A) = 1$.

Random Variables. A *random variable* (r.v.) X on a probability space $(\Omega, \mathcal{F}, \mathbb{P})$ is an $\mathcal{F}$-measurable function $X\colon \Omega \to \mathbb{R} \cup \{-\infty, +\infty\}$, i.e., a function satisfying the condition that for all $d \in \mathbb{R} \cup \{-\infty, +\infty\}$, the set $\{\omega \in \Omega \mid X(\omega) < d\}$ belongs to $\mathcal{F}$. By convention, we abbreviate $+\infty$ as ∞.

Expectation. The *expected value* of a random variable X on a probability space $(\Omega, \mathcal{F}, \mathbb{P})$, denoted by $\mathbb{E}(X)$, is defined as the Lebesgue integral of X w.r.t $\mathbb{P}$, i.e., $\mathbb{E}(X) := \int X \, \mathrm{d}\mathbb{P}$; the precise definition of Lebesgue integral is somewhat technical and is omitted here (cf. [Williams 1991, Chapter 5] for a formal definition). In the case that $ran\, X = \{d_0, d_1, \ldots, d_k, \ldots\}$ is countable with distinct d_k's, we have that $\mathbb{E}(X) = \sum_{k=0}^{\infty} d_k \cdot \mathbb{P}(X = d_k)$.

To present the semantics, we also need the notion of *valuations*.

Valuations. Let V be a finite set of variables with an implicit linear order over its elements. A *valuation* on V is a vector $\mathbf{b}$ in $\mathbb{R}^{|V|}$ such that for each $1 \leq i \leq |V|$, the i-th coordinate of $\mathbf{b}$, denoted by $\mathbf{b}[i]$, is the value for the i-th variable in the implicit linear order on V. For the sake of convenience, we write $\mathbf{b}[y]$ for the value of a variable y in a valuation $\mathbf{b}$.

Program and Sampling Valuations. Let V_{p} (resp. V_{r}) be the set of program (resp. sampling) variables appearing in a probabilistic program, respectively. A *program valuation* (or *program state*) is a valuation on V_{p}. A *sampling valuation* is a valuation on V_{r}. Given a program valuation $\mathbf{b}$ and a boolean expression Φ, the satisfaction relation $\models$ is defined in the standard way so that we have $\mathbf{b} \models \Phi$ iff Φ holds when program variables in Φ are substituted by their corresponding values in $\mathbf{b}$.

Now we give a brief description of the semantics for probabilistic programs. We follow the standard operational semantics through Markov chains. Given a probabilistic program (without non-determinism), its semantics is given as a general state-space Markov chain (GSSMC) [Meyn and Tweedie 1993, Chapter 3], where (i) the state space consists of all pairs of program counters and program valuations for which the program counter refers to the next command to be executed and the program valuation specifies the current values for the program variables, and (ii) the kernel function that specifies the stochastic transitions between states is given by the individual commands

in the program. For any initial state $\mathfrak{c} = (\mathsf{in}, \mathbf{b})$ where in is the program counter of the first command and $\mathbf{b}$ is the input program valuation, each probabilistic program induces a unique probability space through its corresponding GSSMC, where the sample space consists of all infinite sequences of states in the GSSMC (as *runs*), the sigma-algebra is generated by all *cylinder* sets of runs induced by finite Cartisian products of measurable subsets of the state space, and the probability measure is uniquely determined by the kernel function and the initial state. The detailed semantics can be found in [Chakarov and Sankaranarayanan 2013; Chatterjee et al. 2018c; Fu and Chatterjee 2019].

Under our semantics, we denote by $\mathbb{P}_{\mathbf{b}}$ the probability measure for a probabilistic program with the input program valuation $\mathbf{b}$ (note that the program counter in is determined by the program), and by $\mathbb{E}_{\mathbf{b}}(-)$ the expectation under the probability measure $\mathbb{P}_{\mathbf{b}}$.

Simple While Loops. In this paper, we focus on (sequential composition of) simple probabilistic while loops and investigate sound approaches for proving expected sensitivity over such programs. A *simple* (probabilistic) while loop is of the form

$$\textbf{while } \Phi \textbf{ do } P \textbf{ od} \tag{1}$$

where Φ is the *loop guard* and the loop body P is a program without nested while loops. As simple while loops are syntactically restricted, we present succinct notions for such programs.

Update Functions. Given a simple while loop in the form (1) with the disjoint sets V_{p} and V_{r} of program and sampling variables, we abstract away detailed executions of the loop body P by an *update function* $F : \mathbf{L} \times \mathbb{R}^{|V_{\mathrm{p}}|} \times \mathbb{R}^{|V_{\mathrm{r}}|} \to \mathbb{R}^{|V_{\mathrm{p}}|}$ that describes the input-output relationship for one iteration of the loop body as follows. First, we let $\mathcal{L}$ be the set of all program counters that refer to a probabilistic branch (i.e., **if prob**(p) ...) in the loop body of P. Then we define $\mathbf{L}$ to be the set of all functions from $\mathcal{L}$ into the choices of branches (i.e., **then**- or **else**-branch). Informally, such a function specifies for each probabilistic branch in P which branch is chosen in the current loop iteration. Finally, the update function F simply gives the program valuation $F(\ell, \mathbf{b}, \mathbf{r})$ after the current loop iteration given (i) an element $\ell \in \mathbf{L}$ that specifies the probabilistic choices for probabilistic branches, (ii) a program valuation $\mathbf{b}$ that specifies the values for program variables before the current loop iteration and (iii) a sampling valuation $\mathbf{r}$ that gives all the sampled values for the sampling variables in the current loop iteration. In this way, we abstract away the detailed execution within the loop body P and represent it simply by an update function. Note that as the loop body P does not involve nested while loops, one can compute its update function symbolically through a recursive algorithm on the structure of P.

Runs. We also simplify the notion of runs over simple while loops. A *run* for a loop in the form (1) is an infinite sequence $\{\mathbf{b}_n\}_{n\geq 0}$ of program valuations such that each $\mathbf{b}_n$ is the program valuation right before the $(n+1)$-th loop iteration. Note that if $\mathbf{b}_n \models \Phi$, then $\mathbf{b}_{n+1} = F(\ell_n, \mathbf{b}_n, \mathbf{r}_n)$ where ℓ_n (resp. $\mathbf{r}_n$) specifies the probabilistic resolution to all the probabilistic branches (resp. the sampled values for the sampling variables) at the $(n+1)$-th loop iteration, respectively; otherwise, $\mathbf{b}_{n+1} = \mathbf{b}_n$.

Notations. To ease the use of notations, we always use $\mathbf{b}$ for a program valuation, $\mathbf{r}$ for a sampling valuation and ℓ for an element in $\mathbf{L}$, with possible super-/sub-scripts. Given a simple while loop Q in the form (1), we always use V_{p} for its set of program variables, V_{r} for sampling variables, F for the update function, Φ for the loop guard, and P for the loop body. Moreover, we denote by $[\![\Phi]\!]$ the set $\{\mathbf{b} \mid \mathbf{b} \models \Phi\}$ of program valuations that satisfy the loop guard Φ.

To reason about expected sensitivity of simple while loops, we require that the loop body is Lipschitz continuous. This requirement is standard and corresponds to the "η-expansiveness" introduced in

```
while x ≤ 1000 do
  x := x + r
od
```

Fig. 1. Running Example

```
while Φ do
  i := unif[1, ..., n];
  w := w − γ · ∇G_i(w)
od
```

Fig. 2. An SGD Algorithm

```
while y ≤ x ∧ y ≥ 0 do
  y := y − 1
od
```

Fig. 3. A Counterexample for Sensitivity

[Hardt et al. 2016, Definition 2.3]. This continuity condition needs the standard notion of *metrics* that measures the distance between two program valuations, as follows.

Metrics. A *metric* is a function $\mathfrak{d} : \mathbb{R}^{|V_p|} \times \mathbb{R}^{|V_p|} \to [0, \infty)$ that satisfies (i) $\mathfrak{d}(\mathbf{b}, \mathbf{b}') = 0$ iff $\mathbf{b} = \mathbf{b}'$, (ii) $\mathfrak{d}(\mathbf{b}, \mathbf{b}') = \mathfrak{d}(\mathbf{b}', \mathbf{b})$ (*symmetry*) and (iii) $\mathfrak{d}(\mathbf{b}, \mathbf{b}') \leq \mathfrak{d}(\mathbf{b}, \mathbf{b}'') + \mathfrak{d}(\mathbf{b}'', \mathbf{b}')$ (*triangle inequality*). Informally, $\mathfrak{d}(\mathbf{b}, \mathbf{b}')$ is interpreted as the distance between the two program valuations. For example, we can define $\mathfrak{d}$ either through the max norm by $\mathfrak{d}(\mathbf{b}, \mathbf{b}') := \|\mathbf{b} - \mathbf{b}'\|_\infty$ where the max norm $\|\cdot\|_\infty$ is given as $\|\mathbf{b}''\|_\infty := \max_{z \in V_p} |\mathbf{b}''[z]|$, or through the Euclidean norm by $\mathfrak{d}(\mathbf{b}, \mathbf{b}') := \|\mathbf{b} - \mathbf{b}'\|_2$ where $\|\cdot\|_2$ is given as $\|\mathbf{b}''\|_2 := \sqrt{(\mathbf{b}'')^T \mathbf{b}''}$. In this paper, we consider metrics that are comparable with the max norm, i.e., there exist real constants $D_1, D_2 > 0$ such that

$$D_1 \cdot \|\mathbf{b} - \mathbf{b}'\|_\infty \leq \mathfrak{d}(\mathbf{b}, \mathbf{b}') \leq D_2 \cdot \|\mathbf{b} - \mathbf{b}'\|_\infty \ . \tag{2}$$

Note that the comparability is naturally satisfied for metrics derived from norms of finite dimension.

Below we describe the continuity of the loop body under a metric $\mathfrak{d}$.

Definition 2.1 (Lipschitz Continuity L of the Loop Body). We say that the loop body of a simple while loop in the form (1) is *Lipschitz continuous* if there exists a real constant $L > 0$ such that

(B1) $\forall \ell \, \forall \mathbf{r} \, \forall \mathbf{b}, \mathbf{b}' \colon \ [\mathbf{b}, \mathbf{b}' \models \Phi \Rightarrow \mathfrak{d}(F(\ell, \mathbf{b}, \mathbf{r}), F(\ell, \mathbf{b}', \mathbf{r})) \leq L \cdot \mathfrak{d}(\mathbf{b}, \mathbf{b}')] \ .$

If we can choose $L = 1$ in (B1), then the loop is *non-expansive*; otherwise it is *expansive* (i.e., the minimum L is greater than 1).

Remark 1 (Simple While Loops). In general, any imperative probabilistic program can be transformed equivalently into a simple while loop by adding a variable for the program counter and then simulating the original program through transitions between program counters and valuations. However, the class of simple while loops that can be handled by our approach is restricted to those with Lipschitz-continuous loop body. Thus generally, our approach cannot handle conditional branches that usually breaks the continuity property.

Example 2.2 (The Running Example). Consider the simple while loop in Figure 1. In the program, x is a program variable and r is a sampling variable. In every loop iteration, the value of x is increased by a value sampled w.r.t the probability distribution of r until it is greater than 1000. There is no probabilistic branch so $\mathbf{L}$ is a singleton set that only contains the empty function. The update function F for the loop body is then given by $F(\ell, \mathbf{b}, \mathbf{r})[x] = \mathbf{b}[x] + \mathbf{r}[r]$ for program valuation $\mathbf{b}$ and sampling valuation $\mathbf{r}$, where ℓ is the only element in $\mathbf{L}$. By definition, the loop is non-expansive.

3 EXPECTED SENSITIVITY ANALYSIS OF PROBABILISTIC PROGRAMS

In this paper, we focus on averaged sensitivity which is one of the most fundamental sensitivity notions in expected sensitivity analysis of probabilistic programs. Informally, averaged sensitivity compares the distance between the expected outcomes from two close-by input program valuations. The notion of averaged sensitivity has an important applicational value in that it can be used to

model algorithmic stability in many machine-learning algorithms (see e.g. [Bousquet and Elisseeff 2002]).

Below we illustrate the notion of averaged sensitivity formally. To ensure well-definedness, we only consider probabilistic programs that terminate with probability one (i.e., with *almost-sure termination* [Chakarov and Sankaranarayanan 2013]) for all input program valuations. Furthermore, as our approach will rely on ranking supermartingales, we actually require that the probabilistic programs we consider terminate with finite expected termination time [Chatterjee et al. 2018c]. Below we fix a probabilistic program Q and a metric $\mathfrak{d} : \mathbb{R}^{|V_p|} \times \mathbb{R}^{|V_p|} \to [0, \infty)$.

Definition 3.1 (Averaged Sensitivity [Barthe et al. 2018; Bousquet and Elisseeff 2002]). We say that the program Q is *averaged affine-sensitive* over a subset $U \subseteq \mathbb{R}^{|V_p|}$ of input program valuations if there exist real constants $A, B \geq 0$ and $\theta \in (0, \infty]$ such that for all program variables z and $\mathbf{b}, \mathbf{b}' \in U$,

$$\text{if } \mathfrak{d}(\mathbf{b}, \mathbf{b}') \leq \theta \text{ then } |\mathbb{E}_{\mathbf{b}}(Z) - \mathbb{E}_{\mathbf{b}'}(Z')| \leq A \cdot \mathfrak{d}(\mathbf{b}, \mathbf{b}') + B \tag{3}$$

where Z, Z' are random variables representing the values of z after the execution of the program Q under the input program valuations $\mathbf{b}, \mathbf{b}'$, respectively. Furthermore, if we can choose $B = 0$ in (3), then the program Q is said to be *averaged linear-sensitive* in the program variable z.

In the definition, the constants A, B are sensitivity coefficients, while θ is the threshold below which the sensitivity is applicable; if $\theta = \infty$ then the sensitivity is applicable regardless of the distance between $\mathbf{b}, \mathbf{b}'$. Informally, a program Q is averaged affine-sensitive if the difference in the expected value of any program variable z after the termination of Q is bounded by an affine function in the difference of the input program valuations. Likewise, the program is averaged linear-sensitive if the difference can be bounded by a linear function. In this way, we consider the expected sensitivity of the return values where each program variable represents an individual return value. Note that another subtle issue arising from the well-definedness is that the random variables Z, Z' in (3) may not be integrable. In the following, we will always guarantee that the random variables are integrable.

As we only consider averaged sensitivity, in the rest of the paper we will refer to *averaged affine-/linear-sensitivity* simply as *expected affine-/linear-sensitivity*. It is worth noting that in [Barthe et al. 2018], a coupling-based definition for expected sensitivity is proposed. Compared with their definition, our definition treats expected sensitivity directly and do not consider couplings.

4 MOTIVATING EXAMPLES

In the following, we show several motivating examples for expected sensitivity analysis of probabilistic programs. We consider in particular probabilistic programs with a randomized number of loop iterations that also depends on the input program valuation. As existing results [Barthe et al. 2018; Hardt et al. 2016; Huang et al. 2018b] only consider probabilistic loops with a fixed number of loop iterations, none of the examples in this section can be handled by these approaches.

