
AFRL-RI-RS-TR-2019-031

# AMORTIZED INFERENCE FOR PROBABILISTIC PROGRAMS

STANFORD UNIVERSITY

*FEBRUARY 2019*

FINAL TECHNICAL REPORT

***APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED***

STINFO COPY

**AIR FORCE RESEARCH LABORATORY**
**INFORMATION DIRECTORATE**

■ **AIR FORCE MATERIEL COMMAND** ■ **UNITED STATES AIR FORCE** ■ **ROME, NY 13441**

# NOTICE AND SIGNATURE PAGE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

This report is the result of contracted fundamental research deemed exempt from public affairs security and policy review in accordance with SAF/AQR memorandum dated 10 Dec 08 and AFRL/CA policy clarification memorandum dated 16 Jan 09. This report is available to the general public, including foreign nations. Copies may be obtained from the Defense Technical Information Center (DTIC) (http://www.dtic.mil).

AFRL-RI-RS-TR-2019-031 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

FOR THE CHIEF ENGINEER:

**/ S /**
WILLIAM E. MCKEEVER
Work Unit Manager

**/ S /**
QING WU
Technical Advisor, Computing
and Communications Division
Information Directorate

This report is published in the interest of scientific and technical information exchange, and its publication does not constitute the Government's approval or disapproval of its ideas or findings.

# REPORT DOCUMENTATION PAGE

*Form Approved*
**OMB No. 0704-0188**



| 1. REPORT DATE *(DD-MM-YYYY)* | 2. REPORT TYPE | 3. DATES COVERED *(From - To)* |
|---|---|---|
| FEBRUARY 2019 | FINAL TECHNICAL REPORT | NOV 2013 – AUG 2018 |

**4. TITLE AND SUBTITLE**

AMORTIZED INFERENCE FOR PROBABILISTIC PROGRAMS

**5a. CONTRACT NUMBER**

**5b. GRANT NUMBER**
FA8750-14-2-0006

**5c. PROGRAM ELEMENT NUMBER**
61101E

**6. AUTHOR(S)**

Noah Goodman

**5d. PROJECT NUMBER**
PPML

**5e. TASK NUMBER**
4S

**5f. WORK UNIT NUMBER**
TA

**7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)**
Stanford University
Jordan Hall, Building 01-420
450 Serra Mall, Stanford, CA 94305

**8. PERFORMING ORGANIZATION REPORT NUMBER**

**9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)**

Air Force Research Laboratory/RITA
525 Brooks Road
Rome NY 13441-4505

**10. SPONSOR/MONITOR'S ACRONYM(S)**
AFRL/RI

**11. SPONSOR/MONITOR'S REPORT NUMBER**
AFRL-RI-RS-TR-2019-031

**12. DISTRIBUTION AVAILABILITY STATEMENT**
Approved for Public Release; Distribution Unlimited. This report is the result of contracted fundamental research deemed exempt from public affairs security and policy review in accordance with SAF/AQR memorandum dated 10 Dec 08 and AFRL/CA policy clarification memorandum dated 16 Jan 09

**13. SUPPLEMENTARY NOTES**

**14. ABSTRACT**

Probabilistic programming holds the promise of revolutionizing computational systems by enabling non-experts to embed sophisticated probabilistic AI: machine learning, natural language processing, and computer vision. Stanford set out to radically accelerate probabilistic programming systems by targeting the full implementation stack from inference algorithms to hardware. They have made significant advances in inference algorithms, compilation techniques, and applications. Many of these advances have been released as open source software and/or transitioned to open source projects carried on by industry partners. This has contributed to major growth in the probabilistic programming community in both academia and industry. They expect in the near future to see further growth and uses in high-value applications across diverse sectors.

