# Usability of Probabilistic Programming Languages

Alan Blackwell*, Luke Church**, Martin Erwig***, James Geddes****, Andy Gordon*****, Maria Gorinova******, Atilim Gunes Baydin*******, Bradley Gram-Hansen *******, Tobias Kohn*, Neil Lawrence*******, Vikash Mansinghka*********, Brooks Paige****, Tomas Petricek**********, Diana Robinson*, Advait Sarkar*****, Oliver Strickson****

*University of Cambridge; **Africa's Voices Foundation; ***Oregon State University; ****The Alan Turing Institute; *****Microsoft Research; ******University of Edinburgh; *******University of Oxford; ********Amazon AI Research; *********MIT; **********University of Kent

## Abstract

This discussion paper presents a conversation between researchers having active interests in the usability of probabilistic programming languages (PPLs), but coming from a wide range of technical and research perspectives. Although PPL development is currently a vigorous and active research field, there has been very little attention to date to basic questions in the psychology of programming. Relevant issues include mental models associated with Bayesian probability, end-user applications of PPLs, the potential for data-first interaction styles, visualisation of model structure and solver behaviour, and many others. We look forward to further discussion with delegates at the PPIG workshop.

## Introduction

This discussion has been convened to consider open research questions, priorities and potential design guidelines relevant to the usability of probabilistic programming languages (PPLs). It provides a short introduction, for the PPIG audience, to the conceptual and operational principles that underlie PPLs. It then discusses two alternative perspectives: firstly an applications perspective, in which there might be potential for broader application of PPL methods in the context of end-user programming if the languages were usable by a wider range of people; and secondly an educational perspective, in which we consider whether PPLs might be a valuable tool for teaching principles of probability, or even as an introduction to programming that takes a fundamentally probabilistic rather than deterministic or imperative view of how computation should be conceived. Finally, two sections present case studies following these perspectives - a visualisation approach that may have educational value, and a "furthest-first" approach to applications.

## Background and History

Probabilistic programming is a paradigm (generally embedded within conventional languages) in which the program is constructed as a model defined in terms of relationships between random variables (note this is not program synthesis, or programming by example, which are topics of interest at PPIG, but not the subject of this paper). Typical variables might be a sample data set, observable system output, or latent variables. A key distinction in relation to conventional programming languages is that variables do not have a single value, but should be regarded as defining (or sampling from) a probability distribution of likely values. Program execution consists of making inferences over the structure using a variety of methods — for example Markov chain Monte Carlo (MCMC) (Wingate 2011) or variational inference (Blei 2017). From a user perspective, the overall paradigm is declarative (it might be compared to logic

programming, involving far more extensive mathematical functions), although most PPLs are hosted within a functional or imperative language that allows more conventional expression of data transformations, I/O and so on. Note also that expert programmers of declarative languages (notoriously in the case of the Prolog “cut”) must read them imperatively in order to anticipate execution - a point that has been made by several people (remembering Prolog) when they interact with the PPL community.

There exist a number of different approaches to probabilistic programming that are built around a variety of semantics and inference engines. Broadly speaking, we can collapse these languages into two sets, one set being the first-order probabilistic programming languages (FOPPLs) (van de Meent et al. 2018), the other being the set of universal, or higher-order probabilistic programming languages (HOPPLs) (Staton et al. 2016). FOPPLs such as Stan (Gelman, Lee, and Guo 2015), BUGS (Spiegelhalter et al. 1996), Infer.NET (Minka et al. 2013) and LF-PPL (Zhou et al. 2019) directly constrain the set of models that the user can express as programs, so that the inference performed in such programs is more predictable. But, because of this restriction our modelling capabilities are limited, hence the construction of HOPPLs, universal languages that allow users to express any model imaginable in a Turing-complete fashion. Universal languages such as Church (Goodman et al. 2012), Anglican (Wood, van de Meent, and Mansinghka 2014), Pyro (Bingham et al. 2018), TensorFlow Probability (Dillon et al. 2017, Tran et al. 2017) and PyProb (Baydin et al. 2019) provide users with the ability to compose programs that generate any arbitrary model. But, at the cost  that it is impractical to guarantee correctness of the inference result.

Research in this field has primarily been driven by the desire for effective tools to enable statistical and machine learning research, and there has been little specialist attention to studying the usability of PPLs, or designing features that enhance usability. There has also been relatively little attention to PPLs in the human-centric computing, software engineering, software visualisation or visual languages communities, with the exception of a small number of experiments conducted by authors of this paper (systems built by Gorinova and Erwig are discussed below).

## The Idea of a Probabilistic Programming Language

The diversity of technical approaches just described mean that there is no single conception of what probabilistic programming provides. There is even less consensus on where PPL approaches might take us in future (for example, when used as an implementation platform for experiments with deep generative models). Nevertheless, it is useful to consider why this paradigm offers a distinctive intellectual appeal, in terms of the role of computation within a scientific enquiry. Let’s consider one of the possible styles of probabilistic programming, in which we focus on simplicity, interpretability, and causality.

Our modern understanding of the world started with a revolutionary insight and discovery. While analysing the data of the position of stars and planets in the night skies, astronomer Johannes Kepler found a function that reliably describes the data, and hence models the movements of planets in the solar system. A simple function was able to capture the big data, supported predictions, and led to new scientific insights.

Finding a function that describes a given set of data has since appeared in different shapes and forms. Carl Gauss had a clear model of how the function should look like, but had to deal with imprecision and uncertainty in the data when he developed least-squares linear regression. And Joseph Fourier's description of data through trigonometric functions builds a basis of today's signal processing. In contrast to Kepler's great feat of finding a new model, subsequent methods have mostly focused on adapting a known (or assumed) model to the data.

In recent years, AI research has made significant impact with neural networks - a universal set of functions that can model a wide variety of data. With higher power computing systems, deep and complex networks can describe many data sets with surprising precision. However, there is a catch: in their universality, neural networks provide little insight about the actual underlying models behind the data. We often struggle to understand exactly how a neural network describes a given set of data: the price of universality.