Example 4.1 (Mini-roulette). A particular gambler's-ruin game is called *mini-roulette*, which is a popular casino game based on a 13-slot wheel. A player starts the game with x amount of chips. He needs one chip to make a bet and he bets as long as he has chips. If he loses a bet, the chip will not be returned, but a winning bet will not consume the chip and results in a specific amount of (monetary) reward, and possibly even more chips. The following types of bets can be placed at each round. (1) *Even-money bets*: In these bets, 6 specific slots are chosen. Then the ball is rolled and the player wins the bet if it lands in one of the 6 slots. So the player has a winning probability of $\frac{6}{13}$. Winning them gives a reward of two unit and one extra chip. (2) *2-to-1 bets*: these bets

```
while x ≥ 1 do
  if prob(6/65) then
     x := x + 1 ; w := w + 2
  else if prob(4/59) then
       x := x + 2 ; w := w + 3
  else if prob(3/55) then
       x := x + 3 ; w := w + 4
  else if prob(2/52) then
       x := x + 5 ; w := w + 5
  else if prob(1/50) then
       x := x + 11 ; w := w + 6
  else x := x − 1
  fi fi fi fi fi od
```

```
while x ≥ 1 do
  if prob(6/65) then
     x := x + r1 ; w := w + 2
  else if prob(4/59) then
       x := x + r2 ; w := w + 3
  else if prob(3/55) then
       x := x + r3 ; w := w + 4
  else if prob(2/52) then
       x := x + r4 ; w := w + 5
  else if prob(1/50) then
       x := x + r5 ; w := w + 6
  else x := x − r6
  fi fi fi fi fi od
```

Fig. 4. A Mini-roulette example (left) and its continuous variant (right)

correspond to 4 chosen slots and winning them gives a reward of 3 and 2 extra chips. (3) *3-to-1, 5-to-1 and 11-to-1 bets*: These are defined similarly and have winning probabilities of $\frac{3}{13}$, $\frac{2}{13}$ and $\frac{1}{13}$ respectively. Suppose at each round, the player chooses each type of bets with the same probability (i.e. chooses each type with probability $\frac{1}{5}$). The probabilistic program for this example is shown in Figure 4(left), where the program variable x represents the amount of chips and the program variable w records the accumulated rewards. (In the program we consider that x can take a real value.) We also consider a continuous variant of the mini-roulette example in Figure 4(right), where we replace increments to the variable x by uniformly-distributed sampling variables $r_i (i = 1, \ldots, 6)$ and one may choose $r_1 \sim \mathrm{unif}(1, 2)$, $r_2 \sim \mathrm{unif}(2, 3)$, $r_3 \sim \mathrm{unif}(3, 4)$, $r_4 \sim \mathrm{unif}(4, 5)$, $r_5 \sim \mathrm{unif}(8, 9)$, $r_6 \sim \mathrm{unif}(1, 2)$ or other uniform distributions that ensure the termination of the program. Note that the number of loop iterations in all the programs in Figure 4 is randomized and depends on the input program valuation as the loop guard is $x \geq 1$ and the increment/decrement of x is random in each loop iteration. In both the examples, we consider the expected sensitivity in the output program variable w that records the accumulated reward.

Example 4.2 (Multi-room Heating). We consider a case study on multi-room heating from [Abate et al. 2010], modelled as a stochastic hybrid system that involves discrete and probabilistic dynamics. In the case study, there are n rooms each equipped with a heater. The heater can heat the room and the heat can be transferred to another room if the rooms are adjacent. We follow the setting from [Abate et al. 2010] that the average temperature of each room, say room i, evolves according to the following stochastic difference equation that describes the transition from the k-th time step to the $(k + 1)$-th time step, with constant time-interval Δt:

$$x_i(k+1) = x_i(k) + b_i(x_a - x_i(k)) + \textstyle\sum_{i \neq j} a_{ij}(x_j(k) - x_i(k)) + c_i + w_i(k) \tag{4}$$

where (i) x_a represents the ambient temperature (assumed to be constant and equal for the whole building), (ii) the quantities b_i, a_{ij}, c_i are nonnegative constants representing respectively the average heat transfer rate from room i to the ambient (i.e., b_i), to adjacent rooms $j \neq i$ (i.e., a_{ij}'s), supplied to room i by the heater (i.e., c_i), and (iii) $w_i(k)$ is the noise that observes a predefined probability distribution, such as Gaussian, Poisson or uniform distribution, etc. In this paper, we consider two simplified scenarios. The first is a single-room heating modelled as the probabilistic program in Figure 5, where the program variable x represents the current room temperature, the constants x_a, b, c are as in (4) and w is the noise (as a sampling variable); the goal in the first scenario

is to raise the room temperature up to 20 °C. We assume that the starting room temperature is between 0 °C and 20 °C. The second is a double-room heating modelled in Figure 6, where the heater of the main room is on and the heater for the side room is off. In the figure, the program variable x_1 (i.e., x_2) represents the temperature for the main room (resp. the side room), respectively; the constants x_a, b_i, c_i, a_{ij} are as in (4); the sampling variables w_1, w_2 represent the noises. In the program, we adopt the succinct form of simultaneous vector assignment for updates to x_1, x_2. In both scenarios, we have a loop counter n that records the number of stages until the (main) room reaches 20 °C. We consider in particular the expected sensitivity w.r.t the total number of stages as recorded in n, for which we assume that the value of n always starts with 0.

while $0 \le x \le 20$ **do**
  $x := x + b * (x_a - x) + c + w$;
  $n := n + 1$
od

Fig. 5. Single-Room Heating

while $0 \le x_1 \le 20 \wedge 0 \le x_2 \le 20$ **do**
  $\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} := \begin{pmatrix} x_1 + b_1 * (x_a - x_1) + a_{12} * (x_2 - x_1) + c_1 + w_1 \\ x_2 + b_2 * (x_a - x_2) + a_{21} * (x_1 - x_2) + w_2 \end{pmatrix}$;
  $n := n + 1$
od

Fig. 6. Double-Room Heating

Example 4.3 (Stochastic Gradient Descent). The most widely-used method in machine learning is *stochastic gradient descent* (SGD). The general form of an SGD algorithm is illustrated in Figure 2 on Page 6. In the figure, an SGD algorithm with n training data is modelled as a simple while loop, where (i) unif$[1, \dots, n]$ is a sampling variable whose value is sampled uniformly from $1, 2, \dots, n$, (ii) $\mathbf{w}$ is a vector of program variables that represents parameters to be learned, (iii) i is a program variable that represents the sampled index of the training data, and (iv) γ is a positive constant that represents the *step size*. The symbol ∇ represents the *gradient*, while each G_i $(1 \le i \le n)$ is the loss function for the ith training data. By convention, the total loss function G is given as the expected sum of all G_i's, i.e., $G := \frac{1}{n} \sum_i G_i$. At each loop iteration, a data i is chosen uniformly from all n training data and the parameters in $\mathbf{w}$ are adjusted by the product of the step size and the gradient of the ith loss function G_i. The loop guard Φ can either be practical so that a fixed number of iterations is performed (as is analyzed in existing approaches [Barthe et al. 2018; Hardt et al. 2016; Huang et al. 2018b]), or the local criteria that the magnitude $\|\nabla G\|_2$ of the gradient of the total loss function G is small enough, or the global criteria that the value of G is small enough. In this paper, we consider the global criteria, i.e., the loop guard is of the form $G(\mathbf{w}) \ge \zeta$ where ζ is the threshold for "small enough". Note that the SGD algorithm with the global criteria has randomized loop iterations which depends on the initial parameters.

5 PROVING EXPECTED SENSITIVITY FOR NON-EXPANSIVE SIMPLE LOOPS

In this section, we demonstrate a sound approach for proving expected sensitivity over non-expansive simple while loops, whose number of loop iterations is randomized and depends on the input program valuation. The main difficulty is that when the number of loop iterations depends on both the randomized execution and the input program valuation, the executions from two close-by input program valuations may be *non-synchronous* in the sense that they do not terminate at the same time. The following example illustrates this situation.

Example 5.1 (Non-synchronicity). Consider our running example in Figure 1, where the sampling variable r observes the Dirac distribution such that $\mathbb{P}(r = 1) = 1$, so that the program is completely deterministic. Choose the initial inputs x_1^*, x_2^* by setting $x_1^* = 1 - \epsilon$ and $x_2^* = 1 + \epsilon$, where $\epsilon > 0$ can be sufficiently small. Since we add 1 to the value of x in each loop iteration, the output value

x_2^{out} under the input x_2^* equals $1000 + \epsilon$, while at the same step the execution from x_2^* stops, the execution from x_1^* does not terminate as the corresponding value is $1000 - \epsilon$. Note that the final output from x_1^* is $1001 - \epsilon$.

The non-synchronicity prevents us from inferring the total expected sensitivity from the local sensitivity incurred in each loop iteration. To address this issue, we explore a martingale-based approach. In previous results such as [Chakarov and Sankaranarayanan 2013; Chatterjee et al. 2018c], martingales have been successfully applied to prove termination properties of probabilistic programs. Besides qualitative termination properties, martingales can also derive tight quantitative upper/lower bounds for expected termination time and resource usage [Chatterjee et al. 2018a,c; Ngo et al. 2018; Wang et al. 2019b]. In this paper, we utilize the quantitative feature of martingales to bound the difference caused by non-synchronous situations.

We first recall the notion of *ranking-supermartingale maps* (RSM-maps), a core notion in the application of martingale-based approaches to probabilistic programs. As we consider simple while loops as the basic building block of probabilistic programs, we present a simplified version for simple while loops. Below we fix a simple while loop Q in the form (1).

Definition 5.2 (RSM-maps [Chakarov and Sankaranarayanan 2013; Chatterjee et al. 2018c]). A *ranking-supermartingale map* (RSM-map) is a Borel-measurable function $\eta : \mathbb{R}^{|V_p|} \to \mathbb{R}$ such that there exist real numbers $\epsilon > 0, K \leq 0$ satisfying the following conditions:

(A1) $\forall \mathbf{b} : \big(\mathbf{b} \models \Phi \Rightarrow \eta(\mathbf{b}) \geq 0\big)$;
(A2) $\forall \mathbf{b}\, \forall \ell\, \forall \mathbf{r} : \big((\mathbf{b} \models \Phi \wedge F(\ell, \mathbf{b}, \mathbf{r}) \not\models \Phi) \Rightarrow K \leq \eta(F(\ell, \mathbf{b}, \mathbf{r})) \leq 0\big)$;
(A3) $\forall \mathbf{b} : \big(\mathbf{b} \models \Phi \Rightarrow \mathbb{E}_{\mathbf{r},\ell}(\eta(F(\ell, \mathbf{b}, \mathbf{r}))) \leq \eta(\mathbf{b}) - \epsilon\big)$;

where $\mathbb{E}_{\mathbf{r},\ell}(\eta(F(\ell, \mathbf{b}, \mathbf{r})))$ is the expected value of $\eta(F(\ell, \mathbf{b}, \mathbf{r}))$ such that $\mathbf{b}$ is treated as a constant vector and $\mathbf{r}$ (resp. ℓ) observes the joint probability distributions of sampling variables (resp. the probabilities of the probabilistic branches), respectively.

Informally, (A1) specifies that the RSM-map should be non-negative before program termination, (A2) specifies the condition at loop termination, and (A3) specifies the *ranking* condition that the expected value of the RSM-map should decrease (by the positive amount ϵ) after each loop iteration.

The existence of an RSM-map provides a finite upper bound on the expected termination time of a probabilistic program [Chatterjee et al. 2018c] (see Theorem C.1 in [Wang et al. 2019a, Appendix C]). In this way, an RSM-map controls the randomized number of loop iterations. However, simply having an upper bound for the expected termination time is not enough, as what we need to bound is the difference between the expected values in non-synchronous situations. To resolve the non-synchronicity, we need some additional conditions. The first is the *bounded-update* requiring that the value-change in one loop iteration is bounded. The second is the *RSM-continuity* specifying that the RSM-map should be Lipschitz continuous over the loop guard. Below we fix a metric $\mathfrak{d}$.

Definition 5.3 (Bounded Update d). We say that a simple while loop Q has *bounded update* if there exists a real constant $d \geq 0$ such that

(B2) $\forall \ell\, \forall \mathbf{b}\, \forall \mathbf{r} : \big(\mathbf{b} \models \Phi \Rightarrow \mathfrak{d}(\mathbf{b}, F(\ell, \mathbf{b}, \mathbf{r})) \leq d\big)$.

The bounded-update condition simply bounds the change of values during each loop iteration. This condition is standard as it comes from the "σ-finiteness" proposed in the analysis of stochastic gradient descent [Hardt et al. 2016, Definition 2.4].

Definition 5.4 (RSM-continuity M). An RSM-map η has *RSM-continuity* if there exists a real constant $M > 0$ such that

(B3) $\forall \mathbf{b}, \mathbf{b}' : |\eta(\mathbf{b}) - \eta(\mathbf{b}')| \leq M \cdot \mathfrak{d}(\mathbf{b}, \mathbf{b}')$.

By definition, the RSM-continuity bounds the difference of the RSM-map value proportionally in the metric $\mathfrak{d}$ when the program valuations $\mathbf{b}, \mathbf{b}'$ are close. This condition is used to bound the difference in non-synchronous situations and is naturally satisfied if the RSM-map is linear. Although this condition seems a bit restrictive, latter we will show that it can be relaxed (see Remark 4 and Remark 7). We keep the condition in its current form for the sake of brevity.

Below we first present our result for affine sensitivity, then linear sensitivity. We fix a metric $\mathfrak{d}$.

5.1 Proving Expected Affine-Sensitivity

The main result for proving expected affine-sensitivity of non-expansive simple loops is as follows.

Theorem 5.5. *A non-expansive simple while loop Q in the form (1) is expected affine-sensitive over its loop guard $[\![\Phi]\!]$ if we have that*

- *Q has bounded update, and*
- *there exists an RSM-map for Q that has RSM-continuity.*

In particular, we can choose $\theta = \infty$ and $A = 2 \cdot \frac{d \cdot M + \epsilon}{\epsilon \cdot D_1}, B = -2 \cdot \frac{d \cdot K}{\epsilon \cdot D_1}$ in (3), where the parameters d, M, ϵ, K, D_1 are from Definition 5.2, Definition 5.3, Definition 5.4 and (2).

Proof Sketch. Choose any program variable z. Let d be a bound from Definition 5.3, and η be an RSM-map with the parameters ϵ, K from Definition 5.2 that has RSM-continuity with a constant M from Definition 5.4. Consider input program valuations $\mathbf{b}, \mathbf{b}'$ such that $\mathbf{b}, \mathbf{b}' \models \Phi$. Let $\delta := \mathfrak{d}(\mathbf{b}, \mathbf{b}')$. Denote by $T_{\mathbf{b}''}$ (resp. $Z_{\mathbf{b}''}$) the random variable for the number of loop iterations (resp. the value of z after the execution of Q) from an input program valuation $\mathbf{b}''$, respectively. Define $\mathbf{W}_{\mathbf{b}''}$ as the vector of random variables that represents the program valuation after the execution of Q, starting from $\mathbf{b}''$. We illustrate the main proof idea through clarifying the relationships between any runs $\omega = \{\mathbf{b}_n\}_{n \geq 0}$, $\omega' = \{\mathbf{b}'_n\}_{n \geq 0}$ that start from respectively $\mathbf{b}, \mathbf{b}'$ (i.e., $\mathbf{b}_0 = \mathbf{b}$ and $\mathbf{b}'_0 = \mathbf{b}'$) and follow the *same* probabilistic branches and sampled values in every loop iteration. Consider at a step n the event $\min\{T_{\mathbf{b}}, T_{\mathbf{b}'}\} \geq n$ holds (i.e., both the executions do not terminate before the nth loop iteration). We have the following cases:

Case 1. Both $\mathbf{b}_n$ and $\mathbf{b}'_n$ violate the loop guard Φ, i.e., $\mathbf{b}_n, \mathbf{b}'_n \models \neg\Phi$. This case describes that the loop Q terminates exactly after the nth loop iteration for both the executions. From the non-expansiveness, we obtain directly that $\mathfrak{d}(\mathbf{b}_n, \mathbf{b}'_n) \leq \delta$. Hence $|\mathbf{b}_n[z] - \mathbf{b}'_n[z]| \leq \frac{\mathfrak{d}(\mathbf{b}_n, \mathbf{b}'_n)}{D_1} \leq \frac{\delta}{D_1}$.

