Report from Dagstuhl Seminar 15181

Challenges and Trends in Probabilistic Programming

Edited by

Gilles Barthe[1], Andrew D. Gordon[2], Joost-Pieter Katoen[3], and Annabelle McIver[4]

1 **IMDEA Software – Madrid, ES,** gjbarthe@gmail.com
2 **Microsoft Research UK – Cambridge, GB,** adg@microsoft.com
3 **RWTH Aachen University, DE,** katoen@cs.rwth-aachen.de
4 **Macquarie University – Sydney, AU,** annabelle.mciver@mq.edu.au

Abstract

This report documents the program and the outcomes of Dagstuhl Seminar 15181 "Challenges and Trends in Probabilistic Programming". Probabilistic programming is at the heart of machine learning for describing distribution functions; Bayesian inference is pivotal in their analysis. Probabilistic programs are used in security for describing both cryptographic constructions (such as randomised encryption) and security experiments. In addition, probabilistic models are an active research topic in quantitative information now. Quantum programs are inherently probabilistic due to the random outcomes of quantum measurements. Finally, there is a rapidly growing interest in program analysis of probabilistic programs, whether it be using model checking, theorem proving, static analysis, or similar. Dagstuhl Seminar 15181 brought researchers from these various research communities together so as to exploit synergies and realize cross-fertilisation.

Seminar April 27–30, 2015 – http://www.dagstuhl.de/15181

1998 ACM Subject Classification D.1 Programming Techniques, D.2.4 Software/Program Verification, D.3 Programming Languages, D.4.6 Security and Protection, F. Theory of Computation, G.1.6 Optimization, G.3 Probability and Statistics, I.2.2 Automatic Programming, I.2.4 Knowledge Representation Formalisms and Methods

Keywords and phrases Bayesian networks, differential privacy, machine learning, probabilistic programs, security, semantics, static analysis, verification

Digital Object Identifier 10.4230/DagRep.5.4.123

Edited in cooperation with Benjamin Kaminski

1 Executive Summary

Gilles Barthe
Andrew D. Gordon
Joost-Pieter Katoen
Annabelle McIver

Probabilistic programming languages

Probabilistic programs are programs, written in languages like C, Java, LISP, or ML, with two added constructs: (1) the ability to draw values at random from probability distributions, and (2) the ability to condition values of variables in a program through observations. A variety of probabilistic programming languages have been defined such as `Church`, `Infer.NET`, and IBAL. Church is based on the Lisp model of the lambda calculus, containing pure Lisp as its deterministic subset, whereas Infer.NET is a Microsoft developed language akin to C# and compiles probabilistic programs into inference code[1]. Probabilistic programs can be used for modelling complex phenomena from biology and social sciences. By doing so, we get the benefits of programming languages (rigorous semantics, execution, testing and verification) to these problem domains. More than a decade ago, McIver and Morgan defined a probabilistic programming language in the style of Dijkstra's guarded command language, referred to as pGCL. Besides the usual language constructs in Dijkstra's GCL such as non-deterministic choice, it features a probabilistic choice where the probability distribution may be parametric. For instance, the assignment $\mathtt{x} \mathrel{+}= 1\ [p]\ \mathtt{skip}$ increments the variable x by one with probability p, and keeps the value of x unchanged with probability $1-p$, where p is an unknown real value from the range $[0, 1]$. Quantum programming languages such as qGCL and a quantum extension of C++ are also related, as their operational semantics is typically a probabilistic model so as to model the effect of measurements on the quantum state.

The importance of probabilistic programming

The applications of probabilistic programs mainly lie in four domains: (1) machine learning, (2) security, (3) randomised algorithms, and – though to a somewhat lesser extent – (4) quantum computing. Whereas the application in the field of randomised algorithms is evident, let us briefly describe the importance for the other three fields.

Machine learning

A Bayesian generative model consists of a prior distribution over some parameters, together with a sampling distribution (or likelihood) that predicts outputs of the model given its inputs and parameters. Bayesian inference in machine learning consists of training such a model to infer the posterior distribution of the parameters and hence to make predictions. In the probabilistic programming approach to Bayesian inference, the user simply writes the prior and sampling distributions as probabilistic programs, and relies on a compiler to generate code to perform inference and make predictions. Such compilers often operate by considering the program as defining a probabilistic graphical model. Graphical models were pioneered by Judea Pearl and others, and are extensively described in the comprehensive text by Koller and Friedman (2009). They are widely used in statistics and machine learning, with diverse application areas including speech recognition, computer vision, biology, and reliability analysis. Probabilistic graphical models allow specification of dependences between random variables via generative models, as well as conditioning of random variables using phenomena or data observed in the real world. A variety of inference algorithms have been developed to analyse and query such models, e.g., Gibbs sampling methods, Metropolis-Hastings and belief propagation. The probabilistic programming approach has seen growing interest within machine learning over the last 10 years and it is believed – see http://probabilistic-programming.org/wiki/Home – that this approach within AI has the potential to fundamentally change the way that community understands, designs, builds, tests and deploys probabilistic systems.