**15. SUBJECT TERMS**

Probabilistic programming language, WebPPL, Pyro, Markov chain Monte Carlo. MCMC, Bayesian Data Analysis.

| 16. SECURITY CLASSIFICATION OF: | | | 17. LIMITATION OF ABSTRACT | 18. NUMBER OF PAGES | 19a. NAME OF RESPONSIBLE PERSON |
|---|---|---|---|---|---|
| a. REPORT | b. ABSTRACT | c. THIS PAGE | | | **WILLIAM E. MCKEEVER** |
| U | U | U | UU | 17 | 19b. TELEPHONE NUMBER (*Include area code*) **N/A** |

# Contents

# 1 Summary

We set out to radically accelerate probabilistic programming systems by targeting the full implementation stack from inference algorithms to hardware. As reviewed below, we have made significant advances in inference algorithms, compilation techniques, and applications. Many of these advances have been released as open source software and/or transitioned to open source projects carried on by industry partners.

# 2 Introduction

Probabilistic programming holds the promise of revolutionizing computational systems by enabling non-experts to embed sophisticated probabilistic AI: machine learning, natural language processing, and computer vision. To achieve this promise, breakthroughs are needed in efficient inference algorithms and implementations. Over the course of the funding period we explored a set of techniques that resulted in speedups at every level of the inference problem: statistical techniques to learn more efficient inference from past experience, compilation techniques to remove overhead from the runtime system and target efficient hardware, and meta-compilation techniques to infer the most optimal hardware-specific implementations. These techniques, outlined below, resulted in several probabilistic programming language (PPL) implementations, advances in algorithms and applications, and advances in related research involving compilation to tightly target given hardware.

# 3 Methods, Assumptions and Procedures

Throughout the course of this project we have conducted basic research with an eye toward publication, open source software release, and possible impact on industry. Our methods reflect standard research methodology in machine learning, with somewhat greater emphasis on system building and evaluation. This twin emphasis on system building and impact is reflected in the open source probabilistic programming systems we have released and the publications resulting.

# 4 Results and Discussion

## 4.1 PPL systems

In order to fully explore and utilize PPLs and associated algorithms it is necessary to build systems that support scaleable inference. We explored, and released into open source, several such systems. The most important are review below.

### 4.1.1 WebPPL

WebPPL is a rich modern PPL embedded in JavaScript. [3] WebPPL compilation is based on first transforming code to continuation-passing style (CPS). This can result in code that includes deeply nested function calls. In many cases, anonymous functions are defined and immediately applied. We added an optimization pass to WebPPL in order to simplify the Javascript code it generates: we detect such cases, flatten them into blocks, and subsequently improve the generated code via optimization passes such as dead code elimination, removal of side-effect free expressions, and tree-based constant folding. In addition, we implemented trampolining to optimize the many tail-calls that result from CPS. The overall speed of the system is improved by one to two orders of magnitude from these additions.

In WebPPL, a store-passing transformation threads a global store through the CPS-transformed code. We extended the language to introduce a mutable global variable, backed by the store. Because the state of the store is explicit in the transformed code, particle filtering and other algorithms can be easily extended to be compatible with these mutable global variables. Mutable variables make it straightforward to implement collapsed conjugate models (e.g. the integrated for of multinomial-dirichlet pairs).

The WebPPL system includes many new algorithms for inference (further described below), as well as an ecosystem of distributions, libraries, and tests. While teaching a course on probabilistic models of cognition at Stanford, using WebPPL, we found a number of small issues and pain points that we were able to solve. For instance, we added a method for selecting the inference method automatically. This uses a simple decision tree to run through methods up to certain pre-defined computational limits. Student projects both stress-tested WebPPL and showcased it as a useful tool for cognitive modeling and data analysis. We have continued to pursue applications of WebPPL to data analysis ad cognitive modeling (particular of natural language use).

WebPPL has emerged as a go-to platform for cognitive modeling and bayesian data analysis (BDA), used and taught at top universities around the world. More generally, we have found flexible BDA to be one of the most useful and used applications for PPLs. This is reflected in adoption by the community, and in the more than 80 citations for the main

WebPPL source. It further supports cutting-edge directions such as optimal experiment design [6].

