

УДК 004.432.4

Baklan I.V.

National Technical University of Ukraine “Igor Sikorsky Kyiv Polytechnic Institute”

Vasilenko V.G.

National Technical University of Ukraine “Igor Sikorsky Kyiv Polytechnic Institute”

Shyrii V.V.

National Technical University of Ukraine “Igor Sikorsky Kyiv Polytechnic Institute”

SYSTEM ANALYSIS OF ABSTRACT DATA TYPES IN PROBABILISTIC PROGRAMMING LANGUAGES

Today, there are quite a few different probabilistic programming languages that to some extent use the concepts of probability theory for their calculations. But we wanted to know what data types exist for solving probabilistic tasks. In the present paper we present a system analysis of abstract data types in selected languages of probabilistic programming. We revealed several moments. The first is that all the languages we choose use the data types of their „parent” programming languages. The second is that for the use of distributions and random variables, the built-in functions or methods in each of the languages are used. And writing your own can cause some difficulties. We tried to depict the main areas of use of modern probabilistic abstract data types in probabilistic programming languages.

Key words: Probabilistic programming, Abstract data types, Probabilistic programming languages, Programming languages.

Introduction. Today there are widespread applications of the probability models used for creation of the modern artificial intelligence in application-oriented statistics or in cognitive science. This results from the fact that they are connected to operation over probabilities and their probable outputs. However, probabilistic models tend to increase their complexity. Therefore, it is necessary to create new tools to provide a new integrated approach to the probabilistic representation of models. Therefore it is necessary to create new tools for support of a new integrated approach to probable representation of models. And probable a programming language it is provided. Languages allow you to create tools for describing complex probability distributions and implement an effective probabilistic output for an arbitrary computer program.

Probabilistic programming languages, in their simple form, extend the well-known deterministic programming language with primitive constructions for random choice [17]. However, over time, there was a creation of new tools for probabilistic inference and the emergence of new complex probabilistic simulation programs. The presence of a large number of probabilistic programming languages led to the idea

that there is a certain programming paradigm, the so-called probabilistic programming.

Analysis of recent research and publications. Basic principles language design and probabilistic programming were given in [8]. Also in this article describes the differences between Probabilistic Programming and Probabilistic Model Checking.

About each of probabilistic programming languages there are relevant article or the corresponding page on the Internet from their authors. Therefore we will list those languages about which we will speak. Namely: Church (MIT BCS/CSAIL) [5; 13], Anglican (MIT, Oxford University and DARPA PPAML) [2–4; 15], Venture (MIT BCS/CSAIL) [9; 11; 18], Infer.Net (Microsoft Research) [12], TensorFlow [6; 13] (Google) with libraries TensorFlow Distributions (Google, Columbia University) [1] and Edward (Columbia University) [7; 8; 16].

The objectives of the research. In each of the languages of probabilistic programming with the help of abstract types, the basic concepts of probability theory are realized: probability space set, random variable, probability, probability distribution. These concepts, in our opinion, must necessarily be implemented in languages of probabilistic programming.

In this article, we will analyze the implementation of the basic concepts of probability theory with abstract data types in probabilistic programming languages. Namely: Church, Anglican, Venture, Infer. Net, TensorFlow with libraries TensorFlow Distributions and Edward.

Church (MIT BCS/CSAIL). Let's start our analysis with the Church. Church – a universal language for describing stochastic generative processes. Church is based on the Lisp model of lambda calculus, containing a pure Lisp as its deterministic subset.

We will provide the partial description of language with [5]: “Church language is based upon a pure subset of functional language Scheme, a Lisp dialect”. What we can understand from the reading: Church uses the same abstract types as Scheme. Feature of Church is the fact that expressions are values and these expressions describe generative processes.

In Church there is one interesting feature – all computation returns to Church in the form of random variable [12]. Or it is possible to tell and so: Church values include Church expressions, and procedures; if $v_1 \dots v_n$ are Church values the list $(v_1 \dots v_n)$ is a Church value. For example, in a situation where one random variable is given, probability distribution is often called a random variable.