Case 2. Exactly one of $\mathbf{b}_n, \mathbf{b}'_n$ violates the loop guard Φ. This is the non-synchronous situation that needs to be addressed through martingales. W.l.o.g., we can assume that $\mathbf{b}_n \models \Phi$ and $\mathbf{b}'_n \models \neg\Phi$. From the upper-bound property of RSM-maps (see Theorem C.1 in [Wang et al. 2019a, Appendix C]), we derive that $\mathbb{E}_{\mathbf{b}_n}(T_{\mathbf{b}_n}) \leq \dfrac{\eta(\mathbf{b}_n) - K}{\epsilon}$. From the bounded-update condition (B2) and the triangle inequality of metrics, we have that $|\mathbf{b}_n[z] - Z_{\mathbf{b}_n}| \leq \frac{1}{D_1} \cdot \mathfrak{d}(\mathbf{b}_n, \mathbf{W}_{\mathbf{b}_n}) \leq \frac{d}{D_1} \cdot T_{\mathbf{b}_n}$. Thus, we obtain that

$$|\mathbb{E}_{\mathbf{b}_n}(Z_{\mathbf{b}_n}) - \mathbf{b}_n[z]| \leq \mathbb{E}_{\mathbf{b}_n}(|\mathbf{b}_n[z] - Z_{\mathbf{b}_n}|) \leq \mathbb{E}_{\mathbf{b}_n}\left(\frac{d}{D_1} \cdot T_{\mathbf{b}_n}\right) \leq \frac{d}{D_1} \cdot \frac{\eta(\mathbf{b}_n) - K}{\epsilon} \quad . \tag{5}$$

By the non-expansiveness, we have $\mathfrak{d}(\mathbf{b}_n, \mathbf{b}'_n) \leq \delta$. Then by the RSM-continuity (B3), we have $|\eta(\mathbf{b}_n) - \eta(\mathbf{b}'_n)| \leq M \cdot \delta$. Furthermore, from (A2) we have $\eta(\mathbf{b}'_n) \leq 0$. So we obtain that $\eta(\mathbf{b}_n) \leq M \cdot \delta$. It follows that

$$\begin{aligned} |\mathbb{E}_{\mathbf{b}_n}(Z_{\mathbf{b}_n}) - \mathbb{E}_{\mathbf{b}'_n}(Z_{\mathbf{b}'_n})| &= |\mathbb{E}_{\mathbf{b}_n}(Z_{\mathbf{b}_n}) - \mathbf{b}'_n[z]| \\ &\leq |\mathbb{E}_{\mathbf{b}_n}(Z_{\mathbf{b}_n}) - \mathbf{b}_n[z]| + |\mathbf{b}_n[z] - \mathbf{b}'_n[z]| \\ &\leq \frac{d}{D_1} \cdot \frac{M \cdot \delta - K}{\epsilon} + \frac{\delta}{D_1} = \frac{d \cdot M + \epsilon}{\epsilon \cdot D_1} \cdot \delta - \frac{d \cdot K}{\epsilon \cdot D_1} \ . \end{aligned}$$

Case 3. Neither $\mathbf{b}_n$ nor $\mathbf{b}'_n$ violates the loop guard Φ. In this case, the loop Q will continue from both $\mathbf{b}_n$ and $\mathbf{b}'_n$. Then in the next iteration, the same analysis can be carried out for the next program valuations $\mathbf{b}_{n+1}, \mathbf{b}'_{n+1}$, and so forth.

From Theorem C.1 (in [Wang et al. 2019a, Appendix C]), the probability that the third case happens infinitely often equals zero. Thus, the sensitivity analysis eventually reduces to the first two cases. In these two cases, the difference contributed to the total expected sensitivity $|\mathbb{E}_{\mathbf{b}}(Z_{\mathbf{b}}) - \mathbb{E}_{\mathbf{b}'}(Z_{\mathbf{b}'})|$ when one of the runs terminates after the nth loop iteration is at most $\mathbb{P}(T_{\mathbf{b}} = n \vee T_{\mathbf{b}'} = n) \cdot \left(\frac{d \cdot M + \epsilon}{\epsilon \cdot D_1} \cdot \delta - \frac{d \cdot K}{\epsilon \cdot D_1}\right)$ which is no greater than $(\mathbb{P}(T_{\mathbf{b}} = n) + \mathbb{P}(T_{\mathbf{b}'} = n)) \cdot \left(\frac{d \cdot M + \epsilon}{\epsilon \cdot D_1} \cdot \delta - \frac{d \cdot K}{\epsilon \cdot D_1}\right)$. Then by a summation over all n's, we derive the desired result that $|\mathbb{E}_{\mathbf{b}}(Z_{\mathbf{b}}) - \mathbb{E}_{\mathbf{b}'}(Z_{\mathbf{b}'})| \leq A \cdot \delta + B$ where $A := 2 \cdot \frac{d \cdot M + \epsilon}{\epsilon \cdot D_1}$ and $B := -2 \cdot \frac{d \cdot K}{\epsilon \cdot D_1}$. The detailed proof requires an explicit representation of the expected values through Lebesgue integral (see [Wang et al. 2019a, Appendix C]). In particular, the integral representation allows us to consider the *same* probabilistic branches and sampled values in each loop iteration. Another subtle point is that the integrability of the random variables Z, Z' in (3) are guaranteed by the bounded-update condition and finite expected termination time from Theorem C.1. □

Remark 2. Although in the statement of Theorem 5.5 we do not bound the constant B, the result is non-trivial as it applies to all program valuations that satisfy the loop guard. This is because input program valuations in the satisfaction set may lead to unbounded expected outcome as the expected number of loop iterations depend on input program valuations. Thus, simply raising the value of B does not suffice to bound the unbounded expected outcomes.

Remark 3. In Theorem 5.5 we only consider the sensitivity over the loop guard. The reason is that since we consider loops with randomized and input-dependent loop iterations, sensitivity usually applies to the loop guard only. Consider the (nonprobabilistic) loop in Figure 3 on Page 6. We can construct an RSM-map η by $\eta(x, y) := y$ with parameters $\epsilon = 1, K = -1$. Then by Theorem 5.5 (where we set $d = M = 1$ and $\mathfrak{d}$ to be the max-norm) we derive that the loop is expected affine-sensitive over its loop guard. However, it is straightforward to observe that if we choose two input program valuations $\mathbf{b}, \mathbf{b}'$ such that $\mathbf{b}[x] = n, \mathbf{b}'[x] = n - \epsilon$ and $\mathbf{b}[y] = \mathbf{b}'[y] = n$, where n is a natural number that can be arbitrarily large and ϵ is a positive real number that can be arbitrarily close to zero (so that $\mathbf{b} \models \Phi$ and $\mathbf{b}' \not\models \Phi$), then the expected affine-sentivity does not hold. The reason is that the execution from $\mathbf{b}$ enters the loop and ends with -1 for y, and that from $\mathbf{b}'$ does not enter the loop and hence keeps its input value n.

Example 5.6 (The Running Example). Consider our running example in Figure 1. We choose the sampling variable r to observe the Bernoulli distribution $\mathbb{P}(r = 0) = \mathbb{P}(r = 1) = \frac{1}{2}$ and the metric $\mathfrak{d}$ as the max norm. Then we can construct an RSM-map $\eta(x) = 1000 - x$ with $\epsilon = \frac{1}{2}, K = -1$. Moreover, the RSM-map η has the RSM-continuity with $M = 1$, and the loop has bounded update with $d = 1$. Hence by Theorem 5.5, the loop is expected affine-sensitive over its loop guard.

Example 5.7 (Mini-roulette). We show that the Mini-roulette example in Figure 4(left) is expected affine-sensitive in the program variable w over its loop guard. To show this, we construct the function $\eta(x, w) = 13 \cdot x - 13$ with $\epsilon = 1, K = -13$. We also clarify the following points.

(1) For any values x_1, x_2 to the program variable x before a loop iteration and any $\ell \in \mathbf{L}$ that resolves the probabilistic branches, we have that $|(x_1 + a) - (x_2 + a)| = |x_1 - x_2|$ after the loop iteration where the value of a is determined by the probabilistic branch (i.e for branch 5, $a = 11$). The same applies to the program variable w. Thus the loop is non-expansive.
(2) All increments to x and w are bounded, hence the loop has bounded update, which ensures (B2).
(3) The loop guard $x \geq 1$ implies $\eta(x, w) = 13 \cdot x - 13 \geq 0$, thus (A1) is satisfied. When $x \geq 1$, $\ell \in \mathbf{L}$ and $F(\ell, (x, w), -) < 1$, we have $-13 \leq \eta(F(\ell, (x, w), -)) \leq 0$, ensuring (A2). When $x \geq 1$, $\ell \in \mathbf{L}$, we have $\mathbb{E}_\ell(\eta(F(\ell, (x, w), -))) \leq \eta(x, w) - 1$. Thus η is an RSM-map.
(4) Given any values $x_1, x_2 \geq 1$ and w_1, w_2 to the program variables x, w, we have $|\eta(x_1, w_1) - \eta(x_2, w_2)| = 13 \cdot |x_1 - x_2|$. Thus η has *RSM-continuity*.

By Theorem 5.5, we obtain that the program is expected affine-sensitive over its loop guard.

Remark 4. For proving expected affine sensitivity, one can relax the RSM-continuity to the condition that $\exists C > 0. \forall \mathbf{b}, \mathbf{b}'. [(F(\ell, \mathbf{b}, \mathbf{r}) \models \Phi \wedge F(\ell, \mathbf{b}', \mathbf{r}) \not\models \Phi) \Rightarrow \eta(\mathbf{b}) \leq C]$, so that the difference in non-synchronous situations is guaranteed to be bounded by C.

Remark 5 (Conditional Branches). In certain scenarios, it is possible to extend Theorem 5.5 to conditional branches at which it may happen that one program valuation satisfies the condition of the branch and another close-by valuation does not. Consider a scenario where we are to use piecewise-linear functions to approximate a complex loop body. If the approximation is sufficiently strong to ensure that neighbouring pieces of functions behave similarly, then our approach can handle the scenario as follows. We first relax the Lipschitz continuity in Definition 2.1 to "$\leq L \cdot \mathfrak{d}(\mathbf{b}, \mathbf{b}') + V$" where V is a nonnegative constant that bounds the difference between neighbouring pieces of functions in each loop iteration (which is small for sufficiently strong approximation). This ensures that the final sensitivity would be $A \cdot \mathfrak{d}(\mathbf{b}, \mathbf{b}') + (B + V \cdot \mathbb{E}(T))$ in the non-expansive case where $\mathbb{E}(T)$ is the expected termination time of the loop (that depends on the initial input).

5.2 Proving Expected Linear-Sensitivity

To develop a sound approach for proving expected linear-sensitivity, one possible way is to extend the approach for expected affine-sensitivity. However, simply extending the approach is not correct, as is shown by the following example.

Example 5.8. Consider our running example in Figure 1. We first consider that the sampling variable r observes the Dirac distribution such that $\mathbb{P}(r = 1) = 1$, the same as in Example 5.1. By choosing the same initial values x_1^* and x_2^* from Example 5.1, we have that the outcomes satisfy $|x_1^{\text{out}} - x_2^{\text{out}}| = 1 - 2 \cdot \epsilon$. Hence, we could not find a constant A such that $|x_1^{\text{out}} - x_2^{\text{out}}| \leq A \cdot |x_1^* - x_2^*| = 2 \cdot A \cdot \epsilon$ when $\epsilon \longrightarrow 0$. Similar situation happens even if we have non-Dirac discrete probability distributions. For example, consider now that the sampling variable r observes the distribution such that $\mathbb{P}(r = 0) = \mathbb{P}(r = 1) = 0.5$. Then with the same initial values x_1^* and x_2^*, as the increment to the program variable x is either 0 or 1, we have the same outputs $x_1^{\text{out}}, x_2^{\text{out}}$, refuting expected linear-sensitive.

The reason why we have such a situation in Example 5.8 is again due to the non-synchronous situation where the number of loop iterations depends on the input program valuation. While

proving expected affine-sensitivity we can use a constant B (cf. Definition 3.1) to bound the difference caused by non-synchronous situations, in proving linear-sensitivity we need to set $B = 0$, leading to a difficulty that cannot be resolved by the technique developed for affine-sensitivity. To address this issue, we introduce another Lipschitz continuity w.r.t a given metric $\mathfrak{d}$.

Definition 5.9 (Lipschitz Continuity in Next-step Termination L'). We say that a simple while loop Q in the form (1) is *Lipschitz continuous in next-step termination* if there is a constant $L' > 0$ such that

(B4) $\forall \ell\, \forall \mathbf{b}, \mathbf{b}' : (\mathbf{b}, \mathbf{b}' \models \Phi \Rightarrow \mathbb{P}_{\mathbf{r}}(F(\ell, \mathbf{b}, \mathbf{r}) \models \Phi \wedge F(\ell, \mathbf{b}', \mathbf{r}) \models \neg\Phi) \leq L' \cdot \mathfrak{d}(\mathbf{b}, \mathbf{b}'))$

where given the program valuations $\mathbf{b}, \mathbf{b}'$ before the loop iteration and the resolution ℓ for the probabilistic branches, the value $\mathbb{P}_{\mathbf{r}}(F(\ell, \mathbf{b}, \mathbf{r}) \models \Phi \wedge F(\ell, \mathbf{b}', \mathbf{r}) \models \neg\Phi)$ is the probability regarding the sampled values that after one loop iteration we have $F(\ell, \mathbf{b}, \mathbf{r})$ can still enter the loop, while $F(\ell, \mathbf{b}', \mathbf{r})$ violates the loop guard.