[1] For academic use, it is free to use: http://research.microsoft.com/infernet.

Security

Ever since Goldwasser and Micali – recipients of the ACM Turing Award in 2013 – introduced probabilistic encryption, probability has played a central role in cryptography: virtually all cryptographic algorithms are randomized, and have probabilistic security guarantees. Similarly, perturbing outputs with probabilistic noise is a standard tool for achieving privacy in computations; for instance, differential privacy achieves privacy-preserving data-mining using probabilistic noise. Cryptographic algorithms and differentially private algorithms are implemented as probabilistic programs; more singularly, one common approach for reasoning about these algorithms is using the code-based game-based approach, proposed by Bellare and Rogaway, in which not only the algorithms, but also their security properties and the hardness properties upon which their security relies, are expressed as probabilistic programs , and can be verified using (a relational variant of) Hoare logic. This code-based approach is key to recent developments in verified cryptography. Quantitative information flow is another important field in security where probabilistic programs and models play an important role. Here, the key question is to obtain quantitative statements about the leakage of certain information from a given program.

Quantum computing

Quantum programs are used to describe quantum algorithms and typically are quantum extensions of classical while-programs. Whereas in classical computation, we use a type to denote the domain of a variable, in quantum computation, a type is the state space of a quantum system denoted by some quantum variable. The state space of a quantum variable is the Hilbert space denoted by its type. According to a basic postulate of quantum mechanics, the unique way to acquire information about a quantum system is to measure it. Therefore, the essential ingredient in a quantum program is the ability to perform measurements of quantum registers, i.e., finite sequences of distinct quantum variables. The state space of a quantum register is the tensor product of the state spaces of its quantum variables. In executing the statement `measure M[q]; S`, quantum measurement M will first be performed on quantum register q, and then a sub-program S will be selected to be executed next according to the outcome of the measurement. The essential difference between a measurement statement and a classical conditional statement is that the state of program variables is changed after performing the measurement. As the outcome of a measurement is probabilistic, quantum programs are thus inherently probabilistic.

Program analysis

On the other hand, there is a recent rapidly growing trend in research on probabilistic programs which is more in line with traditional programming languages. This focuses on aspects such as efficient compilation, static analysis, program transformations, and program verification. To mention a few, Cousot *et al.* recently extended the framework of abstract interpretation to probabilistic programs (2012), Gordon *et al.* introduced `Tabular`, a new probabilistic programming language (2014), Di Pierro *et al.* apply probabilistic static analysis (2010), Rajamani, Gordon *et al.* have used symbolic execution to perform Bayesian reasoning on probabilistic programs with loops (2013), Katoen, McIver *et al.* have developed invariant synthesis technique for linear probabilistic programs (2010), and Geldenhuys *et al.* considered probabilistic symbolic execution (2012).

Achievements of this seminar

The objective of the seminar was a to bring together researchers from four separate (but related) communities to learn from each other, with the expectation that a better understanding between these communities would open up new opportunities for research and collaboration.

Participants attending the seminar represented all four themes of the original proposal: machine learning, quantitative security, (probabilistic) program analysis and quantum computing. The programme consisted of both tutorials and presentations on any topic within these themes. The tutorials provided a common ground for discussion, and the presentations gave insight into the current state of an area, and summarised the challenges that still remain. The tutorial topics were determined by consulting the participants prior to the seminar by means of a questionnaire.

Although the programme was primarily constructed around the tutorials and standard-length presentations (each around 30 minutes), the organisers made sure that time was always available for short, impromptu talks (sometimes of only 5 minutes) where participants were able to outline a relevant challenge problem or to draw attention to a new research direction or connection that had become apparent during the meeting.

This open forum for exploring links between the communities has led to the following specific achievements:

1. An increased understanding between the disciplines, especially between program verification and probabilistic programming.
2. A demonstration that the mathematical models for reasoning about machine learning algorithms and quantitative security are very similar, but that their objectives are very different. This close relationship at a foundational level suggests theoretical methods to tackle the important challenge of understanding privacy in a data mining context.
3. Evidence that probabilistic programming, analysis and verification of probabilistic programs, can have a broad impact in the design of emerging infrastructures, such as software-defined networks.

The feedback by the participants was very positive, and it was encouraged to organise a workshop or similar event in the future to foster the communication between the different communities, in particular between program verification and probabilistic programming.

We were aware of many new conversations between researchers inspired by the formal talks as well as the mealtime discussions. Already at least one paper (see below) with content inspired by the meeting is accepted for publication, and we are aware of several other new lines of work.

Acknowledgement

The organisers thank Benjamin Kaminski for his support in compiling this report and in several organisational issues.