#### 4.1.2 Pyro

As our inference work turned to optimization-based techniques (described below) we found that it was very difficult to accelerate tensor processing in javascript, we instead took the lessons learned from WebPPL and used them to implement a new PPL directly in Python, using the Pytorch tensor libraries. This was a joint project with the Uber AI Labs, representing a point of technology transfer, as well as an exciting research direction. This resulted in the open source release of the Pyro PPL by Uber (see blog post at https://eng.uber.com/pyro/). [1] The Pyro project has proven to be an extremely good way forward for modern deep bayesian modeling with variational inference. Pyro has now become a large and vibrant community project, with almost 5000 stars on GitHub, an active message board, and contributions from nearly a dozen external contributors.

### 4.2 Inference techniques

#### 4.2.1 MCMC

Markov chain Monte Carlo (MCMC) inference, and especially the relatively simple Metropolis Hastings (MH) algorithm, form the basis for many implementations of scaleable inference in PPLs, especially for statistical data analysis models. Lightweight, source-to-source transformation approaches to implementing MCMC for probabilistic programming languages are popular for their simplicity, support of existing deterministic code, and ability to execute on existing fast runtimes. However, they are also slow, requiring a complete re-execution of the program on every Metropolis Hastings proposal. We developed a new extension to the lightweight approach, C3, which enables efficient, incrementalized re-execution of MH proposals. C3 is based on two core ideas: transforming probabilistic programs into continuation passing style (CPS), and caching the results of function calls. We found that on several common models, C3 reduces proposal runtime by 20-100x, in some cases reducing runtime complexity from linear in model size to constant. We also demonstrate nearly an order of magnitude speedup on a complex inverse procedural modeling application. The C3 system

is fully implemented and available as part of the open-source Webppl language. This work appeared at AIStats2016 [10].

### 4.2.2 HMC

In contrast to MH, Hamiltonian Monte Carlo (HMC) uses gradient information to make better changes to the model state at each step. We implemented HMC in both WebPPL and a low-level, high-performance probabilistic programming language called Quicksand (the source code and documentation can be found at http://dritchie.github.io/quicksand/)

We used Quicksand to explore applications of HMC in design for computer graphics, in particular the creation of stable rigid block structures. We presented a system for generating suggestions from highly-constrained, continuous design spaces. We formulated suggestion as sampling from a probability distribution; constraints are represented as factors that concentrate probability mass around sub-manifolds of the design space. These sampling problems are intractable using typical random walk MCMC techniques, so we adopted HMC. We evaluated its ability to efficiently generate suggestions for two different, highly-constrained example applications: vector art coloring and designing stable stacking structures. This work appeared at Eurographics 2015 [8]

### 4.2.3 SMC

We extended our inference suite by exploring the idea that sequential monte carlo (SMC) techniques, such as particle filtering, will be needed to scale up inference, particularly in light of two problem types: filtering problems (like the UAV challenge problem), and semantic parsing (roughly parsing a sentence into a logical denotation). We considered different ways to implement SMC for PPLs as simply as possible. In considering this problem, it became clear that what we needed was a co-routine approach to inference, where the execution future would be made explicit so that it could be copied and manipulated. To do this in a flexible way, we decided to try continuation passing style (CPS) conversion. We implemented the CPS conversion as a code transform for a purely functional subset of javascript, and used it as the basis of a rather compact PPL implementation. This lead us to WebPPL, and makes very clear what must be added to make a PPL from a PL: a primitive distribution type (ERPs), a sample keyword to sample from ERPs, a factor keyword to re-weight execution paths, and marginal inference operators that compute the marginal distribution on return values form a computation. WebPPL implements a number of inference algorithms, and seems to be a useful language for exploring many examples of interest: in particular, combinations of semantic parsing, pragmatic inference, and visual understanding.

