Research Statement

Di Wang

Research Mission

It is becoming increasingly popular for computer programs and systems to incorporate *randomness* to solve problems in a wide range of domains. In algorithm and system design, people use *randomization* to improve efficiency (as in randomized algorithms) and break symmetry (as in distributed protocols). Many software systems need to deal with *uncertainty*; for example, cyber-physical systems operate in the presence of changing environments and sensor errors, and quantum computers are faced with errors introduced by quantum gates. To model such uncertainty, people implement *probabilistic programs* (prob. programs) that manipulate prob. distributions. Moreover, to automate *Bayesian inference*, which is an approach to statistical machine learning, people develop statistical prob. programming systems (such as Pyro [BCJ$^+$18] and Stan [CGH$^+$17]).

Incorporation of randomness presents new challenges to important software-development tasks such as *analysis* of the correctness and efficiency of the programs and *design* of programming paradigms that are general and usable. Programming-language techniques—such as semantics, verification, compilation, type systems, and program abstractions—have been successfully employed in the analysis and design of traditional *non-probabilistic* programs and systems. However, those techniques are usually not directly applicable in the *probabilistic* setting. Rigorously modeling randomness is complex, and formally analyzing and verifying quantitative properties and statistical guarantees in the presence of randomness is in many ways more challenging than analyzing non-probabilistic programs. As a result, there are many open problems and research opportunities in this area.

Mission and Approach *My research mission* is to develop universal and easy-to-use abstractions and paradigms for programming randomness in software systems, and programming-language-level integrations to automatically analyze, optimize, and synthesize prob. programs. *My approach* is (i) using mathematical rigor to formally describe the semantics and lay the foundations for developing abstractions of prob. programs, (ii) devising new techniques that combine research in programming languages, probability theory, and machine learning to automatically analyze and verify properties of prob. systems, and (iii) applying my research to practical applications that involve randomness.

Research Accomplishments *I have developed an effective toolkit for rigorous and automatic analysis of the correctness and efficiency of multiple kinds of prob. programs.* This includes a new algebraic denotational semantics for incorporating nondeterminism [1], a general and flexible static-analysis framework [2], the first automatic type-based analysis of expected-cost bounds for prob. functional programs [3], and the first automatic derivation of symbolic interval bounds on central moments of cost accumulators [4]. Recently, I have investigated statistical prob. programming languages and proposed a new coroutine-based paradigm for implementing provably correct programmable inference [5]. As a Ph.D. student, I have also worked on analyzing other quantitative properties, such as resource consumption of programs [6; 7; 8]. My research involves diverse areas such as domain theory, probability theory, semantics, formal methods, type systems, constraint solving, Bayesian inference, and program synthesis. My results have been published in the most prestigious conferences in the area, including *PLDI*, *POPL*, and *ICFP*.

Future Work My plan for the future is to establish a research program around the theory and practice of incorporating randomness in software systems. I will build on my successful past work on the analysis and design of prob. programs, and focus on novel applications in areas where the computer programs and systems usually involve randomness, such as machine learning, cyber-physical systems, intermittent computation, and quantum computing. I will also continue to work on the most challenging open problems in the foundations for formal reasoning about randomness in software systems.

Background and Significance

People have been incorporating randomness in software systems via implementing prob. programs that can either execute to behave randomly, or be used to specify statistical models that describe noisy data. Precise understanding of the quantitative aspects and statistical properties of prob. programs is essential for reasoning about randomness.

Quantitative Program Analysis for Randomness To model randomness and uncertainty in algorithms and systems, it is a common approach to implement a *prob. program* that can draw random samples from prob. distributions and involve random control flows. Probabilities are quantitative objects; thus, many fundamental and important analysis tasks for prob. programs are quantitative: *What is the probability that an assertion holds? Is there any expression that is an invariant under expectation? What is the expected time complexity of a program?* People can build useful quantitative analyses upon those kinds of fundamental tasks, such as verifying fairness of decision-making programs [ADD+17], proving differential privacy [BKO+12], and analyzing the reliability of approximate programs [CMR13].