References

1 A. Ścibior, Z. Ghahramani and A. D. Gordon. *Practical Probabilistic Programming with Monads.* ACM SIGPLAN Haskell Symposium 2015, Vancouver, Canada, 3–4 September 2015.

2 Table of Contents

3 Overview of Talks

3.1 Proving Differential Privacy in Hoare Logic

Gilles Barthe (IMDEA Software, Spain)

Main reference Gilles Barthe, Marco Gaboardi, Emilio Jesús Gallego Arias, Justin Hsu, César Kunz, Pierre-Yves Strub: Proving Differential Privacy in Hoare Logic. CSF 2014

Differential privacy is a rigorous privacy policy which provides individuals strong guarantees in the context of privacy-preserving data mining. Thanks to its rigorous definition, differential privacy is amenable to formal verification. Using a notion of (ϵ, δ)-lifting which generalizes the standard definition of lifting used in probabilistic process algebra, we develop a relational program logic to prove that probabilistic computations are differentially private.

3.2 Reasoning about Approximate and Uncertain Computation

Michael Carbin (Microsoft Research – Redmond, US)

Joint work of Carbin, Michael; Misailovic, Sasa; Rinard, Martin
Main reference M. Carbin, S. Misailovic, M. C. Rinard, "Verifying Quantitative Reliability for Programs that Execute on Unreliable Hardware," in Proc. of 28th ACM SIGPLAN Conf. on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA/SPLASH'13), pp. 33–52, ACM, 2013.
URL http://dx.doi.org/10.1145/2509136.2509546

Many modern applications implement large-scale computations (e.g., machine learning, big data analytics, and financial analysis) in which there is a natural trade-off between the quality of the results that the computation produces and the performance and cost of executing the computation.

Exploiting this fact, researchers have recently developed a variety of new mechanisms that automatically change the structure and execution of an application to enable it to meet its performance requirements. Examples of these mechanisms include skipping portions of the application's computation and executing the application on fast and/or energy-efficient unreliable hardware systems whose operations may silently produce incorrect results.

I present a program verification and analysis system, Rely, whose novel verification approach makes it possible to verify the safety, security, and accuracy of the approximate applications that these mechanisms produce. Rely also provides a program analysis that makes it possible to verify the probability that an application executed on unreliable hardware produces the same result as if it were executed on reliable hardware.

3.3 Equivalence of (Higher-Order) Probabilistic Programs

Ugo Dal Lago (University of Bologna, IT)

We introduce program equivalence in the context of higher-order probabilistic functional programs. The canonical notion of equivalence, namely context equivalence, has the nice property of prescribing equivalent programs to behave the same in any context, but has the obvious drawback of being based on a universal quantification over all contexts. We show how the problem can be overcome by going through a variation of Abramsky's applicative bisimulation. We finally hints at the role of equivalence in cryptographic proof.

3.4 A Topological Quantum Calculus

Alessandra Di Pierro (University of Verona, IT)

The work by Richard Feynman in the 1980s, and by Seth Lloyd and many others starting in the 1990s showed how a wide range of realistic quantum systems can be simulated by using quantum circuits, i.e. a quantum computer. In 1989, Edward Witten established a connection between problem solving and quantum field theories; he discovered a strong analogy between the Jones polynomial (an important knot invariant in topology) and Topological Quantum Field Theory (TQFT). Some years later, this discovery inspired a new form of quantum computation, called Topological Quantum Computation (TQC). A topological quantum computer would be computationally as powerful as a standard one. Nevertheless, Witten's discovery of the connection between TQFT and the value of the Jones polynomial at particular roots of unity implicitly suggested an efficient quantum algorithm for the approximation of the Jones polynomial, a problem which classically belongs to the P# complexity class and for which the standard quantum computing algorithmic techniques currently known do not provide any speed-up. Topological Quantum Computation is based on the existence of two-dimensional particles called anyons, whose statistics substantially differ from what we can observe in a three-dimensional quantum system. The behaviour of anyons can be described via the statistics observed after exchanging one particle with another. This exchange rotates the system's quantum state and produces non trivial phases. The idea of using such systems for computing is due to Alexei Kitaev and dates back to 1997. Since then, TQC has been mainly studied in the realm of physics and mathematics, while only recently the algorithmic and complexity aspects of this computational paradigm has been investigated in the area of computer science. Following this line, in this work we revisit TQC from the perspective of computability theory and investigate the question of computational universality for TQC, namely the definition of a anyonic quantum computer that is able to simulate any program on any other anyonic quantum computer. To this aim we introduce a formal calculus for TQC whose definition uses a language which is neither physical nor categorical but rather logical (if we look at the calculus as an equational theory) or programming-oriented (by considering it as an abstract model of computation). We adopt a formalism similar to the lambda-calculus that we call anyonic lambda-calculus. This calculus is essentially a re-writing system consisting of two transformation rules, namely variable substitution (as in the classical lambda-calculus) and a second one representing the braiding of anyons. The function definition scheme is exactly the same as Church's lambda-calculus. However, differently from the latter, the anyonic lambda-calculus represents an anyonic computer, that is a quantum system of anyons where computation occurs by braiding a fixed number of anyons among them for some fixed time. This is an approximation process that allows us to achieve approximate results, i.e. results that are not exact although their precision can be fixed arbitrarily high. For this calculus we provide an operational semantics in the form of a rewriting system and we show a property of confluence which takes into account the approximate nature of topological quantum computation.