The result of this effort forms the basis of the small web book The Design and Implementation of Probabilistic Programming Languages, at dippl.org. This is based on notes from a summer school class (at ESSLLI 2014). Along the way we've had a number of useful ideas and insights. These include the idea that we can insert "heuristic factors" that help guide inference, as long as they cancel by the end of the program execution.

This idea of heuristic factors lead to a method for controlling the output of procedural modeling programs using Sequential Monte Carlo (SMC). Previous probabilistic methods for controlling procedural models use Markov Chain Monte Carlo (MCMC), which receives control feedback only for completely-generated models. In contrast, SMC receives feedback incrementally on incomplete models, allowing it to reallocate computational resources and converge quickly. To handle the many possible sequentializations of a structured, recursive procedural modeling program, we develop and prove the correctness of a new SMC variant, Stochastically-Ordered Sequential Monte Carlo (SOSMC). We implement SOSMC for general-purpose programs using a new programming primitive: the stochastic future. We have shown that SOSMC reliably generates high-quality outputs for a variety of programs and control scoring functions. For small computational budgets, SOSMC's outputs often score nearly twice as high as those of MCMC or normal SMC. This work appeared at SIGGRAPH2015 [9].

Probabilistic inference algorithms such as SMC provide powerful tools for constraining procedural models in computer graphics, but they require many samples to produce desirable results. In this paper, we show how to create procedural models which learn how to satisfy constraints. We augment procedural models with neural networks which control how the model makes random choices based on the output it has generated thus far. We call such models neurally-guided procedural models. As a pre-computation, we train these models to maximize the likelihood of example outputs generated via SMC. They are then used as efficient SMC importance samplers, generating high-quality results with very few samples. We evaluate our method on L-system-like models with image-based constraints. Given a desired quality threshold, neurally-guided models can generate satisfactory results up to 10x faster than unguided models. [11]

### 4.2.4 Variational and optimization-based inference

Probabilistic programming languages are a powerful modeling tool, able to represent any computable probability distribution. Unfortunately, probabilistic program inference is often intractable, and many PPLs mostly rely on sampling-based methods that don't scale to large amounts of data. A different approach to inference in PPLs is to convert inference into an optimization problem, a set of techniques usually called Variational Inference (VI). This family of algorithms optimizes the parameters of a guide (importance) distribution family to

best approximate the posterior over program executions. The objective function can be one of a variety of upper and lower bounds on the model evidence, and training can be based on samples from the guide distribution (as in "black-box" variational) or from the (approximate) posterior. We performed a number of experiments within this framework, including targeted examples for procedural graphics, pragmatics, and vision-to-language models. Like HMC, VI relies critically on gradient computations. In order to explore these algorithms we implemented (and re-implemented) automatic differentiation (AD) for javascript by operator overloading (using sweet.js macros). This resulted in an AD library in JavaScript that has native support for tensor data types. (We also completed an exploration of using the Torch tensor library (TH.C) to speed up tensor computation in WebPPL. While some improvements resulted, the foreign-function interface overhead was so large that it offset the improvements from numerics alone.)

Even VI does not scale as well as we would like, because data points are treated individually. To alleviate this problem, one could try to learn from past inferences, so that future inferences run faster. This strategy is known as amortized inference; it has been applied to Bayesian networks and deep generative models. We proposed a system for amortized inference in PPLs. In our system, amortization comes in the form of a parameterized guide program. Guide programs have similar structure to the original program, but can have richer data flow, including neural network components. These networks can be optimized so that the guide approximately samples from the posterior distribution defined by the original program. We presented a flexible interface for defining guide programs and a stochastic gradient-based scheme for optimizing guide parameters, as well as some preliminary results on automatically deriving guide programs. We explored in detail the common machine learning pattern in which a "local" model is specified by "global" random values and used to generate independent observed data points; this gives rise to amortized local inference supporting global model learning.This work is reported in [7].