The condition (B4) specifies that when the program valuations $\mathbf{b}, \mathbf{b}'$ are close, the probability that after the current loop iteration one of them stays in the loop while the other jumps out of the loop is small as it is proportional to the distance between $\mathbf{b}, \mathbf{b}'$. This condition handles the non-synchronous situation in the sense that the probability of non-synchronous situations is bounded linearly by the distance between the program valuations before a loop iteration. For simple while loops with only *discrete* probability distributions, this condition is usually not met. This is because in discrete probability distributions there often exists a vector $\mathbf{r}$ of sampled values with a minimum probability $p > 0$ that $F(\ell, \mathbf{b}, \mathbf{r}) \models \Phi$ and $F(\ell, \mathbf{b}', \mathbf{r}) \not\models \Phi$. In some cases such probability p may be very large, e.g., in our running example (Example 5.8), with $\mathbb{P}(r = 1) = 1$ we have that $\mathbb{P}_r(F(\ell, 999 - \epsilon, r) \models \Phi \wedge F(\ell, 999 + \epsilon, r) \models \neg\Phi) = 1$, and with $\mathbb{P}(r = 0) = \mathbb{P}(r = 1) = \frac{1}{2}$ we have that the same probability is $\frac{1}{2}$, when $\epsilon \to 0$. In contrast, loops with *continuous* distributions often satisfy this condition. For example, consider again our running example where r now observes the uniform distribution over the interval $[0, 1]$. Then for any initial values $x'' \leq x' \leq 1000$ for the program variable x, the probability that $x' + r > 1000$ but $x'' + r \leq 1000$ equals the chance that the sampled value of r falls in $(1000 - x', 1000 - x'']$, which is no greater than $|x - x'|$ as the probability density function of r is 1 over the interval $[0, 1]$.

In the following, we show that a large class of *affine* simple while loops with continuous distributions guarantees the (B4) condition. Below we say that an update function F is *affine* if for all $\ell \in \mathbf{L}$, we have that $F(\ell, \mathbf{b}, \mathbf{r}) = \mathbf{B} \cdot \mathbf{b} + \mathbf{C} \cdot \mathbf{r} + \mathbf{c}$ for constant matrices $\mathbf{B}, \mathbf{C}$ and vector $\mathbf{c}$. Moreover, a boolean expression Φ is said to be *affine* if Φ can be equivalently rewritten into a disjunctive normal form (DNF) $\bigvee_{i \in \mathcal{I}} (\mathbf{A}_i \cdot \mathbf{b} \leq \mathbf{d}_i)$ with constant matrices $\mathbf{A}_i$ and vectors $\mathbf{d}_i$ so that for all program valuations $\mathbf{b}$, we have $\mathbf{b} \models \Phi$ iff the disjunctive formula $\bigvee_{i \in \mathcal{I}} (\mathbf{A}_i \cdot \mathbf{b} \leq \mathbf{d}_i)$ holds. The class of simple while loops that guarantees (B4) is as follows.

Lemma 5.10. *Consider a simple while loop Q in the form (1) that satisfies the following conditions:*

(1) both F and Φ are affine and Φ is equivalent to some DNF $\bigvee_{i \in \mathcal{I}} (\mathbf{A}_i \cdot \mathbf{b} \leq \mathbf{d}_i)$;
(2) all sampling variables are continuously-distributed whose probability density functions have bounded values;
(3) for all $i \in \mathcal{I}$, $\ell \in \mathbf{L}$ and program valuations $\mathbf{b} \models \Phi$, the coefficients for the sampling variables $\mathbf{r}$ in $\mathbf{A}_i \cdot F(\ell, \mathbf{b}, \mathbf{r})$ are not all zero at each row, i.e., the truth value of each disjunctive clause in Φ for $F(\ell, \mathbf{b}, \mathbf{r})$ depends on $\mathbf{r}$ at every row.

Then the loop Q is Lipschitz continuous in next-step termination w.r.t any metric $\mathfrak{d}$.

Informally, the lemma guarantees the (B4) condition by requiring that (i) both the update function and the loop guard are affine, (ii) all sampling variables are continuously-distributed, and (iii) the truth value of every linear inequality in the loop guard after the current loop iteration depends on the sampled values. The proof of Lemma 5.10 is elementary, see [Wang et al. 2019a, Appendix D].

Remark 6. We note that Lemma 5.10 serves as a sound condition only, and there are situations where the prerequsite of Lemman 5.10 fails but the condition (B4) still holds. For example, consider that a program variable x is assigned to 1 every time in a loop iteration, and the loop guard involves the condition $x \leq 2$. Then the condition $x \leq 2$ does not affect the truth value of the loop guard since it is always satisfied, but has zero coefficients for all sampling variables. While to derive weaker conditions is possible, in this work we consider Lemma 5.10 as a simple guarantee for ensuring (B4).

Now we demonstrate our approach for proving expected linear-sensitivity. Our first result is a sound approach for proving *local* linear-sensitivity. Below given a program valuation $\mathbf{b} \models \Phi$, a radius $\rho > 0$ and a metric $\mathfrak{d}$, we denote by $U_{\Phi,\mathfrak{d}}(\mathbf{b}, \rho)$ the neighbourhood $\{\mathbf{b}' \mid \mathbf{b}' \models \Phi \wedge \mathfrak{d}(\mathbf{b}, \mathbf{b}') \leq \rho\}$.

Proposition 5.11. *A non-expansive simple while loop Q in the form (1) has expected linear-sensitivity over some neighbourhood $U_{\Phi,\mathfrak{d}}(\mathbf{b}^*, \rho)$ of any given $\mathbf{b}^* \in [\![\Phi]\!]$ if Q has (i) bounded update, (ii) an RSM-map with RSM-continuity and (iii) the* Lipschitz continuity in next-step termination.

The proof resembles the one for expected affine-sensitivity (Theorem 5.5). The obtained coefficient A (cf. Definition 3.1) depends on the expected termination time from the input program valuation $\mathbf{b}^*$. See [Wang et al. 2019a, Appendix D] for details.

Proposition 5.11 gives a sound approach for proving *local* linear-sensitivity in that the coefficient A only works for a small neighbourhood of a given input. A natural question arises whether we can obtain *global* linear-sensitivity so that the coefficient A works for all program valuations that satisfy the loop guard. The major barrier in the proof of Proposition 5.11 to obtain global linear sensitivity is that in general we can only treat every program valuation uniformly, without distinguishing between program valuations with large and small RSM-map values. To overcome this difficulty, we partition program valuations into *finitely* many classes so that each class shares a common coefficient, but different classes may have different coefficients. Based on the partition, we utilize the inter-relationship between different classes to prove the existence of a collection of coefficients for the expected linear-sensitivity. The partition relies on the *difference-bounded* condition for RSM-maps proposed for proving concentration properties of termination time [Chatterjee et al. 2018c].

Definition 5.12 (The Difference-bounded Condition [Chatterjee et al. 2018c]). We say that an RSM-map η (for a simple while loop) is *difference-bounded* if there exists a constant $c \geq 0$ such that

(A4) $\forall \mathbf{b}\, \forall \ell\, \forall \mathbf{r} : (\mathbf{b} \models \Phi \Rightarrow |\eta(F(\ell, \mathbf{b}, \mathbf{r})) - \eta(\mathbf{b})| \leq c)$.

The difference-bounded condition ensures the concentration property for program termination [Chatterjee et al. 2018c], see Theorem E.1 in [Wang et al. 2019a, Appendix E]. Below we demonstrate how this condition helps to partition the program valuations into finitely-many regions by their corresponding RSM-map values. First we show that this condition derives a minimum positive probability that the value of an RSM-map decreases by a minimum positive amount.

Lemma 5.13. *If η is a difference-bounded RSM-map with the parameters ϵ, c specified in Definition 5.2 and Definition 5.12, then there exists a constant $p \in (0, 1]$ such that*

$(\dagger)$ $\forall \mathbf{b} : \left(\mathbf{b} \models \Phi \Rightarrow \mathbb{P}_{\mathbf{r},\ell}(\eta(F(\ell,\mathbf{b},\mathbf{r})) - \eta(\mathbf{b}) \leq -\frac{1}{2}\cdot\epsilon) \geq p\right)$

where the probability $\mathbb{P}_{\mathbf{r},\ell}(-)$ is taken w.r.t the sampled valuation $\mathbf{r}$ and the resolution ℓ for probabilistic branches, and treats the program valuation $\mathbf{b}$ as a constant vector. In particular, we can take $p := \frac{\epsilon}{2\cdot c-\epsilon}$.

Proof Sketch. The proof is by Markov's inequality. See [Wang et al. 2019a, Appendix D]. □

The finite partition. Based on Lemma 5.13, we partition the satisfaction set $[\![\Phi]\!]$ into finitely many regions. Our aim is to have a partition $R_1, \ldots, R_{n^*}, R_\infty$ based on which we find individual sensitivity coefficients on each R_k. First we choose the smallest natural number n^* such that $(n^*-1)\cdot\frac{1}{2}\cdot\epsilon \leq c+1 < n^*\cdot\frac{1}{2}\cdot\epsilon$. (Note that $n^* \geq 3$ as $\epsilon \leq c$.) Then we define the region $R_k := \{\mathbf{b} \in [\![\Phi]\!] \mid (k-1)\cdot\frac{1}{2}\cdot\epsilon \leq \eta(\mathbf{b}) < k\cdot\frac{1}{2}\cdot\epsilon\}$ for natural numbers $1 \leq k \leq n^*$, and $R_\infty := \{\mathbf{b} \in [\![\Phi]\!] \mid \eta(\mathbf{b}) \geq n^*\cdot\frac{1}{2}\cdot\epsilon\}$. It follows that $[\![\Phi]\!]$ is a disjoint union of all R_k's. Especially, we treat R_∞ as the region for program valuations with "large enough" RSM-map values. After the partitioning, we are to prove that for each R_k there is a coefficient A_k for Definition 3.1, and for different R_k's there may be different A_k's. The following result presents the first step of the proof, where each A_k $(1 \leq k \leq n^*)$ represents the coefficient for R_k and A_∞ for R_∞.

Proposition 5.14. *For any natural number $n \geq 1$, real numbers $C, D \geq 0$ and probability value $p \in (0,1]$, the following system of linear inequalities (with real variables A_k's $(0 \leq k \leq n)$ and A_∞) $(1-p)\cdot A_\infty + C + p\cdot A_0 \leq A_1, \ldots, (1-p)\cdot A_\infty + C + p\cdot A_{n-1} \leq A_n, D = A_0 \leq A_1 \leq \cdots \leq A_n \leq A_\infty$ has a solution.*

Proof Sketch. First, we equate all inequalities but $D = A_0 \leq \cdots \leq A_n \leq A_\infty$, so that we directly get the solution as follows:

- $A_\infty = \frac{1}{p^{n+1}}\cdot(\sum_{m=1}^{n+1} p^{m-1})\cdot C + D = \frac{1}{p^{n+1}}\cdot\frac{1-p^{n+1}}{1-p}\cdot C + D$,
- $A_k = (\frac{1-p^k}{p^{n+1}}\cdot(\sum_{m=1}^{n+1} p^{m-1}) + (\sum_{m=1}^{k} p^{m-1}))\cdot C + D = (\frac{1-p^k}{p^{n+1}}\cdot\frac{1-p^{n+1}}{1-p} + \frac{1-p^k}{1-p})\cdot C + D$ for $1 \leq k \leq n$.

Then we check that $D = A_0 \leq \cdots \leq A_n \leq A_\infty$ holds. See [Wang et al. 2019a, Appendix D]. □

Now we state our main result for proving global linear sensitivity on non-expansive simple loops.

Theorem 5.15. *A non-expansive simple while loop Q in the form (1) has expected linear-sensitivity over its loop guard $[\![\Phi]\!]$ if Q has (i) bounded update, (ii) a difference-bounded RSM-map with RSM-continuity and (iii) the Lipschitz continuity in next-step termination. In particular, we can choose $\theta = \frac{1}{M}$ in (3) where the parameter M is from the RSM-continuity (Definition 5.4).*

Proof Sketch. Choose any program variable z. Denote by T, T' (resp. Z_n, Z'_n) the random variables for the number of loop iterations (resp. the value of z at the n-th step), from two close-by input program valuations $\mathbf{b}, \mathbf{b}'$, respectively. For each natural number $n \geq 0$, we define $\delta_n(\mathbf{b},\mathbf{b}') := \mathbb{E}_{\mathbf{b}}(Z_{T\wedge n}) - \mathbb{E}_{\mathbf{b}'}(Z'_{T'\wedge n})$, where the random variable $T\wedge n$ is defined as $\min\{T, n\}$ and $T'\wedge n$ likewise. We also define $\delta(\mathbf{b},\mathbf{b}') := \mathbb{E}_{\mathbf{b}}(Z_T) - \mathbb{E}_{\mathbf{b}'}(Z'_{T'})$. First, we prove from the Dominated Convergence Theorem that $\lim_{n\to\infty}\delta_n(\mathbf{b},\mathbf{b}') = \delta(\mathbf{b},\mathbf{b}')$. Second, given a difference-bounded RSM-map η with RSM-continuity, we construct the regions R_k's $(1 \leq k \leq n^*)$ and R_∞ as in the paragraph below Lemma 5.13, and solve A_k's and A_∞ from Proposition 5.14. Third, based on the solved A_k's and A_∞, we prove by induction on $n \geq 0$ that for all $k \in \{1, \ldots, n^*, \infty\}$ and all program valuations $\mathbf{b}, \mathbf{b}'$, we have $\delta_n(\mathbf{b},\mathbf{b}') \leq A_k\cdot\mathfrak{d}(\mathbf{b},\mathbf{b}')$ when $\mathbf{b}, \mathbf{b}' \models \Phi$ and $\mathbf{b} \in R_k$. In the inductive proof, (i) we apply Lemma 5.13

to tackle the regions R_k $(1 \leq k \leq n^*)$ and use the inequality $(1-p) \cdot A_\infty + C + p \cdot A_{k-1} \leq A_k$ from Proposition 5.14 to prove the inductive case, and (ii) for R_∞ we ensure the fact that starting from two close-by program valuations in R_∞, the loop will not terminate after the current loop iteration from both the program valuations, as is guaranteed by (A1), (A2), (A4) and the RSM-continuity. Finally, the result follows from taking the limit $n \to \infty$ and the fact that we have finitely many regions. The detailed proof is put in [Wang et al. 2019a, Appendix D]. □

Remark 7. For expected linear sensitivity, one can relax the RSM-continuity as follows. First, we require the relaxed condition in Remark 4 to tackle the non-synchronous situation. Second, we need the relaxed condition $\exists D > 0. \forall \mathbf{b}, \mathbf{b}'. [(\mathbf{b}, \mathbf{b}' \models \Phi \wedge \mathfrak{d}(\mathbf{b}, \mathbf{b}') \leq \theta \wedge \eta(\mathbf{b}) > D) \Rightarrow \eta(\mathbf{b}') > c]$, so that the neighbourbood around $\mathbf{b}$ w.r.t the threshold θ will not lead to termination after one loop iteration.

Example 5.16. We now show that the mini-roulette variant in Figure 4(right) is expected linear-sensitive in the program variable w over its loop guard. To show this, we construct the function $\eta(x, w) = 2.45 \cdot x - 2.45$ with $\epsilon = 1, K = -4.91$. We also clarify the following points.