Like Gauss and Fourier, however, we often do have an understanding of what the model behind the data should look like. For linear regression, for instance, we assume a linear relationship in the data set and could thus replace the potentially huge and complex model of a neural network with a much simpler model featuring just a few parameters. All we need is a method to find meaningful values for the parameters in our model, whatever model we choose. This is one of the valuable opportunities offered by probabilistic programming methods.

Probabilistic programming in this sense combines the descriptive power of simple models with sophisticated methods to adapt the parameters in your model to given data. At the core of probabilistic programming you will find a set of "inference algorithms" not unlike the “learning algorithms” you encounter when training a neural network. However, instead of training a universal neural network using data samples, you write a specific model in a probabilistic programming language and infer its parameters through conditioning on data. As with earlier generations of logic programming, a probabilistic program is not run in the classical sense, but instead makes inferences. AI advocates sometimes say, of neural network optimization, that the system is being ‘taught’, rather than programmed - but to apply this analogy to probabilistic programming neglects the work done by the PPL programmer, and in particular the potential in some languages to implement alternative inference models (a layer of abstraction where the computation is expressed in more conventional imperative form, e.g. PLDI 2018).

**How does statistical modelling relate to probability?**

Classical linear regression is built on the principle of least squares: on the idea that there is a single pair of parameter values for which the "error" between model and data is minimal. In reality, however, linear regression hardly ever returns the true underlying parameters exactly, although we expect the result to be close to the true values, at least if the linear model is a good fit for the data. Our confidence in the proposed parameter values will then also increase when more data points are captured by the function. On the flipside, if the linear model is a bad fit for the data in the first place, the algorithm will never produce truly meaningful and accurate values.

In the context of probabilistic programming, we do not seek a single value for each parameter, then hope that these values together make our model a good fit to the data. It is much more natural to think in terms of probability distributions. Instead of proposing the single best value for each parameter, the inference engine will much rather tell you how probable a certain value is.

Think of it this way: if your model really fits your data, the inference engine will single out a range of values for your parameters that makes the entire model a very probable candidate for describing the data. If the model does not fit your data, the inference engine will find that no specific set of parameter values really stands out, and that nothing will make your model a particularly probable candidate for the data.

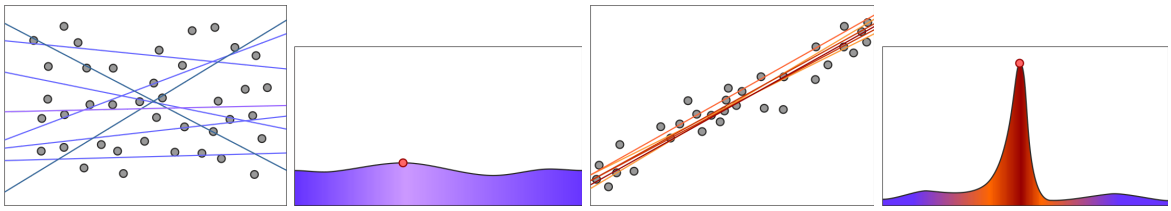


Figure 1: If the linear model is *not* a good fit for the data (on the left), no line really stands out, and all possible parameters have more or less the same probability. Even though we might find a "best" fit, it does not really distinguish itself from the rest. However, if the linear model is a good fit for the data (on the right), then some choices for parameter values are clearly better than others and stand out as highly probable values.

As noted in the introduction to this section, we have presented a relatively straightforward interpretation of probabilistic programming, to illustrate the potential appeal of the paradigm as a scientific tool and educational approach. Current and future developments in PPLs include far more complex approaches to modelling and generation, that might not necessarily retain this intuitive appeal. Nevertheless, this straightforward style of application offers a starting point for broadening access to PPL methods, as discussed in the following sections.

## The end-user development perspective on PPLs

It is often observed that more people create programs in spreadsheets such as Excel than in all other programming languages combined (Scaffidi et al 2005). Many business data processing applications that would have required professional programmers to implement them in the 1960s or 70s are now routinely created by people who have never received any formal training in programming, but are able to use spreadsheets to implement a wide variety of straightforward accountancy and data processing applications. The spreadsheet paradigm is approachable in part because of the way that it offers a concrete perspective on the object of interest (the user's data) rather than on the abstractions of programming. Nevertheless, it is possible to extend the spreadsheet paradigm with sophisticated abstract capabilities such as those of functional programming languages (Peyton Jones et al 2003).

Commercial extensions to the spreadsheet paradigm are generally driven by the business needs of spreadsheet users, who have practical problems to solve rather than being driven by technical curiosity or research agendas. These are defined as end-user programmers (Blackwell 2006, 2017), end-user developers or even end-user software engineers (Ko et al 2011). Increased business interest in the methods of statistical data science suggests that these end-users are likely to find value in PPL capabilities, especially if presented in an interaction context such as a spreadsheet, where users would be able to construct and interpret the behaviour of their program in the context of the data that it relates to. The usability advantages of data-centric presentation have already been observed in previous evolution of statistical applications, such as the adoption of a spreadsheet-style data view in the popular SPSS, when the SPSS-X release under Windows 2.0 included a tabular data editor that became established as the primary user interface for the GUI versions of the product.

Considering PPLs from this data-centric perspective, in terms of the tasks of end-users, raises interesting questions about the boundaries of the programming task. End-users who are creating simple scripts or macros to automate repetitive actions often deal implicitly with the attention investment tradeoff, calculating whether the programming effort will pay off in saved time. Because spreadsheets allow exceptional conditions to be handled by direct manipulation (just changing the cells concerned), spreadsheet programmers are far less likely to devote a lot of effort to anticipating infrequent situations in their code. This is a very familiar situation to data scientists responsible for "data wrangling" - formatting, organising and cleaning data sets for statistical analysis. Although standard research and teaching data

sets have been cleaned in advance, making wrangling distinct from modelling, real-world data science involves a far more ambiguous relationship between the two. It is often difficult to judge whether unexpected data values are errors, outliers, or important clues to an inappropriate model. If probabilistic programming involved closer interaction with original data, this would provide the opportunity for "wrangling" operations to inform the programmed model. It would also provide the opportunity for the mundane tasks of wrangling to be automated through inference, as in the Data Noodles prototype by Gorinova et al. (2016) that allows users to demonstrate how they would like their data to be arranged in a table, then searches for a set of structural transformations that will generate that table.