In general, it is challenging for software developers to perform quantitative analysis tasks on their code, because composing simple distributions can quickly complicate the result distribution, and randomness in the control flow can easily lead to state-space explosion. Moreover, because of the lack of determinacy, multiple runs of a prob. program can execute different paths and produce different results; therefore, dynamic simulation can be ineffective, as it may require a considerable number of runs to approach a good approximation for quantitative analysis.

Statistical Probabilistic Programming Bayesian inference is a method of statistical inference that accounts for the prob. distribution of hypotheses that produce the observed data. It has applications in many fields, including artificial intelligence [Gha15], cognitive science [GKT08], and applied statistics [GCS+13]. Traditionally, when a Bayesian practitioner iterates the design of a statistical model, they need to implement an inference algorithm specific to the model in the current iteration. *Statistical prob. programming* aims to separate model specifications and inference engines, thus speeds up the model-iteration process. The user implements a model as a *prob. program* that specifies a prob. distribution $p(x, y)$ on parameters x and observations y, then provides the program and observed data y to the inference engines, which automatically infer the posterior distribution $p(x \mid y)$. Statistical prob. programming is at the intersection of programming languages, machine learning, and statistics.

There are many prob. programming systems in active development; however, balancing between *expressibility* and *efficiency* remains a challenge for their design and implementation. Domain-specific languages enjoy specialized efficient inference algorithms, but have a restrictive syntax. On the other hand, general-purpose languages extend existing languages to make programming easier, but are often much slower than specialized inference solutions. Moreover, many inference algorithms themselves involve random computation, thus it is also difficult for the users to justify or debug the combination of their model and an inference algorithm.

Research Accomplishments

I have developed an effective toolkit for formal reasoning about prob. programs, and recently proposed a new programming paradigm for provably sound statistical prob. programming.

Formal Reasoning about Probabilistic Programs Noticing that reasoning systems for prob. programs have usually been standalone developments, I devised a generic framework for analyzing prob. programs at compile time in a compositional and flexible way. I proposed a new family of algebraic abstractions, namely *Markov algebras*, and developed a compositional denotational semantics based on those abstractions [1]. Contributions of this work include (i) the novel use of *hyper-graphs*, where each edge has one source but multiple destinations, to capture the natural asymmetry in random control-flows, and (ii) the flexibility to instantiate the semantics with different models of *nondeterminism*, which can arise naturally from prob. models (e.g., the agent for a Markov decision process). Based on Markov algebras, I developed a general framework, which I call PMAF, for designing, implementing, and proving the correctness of static analyses of prob. programs. PMAF can be instantiated from a small analysis specification, automatically and soundly perform *inter*procedural analysis, and carry out versatile analyses, such as performing exact inference or solving Markov decision problems [2]. I also instantiated PMAF with a new analysis that derives expectation invariants, which can be used to infer the *expected running time* (i.e., the average-case efficiency).

Apart from formal methods based on algebraic abstractions, I also took inspiration from *automatic amortized resource analysis* (AARA) [HJ03; HAH11], which is constraint-based and reduces quantitative analysis to linear programming. I proposed an extension of AARA and developed the first type system that automatically infers a polynomial bound on the expected running time of prob. functional programs [3]. A key innovation is the introduction of a *probability-encapsulation* type, which allows a prob. program to manipulate *symbolic* probabilities, whereas most of other automatic systems only support constant probabilities. Another contribution is a soundness proof that establishes the correctness of the bounds with respect to a semantics that includes diverging behavior. In another project, I generalized AARA to reason about *central moments* (i.e., $\mathbb{E}[(X - \mathbb{E}[X])^k]$ for a random variable X and a natural number k), and developed the first fully automatic analysis that derives symbolic interval bounds on central moments of *cost accumulators* (e.g., running time and cash flow) for programs with recursion and continuous distributions [4]. With central moments, I proposed a technique that bounds probabilities of the form $\mathbb{P}[X \geq d]$ or $\mathbb{P}[|X - \mathbb{E}[X]| \geq d]$, and applied it in a case study to identify side-channel vulnerabilities. I proved the soundness of the central-moment analysis using a new extension to the *Optional Stopping Theorem* from probability theory [9].