(1) For any values x_1, x_2 to the program variable x before a loop iteration and any $\ell \in \mathbf{L}$ that resolves the probabilistic branches, we have that $|(x_1 + r_i) - (x_2 + r_i)| = |x_1 - x_2|$ after the loop iteration where the value of r_i is decided by the executed branch and its distribution(i.e for branch 5, $r_i := r_5 \sim unif(8, 9)$). The same applies to the program variable w. Thus the loop body is non-expansive.
(2) All increments to x and w are bounded, hence the loop has bounded update, which ensures (B2).
(3) The loop guard $x \geq 1$ implies $\eta(x, w) = 2.45 \cdot x - 2.45 \geq 0$, thus (A1) is satisfied. When $x \geq 1$, $\ell \in \mathbf{L}$ and $F(\ell, (x, w), -) < 1$, we have $-4.91 \leq \eta(F(\ell, (x, w), -)) \leq 0$, ensuring (A2). When $x \geq 1$, $\ell \in \mathbf{L}$, we have $\mathbb{E}_{\mathbf{r}, \ell}(\eta(F(\ell, (x, w), \mathbf{r}))) \leq \eta(x, w) - 1$, ensuring (A3). Thus, η is an RSM-map.
(4) Given any values $x_1, x_2 \geq 1$ and w_1, w_2 to the program variables x, w, we have $|\eta(x_1, w_1) - \eta(x_2, w_2)| = 2.45 \cdot |x_1 - x_2|$. Thus η has *RSM-continuity*.
(5) When $x \geq 1$, we have $|\eta(F(\ell, (x, w), \mathbf{r})) - \eta(x, w)| \leq |\eta(F(\ell, (x, w), r_5)) - \eta(x, w)| = |2.45 \cdot (x + r_5) - 2.45 - 2.45 \cdot x + 2.45| \leq 2.45 \cdot 9 = 22.05$, ensuring (A4). Thus, η is *difference-bounded*.
(6) Due to the fact that both the update function and the loop guard are affine, all sampling variables are bounded continuously-distributed, and the coefficients for the current sampling variables are not all zero in the loop guard of the next iteration, we can verify that the loop has the Lipschitz continuity in next-step termination by Lemma 5.10.

Then by Theorem 5.15, we can conclude that this probabilistic program is expected linear-sensitive over its loop guard.

6 PROVING EXPECTED SENSITIVITY FOR EXPANSIVE SIMPLE WHILE LOOPS

In this section, we show how our sound approach for proving expected sensitivity of non-expansive loops can be extended to expansive simple while loops. We first illustrate the main difficulty, and then enhance RSM-maps to be difference-bounded and show how they can address the difficulty.

The main difficulty to handle expansive loops is that the difference between two program valuations may tend to infinity as the number of loop iterations increases. For example, consider a simple while loop where at every loop iteration (i) the value of a program variable z is tripled and (ii) the loop terminates immediately after the current loop iteration with probability $\frac{1}{2}$. Then given two different initial values z', z'' for z, we have that

$$\mathbb{E}_{z'}(Z') - \mathbb{E}_{z''}(Z'') = \sum_{n=1}^{\infty} \mathbb{P}(T = n) \cdot 3^n \cdot |z' - z''| = \sum_{n=1}^{\infty} \left(\frac{3}{2}\right)^n \cdot |z' - z''| = \infty.$$

where Z', Z'' are given by the same way of Z, Z' as in (3) and T is the termination time random variable. Thus the expected-sensitivity properties do not hold for this example, as the increasing speed of z is higher than that for program termination. To cope with this point, we consider again RSM-maps to be difference-bounded, as in Definition 5.12. The main idea is to use the exponential decrease from difference-bounded RSM-maps (Theorem E.1 in [Wang et al. 2019a, Appendix E]) to counteract the unbounded increase in the difference between input program valuations.

Below we illustrate the main result of this section. Recall that given a program valuation $\mathbf{b}$, a radius $\rho > 0$ and a metric $\mathfrak{d}$, we denote by $U_{\Phi,\mathfrak{d}}(\mathbf{b}, \rho)$ the neighbourhood $\{\mathbf{b}' \mid \mathbf{b}' \models \Phi \wedge \mathfrak{d}(\mathbf{b}, \mathbf{b}') \leq \rho\}$.

Theorem 6.1. *Consider a simple while loop Q in the form (1) that satisfies the following conditions:*

- *the loop body P is Lipschitz continuous with a constant L specified in Definition 2.1, and has bounded update;*
- *there exists a difference-bounded RSM-map η for Q with RSM-continuity and parameters ϵ, K, c from Definition 5.2 and Definition 5.12 such that $L < \exp(\frac{3\cdot\epsilon^2}{8\cdot c^2})$.*

Then for any program valuation $\mathbf{b}^*$ such that $\mathbf{b}^* \models \Phi$ and $\eta(\mathbf{b}^*) > 0$, there exists a radius $\rho > 0$ such that the loop Q is expected affine-sensitive over $U_{\Phi,\mathfrak{d}}(\mathbf{b}^*, \rho)$. In particular, we can choose in Definition 3.1 that

$$A := 2 \cdot A' \cdot L^N + 2 \cdot A' \cdot L^N \cdot \exp\left(-\frac{\epsilon\cdot\eta(\mathbf{b}^*)}{8\cdot c^2}\right) \cdot \sum_{n=1}^{\infty} \left(L \cdot \exp\left(-\frac{3\cdot\epsilon^2}{8\cdot c^2}\right)\right)^n$$

$$B := 2 \cdot B' + 2 \cdot B' \cdot \exp\left(-\frac{\epsilon\cdot\eta(\mathbf{b}^*)}{8\cdot c^2}\right) \cdot \sum_{n=1}^{\infty} \exp\left(-\frac{3\cdot\epsilon^2}{8\cdot c^2} \cdot n\right)$$

where $A' = \frac{d\cdot M+\epsilon}{D_1\cdot\epsilon}$, $B' = -\frac{d\cdot K}{D_1\cdot\epsilon}$ and $N = \lfloor 4 \cdot \frac{\eta(\mathbf{b}^*)}{\epsilon} \rfloor + 1$, for which the parameters d, M, ϵ, K, D_1 are from Definition 5.2, Definition 5.3, Definition 5.4 and (2).

The proof resembles the one for Theorem 5.5 and compares L with the exponential-decreasing factor $\exp(\frac{3\cdot\epsilon^2}{8\cdot c^2})$, see [Wang et al. 2019a, Appendix E] for the detailed proof. Note that in the statement of the theorem we do not care for θ, this is because we have already restricted the threshold to the neighbourhood $U_{\Phi,\mathfrak{d}}(\mathbf{b}^*, \rho)$.

Theorem 6.1 cannot be directly extended to linear sensitivity as the technique to derive linear sensitivity (e.g. Theorem 5.15) requires non-expansiveness as an important prerequisite. We leave a more detailed investigation of the expansive case (including the linear sensitivity) as a future work.

Summary of Prerequisites for Expected Sensitivity. In Table 1, we summarize the prerequisites for expected sensitivity. The first column specifies the program type (i.e. non-expansive/expansive loops), the second column specifies the sensitivity type (i.e. expected affine/linear-sensitive) for the program, the third column specifies the related theorem for this expected sensitivity of the program, and the last column contains all the prerequisites of this expected sensitivity.

Remark 8. All our results cover the degenerate case where the number of loop iterations is fixed and bounded. To see this, suppose that the number of loop iterations is fixed to be n, and there is a program variable i that serves as the loop counter. Then we can choose $n - i$ as a (difference-bounded) RSM-map that is independent of the program variables other than i, so that our result for expected affine-sensitivity (Theorem 5.5) holds directly for this degenerate case. Furthermore, the condition (B4) is satisfied directly as the termination depends only on the loop counter i, so that our linear-sensitivity result (Theorem 5.15) holds for the degenerate case; for expansive loops, we even do not need to check whether $L < \exp(\frac{3\cdot\epsilon^2}{8\cdot c^2})$ in Theorem 6.1 as the number of loop iterations is bounded.

Table 1. Overview of Expected Sensitivity Results and Their Prerequisites

Program	Type	Theorem	Prerequisites
non-expansive	expected affine-sensitivity	Theorem 5.5	basic prerequisites*
non-expansive	expected linear-sensitivity	Theorem 5.15	basic prerequisites* additional prerequisites**
expansive	expected affine-sensitivity	Theorem 6.1	$L < \exp(\frac{3\cdot\epsilon^2}{8\cdot c^2})$ (Definition 2.1) basic prerequisites* difference-bounded condition (Definition 5.12)

* basic: bounded update (Definition 5.3), RSM-map (Definition 5.2), RSM-continuity (Definition 5.4)
** additional: difference-bounded condition (Definition 5.12), Lipschitz continuity in next-step termination (Definition 5.9)

7 SEQUENTIAL COMPOSITION OF SIMPLE WHILE LOOPS

In this section, we demonstrate the compositionality of our martingale-based approach for proving expected sensitivity of probabilistic programs. We follow the previous work [Barthe et al. 2018] to consider the sequential composition of probabilistic programs. We show that under the same side condition from [Barthe et al. 2018], our approach is compositional under sequential composition.

We first show that the compositionality under sequential composition does not hold in general. The main point is that if the output range of a preceding program Q does not match the input range over which the latter program Q' is expected sensitive, then the global expected sensitivity for the sequential composition $Q; Q'$ may not hold. A detailed example is as follows.

Example 7.1. Consider the sequential composition $Q =$ **skip**$; Q'$ where Q' is the simple while loop from Remark 3. We know that **skip** is expected sensitive over all input program valuations. From Theorem 5.5 and Remark 3, we have Q' is expected affine-sensitive only over its loop guard. Then the program Q is not expected affine-sensitive over all input program valuations, as one can choose two input program valuations such that one satisfies the loop guard of Q' but the other does not.

Thus, in order to ensure compositionality, we need to require that the output range of the preceding program should match the input sensitivity range of the latter program, as is also required in [Barthe et al. 2018]. Under this side condition, we prove that our approach is compositional over sequential composition of non-expansive simple while loops. Below for a probabilistic program Q, we denote by $\mathsf{out}(Q)$ the set of all possible outcome program valuations after the execution of Q under some input program valuation in the satisfaction set of its loop guard.

Theorem 7.2. *Consider a non-expansive simple while loop Q with bounded-update and an RSM-map with RSM-continuity, and a general program Q' that has expected affine-sensitivity over a subset U of input program valuations with threshold θ in (3). If $\mathsf{out}(Q) \subseteq U$ and assuming integrability in (3), then the sequential composition $Q; Q'$ is expected affine-sensitive over the satisfaction set of the loop guard of Q with threshold θ.*

Proof Sketch. The proof is basically an extension to the previous proof for Theorem 5.5. We consider the same three cases from the previous proof, and use the expected affine-sensitivity from Q' to derive the new sensitivity coefficients. See [Wang et al. 2019a, Appendix F] for details. □

Theorem 7.2 presents a general compositional result where the program Q' can be an arbitrary probabilistic program. By extending the proof for Theorem 5.15 in a similar way as from Theorem 5.5

to Theorem 7.2, we can also derive a compositional result for expected linear-sensitivity. However, we now need to consider Q' as a sequential composition of simple while loops and impose *linearity* on the RSM-maps. (An RSM-map is *linear* if it can be expressed as a linear combination of program variables and a possible constant term.) Then for a sequential composition $Q = Q_1; \ldots; Q_n$ of simple while loops, we require the condition (‡) that (i) each Q_i having bounded update and a *linear* RSM-map η_i that witnesses its expected linear-sensitivity (i.e., that satisfies the conditions (A1) – (A4).) and (ii) $\mathrm{out}(Q_i) \subseteq [\![\Phi_{i+1}]\!]$ for all i, where Φ_{i+1} is the loop guard of Q_{i+1}. By extending the proof for Theorem 5.15 (see [Wang et al. 2019a, Appendix F]), we establish the following theorem.

Theorem 7.3. *Consider a non-expansive simple while loop Q with loop guard Φ that has (i) bounded-update, (ii) a difference-bounded linear RSM-map with RSM-continuity, and (iii) the Lipschitz continuity in next-step termination. Then for any sequential composition Q' of simple while loops that (a) satisfies the condition (‡) (defined right before the theorem) and (b) has expected linear-sensitivity over a subset U of input program valuations, if $[\![\Phi]\!] \cup \mathrm{out}(Q) \subseteq U$, then the sequential composition $Q; Q'$ is expected linear-sensitive over the satisfaction set of the loop guard of Q.*

By an iterated application of Theorem 7.2 and Theorem 7.3 (i.e., loop-by-loop), we obtain directly the compositionality over sequential composition of non-expansive simple while loops.

Remark 9 (Expansive Loops). Up till now we only consider non-expansive loops. The main issue arising from expansive loops is that the expected sensitivity is restricted to a small neighbourhood of a fixed input program valuation, and the sensitivity coefficients often depend on the input program valuation. More precisely, these coefficients may be exponential in general (see Theorem 6.1). Thus, compositionality for expansive loops depends on the exact post probability distribution after the execution of the preceding loops. To overcome this difficulty, new techniques need to be developed and we plan it as a future work.

Remark 10 (Comparison with [Barthe et al. 2018]). A similar compositional result is established in [Barthe et al. 2018, Proposition 4.3] for loops with a fixed number of loop iterations. The approach is similar to ours as it also considers sequential composition and requires that the output range of the preceding loop should match the input sensitivity range of the latter loop, and treats each individual program in the sequential composition separately. The only difference is that they prove directly that their coupling-based sensitivity has the compositional property regardless of the detailed program structure, while our approach requires an explicit RSM-map for each loop. This is however due to the fact that our approach considers the more complex situation that the loop iterations are randomized and depend on the input program valuation.

Remark 11 (Compositionality). We would like to note that the level of compositionality depends on the *side condition* in an approach. Some authors insist that compositionality should require no side condition, while other authors allow side conditions [Kupferman and Vardi 1997]. Our approach, like the approach in [Barthe et al. 2018], has the least side condition, as we only require that the output range of the preceding program matches the input sensitivity range of the latter. Our result is also different from the original one in [Barthe et al. 2018] as in our case we need to tackle the non-trivial point of non-synchronicity (see Example 5.1).

8 AN AUTOMATED APPROACH THROUGH RSM-SYNTHESIS ALGORITHMS

In this section, we describe an automated algorithm that, given a non-expansive probabilistic loop Q in the form (1), synthesizes an RSM-map with extra conditions required for proving expected sensitivity. We consider affine programs whose loop guard and update function are affine, and linear templates for an RSM-map. Our algorithm runs in polynomial time and reduces the problem

of RSM-map synthesis to linear programming by applying Farkas' Lemma. We strictly follow the framework from previous synthesis algorithms [Chakarov and Sankaranarayanan 2013; Chatterjee et al. 2016, 2018a,c, 2017; Feng et al. 2017; Wang et al. 2019b]. As our synthesis framework is not novel, we describe only the essential details of the algorithm. Below we first recall Farkas' Lemma.

Theorem 8.1 (Farkas' Lemma [Farkas 1894]). *Let* $\mathbf{A} \in \mathbb{R}^{m\times n}$, $\mathbf{b} \in \mathbb{R}^m$, $\mathbf{c} \in \mathbb{R}^n$ *and* $d \in \mathbb{R}$. *Suppose that* $\{\mathbf{x} \in \mathbb{R}^n \mid \mathbf{A}\mathbf{x} \le \mathbf{b}\} \neq \emptyset$. *Then* $\{\mathbf{x} \in \mathbb{R}^n \mid \mathbf{A}\mathbf{x} \le \mathbf{b}\} \subseteq \{\mathbf{x} \in \mathbb{R}^n \mid \mathbf{c}^{\mathrm{T}}\mathbf{x} \le d\}$ *iff there exists* $\mathbf{y} \in \mathbb{R}^m$ *such that* $\mathbf{y} \ge \mathbf{0}$, $\mathbf{A}^{\mathrm{T}}\mathbf{y} = \mathbf{c}$ *and* $\mathbf{b}^{\mathrm{T}}\mathbf{y} \le d$.