One example of such amortized inference is Variational Autoencoders (VAEs), which learn representations of data by jointly training a probabilistic encoder and decoder network. Typically these models encode all features of the data into a single variable. We became interested in learning disentangled representations that encode distinct aspects of the data into separate variables. We proposed to learn such representations using model architectures that generalize from standard VAEs, employing a general graphical model structure in the encoder and decoder, much as in our above technical report. This allows us to train partially-specified models that make relatively strong assumptions about a subset of interpretable variables and rely on the flexibility of neural networks to learn representations for the remaining variables. We further defined a general objective for semi-supervised learning in this model class, which can be approximated using an importance sampling procedure. We evaluated our framework's ability to learn disentangled representations, both by qualita-

tive exploration of its generative capacity, and quantitative evaluation of its discriminative ability on a variety of models and datasets [16].

Further extending this line of research, we investigated deep generative models for multi-modal data. Multiple modalities often co-occur when describing natural phenomena. Learning a joint representation of these modalities should yield deeper and more useful representations. Previous generative approaches to multi-modal input either do not learn a joint distribution or require additional computation to handle missing data. We introduced a multimodal variational autoencoder (MVAE) that uses a product-of-experts inference network and a sub-sampled training paradigm to solve the multi-modal inference problem. Notably, our model shares parameters to efficiently learn under any combination of missing modalities. We applied the MVAE on many datasets and matched state-of-the-art performance using many fewer parameters. In addition, we found that the MVAE is directly applicable to weakly supervised learning, and is robust to incomplete supervision. We considered two case studies in detail: one of learning image transformations – edge detection, colorization, segmentation – as a set of modalities, the second one of machine translation between two languages. We found appealing results across this range of tasks [19].

### 4.2.5 Coarse-to-fine inference

The goal of this project was to exploit the (implicit or explicit) hierarchical structure of a probabilistic model to speed up probabilistic inference: we reason first on an abstract (coarse) level, then refine our reasoning using the most promising avenues found when reasoning abstractly. Many practical techniques for probabilistic inference require a sequence of distributions that interpolate between a tractable distribution and an intractable distribution of interest. Usually, the sequences used are simple, e.g., based on geometric averages between distributions. When models are expressed as probabilistic programs, the models themselves are highly structured objects that can be used to derive annealing sequences that are more sensitive to domain structure. We proposed an algorithm for transforming probabilistic programs to coarse-to-fine programs which have the same marginal distribution as the original programs, but generate the data at increasing levels of detail, from coarse to fine. We applied this algorithm to an Ising model, its depth-from-disparity variation, and a factorial hidden Markov model. We found preliminary evidence that the use of coarse-to-fine models can make existing generic inference algorithms more efficient. [17]

## 4.3 Hardware for inference

We performed experiments on optimal compilation and accelerated hardware backends for probabilistic programming languages. The language used, Sliver, is not as expressive as a

full PPL. In particular, higher-order functions are only allowed in a limited form: looping primitives. Functions can only be fed to looping primitives, which take loop bounds as arguments and have void return type. We implemented a trace-based compiler for Silver that targets GPU's. In this case, we exploited the fact that traces do not contain control flow and GPU's can run such straight-line code very fast. We are then able to run many samples in parallel, either using MH and running independent chains, or plain Monte Carlo. We observed that a 32-site Ising model and a 12-item bin-packing problem ran on the order of $10^9$ parallel samples per second on a state-of-the-art GPU (NVIDIA GTX Titan).

Another set of experiments concerned a compiler that made better use of the fact that programs in Sliver yield exact information when exposed to a CFA2 analysis. This was a whole-program optimizing compiler in the tradition of Stalin Scheme and MLton, which also took advantage of control flow information for the whole program. Our approach was to compile the program to a finite-state machine (FSM), which fixed all control flow, and then compiled the FSM to a lower-level language (in our case, C++). For MCMC on a 32-site Ising model, we found that the resulting compiled program ran about as fast as the 32-site Ising model produced by tracing out the whole model. This is promising, as it shows that we can achieve performance similar to tracing using a compile-time set of techniques.