References

1 Pachos, J. K., *Introduction to Topological Quantum Computation,* Cambridge U.P., 2012.
2 Wang, Z., *Topological Quantum Computation,* American Mathematical Soc., 2010.
3 Witten, E., *Topological quantum field theory*, Commun. Math. Phys **117** (1988), pp. 353–386.
4 Aharonov, D., V. Jones and Z. Landau, *A polynomial quantum algorithm for approximating the jones polynomial*, in: Proceedings of STOC'06 (2006), pp. 427–436.
5 Barendregt, H. P., *The Lambda Calculus*, Studies in Logic and the Foundations of Mathematics **103**, North-Holland, 1991, revised edition.
6 Freedman, M. H., A. Kitaev, M. J. Larsen and Z. Wang, *Topological Quantum Computation*, Physical Review Letters **40** (2001), p. 120402.
7 Jones, V. F. R., *A polynomial invariant for knots via Von Neumann algebras*, Bull. American Mathematical Society (New Series) **12** (1985), pp. 103–111.
8 Kitaev, A., *Fault-tolerant quantum computation by anyons*, Annals of Physics **303** (1997).

3.5 Dyna: A Circuit Programming Language for Statistical AI

Jason Eisner (Johns Hopkins University, US)

The Dyna programming language is intended to provide an declarative abstraction layer for building systems in ML and AI. A Dyna program specifies a generalized circuit that defines named quantities from other named quantities, using weighted Horn clauses with aggregation. The Dyna runtime must efficiently find a fixpoint of this circuit and maintain it under changes to the inputs. The language is an extension of logic programming with non-boolean values, evaluation, aggregation, types, and modularity. We illustrate how Dyna supports design patterns in AI, allowing extremely concise specifications of various algorithms, and we discuss the implementation decisions that are left to the system. Finally, we also sketch a preliminary design for P-Dyna, a probabilistic modeling language that can be embedded within Dyna and is based on augmenting Dyna's circuits with randomness.

3.6 Probabilistic Termination

Luis Maria Ferrer Fioriti (Universität des Saarlandes, DE)

Joint work of Ferrer Fioriti, Luis María; Hermanns, Holger
Main reference L. M. Ferrer Fioriti, H. Hermanns, "Probabilistic Termination: Soundness, Completeness, and Compositionality," in Proc. of the 42nd Annual ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages (POPL'15), pp. 489–501, ACM, 2015.
URL http://dx.doi.org/10.1145/2676726.2677001

We propose a framework to prove almost sure termination for probabilistic programs with real valued variables. It is based on ranking supermartingales, a notion analogous to ranking functions on nonprobabilistic programs. The framework is proven sound and complete for a meaningful class of programs involving randomization and bounded nondeterminism. We complement this foundational insight by a practical proof methodology, based on sound conditions that enable compositional reasoning and are amenable to a direct implementation using modern theorem provers. This is integrated in a small dependent type system, to overcome the problem that lexicographic ranking functions fail when combined with randomization. Among others, this compositional methodology enables the verification of probabilistic programs outside the complete class that admits ranking supermartingales.

3.7 Computability of conditioning: approximate inference and conditional independence

Cameron Freer (MIT – Cambridge, US)

Joint work of Ackerman, Nathanael; Avigad, Jeremy; Freer, Cameron; Roy, Daniel; Rute, Jason
Main reference N. L. Ackerman, C. E. Freer, D. M. Roy, "On the computability of conditional probability," arXiv:1005.3014v2 [math.LO], 2011.
URL http://arxiv.org/abs/1005.3014v2

Here we address three key questions at the theoretical and algorithmic foundations of probabilistic programming – and probabilistic modeling more generally – that can be answered using tools from probability theory, computability and complexity theory, and non-parametric Bayesian statistics. Which Bayesian inference problems can be automated, and which cannot? Can probabilistic programming languages represent the stochastic processes at the core of state-of-the-art nonparametric Bayesian models? And if not, can we construct useful approximations?