Towards Sound Statistical Probabilistic Programming One goal of statistical prob. programming is to help developers who have domain expertise, but lack experience in machine learning or statistics. Because there is not a single known inference algorithm that works well for all models, several prob. programming systems have recently added support for *programmable inference* to allow users to use domain knowledge to customize the inference process [MSH+18; CSL+19; BCJ+18; GXG18]. One popular customization mechanism involves user-supplied *guide programs*, such as *proposals* for Monte-Carlo methods (e.g., [GGT15]) and *approximating distributions* for variational inference (e.g., [FJB+19]). Implementing guide programs is nontrivial and error-prone [Pyr21], and guide programs must be *compatible* with model programs for inference to be sound.

I have focused on one pervasive but challenging condition for model-guide compatibility: *absolute continuity*, which can be guaranteed from a semantic property that the model and guide programs define prob. distributions with the same support [5]. I developed a prob. programming system that supports branching and recursion in its programming paradigm, with novel *guide types* that ensure well-typed model-guide pairs are guaranteed to enjoy absolute continuity. In my system, for the first time, model and guide programs are implemented as *coroutines* that communicate with each other to synchronize the set of random variables they sample during their execution. For good usability, the model and guide are still decoupled, but they are bridged via a guide type that enforces a communication protocol between them. An efficient algorithm infers guide types from code so that users do not have to specify the types. Other contributions are the proof that guide types ensure safety of coroutine communication, and the justification of the soundness of importance sampling, Markov-Chain Monte Carlo, and variational inference, via guide types. In my development, I took inspiration from *session types* [Hon93] that account for communication protocols in concurrent systems, and my approach opens the possibility of adapting programming-language techniques for concurrent computing to reason about the process of Bayesian inference.

Research Plan

My research plan is to establish a research program around sound and efficient incorporation of randomness in practical applications. I have demonstrated with my work on verification of soundness of statistical prob. programming [5] that programming-language techniques and proper abstractions can be adapted to tackling problems in different research areas. At the same time, I will continue to work on foundations of prob. programming.

1. Modeling Noisy Energy Consumption of Intermittent Programs *Energy-harvesting* systems, which are broadly used in the internet of things and implantable devices (e.g., [LKB+19]), collect energy from their environment to complete tasks without a battery. It is challenging to implement programs for energy-harvesting devices, because they operate *intermittently* when energy becomes available. *Task-based* intermittent programming [LR15] ensures correct execution of long-running programs, but require the developer to decompose the program into tasks, where each task consumes less energy than the energy capacity. However, modeling energy consumption is difficult because there is much *uncertainty* in the energy cost, such as environmental noises and unpredictable events from sensors.

I propose to apply quantitative analysis of prob. programs to precisely analyze the energy consumption of task-based intermittent programs. I plan to divide the problem into two parts: (i) given a basic-block-level statistical energy model, develop static-analysis techniques to automatically derive moments or tail bounds on the energy consumption of a task, and (ii) given the architectural specification of a device, develop static and dynamic methods to extract a device-specific statistical energy model. For the first part, I plan to devise new constructs to encode energy models into programs, and extend my work on central-moment analysis [4] to derive moments or tail bounds on energy consumption. Noting that intermittent programs usually do *not* feature recursion or unbounded loops, it is promising to use recurrence relations to derive moment bounds of non-polynomial forms, and concentration inequalities (e.g., the Chernoff-Hoeffding bound) to bound tail probabilities. For the second part, I plan to explore symbolic simulation on the hardware specification, dynamic profiling that collects statistics on energy measurements, and statistical modeling of noises and events. I envision that working in this part, I can take inspiration from and make innovations in formal verification of large-scale systems and environmental modeling for cyber-physical systems.