Finally, we also explored compilation to digital logic circuits realized on the field programmable gate array (FPGA). Our Sliver compiler can easily target digital logic circuits, as it starts from a representation of the computation as a finite state machine. We were able to successfully compile a few simple models, such as the 1-D Ising model, to the FPGA. One aspect of our FPGA compilation approach is to compile entire extended basic blocks of the original program to a single hardware state transition, rather than one instruction for each primitive operation. This can result in more computation per FPGA clock cycle. Following on to the PPAML program, our FPGA effort has been scaled up significantly because Mark Horowitz and Pat Hanrahan were funded by Intel to start a science and technology center on Agile Hardware Design.

## 4.4 Stochastic super-optimization

The optimization of short sequences of loop-free, fixed-point assembly code sequences is an important problem in highperformance computing. However, the competing constraints of transformation correctness and performance improvement often force even special purpose compilers to produce sub-optimal code. We have shown that by encoding these constraints as terms in a cost function, and using a Markov Chain Monte Carlo sampler to rapidly explore the space of all possible code sequences, we are able to generate aggressively optimized versions of a given target code sequence. Beginning from binaries compiled by "llvm ?O0", we are able to produce provably correct code sequences that either match or outperform

code produced by "gcc ?O3", "icc ?O3", and in some cases expert handwritten assembly. [13]

We described a general framework c2i for generating an invariant inference procedure from an invariant checking procedure. Given a checker and a language of possible invariants, c2i generates an inference procedure that iteratively invokes two phases. The search phase uses randomized search to discover candidate invariants and the validate phase uses the checker to either prove or refute that the candidate is an actual invariant. To demonstrate the applicability of c2i, we use it to generate inference procedures that prove safety properties of numerical programs, prove non-termination of numerical programs, prove functional specifications of array manipulating programs, prove safety properties of string manipulating programs, and prove functional specifications of heap manipulating programs that use linked list data structures. [14]

The aggressive optimization of heavily used kernels is an important problem in high-performance computing. However, both general purpose compilers and highly specialized tools such as superoptimizers often do not have sufficient static knowledge of restrictions on program inputs that could be exploited to produce the very best code. For many applications, the best possible code is conditionally correct: the optimized kernel is equal to the code that it replaces only under certain preconditions on the kernel?s inputs. The main technical challenge in producing conditionally correct optimizations is in obtaining non-trivial and useful conditions and proving conditional equivalence formally in the presence of loops. We combined abstract interpretation, decision procedures, and testing to yield a verification strategy that can address both of these problems. This approach yields a superoptimizer for x86 that in our experiments produces binaries that are often multiple times faster than those produced by production compilers. [15]

Software fault isolation (SFI) is an important technique for the construction of secure operating systems, web browsers, and other extensible software. We demonstrate that superoptimization can dramatically improve the performance of Google Native Client, a SFI system that ships inside the Google Chrome Browser. Key to our results are new techniques for superoptimization of loops: we propose a new architecture for superoptimization tools that incorporates both a fully sound verification technique to ensure correctness and a bounded verification technique to guide the search to optimized code. In our evaluation we optimize 13 libc string functions, formally verify the correctness of the optimizations and report a median and average speedup of 25% over the libraries shipped by Google. [2]