Intuitively, Farkas' Lemma transforms the inclusion problem of a nonempty polyhedron within a halfspace into a feasibility problem of a system of linear inequalities. As a result, one can decide the inclusion problem in polynomial time through linear programming.

The RSM-synthesis Algorithm. Our algorithm has the following four steps:

(1) *Template.* The algorithm sets up a column vector $\mathbf{a}$ of $|V_{\mathrm{p}}|$ fresh variables and a fresh scalar variable b such that the template for an RSM-map η is $\eta(\mathbf{b}) = \mathbf{a}^{\mathrm{T}} \cdot \mathbf{b} + b$. Note that since we use linear templates, the RSM-continuity condition is naturally satisfied.
(2) *Constraints on* $\mathbf{a}$ *and* b. The algorithm first encodes the condition (A1) for the template η as the inclusion assertion $\{\mathbf{b} \mid \mathbf{b} \models \Phi\} \subseteq \{\mathbf{b} \mid \mathbf{c}_1^{\mathrm{T}} \cdot \mathbf{b} \le d_1\}$ where $\mathbf{c}_1, d_1$ are unique linear combinations of unknown coefficients $\mathbf{a}, b$ satisfying that $\mathbf{c}_1^{\mathrm{T}} \cdot \mathbf{b} \le d_1 \Leftrightarrow \eta(\mathbf{b}) \ge 0$. Next, the algorithm encodes the condition (A2) as the inclusion assertion $\{(\mathbf{b}, \mathbf{r}) \mid \mathbf{b} \models \Phi \wedge F(\ell, \mathbf{b}, \mathbf{r}) \not\models \Phi\} \subseteq \{(\mathbf{b}, \mathbf{r}) \mid K \le \eta(F(\ell, \mathbf{b}, \mathbf{r})) \le 0\}$ parameterized with $\mathbf{a}, b, K$ for every $\ell \in \mathbf{L}$, where K is a fresh unknown constant. Then the algorithm encodes (A3) as $\{\mathbf{b} \mid \mathbf{b} \models \Phi\} \subseteq \{\mathbf{b} \mid \mathbf{c}_2^{\mathrm{T}} \cdot \mathbf{b} \le d_2\}$ where $\mathbf{c}_2, d_2$ are unique linear combinations of unknown coefficients $\mathbf{a}, b$ satisfying that $\mathbf{c}_2^{\mathrm{T}} \cdot \mathbf{b} \le d_2 \Leftrightarrow \mathbb{E}_{\mathrm{r},\ell}(\eta(F(\ell, \mathbf{b}, \mathbf{r}))) \le \eta(\mathbf{b}) - \epsilon$. The algorithm can also encode (A4) as $\{(\mathbf{b}, \mathbf{r}) \mid \mathbf{b} \models \Phi\} \subseteq \{(\mathbf{b}, \mathbf{r}) \mid |\eta(F(\ell, \mathbf{b}, \mathbf{r})) - \eta(\mathbf{b})| \le c\}$ parameterized with $\mathbf{a}, b, c$ for every $\ell \in \mathbf{L}$, where c is a fresh unknown constant. All the inclusion assertions (with parameters $\mathbf{a}, b, K, \epsilon, c$) are grouped *conjunctively* so that these inclusions should all hold.
(3) *Applying Farkas' Lemma.* The algorithm applies Farkas' Lemma to all the inclusion assertions from the previous step and obtains a system of linear inequalities with the parameters $\mathbf{a}, b, K, \epsilon, c$, where we over-approximate all strict inequalities (with '$<$') by non-strict ones (with '$\le$').
(4) *Constraint Solving.* The algorithm calls a linear programming (LP) solver on the linear program consisting of the system of linear inequalities generated in the previous step.

Besides the RSM-synthesis, we guarantee the non-expansiveness either directly from the structure of the program or by manual inspections (note that it can also be verified automatically through SMT solvers on the first order theory of reals). We check the bounded-update condition by a similar application of Farkas' Lemma (but without unknown parameters), and the Lipschitz continuity in next-step termination by Lemma 5.10. If the output of the algorithm is successful, i.e. if the obtained system of linear inequalities is feasible, then the solution to the LP obtains concrete values for $\mathbf{a}, b, K, \epsilon, c$ and leads to a concrete (difference-bounded) RSM-map η.

As our algorithm is based on LP solvers, we obtain polynomial-time complexity of our algorithm.

Theorem 8.2. *Our RSM-synthesis algorithm has polynomial-time complexity.*

Example 8.3. Consider the mini-roulette example showed in Figure 4(left) (Page 8).

(1) The algorithm sets a linear template $\eta(x, w) := a_1 \cdot x + a_2 \cdot w + a_3$.

(2) The algorithm encodes the conditions (A1)–(A3) as the inclusion assertions:

$$(A1)\quad \{(x,w) \mid x \geq 1 \wedge w \geq 0\} \subseteq \{(x,w) \mid -a_1 \cdot x - a_2 \cdot w \leq a_3\}$$
$$(A2)\quad \{(x,w) \mid x \geq 1 \wedge w \geq 0 \wedge x < 2\} \subseteq \{(x,w) \mid K \leq a_1 \cdot (x-1) + a_2 \cdot w + a_3 \leq 0\}$$
$$(A3)\quad \{(x,w) \mid x \geq 1 \wedge w \geq 0\} \subseteq \{(x,w) \mid 0 \leq \tfrac{1}{13} a_1 - \tfrac{4}{5} a_2 - \epsilon\}$$

(3) The algorithm applies Farkas' Lemma to all the inclusion assertions generated in the previous step and obtains a system of linear inequalities involving the parameters $a_1, a_2, a_3, K, \epsilon$, where we over-approximate all strict inequalities (with '$<$') by non-strict ones (with '$\leq$').
(4) The algorithm calls a linear programming (LP) solver on the linear program consisting of the system of linear inequalities generated in the previous step.

Finally, the algorithm outputs an optimal answer $\eta(x) = 13 \cdot x - 13$ with $\epsilon = 1, K = -13$(see Example 5.7). We can verify this η is an RSM-map with RSM-continuity. Due to the fact that this loop is non-expansive and has bounded-update, we can conclude that this loop is expected affine-sensitive over its loop guard by Theorem 5.5.

Example 8.4. Consider the mini-roulette variant example showed in Figure 4(right) (Page 8).

(1) The algorithm sets a linear template $\eta(x,w) := a_1 \cdot x + a_2 \cdot w + a_3$.
(2) The algorithm encodes the conditions (A1)–(A4) as the inclusion assertions:

$$(A1)\quad \{(x,w) \mid x \geq 1 \wedge w \geq 0\} \subseteq \{(x,w) \mid -a_1 \cdot x - a_2 \cdot w \leq a_3\}$$
$$(A2)\quad \{((x,w),r_6) \mid x \geq 1 \wedge w \geq 0 \wedge x < 3\} \subseteq \{((x,w),r_6) \mid K \leq a_1 \cdot (x - r_6) + a_2 \cdot w + a_3 \leq 0\}$$
$$(A3)\quad \{(x,w) \mid x \geq 1 \wedge w \geq 0\} \subseteq \{(x,w) \mid 0 \leq \tfrac{53}{130} a_1 - \tfrac{4}{5} a_2 - \epsilon\}$$
$$(A4)\quad \{(x,w) \mid x \geq 1 \wedge w \geq 0\} \subseteq \{(x,w) \mid |a_1 \cdot r_1 + 2a_2| \leq c \wedge |a_2 \cdot r_2 + 3a_2| \leq c \wedge |a_3 \cdot r_3 + 4a_2| \leq c \wedge |a_4 \cdot r_4 + 5a_2| \leq c \wedge |a_5 \cdot r_5 + 6a_2| \leq c \wedge |a_6 \cdot r_6| \leq c\}$$

(3) The algorithm applies Farkas' Lemma to all the inclusion assertions generated in the previous step and obtains a system of linear inequalities involving the parameters $a_1, a_2, a_3, K, \epsilon$, where we over-approximate all strict inequalities (with '$<$') by non-strict ones (with '$\leq$').
(4) The algorithm calls a linear programming (LP) solver on the linear program consisting of the system of linear inequalities generated in the previous step.

Finally, the algorithm outputs an optimal answer $\eta(x,w) = 2.45 \cdot x - 2.45$ with $\epsilon = 1, K = -4.91$(see Example 5.16). We can verify this η a difference-bounded RSM-map with RSM-continuity. In this example, we find both the update function and the loop guard are affine, all sampling variables are bounded continuously-distributed, and the coefficients for the current sampling variables are not all zero in the loop guard of the next iteration, so we can conclude this loop has the Lipschitz continuity in next-step termination by Lemma 5.10. Therefore, by Theorem 5.15, we can obtained that this loop is expected linear-sensitive over its loop guard.

Remark 12 (Scalability). As our approach is based on martingale synthesis, the scalability of our approach relies on the efficiency of martingale synthesis algorithms [Chakarov and Sankaranarayanan 2013; Chatterjee et al. 2016, 2018a,c, 2017; Feng et al. 2017; Wang et al. 2019b].

9 CASE STUDIES AND EXPERIMENTAL RESULTS

We demonstrate the effectiveness of our approach through case studies and experimental results. First, we consider two case studies of the SGD algorithm. Then in the experimental results, we use existing RSM-synthesis algorithms [Chakarov and Sankaranarayanan 2013; Chatterjee et al. 2016, 2018c] (as is illustrated in the previous section) to synthesize linear RSM-maps for non-expansive simple loops such as the mini-roulette examples, the heating examples and many other examples from the literature. Note that by Theorem 5.5 and Theorem 5.15, the existence of RSM-maps

with proper conditions (such as (A4), (B1)–(B4), etc.) leads to the expected sensitivity of all these examples.

9.1 Case Studies on Stochastic Gradient Descent

For the general SGD algorithm from Figure 2, we view each value i from $1, 2, \ldots, n$ a probabilistic branch with probability $\frac{1}{n}$. By viewing so, we have that the value after one loop iteration is $F(\mathbf{w}, i)$ where F is the update function, $\mathbf{w}$ is the program valuation before the loop iteration and i is the random integer sampled uniformly from $1, 2, \ldots, n$. In the case studies, we consider the metric defined from the Euclidean distance. To prove the expected affine-sensitivity of the SGD algorithm, we recall several properties for smooth convex functions (see e.g. [Nesterov 2004]).

Definition 9.1 (Smooth Convex Functions). A continuous differentiable function $f : \mathbb{R}^n \to \mathbb{R}$ is *convex* if for all $u, v \in \mathbb{R}^n$, we have that $f(v) \geq f(u) + \langle (\nabla f)(u), v - u \rangle$.

In the following, we denote by $\mathcal{F}^1(\mathbb{R}^n)$ the class of convex continuous differentiable Lipschitz-continuous functions, and by $\mathcal{F}_\beta^{1,1}(\mathbb{R}^n)$ the subclass of convex continuous differentiable Lipschitz-continuous functions whose gradient is Lipschitz-continuous with a Lipschitz constant β. We also denote by $\mathbf{0}$ the zero vector. The following results can be found in [Hardt et al. 2016; Nesterov 2004].

Proposition 9.2. *If $f \in \mathcal{F}^1(\mathbb{R}^n)$ and $(\nabla f)(u^*) = \mathbf{0}$, then u^* is the global minimum of f on $\mathbb{R}^n$.*

Proposition 9.3. *For any function $f \in \mathcal{F}_\beta^{1,1}(\mathbb{R}^n)$ and real number $0 < \gamma \leq \frac{2}{\beta}$, we have that the function g defined by $g(u) := u - \gamma \cdot (\nabla f)(u)$ is Lipschitz-continuous with Lipschitz constant 1.*

We consider the SGD algorithm in Figure 2 whose execution time is randomized and depends on the input program valuation. Recall that we consider the loop guard Φ of the SGD algorithm to be $G(\mathbf{w}) \geq \zeta$ so that ζ is the acceptable threshold of the (non-negative) loss function G and the aim of the algorithm is to find a solution $\mathbf{w}^*$ satisfying $G(\mathbf{w}^*) < \zeta$, as described in Example 4.3. Following [Hardt et al. 2016], we assume that each loss function G_i lies in $\mathcal{F}_\beta^{1,1}(\mathbb{R}^n)$ (so that the Euclidean magnitude of its gradient will always be bounded by a constant M), with a single Lipschitz constant β for all i. For practical purpose, we further consider the following assumptions:

(1) the function G has a global minimum $\zeta_{\min}$ at $\mathbf{w}_{\min}$ so that $G(\mathbf{w}_{\min}) = \zeta_{\min}$, which is satisfied by many convex functions;
(2) the parameters $\mathbf{w}$ will always be bounded during the execution of the SGD algorithm, i.e., $\|\mathbf{w}\|_2 \leq R$ for some radius $R > 0$. This can be achieved by e.g. regularization techniques where a penalty term is added to each G_i to prevent the parameters from growing too large;
(3) in order to ensure termination of the algorithm, we consider that the threshold value ζ is strictly greater than the global minimum $\zeta_{\min}$.

Below we show the expected (approximately linear) sensitivity of the SGD algorithm. The first fixes the vector ϑ of n training data that results in a non-expansive loop, while the second considers sensitivity w.r.t the training data and leads to an expansive loop in general.

Case Study A: sensitivity w.r.t initial parameters. In this case study, we fix the training data ϑ. By Proposition 9.3, we have that the loop is non-expansive w.r.t the Euclidean distance if the step size γ is small enough, for all i. Moreover, from the bound M, we have that the loop body has bounded update in $\mathbf{w}$, with a bound $d = M'' \cdot \gamma$ for some constant M'' determined by M.