The same redefinition of boundaries, in a data-centric approach to end-user probabilistic programming, might allow us to revisit the definition of labelling, in the machine learning lifecycle. At present, most supervised learning systems rely on data that has been labelled with a "ground truth" of human interpretation, often obtained via Mechanical Turk, or forced tasks such as ReCAPTCHA. The people who carry out these labelling tasks may have some insight into the modelling assumptions (for example in relation to implicit bias in the judgments they have been asked to make, or explanations of why they made a particular judgment). However, those insights are currently not captured, or even discarded, in conventional machine learning paradigms. There have been some experiments in semi-supervised or mixed-initiative approaches to labelling, for example supporting more dynamic structuring of label categories. However, more sophisticated approaches could be enabled if the data views presented to the labeller offered more direct insight into the structure and behaviour of the model, perhaps even allowing trusted labellers to make incremental adjustments or modifications to the structure. A complementary benefit would be realised by data scientists themselves, who are often advised to spend more time looking at the data, before making assumptions about the structure of the model. Allowing the end-user programmer to contribute to labelling in a way that was continuous with the modelling task allows more sophisticated reasoning across multiple levels of abstraction, in a manner that is analogous to the constant shifts in level of abstraction that are observed in studies of expert programmers (Pennington 1987, 1995)

End-user paradigms such as spreadsheet programming also demonstrate the advantages for learning that result from bridging across levels of abstraction. Modern user interfaces appear more intuitive because a handful of basic principles can be applied in a concrete manner, together with discoverability of more abstract functions and relations. All of these design principles could be applied to implement tools supporting methodologies such as the Bayesian workflow of explore, model, infer, check, repeat (Gabry et al 2019).

We can also consider the potential to generate spreadsheets from PPL specifications. Two of the authors of this report (Geddes and Strickson) are working on a probabilistic programming language -- nocell -- where the result of running a program written in this language is a spreadsheet model applied to the input data. This allows the advantages of spreadsheet models, of understandability and immediacy, to be combined with sophisticated modelling techniques, as well as good software development practices (such as version control and modularity). A further aim is to connect the communities of software-developer data analysts with the wider community of spreadsheet users.

Values in nocell are probability distributions, supporting arithmetic operations and conditioning on observed data. This is motivated in part by the observation that many spreadsheet models are used in situations where capturing uncertainty in the model is beneficial, and that recent advances in PPL and machine language ideas could provide significant value to users of these models, who would otherwise have limited access to tools built on these ideas. This probabilistic approach contrasts with, for example, the type of scenario analysis that is commonly performed, where "typical", "typical-low", "typical-high" and perhaps more extreme values of model inputs are considered, to obtain an idea of the range of values that can be produced by a model (this could still be useful as a /presentational/ tool).

In the nocell approach, the programmer constructs a model as a nocell program, which includes setting appropriate program inputs to probability distributions and perhaps describe observations of their values. When run, this program produces a spreadsheet where the program outputs are evaluated from particular choices of input value, but in addition are annotated with their mean and standard deviation.

An important consideration, driving some of the current work, is how the probability distribution of a value of interest should be represented within the spreadsheet. This should be done in a way that conveys useful information at a level of detail appropriate for a wide audience.

## The educational perspective on PPLs

Languages such as Scratch (Resnick et al 2009) include both an application domain (in the case of Scratch, an architecture for agent-based graphical canvas operations) and an educationally-oriented IDE (in the case of Scratch, a block-syntax editor and a library browser). In case of Scratch, one of its major assets is the graphics application domain, echoing the emphasis on graphics in many earlier educational languages from Logo to AgentSheets and Alice. Such languages bring a Piagetian perspective to computation, for example the Scratch sprite or Logo turtle help the learner to think *syntonically* about program execution, by allowing the learner to reason about a computational agent's behaviour (Watt 1998, Pane 2002). Furthermore, as often advocated by Kay, tangible representations can help provide a concrete manipulable representation that helps learners to reason about abstract relations (Repenning 1996, Edge 2006, Kohn 2019).

Beyond the cognitive and notational advantages of a graphical application domain, a core motivation has been that graphics are fun. Children enjoy drawing, and freedom of graphical expression can help bring a creative and exploratory attitude to the introduction of novel notation systems (Stead 2014). How do we make teaching about (Bayesian) probability fun, through use of an application domain that motivates learners? Much traditional teaching of probability follows the traditions of Bayes himself, and other early theorists, in exploring the mathematical implications of gambling (coin tosses, dice throws etc). Teaching of frequentist statistics is largely grounded in the logic of hypothesis testing, and taught in the service of biology or psychology. Might contemporary problems of data science be more motivational as an application domain for education? For example, local children are affected by traffic speed on a road outside their school. The council reports an average speed slightly under the speed limit as evidence that there is no danger. Would children be motivated by gaining access to the council's raw data, and exploring the implications of those distributions for themselves?

A probabilistic programming IDE might potentially include live visualisations of the model; direct dependencies and probability distributions, highlight conditional independencies, as well as provide tools for visual or numerical diagnostics. Some of these ideas have been explored by previous work. For example, Gorinova et al (2016) present a live, multiple-representation environment (MRE) for the probabilistic programming language Infer.NET. Alongside the Infer.NET code, the environment maintains a visualisation of the program as a Bayesian network (a directed acyclic graph, encoding the conditional dependencies between variables). The marginal distribution of each variable is also visualised. Gorinova et al (2016) show that, when presented with debugging and program description tasks, users inexperienced in probabilistic modelling are faster and more confident when using the MRE compared to when using a conventional programming environment. Participants were also more likely to give a higher-level description of the dependencies in the model when using the MRE, as opposed to a lower-level code description. This suggests that live visualisations can be a useful way of teaching core concepts in Bayesian reasoning, and of drawing a clear distinction between conventional and probabilistic programming.