This is an ongoing project with *Brandon Lucia*, *Limin Jia*, and *Jan Hoffmann* (*Carnegie Mellon University*).

2. Diagnosing and Optimizing Programmable Bayesian Inference My work on sound programmable inference [5] focuses on the compatibility between model and guide programs. However, for a fixed model and a fixed inference algorithm, different compatible guides can dramatically influence the efficiency of the inference (e.g., [GGT15; Pyr21]). This fact inspires me to develop static-analysis techniques to diagnose and optimize programmable inference.

For efficiency diagnosis, I plan to research *relational* static analysis to compare quantitative aspects of two candidate guide programs for a given model. For example, variational inference usually involves Monte-Carlo estimation of stochastic gradients, and the efficiency is likely to be poor if the estimation has *high variance*. To compare two guide programs used for variational inference, one possible approach is to represent the gradient estimation as a prob. program with a cost counter that tracks the accumulation of gradients, and then statically bound the variance of the cost counter via a central-moment analysis [4]. I also want to investigate guide comparison for other inference algorithms such as Sequential Monte Carlo [DDJ06] with user-implemented proposal distributions.

For optimizing programmable inference, an ambitious approach that I plan to study is to automatically *synthesize* a guide program for a given model. Pyro [BCJ$^+$18] has supported automatic guide generation, but it is a runtime technique and requires that the user provide a strategy (e.g., the guide always samples from a Normal distribution). Instead, I plan to develop compile-time guide synthesis based on two components: (i) the quantitative relational analysis for efficiency diagnosis, and (ii) program synthesis that is type based (e.g., from guide types [5]) and takes quantitative constraints as its input. Together with my collaborators, we have developed an algorithm that synthesizes programs from resource-aware refinement types [7; 8]. Noting that quantitative constraints generated from resource analysis are similar to those generated from probabilistic analysis, I plan to investigate how the techniques for resource-aware synthesis can be adapted to synthesize prob. programs.

3. Taking the Next Steps in Reasoning about Randomness *Analysis of randomness in distributed systems.* There is a rise of modern distributed systems such as blockchains, cloud computing platforms, and federal machine learning, and many of them are naturally probabilistic. For example, distributed algorithms use randomization internally to break symmetry, and some message-passing systems use prob. distributions to model uncertainty of external events (e.g., messages being dropped). Formal reasoning about prob. distributed systems is challenging: the difficulty has been recently demonstrated in a prob. message-passing system by me and my colleagues [10]. I plan to study formal semantic models for different kinds of prob. distributed systems and develop analysis and verification tools for them.

Foundations of Markov algebras. I have proposed Markov algebras as the foundation of my work on the static analysis framework for prob. programs [1; 2]. As the *Kleene algebra* [Koz91] becomes an algebraic foundation for verification of non-probabilistic programs, I plan to develop the theory of Markov algebras to form an algebraic foundation for verification of prob. programs. To this end, I will investigate how to formulate an axiom system, prove soundness and completeness, and design a decision procedure of Markov algebras. I also plan to study applications of Markov algebras in concrete verification tasks, and to extend Markov algebras to reason about quantum computation.

In addition to the research directions described above, I am also open for new ideas and collaborations; in particular, with researchers outside of my area of expertise.

Referenced Own Publications

[1] D. Wang, J. Hoffmann, and T. Reps. 2019. A Denotational Semantics for Low-Level Probabilistic Programs with Nondeterminism. *Electr. Notes Theor. Comp. Sci.*, 347, (November 2019). Proceedings of the Thirty-Fifth Conference on the Mathematical Foundations of Programming Semantics. DOI: 10.1016/j.entcs.2019.09.016.

[2] D. Wang, J. Hoffmann, and T. Reps. 2018. PMAF: An Algebraic Framework for Static Analysis of Probabilistic Programs. In *Prog. Lang. Design and Impl.* (PLDI'18). DOI: 10.1145/3192366.3192408.