3.8 Tabular: A Schema-Driven Probabilistic Programming Language

Andrew D. Gordon (Microsoft Research UK – Cambridge, GB)

Joint work of Gordon, Andrew D.; Graepel, Thore; Rolland, Nicolas; Russo, Claudio; Borgstrom, Johannes; John Guiver, John
Main reference A. D. Gordon, T. Graepel, N. Rolland, C. Russo, J. Borgstrom, J. Guiver, "Tabular: A Schema-driven Probabilistic Programming Language," in Proc. of the 41st ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages (POPL'14), pp. 321-334, ACM, 2014.
URL http://dx.doi.org/10.1145/2535838.2535850

We propose a new kind of probabilistic programming language for machine learning. We write programs simply by annotating existing relational schemas with probabilistic model expressions. We describe a detailed design of our language, Tabular, complete with formal semantics and type system. A rich series of examples illustrates the expressiveness of Tabular. We report an implementation, and show evidence of the succinctness of our notation relative to current best practice. Finally, we describe and verify a transformation of Tabular schemas so as to predict missing values in a concrete database. The ability to query for missing values provides a uniform interface to a wide variety of tasks, including classification, clustering, recommendation, and ranking.

3.9 Conditioning in Probabilistic Programming

Friedrich Gretz (RWTH Aachen, DE)

Joint work of Gretz, Friedrich; Jansen, Nils; Kaminski, Benjamin Lucien; Katoen, Joost-Pieter; McIver, Annabelle; Olmedo, Federico
Main reference F. Gretz, N. Jansen, B. L. Kaminski, J.-P. Katoen, A. McIver, F. Olmedo, "Conditioning in Probabilistic Programming," arXiv:1504.00198v1 [cs.PL] , 2015.
URL http://arxiv.org/abs/1504.00198v1

In practical applications of probabilistic programming such as machine learning often the goal is to infer parameters of a probabilistic model from observed data. The used inference methods are entirely based on sampling and statistical methods. At the same time probabilistic programs in the realm of formal methods have a formal semantics that precisely captures the distribution generated by a program. First formal analysis techniques for such programs are emerging. Thus the question is if we can bring together two areas and apply formal methods to machine learning. Our work goes in this direction by introducing observations in a minimalistic core probabilistic language called pGCL. We are able to extend two existing equivalent semantics to conditional probability distributions. Our semantics are sound even for programs that do not necessarily terminate with probability one. We explain how non-determinism in the model can be handled in the operational semantics and why it is problematic for denotational semantics. We conclude with applications of our semantics. For one, we show how in principle we can reason about properties of probabilistic programs with observations. Second, we show how our semantics enable us to formally proof the validity of program transformations which are useful in practise.

3.10 On the Hardness of Almost-Sure Termination

Benjamin Kaminski (RWTH Aachen, DE)

Joint work of Kaminski, Benjamin; Katoen, Joost-Pieter

We study the computational hardness of computing expected outcomes and deciding (universal) (positive) almost–sure termination of probabilistic programs. It is shown that computing lower and upper bounds of expected outcomes is Σ_1^0– and Σ_2^0–complete, respectively. Deciding (universal) almost–sure termination as well as deciding whether the expected outcome of a program equals a given rational value is shown to be Π_2^0–complete. Finally, it is shown that deciding (universal) positive almost–sure termination is Σ_2^0–complete (Π_3^0–complete).

3.11 Distinguishing Hidden Markov Chains

Stefan Kiefer (University of Oxford, UK)

Hidden Markov Chains (HMCs) are commonly used mathematical models of probabilistic systems. They are specified by a Markov Chain, capturing the probabilistic behavior of a system, and an output function specifying the outputs generated from each of its states. One of the important problems associated with HMCs is the problem of identification of the source of outputs generated by one of a number of known HMCs. We report on progress on this problem.

3.12 Tutorial on Probabilistic Programming Languages

Angelika Kimmig (KU Leuven, BE)

Probabilistic programming languages combine programming languages with probabilistic primitives as well as general purpose probabilistic inference techniques. They thus facilitate constructing and querying complex probabilistic models. This tutorial provides a gentle introduction to the field through a number of core probabilistic programming concepts. It focuses on probabilistic logic programming (PLP), but also connects to related areas such as statistical relational learning and probabilistic databases. The tutorial illustrates the concepts through examples, discusses the key ideas underlying inference in PLP, and touches upon parameter learning, language extensions, and applications in areas such as bioinformatics, object tracking and information processing.

An interactive tutorial can be found at https://dtai.cs.kuleuven.be/problog/.

References

1 Luc De Raedt and Angelika Kimmig. *Probabilistic (logic) programming concepts.* Machine Learning, 2015. http://dx.doi.org/10.1007/s10994-015-5494-z

3.13 Rational Protection against Timing Attacks

Boris Köpf (IMDEA Software – Madrid, ES)

Joint work of Doychev, Goran; Köpf, Boris
Main reference G. Doychev, B. Köpf, "Rational Protection against Timing Attacks," in Proc. of the IEEE 28th Computer Security Foundations Symp. (CSF'15), pp. 526–536, IEEE, 2015; revised version available from author's webpage.
URL http://dx.doi.org/10.1109/CSF.2015.39
URL http://software.imdea.org/~bkoepf/papers/csf15.pdf

Timing attacks can effectively recover keys from cryptosystems. While they can be defeated using constant-time implementations, this defensive approach comes at the price of a performance penalty. One is hence faced with the problem of striking a balance between performance and security against timing attacks.