At PPIG 2018, Andrea diSessa made the provocative suggestion that school science lessons such as physics should in future be taught by students constructing their own computational simulations of the phenomenon, rather than through algebraic analysis and fitting of experimental observations. Scientific simulation has been a common application domain for educational programming languages in the past, for example in Repenning's AgentSheets, and Cypher's KidSim. We might imagine the possibility that teaching of probability in schools could be better achieved through modelling in a PPL. Indeed, Goodman and Tenenbaum's *Probabilistic Models of Cognition* includes interactive code examples implemented in WebPPL, Lee and Wagenmakers text on *Bayesian Cognitive Modelling* uses BUGS, and Andrew Gelman's courses at Columbia such as *Statistics GR6103* use modelling in Stan.

Many research users of PPLs have been mathematicians, meaning that the "natural" conceptualisations they are working with may be relatively sophisticated in mathematical terms. In the end-user application field, how far do our priorities shift from support for people who are familiar with the abstract operations, to those who have to treat the models and inferences as black box behaviour? What is the minimum conceptual framework for thinking about the behaviour of Bayesian models?

We can contrast this educational perspective with the suggestion that students should be given a "black box" understanding of how supervised machine learning systems work, as in research by Hitron et al (2019). This project provided students with an experimental environment in which they were able to collect and label (gesture) data, train a classifier, and evaluate the resulting system behaviour. Through experimentation with the system, students did gain improved understanding of supervised learning. We should consider such results in relation to the ICT/computer science debate in school curriculum. The "ICT" perspective was that it was sufficient for students to know how to use applications like Powerpoint or Word, and not necessary to understand how these work internally (i.e. to learn programming). This policy has now been overturned, in favour of teaching at a more fundamental level. On which side of this dichotomy might machine learning fall in future? Will training and using an ML system (for example, predictive text, or spam filtering) be a routine everyday task analogous to the use of Word or Powerpoint, or will it be a sophisticated intellectual task, providing a conceptual foundation for science and engineering? Will students benefit from being able to build new classifiers (using a PPL), or should a standard model be used to describe behaviour, for example in terms of feature selection, convergence, stability and generalisation?

## Case study of human-centric design on PPL principles

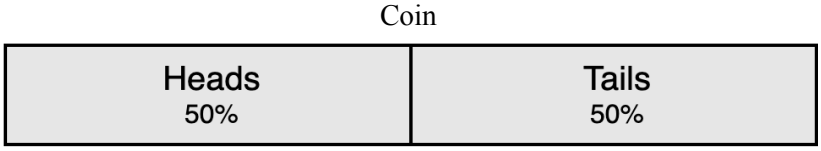
Here we briefly present an approach to visualize basic computations with probabilistic values, intended to make probabilistic programming accessible to a non-specialist audience, based on a notation introduced in (Erwig and Walkingshaw 2013). The notation is not a comprehensive visualization for PPLs yet; in particular, the visualization of inference requires extensions that we plan to work on in the future. It will also be interesting to compare this approach to probability visualisations such as (Cheng 2011). However, we believe the notation provides a simple metaphor for understanding probabilistic computations and can therefore be the basis for a discussion of a more comprehensive approach to visualizing the execution of PPL programs.

Understanding why and how programs produce their results is important for programmers to be confident in their work and for users to be sure that decisions they make based on a program's results are valid. Explaining the computation that results from executing a (non-probabilistic) program is a challenging task already, and this task becomes only more difficult for probabilistic programs, since probabilistic values have a more complicated nature and behave differently from deterministic values under transformations.

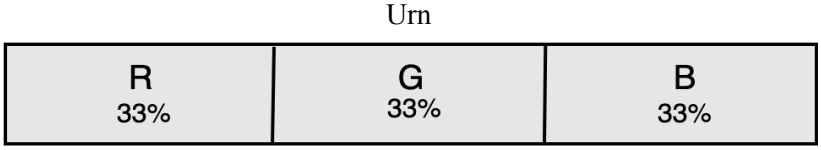
(The term *probabilistic value* in this context is synonymous with *probability distribution*; it is used to emphasize its role as an object of computational manipulation.)

A general approach to explaining program behavior is based on tracing values or states as they undergo changes when manipulated by a specific program. To apply this approach to the explanation of probabilistic programs we first need a representation of probabilistic values that can be the basis traces. A probabilistic value is a mapping from a set of values to numbers in the range [0, 1]. For example, the result of a fair coin flip can be represented by the mapping {Heads ↦ 0.5, Tails ↦ 0.5}.

Spatial partitions can serve as a simple visual metaphor for illustrating *discrete* probabilistic values (that is, a mapping from a *finite* set of discrete values to numbers in [0,1]). In this representation, we divide a region into blocks, one for each value of the probabilistic value, so that the area occupied by each value corresponds to its probability (Erwig and Walkingshaw 2013). The shape of the area does not matter in principle, but horizontally extended rectangles support the drawing of traces rather well. Here is how the probabilistic coin flip value looks like in this notation.




We call such a drawing a *probabilistic partition*, or *prop* for short. This notation captures three important features of a probabilistic value, namely (A) the fact that it may consist of multiple values, (B) that each value has a distinctive probability associated with it, and (C) that the probabilities of all values sum up to 1. (The exact position of each value as well as the order among values does not matter.) Note that showing the value probabilities, while helpful to users, is redundant as far as the spatial representation goes, since they are derived from the relative sizes of the partition blocks. Here is another example that represents the result of drawing a ball from an urn that contains the same number of red, green, and blue balls. Of course, the shown percentages are only approximations; depending on user preferences, using a higher precision or showing fractions may be preferable.




The most basic operation on a probabilistic value is to inquire about the probability of an event, which means to look up the probability in the mapping. In the prop representation this amounts to measuring the size of the occupied area. In general, an event is given by a predicate that selects a subset of values, and the probability for the event is given by the total relative size of the blocks corresponding to the subset of values. For example, the probability of picking either a green or blue ball is represented by ⅔ of the Urn rectangle.