The aggressive optimization of floating-point computations is an important problem in high-performance computing. Unfortunately, floating-point instruction sets have complicated semantics that often force compilers to preserve programs as written. We present a method that treats floating-point optimization as a stochastic search problem. We demonstrated the ability to generate reduced precision implementations of Intel?s handwritten C

numeric library which are up to 6 times faster than the original code, and achieve end-to-end speedups of over 30% on a direct numeric simulation and a ray tracer by optimizing kernels that can tolerate a loss of precision while still remaining correct. Because these optimizations are mostly not amenable to formal verification using the current state of the art, we presented a stochastic search technique for characterizing maximum error. The technique comes with an asymptotic guarantee and provides strong evidence of correctness. [12]

Reasoning about floating-point is difficult and becomes only more so if there is an interplay between floating-point and bit-level operations. Even though real-world floating-point libraries use implementations that have such mixed computations, no systematic technique to verify the correctness of the implementations of such computations is known. In this paper, we presented the first general technique for verifying the correctness of mixed binaries, which combines abstraction, analytical optimization, and testing. The technique provides a method to compute an error bound of a given implementation with respect to its mathematical specification. We apply our technique to Intel?s implementations of transcendental functions and prove formal error bounds for these widely used routines. [5]

The x86-64 ISA sits at the bottom of the software stack of most desktop and server software. Because of its importance, many software analysis and verification tools depend, either explicitly or implicitly, on correct modeling of the semantics of x86-64 instructions. However, formal semantics for the x86-64 ISA are difficult to obtain and often written manually through great effort. We described an automatically synthesized formal semantics of the input/output behavior for a large fraction of the x86-64 Haswell ISA's many thousands of instruction variants. The key to our results is stratified synthesis, where we use a set of instructions whose semantics are known to synthesize the semantics of additional instructions whose semantics are unknown. As the set of formally described instructions increases, the synthesis vocabulary expands, making it possible to synthesize the semantics of increasingly complex instructions. Using this technique we automatically synthesized formal semantics for 1,795 instruction variants of the x86-64 Haswell ISA. We evaluated the learned semantics against manually written semantics (where available) and found that they are formally equivalent with the exception of 50 instructions, where the manually written semantics contain an error. We further found the learned formulas to be largely as precise as manually written ones and of similar size. [4]

## 4.5 Education and outreach

Over the course of the project we engaged in a number of education and outreach efforts. These included the PPAML 2016 summer school, a tutorial on BDA in WebPPL to be presented at CogSci2018, and the online textbooks probmods.org and dippl.org. In preparation for the PPAML 2016 summer school, we cleaned up and revised a large amount of the doc-

umentation, system, and interface for our WebPPL PPS. We also created materials for the lectures and exercises which are hosted in another web book.

# 5 Conclusions

Probabilistic programming holds the promise of revolutionizing computational systems by enabling non-experts to embed sophisticated probabilistic AI: machine learning, natural language processing, and computer vision. We set out to radically accelerate probabilistic programming systems by targeting the full implementation stack from inference algorithms to hardware. We have made significant advances in inference algorithms, compilation techniques, and applications. Many of these advances have been released as open source software and/or transitioned to open source projects carried on by industry partners. This has contributed to major growth in the probabilistic programming community in both academia and industry. We expect in the near future to see further growth and uses in high-value applications across diverse sectors.

# 6 References


[1] Eli Bingham, Jonathan P. Chen, Martin Jankowiak, Fritz Obermeyer, Neeraj Pradhan, Theofanis Karaletsos, Rohit Singh, Paul A. Szerlip, Paul Horsfall, and Noah D. Goodman. Pyro: Deep universal probabilistic programming. *JMLR*, abs/1810.09538, 2019.

[2] Berkeley Churchill, Rahul Sharma, JF Bastien, and Alex Aiken. Sound loop superoptimization for google native client. In *ACM SIGPLAN Notices*, volume 52, pages 313–326. ACM, 2017.

[3] N. D. Goodman and A. Stuhlmüller. *The Design and Implementation of Probabilistic Programming Languages*. 2015.

[4] Stefan Heule, Eric Schkufza, Rahul Sharma, and Alex Aiken. Stratified synthesis: automatically learning the x86-64 instruction set. In *ACM SIGPLAN Notices*, volume 51, pages 237–250. ACM, 2016.