[3] D. Wang, D. M. Kahn, and J. Hoffmann. 2020. Raising Expectations: Automating Expected Cost Analysis with Types. *Proc. ACM Program. Lang.*, 4, ICFP, (August 2020). DOI: 10.1145/3408992.

[4] D. Wang, J. Hoffmann, and T. Reps. 2021. Central Moment Analysis for Cost Accumulators in Probabilistic Programs. In *Prog. Lang. Design and Impl.* (PLDI'21). DOI: 10.1145/3453483.3454062.

[5] D. Wang, J. Hoffmann, and T. Reps. 2021. Sound Probabilistic Inference via Guide Types. In *Prog. Lang. Design and Impl.* (PLDI'21). DOI: 10.1145/3453483.3454077.

[6] D. Wang and J. Hoffmann. 2019. Type-Guided Worst-Case Input Generation. *Proc. ACM Program. Lang.*, 3, POPL, (January 2019). DOI: 10.1145/3290326.

[7] T. Knoth, D. Wang, N. Polikarpova, and J. Hoffmann. 2019. Resource-Guided Program Synthesis. In *Prog. Lang. Design and Impl.* (PLDI'19). DOI: 10.1145/3314221.3314602.

[8] T. Knoth, D. Wang, A. Reynolds, J. Hoffmann, and N. Polikarpova. 2020. Liquid Resource Types. *Proc. ACM Program. Lang.*, 4, ICFP, (August 2020). DOI: 10.1145/3408988.

[9] D. Wang, J. Hoffmann, and T. Reps. 2021. Expected-Cost Analysis for Probabilistic Programs and Semantics-Level Adaption of Optional Stopping Theorems. (2021). https://arxiv.org/abs/2103.16105.

[10] A. Das, D. Wang, and J. Hoffmann. 2021. Probabilistic Resource-Aware Session Types. (2021). https://arxiv.org/abs/2011.09037.

External References

[ADD+17] A. Albarghouthi, L. D'Antoni, S. Drews, and A. V. Nori. 2017. FairSquare: Probabilistic Verification of Program Fairness. *Proc. ACM Program. Lang.*, 1, OOPSLA, (October 2017). DOI: 10.1145/3133904.

[BCJ+18] E. Bingham, J. P. Chen, M. Jankowiak, F. Obermeyer, N. Pradhan, T. Karaletsos, R. Singh, P. Szerlip, P. Horsfall, and N. D. Goodman. 2018. Pyro: Deep Universal Probabilistic Programming. *J. Machine Learning Research*, 20, 1, (January 2018). https://dl.acm.org/doi/10.5555/3322706.3322734.

[BKO+12] G. Barthe, B. Köpf, F. Olmedo, and S. Zanella Béguelin. 2012. Probabilistic Relational Reasoning for Differential Privacy. In *Princ. of Prog. Lang.* (POPL'12). DOI: 10.1145/2492061.

[CGH+17] B. Carpenter, A. Gelman, M. D. Hoffman, D. Lee, B. Goodrich, M. Betancourt, M. Brubaker, J. Guo, P. Li, and A. Riddell. 2017. Stan: A Probabilistic Programming Language. *J. Statistical Softw.*, 76, 1. DOI: 10.18637/jss.v076.i01.

[CMR13] M. Carbin, S. Misailovic, and M. C. Rinard. 2013. Verifying Quantitative Reliability for Programs that Execute on Unreliable Hardware. In *Object-Oriented Prog., Syst., Lang., and Applications* (OOPSLA'13). DOI: 10.1145/2509136.2509546.

[CSL+19] M. F. Cusumano-Towner, F. A. Saad, A. K. Lew, and V. K. Mansinghka. 2019. Gen: A General-Purpose Probabilistic Programming System with Programmable Inference. In *Prog. Lang. Design and Impl.* (PLDI'19). DOI: 10.1145/3314221.3314642.