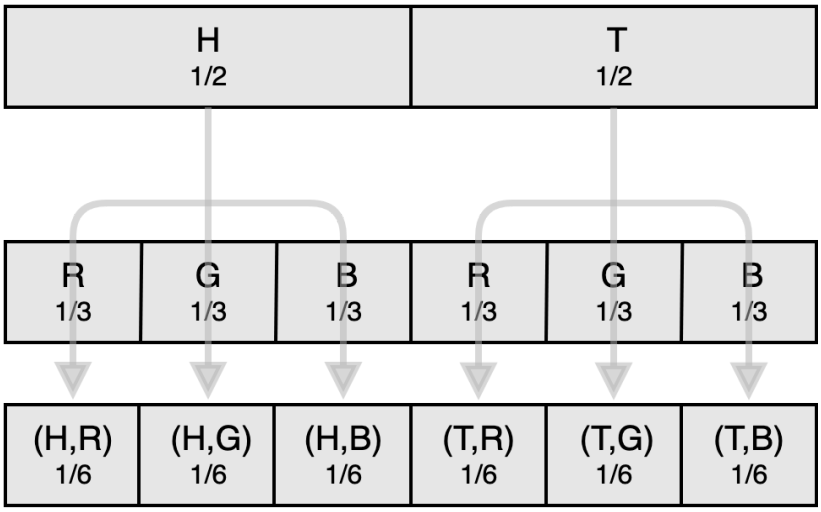
The programming with probabilistic values involves the transformation of probabilistic values, which raises the question of how to represent (the effect of) operations on probabilistic values. The first basic operation is ∩ to compute the joint probability of two events, which amounts to a combination of two probabilistic values such as Coin and Urn into one value Coin ∩ Urn. In terms of the employed spatial metaphor, the two props have to be superimposed to create a new prop in which the value combinations have to "share" the common space. For example, if we consider the joint probability of throwing a coin

and drawing a ball from an urn that contains the same number of red, green, and blue balls, we obtain the following prop (abbreviating Heads and Tails).

Coin ∩ Urn

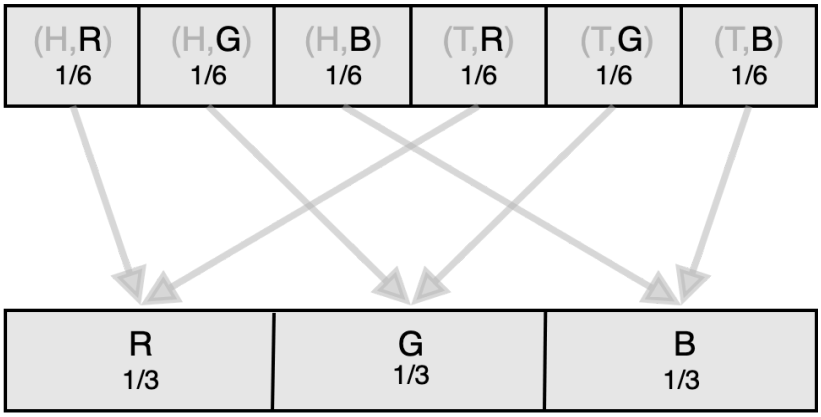
| (H,R) 1/6 | (H,G) 1/6 | (H,B) 1/6 | (T,R) 1/6 | (T,G) 1/6 | (T,B) 1/6 |
|---|---|---|---|---|---|

The values of the input props are represented as pairs, which reflects the ordering of the arguments of the ∩ operation. The generation of the joint probabilities can be illustrated using a flow diagram in which multihead arrows that indicate the flow of values from the first input prop via all the values of the second prop to the resulting prop.



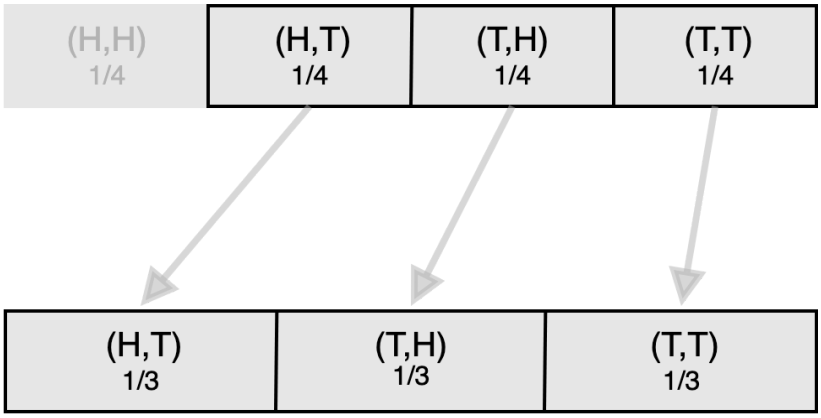

The creation of the result prop can be broken down into a sequence of three basic spatial operations. First, a copy of the second argument prop is created for each value in the first argument prop. Second, each copy is horizontally shrunk to the width of the block of its corresponding value from the first argument prop. Finally, the two partitions are intersected, and the combination of the values and the product of their probabilities annotate the blocks of the resulting partition.

The computation of marginal probability distributions can be also illustrated by a flow diagram. In this case, however, the arrows indicate a spatial union operation of all blocks that have the same value once a value has been marginalized out. For example, we can illustrate the computation of the marginal distribution for Urn as follows.




For each value of Urn we obtain two blocks (one for each value of Coin), and those two blocks are merged into one with a common label. The addition of the probabilities is naturally supported by the spatial metaphor, since the probabilities are homomorphic to the areas.

The computation of conditional probability distributions, which is probably (no pun intended) the most difficult to understand, can be broken down into three steps in the prop representation. We believe that it is this decomposition into simpler operations that provides the explanatory value of the prop representation. As an example, consider the following scenario: While throwing two coins, one comes up Tails. What is the probability that the other one is Heads? Many people believe that the probability is 50%, whereas it is actually 67%. We can illustrate this computation by starting with a joint probability for two coins. The first step is to select the blocks that correspond to the event "one comes up Tails," which are three blocks except the one containing two Heads. The three blocks define the probability space against which the query "What is the probability that the other one is Heads?" is posed. In the second step, the exclusion of the (H,H) block requires a resizing of the remaining blocks to occupy the whole probability space, which leads to a prop with 3 blocks that each have an associated probability of ⅓.