This talk presents a game-theoretic approach to the problem, for the case of cryptosystems based on discrete logarithms. Namely, we identify the optimal countermeasure configuration as an equilibrium in a game between a resource-bounded timing adversary who strives to maximize the probability of key recovery, and a defender who strives to reduce the cost while maintaining a certain degree of security. The key novelty in our approach are bounds for the probability of key recovery, which are expressed as a function of the countermeasure configuration and the attack strategy of the adversary.

We put our techniques to work for a library implementation of ElGamal. A highlight of our results is that we can formally justify the use of an aggressively tuned but (slightly) leaky implementation over a defensive constant-time implementation, for some parameter ranges. The talk concludes with an outlook on how static analysis, probabilistic programming, and machine learning can help with performing similar analyses for more general classes of programs.

References

1 Goran Doychev and Boris Köpf. *Rational Protection against Timing Attacks.* 28th IEEE Computer Security Foundations Symposium (CSF). 2015

3.14 Probabilistic Programming for Security

Piotr Mardziel (University of Maryland – College Park, US)

Probabilistic inference is a powerful tool for reasoning about hidden data from restricted observations and probabilistic programming is a convenient means of expressing and mechanizing this process. Likewise the same approaches can used to model adversaries learning about secrets. Security, however, often relies on formal guarantees not typical in machine learning applications. In this talk we will compare and contrast the two applications of probabilistic programming and present our work on approximate probabilistic inference that is sound relative to quantitative measures of information security.

3.15 Three Tokens Suffice

Joel Ouaknine (University of Oxford, GB)

Herman's self-stabilisation algorithm, introduced 25 years ago, is a well-studied synchronous randomised protocol for enabling a ring of N processes collectively holding any odd number of tokens to reach a stable state in which a single token remains. Determining the worst-case expected time to stabilisation is the central outstanding open problem about this protocol. It is known that there is a constant h such that any initial configuration has expected stabilisation time at most hN2. Ten years ago, McIver and Morgan established a lower bound of $4/27 \approx 0.148$ for h, achieved with three equally-spaced tokens, and conjectured this to be the optimal value of h. A series of papers over the last decade gradually reduced the upper bound on h, with the present record (achieved last year) standing at approximately 0.156. In a paper currently under review, we prove McIver and Morgan's conjecture and establish that $h = 4/27$ is indeed optimal.

In the talk, I would like to describe Herman's protocol, consider examples, discuss related work and some of the history of the problem, and present a very brief schematic overview of the approach.

3.16 The Design and Implementation of Figaro

Avi Pfeffer (Charles River Analytics – Cambridge, US)

In this talk, I present some of the motivations for the design of the Figaro probabilistic programming system (PPS) and describe the approach to implementing the system, particularly in regards to factored inference algorithms. Figaro is a PPS that is able to represent a very wide range of probabilistic models and provides automated inference algorithms for reasoning with those models. Figaro is designed to be easy to integrate with applications and data and to support many modeling frameworks, like functional and object-oriented paradigms, directed and undirected models, hybrid models with discrete and continuous variables, and dynamic models. Figaro is designed as an embedded library in Scala; you write Scala programs to construct and operate on Figaro models. This provides numerous advantages such as support for integration and the ability to construct models programmatically. Figaro has been applied to a number of applications in areas like cyber security, climate prediction, and system health monitoring.

Many PPSs use sampling algorithms such as Markov chain Monte Carlo for inference and Figaro also provides such algorithms. However, in Figaro, we are trying to make factored inference algorithms like variable elimination and belief propagation viable for probabilistic programming. These algorithms are often the best performing for graphical models, but they can be difficult to apply to probabilistic programs because they assume a factor graph of fixed, finite structure. We address this problem with two main ideas. First, lazy factored inference partially expands a model to a finite depth and bounds the influence of the unexpanded part of the model on the query, thereby enabling factored algorithms to be applied even when the factor graph is very large or infinite. We have shown the ability to produce bounds on problems where sampling and other factored algorithms cannot operate. Second, structured inference uses the model definition to automatically decompose a difficult factor graph into subproblems. Each of these subproblems can be solved using a different solver. We have shown that using different algorithms on different subproblems can yield a significant improvement in accuracy without incurring additional computation cost.