[5] Wonyeol Lee, Rahul Sharma, and Alex Aiken. Verifying bit-manipulations of floating-point. *ACM SIGPLAN Notices*, 51(6):70–84, 2016.

[6] Long Ouyang, Michael Henry Tessler, D Ly, and Noah D Goodman. webppl-oed: A practical optimal experiment design system. In *Proceedings of the Fortieth Annual Conference of the Cognitive Science Society*, 2018.

[7] Daniel Ritchie, Paul Horsfall, and Noah D. Goodman. Deep Amortized Inference for Probabilistic Programs. Technical report, 2016.

[8] Daniel Ritchie, Sharon Lin, Noah D. Goodman, and Pat Hanrahan. Generating Design Suggestions under Tight Constraints with Gradient-based Probabilistic Programming. In *Proceedings of Eurographics 2015*, 2015.

[9] Daniel Ritchie, B. Mildenhall, N. D. Goodman, and P. Hanrahan. Controlling procedural modeling programs with stochastically-ordered sequential monte carlo. In *SIGGRAPH 2015*, 2015.

[10] Daniel Ritchie, Andreas Stuhlmüller, and Noah D. Goodman. C3: Lightweight incrementalized mcmc for probabilistic programs using continuations and callsite caching. In *AISTATS 2016*, 2016.

[11] Daniel Ritchie, Anna Thomas, Pat Hanrahan, and Noah D. Goodman. Neurally-guided procedural models: Amortized inference for procedural graphics programs using neural networks. In *Advances in Neural Information Processing Systems (NIPS 2016)*, 2016.

[12] Eric Schkufza, Rahul Sharma, and Alex Aiken. Stochastic optimization of floating-point programs with tunable precision. *ACM SIGPLAN Notices*, 49(6):53–64, 2014.

[13] Eric Schkufza, Rahul Sharma, and Alex Aiken. Stochastic program optimization. *Communications of the ACM*, 59(2):114–122, 2016.

[14] Rahul Sharma and Alex Aiken. From invariant checking to invariant inference using randomized search. *Formal Methods in System Design*, 48(3):235–256, 2016.

[15] Rahul Sharma, Eric Schkufza, Berkeley Churchill, and Alex Aiken. Conditionally correct superoptimization. In *ACM SIGPLAN Notices*, volume 50, pages 147–162. ACM, 2015.

[16] N Siddharth, Brooks Paige, Van de Meent, Alban Desmaison, Frank Wood, Noah D Goodman, Pushmeet Kohli, and Philip HS Torr. Learning disentangled representations with semi-supervised deep generative models. In *Advances in Neural Information Processing Systems 30*, 2017.

[17] Andreas Stuhlmüller, Robert X. D. Hawkins, N. Siddharth, and Noah D. Goodman. Coarse-to-fine sequential monte carlo for probabilistic programs. Technical report, 2015.

[18] David Tolpin, Jan-Willem van de Meent, and Frank Wood. Probabilistic programming in anglican. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 308–311. Springer, 2015.

[19] Mike Wu and Noah Goodman. Multimodal generative models for scalable weakly-supervised learning. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 5576–5586. Curran Associates, Inc., 2018.

# 7 List of Acronyms

- probabilistic programming language (PPL)
- continuation-passing style (CPS)
- bayesian data analysis (BDA)
- Hamiltonian Monte Carlo (HMC)
- Markov chain Monte Carlo (MCMC)
- sequential monte carlo (SMC)
- Stochastically-Ordered Sequential Monte Carlo (SOSMC)
- Variational Inference (VI)
- automatic differentiation (AD)
- Variational Autoencoders (VAEs)
- multimodal variational autoencoder (MVAE)
- graphics processing unit (GPU)
- finite-state machine (FSM)
- field programmable gate array (FPGA)