The third step requires the grouping of all blocks that match the query "What is the probability that the other one is Heads?" It is easy to see that this event occupies two blocks which occupy ⅔ of the prop. We can make this step explicit through an additional flow graph that merges the blocks, see (Erwig and Walkingshaw 2013).

At this point we have a method for explaining probabilistic values and computations with them in terms of spatial partitions. Specifically, computing joint probabilities can be explained through the intersection

of partitions, computing marginal probabilities can be explained by relabeling and union of partitions, and computing conditional probabilities can be explained filtering and resizing partitions. Many probabilistic programs are using these basic operations as building blocks, and we can, in principle, apply the partition-based explanation mechanism to explain the execution of such programs. In (Erwig and Walkingshaw 2013) we have used this approach to explain linear programs employing a story-telling metaphor (Erwig 2017), but this approach could also be used to explain non-linear representations as used in probabilistic reasoning in Bayesian networks.

To make this explanation approach practical, more work is required. First, the formalization of props is straightforward by building on our earlier work (Erwig and Schneider 1997). Second, we need a collection of programs as a benchmark for evaluating prop explanations. Third, we probably need several extensions to the notation. For example, to visually explain inference in Bayesian networks we need to represent conditional probability tables. Given the prop notation proposed here, there are some obvious ways for doing it. A more serious challenge is the question of how to scale the notation for bigger programs. This will probably require a notion of partial explanation, see (Cunha et al. 2018) where we also discuss a number of general principles for explanation languages.

## A furthest-first initiative: AI tools for African students and researchers

A common strategy in other fields of usability research and human-centric system design is to consider the "furthest first". This identifies the class of users who are least well served by the current generation of technology or user interface, and gives priority to meeting the needs of those people. It often turns out that a design strategy focused on those who are least well-served results in benefits for all users. Perhaps the most dramatic example of this strategy in programming language research was the Smalltalk language, initially proposed as part of the KiddiKomp project at Xerox PARC (Kay 1996), as one of the first programming languages that would be accessible to children. As it turned out, Smalltalk was more popular among adult programmers than among children (although the underlying principles did continue to benefit very young programmers, in particular through the Smalltalk architecture that underpins the Scratch language). But an even more dramatic outcome of the Smalltalk project was the way in which it required the developers to rethink many other aspects of the programming user interface, leading to the invention of icons, windows, menus, and many other elements of the modern GUI. A furthest-first approach to programming language research can have extraordinarily far-reaching impact.

One of the authors (Blackwell) is currently planning a year-long project, investigating the requirements for probabilistic programming among a population that are currently not well-served by existing languages for AI and data science research. He plans to collaborate with programmers, end-users and students in four different African countries (Uganda, Kenya, Ethiopia and Namibia). This builds on work by Church and others (Church, Simpson et al 2018) designing new tools and architectures for social science, public health and humanitarian research using text data obtained via SMS from regions with poor communications infrastructure. The application of AI methods in such contexts is often intended to empower local actors rather than follow the typical business models of software start-ups, meaning that greater access to configuration and control through accessible programming languages could be particularly important. Economic and political models may also differ from those in typical software technology contexts, for example considering whether those who contribute cognitive labour as a condition of access to media should be paid for their work. The specific aspirations of people in low income countries are also likely to be different, in shaping the imagination of what AI systems can possibly do - providing tools that allow people to explore their own imaginative ideas therefore offers support for innovations that might not have been anticipated in corporate laboratories or universities in wealthy countries.

Some research issues that may be productive in these furthest-first contexts include:

- Redraw system boundaries to consider interaction between labelling and modelling
- Consider reform of school curricula in maths and probability
- Explore AI as enabling structural innovation, not data science as statistical bureaucracy
- Acknowledge the economic and political tensions in cognitive labour such as labelling

We have already noted the specifically educational challenges and opportunities in the design of PPLs. These may present differently in low-income countries, and in relation to the mathematics curriculum taught in those countries. One interesting possibility is the role of probabilistic models in public discourse and activism, for example Carroll and Rosson's investigation of end-user development practices as part of participatory design for community informatics (2007). Experiments such as use of AgentSheets to discuss local community policy (Arias et al 1999) demonstrate the ways that simulation models might be integrated into other social contexts. We might describe this as "broad learning" in contrast to "deep learning", where a wider range of people are able to participate in the definition of models, rather than simply providing training data labels.

If teaching resources are limited, we might also consider following the design strategy of Sonic Pi and other educational languages, in which all tutorial content is integrated into the IDE itself. Sonic Pi has been successfully applied in an African context during the CodeBus Africa project that toured schools in 10 African countries over 100 days in 2017 (Bakić et al 2018).

## Conclusion

This discussion represents work in progress, and we expect discussion to continue at the PPIG workshop. Although some initial advances have been reported here (summarising earlier publications), this paper should primarily regarded as a manifesto for promising research directions in the development of more usable PPLs. Several of the authors have substantial research projects in progress, and readers interested in this topic are encouraged to follow developments from those who have contributed.

A key message emerging from our discussions is that the core principles in PPLs are going to be relevant to several very different classes of programmer, and that each of these classes will have very different usability requirements. At present, most users of PPLs are researchers. Researchers do have usability needs, including straightforward considerations of effective software engineering tools (debuggers, tracers, smart editors and so on). It would be possible to carry out more comprehensive task analysis of research work processes, for example as in Marasoiu's study of data scientists, to identify the activity profiles of these researchers and identify ways to optimise tools and notations that suit those profiles. It would also be possible to study the design representations that they already use, and integrate versions of these more closely into data science tools (for example, as in Gordon et al's (2014) Tabular alternative to "plates and gates" diagrams). A second class of programmer is the person who needs to define, explore and apply probabilistic models, but is unlikely to have specialised training (the end-user case). For this class, building on familiar representation conventions such as spreadsheets is likely to be valuable, in addition to supporting more casual and data-centric workflows. A third class is the pedagogic application, where the users are students acquiring an understanding of data science methods or even simple principles of Bayesian probability. For this class, conceptual clarity, minimal distracting syntax, and correspondence to naturalistic descriptions is likely to be important. Many standard considerations of software engineering, including debugging, version control etc have little relevance to the pedagogic situation, beyond the simple need to avoid distraction.