3.17 Dual Abstractions of Hidden Markov Models: A Monty Hell Puzzle

Tahiry Rabehaja (Macquarie University – Sydney, AU)

Hidden Markov Models, HMMs, are mathematical models of Markov processes whose state is hidden but from which information can leak via channels. They are typically represented as 3-way joint probability distributions. We use HMMs as denotations of probabilistic hidden state sequential programs, after recasting them as “abstract" HMMs, computations in the Giry monad, and equipping them with a partial order of increasing security. We then present uncertainty measures as a generalisation of the extant diversity of probabilistic entropies, and we propose characteristic analytic properties for them. Based on that, we give a “backwards", uncertainty-transformer semantics for HMMs, dual to the “forwards" abstract

HMMs. The backward semantics is specifically aimed towards a source level reasoning method for probabilistic hidden state sequential programs. [Joint work with Annabelle McIver and Carroll Morgan.]

I will be talking about channels, Markov processes and HMMs through a small Monty Hell puzzle. We will see that they are pieces of the single unified framework of abstract HMMs which in turn admit backward interpretations as UM-transformers. The transformer semantics constitutes the logical basis towards a source level quantitative analysis of programs with hidden states.

3.18 Types and Modules for Probabilistic Programming Languages

Norman Ramsey (Tufts University – Medford, US)

Many probabilistic programming languages include only core-language constructs for deterministic computation, plus primitives for probabilistic modeling and inference. We hypothesize that, like many other special-purpose languages, probabilistic languages could benefit from linguistic apparatus that has been found to be helpful in more general settings – in particular, types and modules. To support this hypothesis, we introduce the model, which resembles an ML module, but which in addition to a type part and a value part, also enjoys a distribution part. These parts are described in a model type, which is analogous to an ML signature or interface. In both a model and its type, distribution part is described compositionally by a collection of bindings to random variables. To explore the values of these ideas, we present a family of model types, at different levels of abstraction, and a corresponding model, of a problem in seismic detection (provided by Stuart Russell). Many challenges remain, of which the most pressing may be specifying the desire to learn a predictive posterior distribution.

3.19 Conditioning by Lazy Partial Evaluation

Chung-chieh Shan (Indiana University – Bloomington, US)

We review how to define measures mathematically, express them as programs, and run them as samplers. We then show how to define conditioning mathematically and implement it as a program transformation.

3.20 NetKAT – A Formal System for the Verification of Networks

Alexandra Silva (Radboud University Nijmegen, NL)

This talk will describe NetKAT, a formal system to program and verify networks. I will describe work from the following two articles:

1. Carolyn Jane Anderson, Nate Foster, Arjun Guha, Jean-Baptiste Jeannin, Dexter Kozen, Cole Schlesinger, and David Walker, NetKAT: Semantic Foundations for Networks. POPL 14.
2. Nate Foster, Dexter Kozen, Matthew Milano, Alexandra Silva, and Laure Thompson. A Coalgebraic Decision Procedure for NetKAT. POPL 15.

3.21 WOLFE: Practical Machine Learning Using Probabilistic Programming and Optimization

Sameer Singh (University of Washington – Seattle, US)

Performing machine learning with existing toolkits on large datasets is quite a frustrating experience: each toolkit focuses on its own subclass of machine learning techniques, have their own different interface of how much of the underlying system is surfaced to the user, and don't support the iterative development that is required to tune machine learning algorithms and achieve satisfactory predictors.

In this talk we present Wolfe, a declarative machine learning stack consisting of three crucial components: (1) Language: a math-like syntax embedded in Scala to concisely specify arbitrarily complex machine learning systems that unify most existing, and future, techniques, (2) Interpreter that transforms the declarative description into efficient code that scales to large-datasets, and (3) REPL: A new iPython-like IDE for Scala that supports the unique features for machine learning such as visualizing structured data, probability distributions, and state of optimization.

3.22 Recent Results in Quantitative Information Flow

Geoffrey Smith (Florida International University – Miami, US)

Main reference G. Smith, "Recent Developments in Quantitative Information Flow (Invited Tutorial)," in Proc. of the 30th Annual ACM/IEEE Symp. on Logic in Computer Science (LICS'15), pp. 23–31, IEEE, 2015; pre-print available from author's webpage.
URL http://dx.doi.org/10.1109/LICS.2015.13
URL http://users.cis.fiu.edu/~smithg/papers/lics15.pdf

In computer security, it is frequently necessary in practice to accept some leakage of confidential information. This motivates the development of theories of Quantitative Information Flow aimed at showing that some leaks are "small" and therefore tolerable. We describe the fundamental view of channels as mappings from prior distributions on secrets to hyper-distributions, which are distributions on posterior distributions, and we show how g-leakage provides a rich family of operationally-significant measures of leakage. We also discuss two approaches to achieving robust judgments about leakage: notions of capacity and a robust leakage ordering called composition refinement.