[DDJ06] P. Del Moral, A. Doucet, and A. Jasra. 2006. Sequential Monte Carlo Samplers. *J. Royal Statistical Society*, 68, 3, (January 2006). https://www.jstor.org/stable/3879283.

[FJB+19] A. Foster, M. Jankowiak, E. Bingham, P. Horsfall, Y. W. Teh, T. Rainforth, and N. D. Goodman. 2019. Variational Bayesian Optimal Experimental Design: Efficient Automation of Adaptive Experiments. In *Neural Info. Processing Syst.* (NIPS'19). https://arxiv.org/abs/1903.05480.

[GCS+13] A. Gelman, J. B. Carlin, H. S. Stern, D. B. Dunson, A. Vehtari, and D. B. Rubin. 2013. *Bayesian Data Analysis*. Chapman and Hall/CRC. DOI: 10.1201/b16018.

[GGT15] S. Gu, Z. Ghahramani, and R. E. Turner. 2015. Neural Adaptive Sequential Monte Carlo. In *Neural Info. Processing Syst.* (NIPS'15). https://dl.acm.org/doi/10.5555/2969442.2969533.

[Gha15] Z. Ghahramani. 2015. Probabilistic machine learning and artificial intelligence. *Nature*, 521, (May 2015). DOI: 10.1038/nature14541.

[GKT08] T. L. Griffiths, C. Kemp, and J. B. Tenenbaum. 2008. Bayesian Models of Cognition. In *The Cambridge Handbook of Computational Psychology*. Cambridge University Press. DOI: 10.1017/CBO9780511816772.006.

[GXG18] R. Ge, K. Xu, and Z. Ghahramani. 2018. Turing: A Language for Flexible Probabilistic Inference. In *Artificial Intelligence and Statistics* (AISTATS'18).

[HAH11] J. Hoffmann, K. Aehlig, and M. Hofmann. 2011. Multivariate Amortized Resource Analysis. In *Princ. of Prog. Lang.* (POPL'11). DOI: 10.1145/1926385.1926427.

[HJ03] M. Hofmann and S. Jost. 2003. Static Prediction of Heap Space Usage for First-Order Functional Programs. In *Princ. of Prog. Lang.* (POPL'03). DOI: 10.1145/604131.604148.

[Hon93] K. Honda. 1993. Types for Dyadic Interaction. In *Int. Conf. on Concurrency Theory* (CONCUR'93). DOI: 10.1007/3-540-57208-2_35.

[Koz91] D. Kozen. 1991. A Completeness Theorem for Kleene Algebras and the Algebra of Regular Events. In *Logic in Computer Science* (LICS'91). DOI: 10.1109/LICS.1991.151646.

[LKB+19] Y. Lee, G. Kim, S. Bang, Y. Kim, I. Lee, P. Dutta, D. Sylvester, and D. Blaauw. 2019. A modular 1mm^3 die-stacked sensing platform with optical communication and multi-modal energy harvesting. In *Int. Conf. on Solid-State Circuits* (ISSCC'19). DOI: 10.1109/ISSCC.2012.6177065.

[LR15] B. Lucia and B. Ransford. 2015. A Simpler, Safer Programming and Execution Model for Intermittent Systems. In *Prog. Lang. Design and Impl.* (PLDI'15). DOI: 10.1145/2737924.2737978.

[MSH+18] V. K. Mansinghka, U. Schaechtle, S. Handa, A. Radul, Y. Chen, and M. C. Rinard. 2018. Probabilistic Programming with Programmable Inference. In *Prog. Lang. Design and Impl.* (PLDI'18). DOI: 10.1145/3296979.3192409.

[Pyr21] Pyro Contributors. 2021. SVI Part IV: Tips and Tricks — Pyro Tutorials 1.7.0 documentation. (2021). Retrieved 08/29/2021 from http://pyro.ai/examples/svi_part_iv.html.