## A Probabilistic Postscript

Two of our authors (Advait Sarkar and Tobias Kohn) will not be available to attend the PPIG workshop, because they are getting married on the same day. Not to each other. We invite readers to create a PPL model that would estimate the prior likelihood of such an event, for any given research publication, and thus assess the risk that this might occur again in future. We also record our congratulations to Advait and Tobias!

## References

Arias, E. G., Eden, H., Fischer, G., Gorman, A., & Scharff, E. (1999, December). Beyond access: Informed participation and empowerment. In Proceedings of the 1999 conference on Computer support for collaborative learning (p. 2). International Society of the Learning Sciences.

Bakić, I., Puukko, O., Hämäläinen, V., and Subra, R. (2018). CodeBus Africa Study. Aalto Global Impact; Aalto University. Washington, DC : World Bank Group. http://documents.worldbank.org/curated/en/357931528940784006/Codebus-Africa-Study

Baydin, A. G., Pearlmutter, B. A., Radul, A. A., Siskind, J. M. (2018). Automatic Differentiation in Machine Learning: a Survey. Journal of Machine Learning Research (JMLR) 18 (153): 1–43. http://jmlr.org/papers/v18/17-468.html.

Baydin, A. G., Shao, L., Bhimji, W., Heinrich, L., Meadows, L. F., Liu, J., Munk, A., Naderiparizi, S., Gram-Hansen, B., Louppe, G., Ma, M., Zhao, X. Torr, P. H. S., Cranmer, K., Lee, V., Prabhat, Wood, F. (2019). Etalumis: Bringing Probabilistic Programming to Scientific Simulators at Scale. In Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis (SC19), November 17–22, 2019.

Bingham, E., Chen, J. P., Jankowiak, M., Obermeyer, F., Pradham, N., Karaletsos, T., Singh, R., Szerlip, P., Horsfall, P., Goodman, N. D. (2018): Pyro: Deep universal probabilistic programming. Journal of Machine Learning Research, 2018.

Blackwell, A.F. (2006). Psychological issues in end-user programming. In H. Lieberman, F. Paterno and V. Wulf (Eds.), End User Development. Dordrecht: Springer, pp. 9-30

Blackwell, A.F. (2017). End-user developers - what are they like? In F. Paternò and V. Wulf (Eds). New Perspectives in End-User Development. Springer, pp. 121-135.

Blei, D. M., Kucukelbir, A., McAuliffe, J. D.. (2017). Variational inference: A review for statisticians. Journal of the American Statistical Association, 112(518):859–877.

Carpenter, B., Gelman, A., Hoffman, M. D., Lee, D., Goodrich, B., Betancourt, M., Brubaker, M., Guo, J., Li, P., & Riddell, A. (2017). Stan: A probabilistic programming language. *Journal of statistical software*, *76*(1).

Carroll, J. M., & Rosson, M. B. (2007). Participatory design in community informatics. Design studies, 28(3), 243-261.

Carroll, J.M., Rosson, M.B., Isenhour, P., Ganoe, C., Dunlap, D., Fogarty, J., Schafer, W. and Van Metre, C., 2001. Designing our town: MOOsburg. International Journal of Human-Computer Studies, 54(5), pp.725-751.

Cheng, P. C. H. (2011). Probably good diagrams for learning: representational epistemic recodification of probability theory. Topics in Cognitive Science, 3(3), 475-498.

Church, L., Simpson, A., Zagoni, R., Srinivasan, S. and Blackwell, A.F. (2018). Building socio-technical systems for representing citizens voices in humanitarian interventions. In S. Tanimoto, S. Fan, A. Ko and D. Locksa (Eds), Proceedings of the Workshop on Designing Technologies to Support Human Problem Solving. University of Washington. pp. 19-21.

Cunha, J., Dan, M., Erwig, M., Fedorin, D. and Grejuc, A. (2018). Explaining Spreadsheets with Spreadsheets. *ACM SIGPLAN Conf. on Generative Programming: Concepts & Experiences (GPCE'18)*, 161-167.

Dillon, J. V., Langmore, I., Tran, D., Brevdo, E., Vasudevan, S., Moore, D., Patton, B., Alemi, A., Hoffman, M., Saurous, R. A. (2017): Tensorflow distributions. arXiv preprint arXiv:1711.10604.

Du Boulay, B. (1986): Some Difficulties of Learning to Program. Journal of Educational Computing Research, 2(1), pp. 57-73.

Edge, D., & Blackwell, A. (2006). Correlates of the cognitive dimensions for tangible user interface. Journal of Visual Languages & Computing, 17(4), 366-394.

Erwig, M. (2017). *Once Upon an Algorithm - How Stories Explain Computing*. MIT Press, Cambridge, MA.

Erwig, M. and Walkingshaw, E. (2013). A Visual Language for Explaining Probabilistic Reasoning. *Journal of Visual Languages and Computing, Vol. 24*, No. 2, 88-109.

Erwig, M. and Schneider, M. (1997). Partition and Conquer. *3rd Int. Conf. on Spatial Information Theory (COSIT'97)*, LNCS 1329, 389-408.

Gabry, J., Simpson, D., Vehtari, A., Betancourt, M., & Gelman, A. (2019). Visualization in Bayesian workflow. Journal of the Royal Statistical Society: Series A (Statistics in Society), 182(2), 389-402.

Gelman, A., Lee. D., Guo, J. (2015): Stan: A Probabilistic Programming Language for Bayesian Inference and Optimization. Journal of Educational and Behavioral Statistics, 40(5):530-543, 2015.

N. D. Goodman, J. B. Tenenbaum, and The ProbMods Contributors (2016). Probabilistic Models of Cognition (2nd ed.). Retrieved 2019-6-25 from https://probmods.org/

Goodman, N., Mansinghka, V., Roy, D. M., Bonawitz, K., Tenenbaum, J. B.(2012): Church: a language for generative models. arXiv preprint arXiv:1206.3255.