Define the RSM-map η by $\eta(\mathbf{w}) := G(\mathbf{w}) - \zeta$. We show that η is indeed an RSM-map when the step size γ is sufficiently small. Since G is smooth convex, we have from Proposition 9.2 that $\nabla G(\mathbf{w}) = \mathbf{0}$

iff $\mathbf{w}$ is the global minimum on $\mathbb{R}^n$. Thus by the compactness from the radius R, we have that $G(\mathbf{w}) \geq \zeta > \zeta_{\min}$ implies $\|\nabla G(\mathbf{w})\|_2^2 \geq \delta$ for a fixed constant $\delta > 0$. Then by the Mean-Value Theorem, there is a vector $\mathbf{w}'$ on the line segment between $\mathbf{w}$ and $\mathbf{w} - \gamma \cdot \nabla G_i(\mathbf{w})$ such that

$$\begin{aligned}\eta(\mathbf{w} - \gamma \cdot \nabla G_i(\mathbf{w})) - \eta(\mathbf{w}) &= (\nabla G(\mathbf{w}'))^{\mathrm{T}} \cdot (-\gamma \cdot \nabla G_i(\mathbf{w})) \\ &= (\nabla G(\mathbf{w}))^{\mathrm{T}} \cdot (-\gamma \cdot \nabla G_i(\mathbf{w})) + (\nabla G(\mathbf{w}) - \nabla G(\mathbf{w}'))^{\mathrm{T}} \cdot \gamma \cdot \nabla G_i(\mathbf{w}) \ .\end{aligned}$$

By the smoothness and the Cauchy-Schwarz's Inequality, we have that

$$\begin{aligned}|(\nabla G(\mathbf{w}) - \nabla G(\mathbf{w}'))^{\mathrm{T}} \cdot \gamma \cdot \nabla G_i(\mathbf{w})| &\leq \gamma \cdot \|\nabla G(\mathbf{w}) - \nabla G(\mathbf{w}')\|_2 \cdot \|\nabla G_i(\mathbf{w})\|_2 \\ &\leq \gamma \cdot \beta \cdot \|\mathbf{w} - \mathbf{w}'\|_2 \cdot \|\nabla G_i(\mathbf{w})\|_2 \\ &\leq \gamma^2 \cdot \beta \cdot \|\nabla G_i(\mathbf{w})\|_2^2 \leq \gamma^2 \cdot \beta \cdot M^2\end{aligned}$$

It follows that $\mathbb{E}_i(\eta(\mathbf{w}-\gamma\cdot\nabla G_i(\mathbf{w})))-\eta(\mathbf{w}) = \frac{1}{n}\cdot\sum_{i=1}^{n}[\eta(\mathbf{w}-\gamma\cdot\nabla G_i(\mathbf{w}))-\eta(\mathbf{w})] = -\frac{\gamma}{n}\cdot\|\nabla G(\mathbf{w})\|_2^2 + C$ where $|C| \leq \gamma^2\cdot\beta\cdot M^2$. Hence by choosing a step size γ small enough, whenever $G(\mathbf{w}) \geq \zeta$ and the SGD algorithm enters the loop, we have $\mathbb{E}_i(\eta(\mathbf{w} - \gamma \cdot \nabla G_i(\mathbf{w}))) \leq \eta(\mathbf{w}) - \frac{\delta\cdot\gamma}{2\cdot n}$. Thus, we can choose $\epsilon = \frac{\delta\cdot\gamma}{2\cdot n}$ to fulfill the condition (A3). Moreover, by choosing $K = -\gamma \cdot M'$ for some positive constant M' determined by M, we have that when the SGD algorithm terminates, it is guaranteed that $K \leq \eta(\mathbf{w}) \leq 0$. Hence, η is an RSM-map for the SGD algorithm. Then by Theorem 5.5, we obtain the desired result that the SGD algorithm is expected affine-sensitive w.r.t the initial input parameters. In detail, we have the coefficients A_γ, B_γ from Theorem 5.5 that $A_\gamma = 2 \cdot \frac{d\cdot M+\epsilon}{\epsilon\cdot D_1}$, $B_\gamma = -2 \cdot \frac{d\cdot K}{\epsilon\cdot D_1}$. As both d, K, ϵ are proportional to the step size γ, when $\gamma \to 0$, we have that A_γ remains bounded and $B_\gamma \to 0$. Thus, our approach derives that the SGD algorithm is approximately expected linear sensitive (over all $G(\mathbf{w}) \geq \zeta$) when the step size tends to zero.

Case Study B: sensitivity w.r.t both initial parameters and training data. Second, we consider the expected affine-sensitivity around a neighbourhood of initial parameters $\mathbf{w}^*$ and the training data ϑ^*. Similar to the first case study, we consider that the values of the parameters $\mathbf{w}$ are always bounded in some radius and the magnitude of each individual training data is also bounded. A major difference in this case study is that we cannot ensure the non-expansiveness of the loop as the variation in the training data may cause the loop to be expansive. Instead, we consider the general case that the loop is expansive with the Lipschitz constant $L_\gamma = 1 + \gamma \cdot C$ for some $C > 0$.

Define the RSM-map η again as $\eta(\mathbf{w}, \vartheta) := G(\mathbf{w}, \vartheta) - \zeta$. Similarly, we can show that η is an RSM-map when the step size γ is sufficiently small, with parameters ϵ, K, d, M derived in the same way as in the first case study; in particular, both d, K, ϵ, c are proportional to the step size γ and we denote $\epsilon = M_1 \cdot \gamma$. Moreover, the RSM-map η is difference bounded with bound $c = M_2 \cdot \gamma$, where M_2 is a constant determined by M. Then by Theorem 6.1, we obtain that the SGD algorithm is expected affine-sensitive w.r.t both the initial input parameters and the training data. By a detailed calculation, we have the coefficients A_γ, B_γ from Theorem 6.1 that $A_\gamma := 2\cdot A'_\gamma \cdot L_\gamma^{N_\gamma} + 2\cdot A' \cdot L_\gamma^{N_\gamma} \cdot \exp\left(-\frac{\epsilon\cdot\eta(\mathbf{w}^*,\vartheta^*)}{8\cdot c^2}\right) \cdot \sum_{n=1}^{\infty}\left(L_\gamma \cdot \exp\left(-\frac{3\cdot\epsilon^2}{8\cdot c^2}\right)\right)^n$ and $B_\gamma := 2 \cdot B'_\gamma + 2 \cdot B'_\gamma \cdot \exp\left(-\frac{\epsilon\cdot\eta(\mathbf{w}^*,\vartheta^*)}{8\cdot c^2}\right) \cdot \sum_{n=1}^{\infty}\exp\left(-\frac{3\cdot\epsilon^2}{8\cdot c^2}\cdot n\right)$ where $A'_\gamma = \frac{d\cdot M+\epsilon}{D_1\cdot\epsilon}$, $B'_\gamma = -\frac{d\cdot K}{D_1\cdot\epsilon}$ and $N_\gamma = \lfloor 4 \cdot \frac{\eta(\mathbf{w}^*,\vartheta^*)}{\epsilon}\rfloor + 1$. As both d, K, ϵ, c are proportional to the step size γ, we have that A'_γ remains bounded and $B'_\gamma \to 0$. Moreover, we have that

$$L_\gamma^{N_\gamma} = (1 + C\cdot\gamma)^{\lfloor 4\cdot\frac{\eta(\mathbf{w}^*,\vartheta^*)}{M_1\cdot\gamma}\rfloor+1} \leq (1 + C\cdot\gamma)^{4\cdot\frac{\eta(\mathbf{w}^*,\vartheta^*)}{M_1\cdot\gamma}+1} \leq e^{4\cdot\frac{C}{M_1}\cdot\eta(\mathbf{w}^*,\vartheta^*)} \cdot (1 + C\cdot\gamma) \ .$$

where we recall that e is the base for natural logarithm. Hence, $L_\gamma^{N_\gamma}$ remains bounded when $\gamma \to 0$. Furthermore, as $\frac{\epsilon}{c^2} \to \infty$ and $\frac{\epsilon^2}{c^2}$ is constant when $\gamma \to 0$, we obtain that A_γ remain bounded and $B_\gamma \to 0$ when the step size tends to zero. Thus, we can also assert in this case that the SGD

Table 2. Experimental Results for Expected Affine-sensitivity (with $\epsilon = 1, L = 1$)

Example	Time/sec	$\eta(\mathbf{b})$	K	d	M
mini-Roulette	5.97	$13 \cdot x - 13$	-13	11	13
rdwalk	3.91	$-5 \cdot x + 5000$	-5	1	5
prdwalk variant	4.88	$-0.2857 \cdot x + 285.7$	-1.429	5	0.2857
prspeed	4.31	$-1.7143 \cdot x + 1714.3$	-5.143	3	1.7143
race variant	5.43	$-1.43 \cdot h + 1.43 \cdot t$	-4.29	4	2.86
ad. rdwalk 2D	4.55	$-0.77 \cdot x + 0.77 \cdot y$	-2.31	3	1.54
ad. rdwalk 1D Variant	4.49	$2.86 \cdot x$	-2.86	2	2.86
American Roulette	7.66	$20.27 \cdot x - 20.27$	-20.27	35	20.27

algorithm is approximately expected linear-sensitive (around a neighbourhood of the given input parameters and training data) when the step size tends to zero.

9.2 Experimental Results

We implemented our approach in Section 8 and obtained experimental results on a variety of programs. Recall we use Lemma 5.10 to ensure the Lipschitz continuity in next-step termination, and manually check whether a loop is non-expansive (which can also be automated (see Section 8)).

Experimental Examples. We consider examples and their variants from the literature [Abate et al. 2010; Chatterjee et al. 2018b,c; Ngo et al. 2018; Wang et al. 2019b]. Single/double-room heating is from [Abate et al. 2010]. Mini-roulette, American roulette are from [Chatterjee et al. 2018b]. Ad rdwalk 1D, Ad rdwalk 2D are from [Chatterjee et al. 2018c]. rdwalk, prdwalk, prspeed and race are from [Ngo et al. 2018]. Simple-while-loop, pollutant-disposal are from [Wang et al. 2019b]. See [Wang et al. 2019a, Appendix G] for the detailed examples. All the assignment statements in these examples (except for single/double-room heating) are of the form $x := x + r$, where r is a constant or a random variable, so we can find these examples are non-expansive. For single/double-room heating (Figure 5 and Figure 6), we choose the values of the parameters $b, b_1, b_2, a_{12}, a_{21}$ to be small enough, so that we can find the two examples non-expansive by manual inspections. In our experiments, we choose $x_a = 10, b = b_1 = 0.03, c = c_1 = 1.5, w \sim \mathsf{unif}(-0.3, 0.3)$, $b_2 = 0.02, a_{12} = a_{21} = 0.04, w_1 \sim \mathsf{unif}(-0.3, 0.3), w_2 \sim \mathsf{unif}(-0.2, 0.2)$. We use the max-norm as the metric. In this example, as the loop counter n starts always with 0, our experimental results show that the programs are expected affine/linear sensitive in n (i.e., the number of loop iterations).

Implementations and Results. We implemented our approach in Matlab R2018b. The results were obtained on a Windows machine with an Intel Core i5 2.9GHz processor and 8GB of RAM. Examples of expected affine-sensitivity are illustrated in Table 2, where the first column specifies the example and the program variable of concern, the second is the running time (in seconds) for the example, the third column is the RSM-map, and the last columns specify its related constants. Examples of expected linear-sensitivity are illustrated in Table 3 with similar layout.

Remark 13. In this work, we only consider the synthesis of linear RSM-maps to prove expected sensitivity. Given that algorithms for synthesis of polynomial RSM-maps are also present (see [Chatterjee et al. 2016; Feng et al. 2017]), it is also possible to tackle the case studies in Section 9.1 if the number of training data is fixed and the loss function G is polynomial. We plan the automated synthesis of complex RSM-maps for proving expected sensitivity as a future work.

Table 3. Experimental Results for Expected Linear-sensitivity (with $\epsilon = 1, L = 1$)

Example	Time/sec	$\eta(\mathbf{b})$	K	d	M	c
mini-roulette variant	12.69	$2.45 \cdot x - 2.45$	-4.91	9	2.45	22.08
single-room heating	5.21	$-0.833 \cdot x + 16.67$	-1.25	2.1	0.833	1.75
double-room heating	5.64	$-2.27 \cdot x_1 + 45.45$	-3.41	2.87	2.27	6.518
rdwalk variant	4.44	$-2.5 \cdot x + 2500$	-7.5	3	2.5	7.5
prdwalk	4.46	$-0.5714 \cdot x + 571.4$	-2.86	5	0.5714	2.86
prspeed variant	5.00	$-0.5333 \cdot x + 533.3$	-2.67	5	0.5333	2.67
race	5.17	$-2 \cdot h + 2 \cdot t$	-6	4	4	6
simple while loop	3.59	$-2 \cdot x + 2000$	-2	1	2	2
pollutant disposal	4.87	$n - 5$	-3	3	1	3
ad. rdwalk 2D variant	6.07	$-0.606 \cdot x + 0.606 \cdot y$	-2.424	4	1.212	2.424
ad. rdwalk 1D	5.16	$1.11 \cdot x$	-2.22	2	1.11	2.22
American roulette variant	14.95	$2.08 \cdot x - 2.08$	-4.15	35	2.08	72.63

10 RELATED WORK

In program verification Lipschitz continuity has been studied extensively: a SMT-based method for proving program robustness for a core imperative language is presented in [Chaudhuri et al. 2010]; a linear type system for proving sensitivity has been developed in [Reed and Pierce 2010]; approaches for differential privacy in higher-order languages have also been considered [de Amorim et al. 2017; Gaboardi et al. 2013; Winograd-Cort et al. 2017]. For probabilistic programs computing expectation properties has been studied over the decades, such as, influential works on PPDL [Kozen 1985] and pGCL [Morgan et al. 1996]. Various approaches have been developed to reason about expected termination time of probabilistic programs [Chatterjee et al. 2018c; Fu and Chatterjee 2019; Kaminski et al. 2016] as well as to reason about whether a probabilistic program terminates with probability one [Agrawal et al. 2018; Chatterjee et al. 2017; Huang et al. 2018a; McIver et al. 2017]. However, these works focus on non-relational properties, such as, upper bounds expected termination time, whereas expected sensitivity is intrinsically relational. To the best of our knowledge while RSMs have been used for non-relational properties, we are the first to apply them for relational properties. There is also a great body of literature on relational analysis of probabilistic programs, such as, relational program logics [Barthe et al. 2009] and differential privacy of algorithms [Barthe et al. 2012]. However, this line of works does not consider relational expectation properties. There are also several works on relational expectation properties in several specific area, e.g., in the area of masking implementations in cryptography, quantitative masking [Eldib et al. 2015] and bounded moment model [Barthe et al. 2016].

The general framework to consider probabilistic program sensitivity was first considered in [Barthe et al. 2017], and later improved in [Barthe et al. 2018]. Several classical examples such as stochastic gradient descent, population dynamics or Glauber dynamics can be analyzed in the framework of [Barthe et al. 2018]. Another method for the sensitivity analysis of probabilistic programs has been proposed in [Huang et al. 2018b] and they analysed a linear-regression example derived from the SGD algorithm in [Barthe et al. 2018]. For details of literature on relational analysis of probabilistic programs leading to the work of [Barthe et al. 2018] see [Barthe et al. 2018, Section 9].

Below we compare our result in detail with the most related results. Recall that we have discussed the issue of conditional branches at the end of Section 1. Here we compare other technical aspects.

Comparison with [Barthe et al. 2018]. The result of [Barthe et al. 2018] is based on the classical notion of couplings. Coupling is a powerful probabilistic proof technique to compare two distributions X and Y by creating a random distribution W who marginal distributions correspond to X and Y. Given a program with two different inputs x and y, let X_i and Y_i denote the respective probability distribution after the i-th iteration. If the number of iterations is fixed, then coupling can be constructed for each i-th step. However, if the number of iterations is randomized and variable-dependent, then in one case termination could be achieved while the other case still continues with the loop. In such situation, a suitable coupling is cumbersome to obtain. Thus, while coupling present an elegant technique for fixed number of iterations, our approach applies directly to the situation where the number of iterations is randomized as well as variable dependent. The advantage of coupling-based proof rules is that such approach can handle variable-dependent sampling and complex data structures through manual proofs. This leads to the fact that their approach can prove rapid mixing of population dynamics and glauber dynamics, while our approach in its current form cannot handle such examples. A strengthening of our approach to handle these type of examples (through e.g. an integration with coupling) is an interesting future direction.

Comparison with [Huang et al. 2018b]. The result of [Huang et al. 2018b] develops an automated tool based on computer algebra that calculates tight sensitivity bounds for probabilistic programs. As computer algebra requires to unroll every loop into its sequential counterpart without looping, the approach is suitable only for programs with a bounded number of loop iterations and cannot handle variable-dependent randomized loop iterations that typically lead to unbounded loop iterations. In contrast, our approach can handle unbounded variable-dependent randomized loop iterations.