3.23 Tutorial on Probabilistic Programming in Machine Learning

Frank Wood (University of Oxford, GB)

This tutorial covers aspects of probabilistic programming that are of particular importance in machine learning in a way that is meant to be accessible and interesting to programming languages researchers. Example programs and inference are demonstrated in the Anglican programming language and examples of new inference algorithms applicable to inference in probabilistic programming systems, in particular the particle cascade, are provided.

3.24 Quantum Programming: From Superposition of Data to Superposition of Programs

Mingsheng Ying (University of Technology – Sydney, AU)

We extract a novel quantum programming paradigm – superposition of programs – from the design idea of a popular class of quantum algorithms, namely quantum walk-based algorithms. The generality of this paradigm is guaranteed by the universality of quantum walks as a computational model.

A new quantum programming language QGCL is then proposed to support the paradigm of superposition of programs. This language can be seen as a quantum extension of Dijkstra's GCL (Guarded Command Language). Alternation (case statement) in GCL splits into two different notions in the quantum setting: classical alternation (of quantum programs) and quantum alternation, with the latter being introduced in QGCL for the first time. Quantum alternation is the key program construct for realizing the paradigm of superposition of programs.

The denotational semantics of QGCL are defined by introducing a new mathematical tool called the guarded composition of operator-valued functions. Then the weakest precondition semantics of QGCL can straightforwardly derived.

Another very useful program construct in realizing the quantum programming paradigm of superposition of programs, called quantum choice, can be easily defined in terms of quantum alternation. The relation between quantum choices and probabilistic choices is clarified through defining the notion of local variables.

Furthermore, quantum recursion with quantum control flow is defined based on second quantisation method.

We believe that this new quantum programming paradigm can help to further exploit the unique power of quantum computing.

3.25 Counterexample-Guided Polynomial Quantitative Loop Invariants by Lagrange Interpolation

Lijun Zhang (Chinese Academy of Sciences, CN)

We apply multivariate Lagrange interpolation to synthesizing polynomial quantitative loop invariants for probabilistic programs. We reduce the computation of an quantitative loop invariant to solving constraints over program variables and unknown coefficients. Lagrange interpolation allows us to find constraints with less unknown coefficients. Counterexample-guided refinement furthermore generates linear constraints that pinpoint the desired quantitative invariants. We evaluate our technique by several case studies with polynomial quantitative loop invariants in the experiments.

Participants

- Christel Baier
TU Dresden, DE
- Gilles Barthe
IMDEA Software – Madrid, ES
- Johannes Borgström
Uppsala University, SE
- Michael Carbin
Microsoft Res. – Redmond, US
- Aleksandar Chakarov
Univ. of Colorado – Boulder, US
- Ugo Dal Lago
University of Bologna, IT
- Alessandra Di Pierro
University of Verona, IT
- Jason Eisner
Johns Hopkins University – Baltimore, US
- Yuan Feng
University of Technology – Sydney, AU
- Luis Maria Ferrer Fioriti
Universität des Saarlandes, DE
- Cédric Fournet
Microsoft Research UK – Cambridge, GB
- Cameron Freer
MIT – Cambridge, US
- Marco Gaboardi
University of Dundee, GB
- Andrew D. Gordon
Microsoft Research UK – Cambridge, GB
- Friedrich Gretz
RWTH Aachen, DE
- Johannes Hölzl
TU München, DE
- Chung-Kil Hur
Seoul National University, KR
- Benjamin Kaminski
RWTH Aachen, DE
- Joost-Pieter Katoen
RWTH Aachen, DE
- Stefan Kiefer
University of Oxford, GB
- Angelika Kimmig
KU Leuven, BE
- Boris Köpf
IMDEA Software – Madrid, ES
- Pasquale Malacaria
Queen Mary University of London, GB
- Vikash Mansinghka
MIT – Cambridge, US
- Piotr Mardziel
University of Maryland – College Park, US
- Annabelle McIver
Macquarie Univ. – Sydney, AU
- Joel Ouaknine
University of Oxford, GB
- Catuscia Palamidessi
INRIA Saclay – Île-de-France, FR
- David Parker
University of Birmingham, GB
- Avi Pfeffer
Charles River Analytics – Cambridge, US
- Tahiry Rabehaja
Macquarie Univ. – Sydney, AU
- Sriram K. Rajamani
Microsoft Research India – Bangalore, IN
- Norman Ramsey
Tufts University – Medford, US
- Chung-chieh Shan
Indiana University – Bloomington, US
- Alexandra Silva
Radboud Univ. Nijmegen, NL
- Sameer Singh
University of Washington – Seattle, US
- Geoffrey Smith
Florida International University – Miami, US
- Andreas Stuhlmüller
MIT Cambridge & Stanford University, US
- Frank Wood
University of Oxford, GB
- Mingsheng Ying
University of Technology – Sydney, AU
- Lijun Zhang
Chinese Academy of Sciences, CN