Gorinova, M.I., Sarkar, A., Blackwell, A.F. and Syme, D. (2016). A Live, Multiple-Representation Probabilistic Programming Environment for Novices. In Proceedings of CHI 2016, pp. 2533-2537.

Tom Hitron, Yoav Orlev, Iddo Wald, Ariel Shamir, Hadas Erel, and Oren Zuckerman. 2019. Can Children Understand Machine Learning Concepts?: The Effect of Uncovering Black Boxes. In Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems (CHI '19). ACM, New York, NY, USA, Paper 415, 11 pages. DOI: https://doi.org/10.1145/3290605.3300645

Hoffman, M. D., Gelman, A. (2014). The No-U-turn Sampler: Adaptively Setting Path Lengths in Hamiltonian Monte Carlo. Journal of Machine Learning Research, 15(1):1593–1623.

Gabry, J., Simpson, D., Vehtari, A., Betancourt, M., & Gelman, A. (2019). Visualization in Bayesian workflow. Journal of the Royal Statistical Society: Series A (Statistics in Society), 182(2), 389-402.

Gordon, A. D., Graepel, T., Rolland, N., Russo, C., Borgstrom, J., & Guiver, J. (2014). Tabular: a schema-driven probabilistic programming language. In ACM SIGPLAN Notices (Vol. 49, No. 1, pp. 321-334). ACM.

Alan C. Kay. 1996. The early history of Smalltalk. In History of programming languages---II, Thomas J. Bergin, Jr. and Richard G. Gibson, Jr. (Eds.). ACM, New York, NY, USA 511-598. DOI: https://doi.org/10.1145/234286.1057828

Ko, A.J., Abraham, R., Beckwith, L., Blackwell, A.F., Burnett, M., Erwig, M., Lawrence, J., Lieberman, H., Myers, B., Rosson, M.-B., Rothermel, G., Scaffidi, C., Shaw, M., and Wiedenbeck, S. (2011). The State of the Art in End-User Software Engineering. ACM Computing Surveys 43(3), Article 21.

Kohn, T., Komm, D.: Teaching Programming and Algorithmic Complexity with Tangible Machines. In: Pozdniakiv, S., Dagien, V. (eds): Informatics in Schools. Fundamentals of Computer Science and Software Engineering. ISSEP 2018. Lecture Notes in Computer Science, vol. 11169, Springer, Cham.

Le, T. A., Baydin, A. G., Wood, F. (2017): Inference compilation and universal probabilistic programming. In Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (AISTATS), volume 54 of Proceedings of Machine Learning Research, pages 1338–1348, Fort Lauderdale, FL, USA, 2017. PMLR.

Lister, R. (2011). Concrete and Other Neo-Piagetian Forms of Reasoning in the Novice Programmer. 13th Australasian Computer Education Conference (ACE 2011).

Minka, T., Winn, J., Guiver, J., Knowles, D. (2013): Infer.net 2.4, Microsoft Research Cambridge. URL: http://research. microsoft. com/infernet.

Pane, J. F., Myers, B. A., & Garlan, D. (2002). A programming system for children that is designed for usability (Doctoral dissertation, School of Computer Science, Carnegie Mellon University). http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.481.2364

Pennington, N. (1987). Comprehension strategies in programming. In Empirical studies of programmers: second workshop (pp. 100-113). Ablex Publishing Corp..

Pennington, N., Lee, A. Y., & Rehder, B. (1995). Cognitive activities and levels of abstraction in procedural and object-oriented design. Human-Computer Interaction, 10(2), 171-226.

Peyton Jones, S., Blackwell, A and Burnett, M. (2003). A user-centred approach to functions in Excel. In Proceedings International Conference on Functional Programming, pp. 165-176.

Repenning, A., & Ambach, J. (1996). Tactile programming: A unified manipulation paradigm supporting program comprehension, composition and sharing. In Proceedings 1996 IEEE Symposium on Visual Languages (pp. 102-109). IEEE.

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J.S., Silverman, B. and Kafai, Y.B., (2009). Scratch: Programming for all. Comm. ACM, 52(11), 60-67.

Scaffidi, C., Shaw, M., & Myers, B. (2005). Estimating the numbers of end users and end user programmers. In 2005 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC'05) (pp. 207-214). IEEE.

Spiegelhalter, D., Thomas, A., Best, N., Gilks, W. (1996): BUGS 0.5: Bayesian Inference Using Gibbs Sampling Manual (version ii). MRC Biostatistics Unit, Institute of Public Health, Cambridge, UK, pages 1–59,

Stan Development Team. (2017). ShinyStan: Interactive visual and numerical diagnostics and posterior analysis for Bayesian models. *R Package Version*, *2*.

Staton, S., Wood, F., Yang, H., Heunen, C., Jammar, O. (2016): Semantics for probabilistic programming: higher-order functions, continuous distributions, and soft constraints. In 2016 31st Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), pages 1–10. IEEE, 2016.

Stead, A., & Blackwell, A. F. (2014). Learning syntax as notational expertise when using drawbridge. In Proceedings of the Psychology of Programming Interest Group Annual Conference (PPIG 2014) (pp. 41-52).

Tran, D., Hoffman, M. D., Saurous, R. A., Brevdo, E., Murphy, K., Blei, D. M. (2017): Deep probabilistic programming. arXiv preprint arXiv:1701.03757

Van de Meent, J.-W., Paige, B., Yang, H., Wood, F. (2018): An Introduction to Probabilistic Programming. arXiv e-prints, Sep 2018

Watt, S. (1998). Syntonicity and the psychology of programming. Proceedings of PPIG 1998.

Wingate, D., Stuhlmueller, A., Goodman, N. (2011). Lightweight implementations of probabilistic programming languages via transformational compilation. In Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, pages 770–778.

Wood, F., Van de Meent, J.-W., Mansinghka, V. (2014): A new approach to probabilistic programming inference. In Artificial Intelligence and Statistics, pages 1024–1032, 2014.

Zhou, Y., Gram-Hansen, B. J., Kohn, T., Rainforth, T., Yang, H., Wood, F. (2019): LF-PPPL: A low-level first order probabilistic programming language for non-differentiable models. arXiv preprint arXiv:1903.02482