Comparison with a recent arXiv submission. Recently, there is an arXiv submission [Aguirre et al. 2019] that also involves sensitivity analysis of probabilistic programs. Although their approach can handle randomized loop iterations to some extent, the approach requires that the executions from two close-by inputs should *synchronize* strictly, and simply assigns ∞ to *non-synchronous* situations such as one program valuation enters the loop while the other does not (see the definition for if-branch and while loop in [Aguirre et al. 2019, Figure 1]). Moreover, all the examples for illustrating their approach have fixed and bounded number of loop iterations, while all our examples have variable-dependent randomized loop iterations.

11 CONCLUSION

In this work we studied expected sensitivity analysis of probabilistic programs, and presented sound approaches for the analysis of (sequential composition of) probabilistic while loops whose termination time is randomized and depends on the input values, rather than being fixed and bounded. Our approach can be automated and can handle a variety of programs from the literature. An interesting future direction is to extend our approach to a wider class of programs (e.g., programs with expansive loops, conditional branches and variable-dependent sampling, synthesis of complex RSM-maps, etc.). Another important direction is to consider methods for generating tight bounds for sensitivity. Besides, integration with coupling-based approaches and practical issues arising from e.g. floating-point arithmetics would also be worthwhile to address.

ACKNOWLEDGMENTS

We thank anonymous reviewers for helpful comments, especially for pointing to us a scenario of piecewise-linear approximation (Remark 5). The research was partially supported by the National Natural Science Foundation of China (NSFC) under Grant No. 61802254, 61672229, 61832015, 61772336,11871221 and Austrian Science Fund (FWF) NFN under Grant No. S11407-N23 (RiSE/SHiNE). We thank Prof. Yuxi Fu, director of the BASICS Lab at Shanghai Jiao Tong University, for his support.

REFERENCES

Alessandro Abate, Joost-Pieter Katoen, John Lygeros, and Maria Prandini. 2010. Approximate Model Checking of Stochastic Hybrid Systems. *Eur. J. Control* 16, 6 (2010), 624–641. https://doi.org/10.3166/ejc.16.624-641

Sheshansh Agrawal, Krishnendu Chatterjee, and Petr Novotný. 2018. Lexicographic ranking supermartingales: an efficient approach to termination of probabilistic programs. *PACMPL* 2, POPL (2018), 34:1–34:32. https://doi.org/10.1145/3158122

Alejandro Aguirre, Gilles Barthe, Justin Hsu, Benjamin Lucien Kaminski, Joost-Pieter Katoen, and Christoph Matheja. 2019. Kantorovich Continuity of Probabilistic Programs. *CoRR* abs/1901.06540 (2019). arXiv:1901.06540 http://arxiv.org/abs/1901.06540

David J. Aldous. 1983. Random walks on finite groups and rapidly mixing Markov chains. *Séminaire de probabilités de Strasbourg* 17 (1983), 243–297. http://www.numdam.org/item/SPS_1983__17__243_0

Gilles Barthe, François Dupressoir, Sebastian Faust, Benjamin Grégoire, François-Xavier Standaert, and Pierre-Yves Strub. 2016. Parallel Implementations of Masking Schemes and the Bounded Moment Leakage Model. *IACR Cryptology ePrint Archive* 2016 (2016), 912. http://eprint.iacr.org/2016/912

Gilles Barthe, Thomas Espitau, Benjamin Grégoire, Justin Hsu, and Pierre-Yves Strub. 2018. Proving expected sensitivity of probabilistic programs. *PACMPL* 2, POPL (2018), 57:1–57:29. https://doi.org/10.1145/3158145

Gilles Barthe, Benjamin Grégoire, and Santiago Zanella Béguelin. 2009. Formal certification of code-based cryptographic proofs. In *Proceedings of the 36th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2009, Savannah, GA, USA, January 21-23, 2009*. 90–101. https://doi.org/10.1145/1480881.1480894

Gilles Barthe, Benjamin Grégoire, Justin Hsu, and Pierre-Yves Strub. 2017. Coupling proofs are probabilistic product programs. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017, Paris, France, January 18-20, 2017*. 161–174. http://dl.acm.org/citation.cfm?id=3009896

Gilles Barthe, Boris Köpf, Federico Olmedo, and Santiago Zanella Béguelin. 2012. Probabilistic relational reasoning for differential privacy. In *Proceedings of the 39th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2012, Philadelphia, Pennsylvania, USA, January 22-28, 2012*. 97–110. https://doi.org/10.1145/2103656.2103670

Patrick Billingsley. 1995. *Probability and Measure*. JOHN WILEY & SONS.

Olivier Bousquet and André Elisseeff. 2002. Stability and Generalization. *Journal of Machine Learning Research* 2 (2002), 499–526. http://www.jmlr.org/papers/v2/bousquet02a.html

Aleksandar Chakarov and Sriram Sankaranarayanan. 2013. Probabilistic Program Analysis with Martingales. In *CAV 2013*. 511–526.

Krishnendu Chatterjee. 2012. Robustness of Structurally Equivalent Concurrent Parity Games. In *Foundations of Software Science and Computational Structures - 15th International Conference, FOSSACS 2012, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2012, Tallinn, Estonia, March 24 - April 1, 2012. Proceedings*. 270–285. https://doi.org/10.1007/978-3-642-28729-9_18

Krishnendu Chatterjee, Hongfei Fu, and Amir Kafshdar Goharshady. 2016. Termination Analysis of Probabilistic Programs Through Positivstellensatz's. In *Computer Aided Verification - 28th International Conference, CAV 2016, Toronto, ON, Canada, July 17-23, 2016, Proceedings, Part I (Lecture Notes in Computer Science)*, Swarat Chaudhuri and Azadeh Farzan (Eds.), Vol. 9779. Springer, 3–22. https://doi.org/10.1007/978-3-319-41528-4_1

Krishnendu Chatterjee, Hongfei Fu, Amir Kafshdar Goharshady, and Nastaran Okati. 2018a. Computational Approaches for Stochastic Shortest Path on Succinct MDPs. In *IJCAI 2018*. 4700–4707.

Krishnendu Chatterjee, Hongfei Fu, Amir Kafshdar Goharshady, and Nastaran Okati. 2018b. Computational Approaches for Stochastic Shortest Path on Succinct MDPs. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden*. 4700–4707. https://doi.org/10.24963/ijcai.2018/653

Krishnendu Chatterjee, Hongfei Fu, Petr Novotný, and Rouzbeh Hasheminezhad. 2018c. Algorithmic Analysis of Qualitative and Quantitative Termination Problems for Affine Probabilistic Programs. *ACM Trans. Program. Lang. Syst.* 40, 2 (2018), 7:1–7:45. https://doi.org/10.1145/3174800

Krishnendu Chatterjee, Petr Novotný, and Đorđe Žikelić. 2017. Stochastic invariants for probabilistic termination. In *POPL 2017*. 145–160.

Swarat Chaudhuri, Sumit Gulwani, and Roberto Lublinerman. 2010. Continuity analysis of programs. In *Proceedings of the 37th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2010, Madrid, Spain, January 17-23, 2010*. 57–70. https://doi.org/10.1145/1706299.1706308

Arthur Azevedo de Amorim, Marco Gaboardi, Justin Hsu, Shin-ya Katsumata, and Ikram Cherigui. 2017. A semantic account of metric preservation. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017, Paris, France, January 18-20, 2017*. 545–556. http://dl.acm.org/citation.cfm?id=3009890

Josee Desharnais, Vineet Gupta, Radha Jagadeesan, and Prakash Panangaden. 2004. Metrics for labelled Markov processes. *Theor. Comput. Sci.* 318, 3 (2004), 323–354. https://doi.org/10.1016/j.tcs.2003.09.013

Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. 2006. Calibrating Noise to Sensitivity in Private Data Analysis. In *Proceedings of the Third Conference on Theory of Cryptography (TCC'06)*. Springer-Verlag, Berlin, Heidelberg, 265–284.

Cynthia Dwork and Aaron Roth. 2014. The Algorithmic Foundations of Differential Privacy. *Foundations and Trends in Theoretical Computer Science* 9, 3-4 (2014), 211–407. https://doi.org/10.1561/0400000042

Hassan Eldib, Chao Wang, Mostafa M. I. Taha, and Patrick Schaumont. 2015. Quantitative Masking Strength: Quantifying the Power Side-Channel Resistance of Software Code. *IEEE Trans. on CAD of Integrated Circuits and Systems* 34, 10 (2015), 1558–1568. https://doi.org/10.1109/TCAD.2015.2424951

J. Farkas. 1894. A Fourier-féle mechanikai elv alkalmazásai (Hungarian). *Mathematikaiés Természettudományi Értesitö* 12 (1894), 457–472.

Yijun Feng, Lijun Zhang, David N. Jansen, Naijun Zhan, and Bican Xia. 2017. Finding Polynomial Loop Invariants for Probabilistic Programs. In *Automated Technology for Verification and Analysis - 15th International Symposium, ATVA 2017, Pune, India, October 3-6, 2017, Proceedings (Lecture Notes in Computer Science)*, Deepak D'Souza and K. Narayan Kumar (Eds.), Vol. 10482. Springer, 400–416. https://doi.org/10.1007/978-3-319-68167-2_26

Hongfei Fu. 2012. Computing Game Metrics on Markov Decision Processes. In *Automata, Languages, and Programming - 39th International Colloquium, ICALP 2012, Warwick, UK, July 9-13, 2012, Proceedings, Part II (Lecture Notes in Computer Science)*, Artur Czumaj, Kurt Mehlhorn, Andrew M. Pitts, and Roger Wattenhofer (Eds.), Vol. 7392. Springer, 227–238. https://doi.org/10.1007/978-3-642-31585-5_23

Hongfei Fu and Krishnendu Chatterjee. 2019. Termination of Nondeterministic Probabilistic Programs. In *Verification, Model Checking, and Abstract Interpretation - 20th International Conference, VMCAI 2019, Cascais, Portugal, January 13-15, 2019, Proceedings (Lecture Notes in Computer Science)*, Constantin Enea and Ruzica Piskac (Eds.), Vol. 11388. Springer, 468–490. https://doi.org/10.1007/978-3-030-11245-5_22

Marco Gaboardi, Andreas Haeberlen, Justin Hsu, Arjun Narayan, and Benjamin C. Pierce. 2013. Linear dependent types for differential privacy. In *The 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '13, Rome, Italy - January 23 - 25, 2013*. 357–370. https://doi.org/10.1145/2429069.2429113

Moritz Hardt, Ben Recht, and Yoram Singer. 2016. Train faster, generalize better: Stability of stochastic gradient descent. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*. 1225–1234. http://jmlr.org/proceedings/papers/v48/hardt16.html

Mingzhang Huang, Hongfei Fu, and Krishnendu Chatterjee. 2018a. New Approaches for Almost-Sure Termination of Probabilistic Programs. In *Programming Languages and Systems - 16th Asian Symposium, APLAS 2018, Wellington, New Zealand, December 2-6, 2018, Proceedings (Lecture Notes in Computer Science)*, Sukyoung Ryu (Ed.), Vol. 11275. Springer, 181–201. https://doi.org/10.1007/978-3-030-02768-1_11

Zixin Huang, Zhenbang Wang, and Sasa Misailovic. 2018b. PSense: Automatic Sensitivity Analysis for Probabilistic Programs. In *Automated Technology for Verification and Analysis - 16th International Symposium, ATVA 2018, Los Angeles, CA, USA, October 7-10, 2018, Proceedings*. 387–403. https://doi.org/10.1007/978-3-030-01090-4_23

Benjamin Lucien Kaminski, Joost-Pieter Katoen, Christoph Matheja, and Federico Olmedo. 2016. Weakest Precondition Reasoning for Expected Run-Times of Probabilistic Programs. In *Programming Languages and Systems - 25th European Symposium on Programming, ESOP 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings*. 364–389. https://doi.org/10.1007/978-3-662-49498-1_15

Dexter Kozen. 1985. A Probabilistic PDL. *J. Comput. Syst. Sci.* 30, 2 (1985), 162–178. https://doi.org/10.1016/0022-0000(85)90012-1

Orna Kupferman and Moshe Y. Vardi. 1997. Modular Model Checking. In *Compositionality: The Significant Difference, International Symposium, COMPOS'97, Bad Malente, Germany, September 8-12, 1997. Revised Lectures (Lecture Notes in Computer Science)*, Willem P. de Roever, Hans Langmaack, and Amir Pnueli (Eds.), Vol. 1536. Springer, 381–401. https://doi.org/10.1007/3-540-49213-5_14

Annabelle McIver, Carroll Morgan, Benjamin Lucien Kaminski, and Joost-Pieter Katoen. 2017. A new proof rule for almost-sure termination. *Proceedings of the ACM on Programming Languages* 2, POPL (2017), 33.

S.P. Meyn and R.L. Tweedie. 1993. *Markov Chains and Stochastic Stability*. Springer-Verlag, London. available at: probability.ca/MT.

Carroll Morgan, Annabelle McIver, and Karen Seidel. 1996. Probabilistic Predicate Transformers. *ACM Trans. Program. Lang. Syst.* 18, 3 (1996), 325–353. https://doi.org/10.1145/229542.229547

Yurii Nesterov. 2004. *Introductory Lectures on Convex Optimization*. Applied Optimization, Vol. 87. Springer-Verlag US. https://doi.org/10.1007/978-1-4419-8853-9

Van Chan Ngo, Quentin Carbonneaux, and Jan Hoffmann. 2018. Bounded expectations: resource analysis for probabilistic programs. In *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2018, Philadelphia, PA, USA, June 18-22, 2018*. 496–512. https://doi.org/10.1145/3192366.3192394

Jason Reed and Benjamin C. Pierce. 2010. Distance makes the types grow stronger: a calculus for differential privacy. In *Proceeding of the 15th ACM SIGPLAN international conference on Functional programming, ICFP 2010, Baltimore, Maryland, USA, September 27-29, 2010*. 157–168. https://doi.org/10.1145/1863543.1863568

Franck van Breugel and James Worrell. 2006. Approximating and computing behavioural distances in probabilistic transition systems. *Theor. Comput. Sci.* 360, 1-3 (2006), 373–385. https://doi.org/10.1016/j.tcs.2006.05.021

Peixin Wang, Hongfei Fu, Krishnendu Chatterjee, Yuxin Deng, and Ming Xu. 2019a. Proving Expected Sensitivity of Probabilistic Programs with Randomized Variable-Dependent Termination Time. *CoRR* abs/1902.04744 (2019). arXiv:1902.04744 http://arxiv.org/abs/1902.04744

Peixin Wang, Hongfei Fu, Amir Kafshdar Goharshady, Krishnendu Chatterjee, Xudong Qin, and Wenjun Shi. 2019b. Cost analysis of nondeterministic probabilistic programs. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2019, Phoenix, AZ, USA, June 22-26, 2019.*, Kathryn S. McKinley and Kathleen Fisher (Eds.). ACM, 204–220. https://doi.org/10.1145/3314221.3314581

David Williams. 1991. *Probability with Martingales*. Cambridge University Press.

Daniel Winograd-Cort, Andreas Haeberlen, Aaron Roth, and Benjamin C. Pierce. 2017. A framework for adaptive differential privacy. *PACMPL* 1, ICFP (2017), 10:1–10:29. https://doi.org/10.1145/3110254