To specify sets, you can use built-in commands, such as list, vector and map. Using the built-in

Scheme types to represent probabilities, Church uses the type number. And it can be like integer or, if it is necessary to calculate probability, rational.

For calculation of probability a distribution function is used. It returns value from evaluating the body given env and values of formal parameters.

Anglican (MIT, Oxford University and DARPA PPAML). Because Anglican is like the Clojure programming language, it uses the same data types. Here are just Clojure data types are Java data types, which also means that all values in Clojure are regular Java reference objects.

For representation of sets, Anglican, as well as Church, uses the list, vector or hashmap types. Sample method returns a random sample and roughly corresponds to the default implementation of the sample checkpoint.

For storage and work with probability, Anglican uses library `java.lang.BigDecimal` – decimal values or other classes, because Java primitives are usually boxed in Clojure functions. The observe method returns the log probability of the value, which roughly corresponds to the default implementation of the observe checkpoint.

To determine the distribution used macro `defdist`. It takes care of defining a separate type for every distribution so that Clojure multimethods (or overloaded

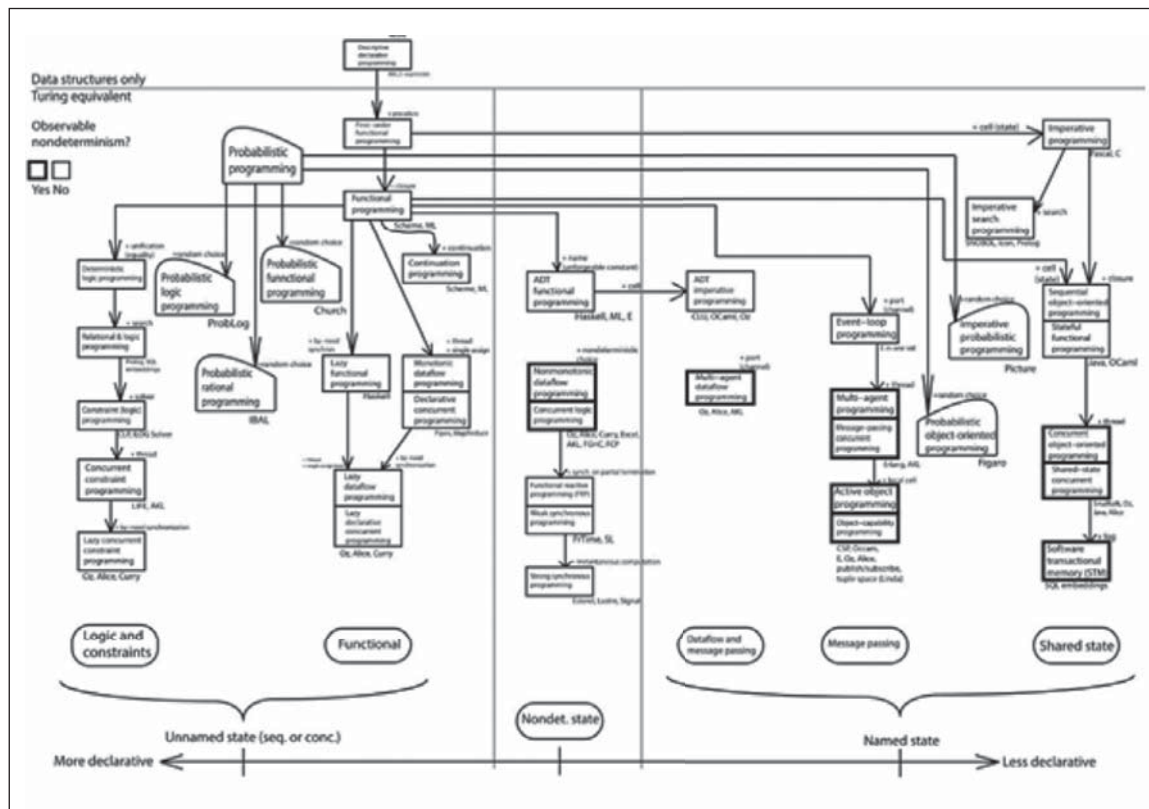


Fig. 1. Classification of programming paradigms [17]

methods) can be dispatched on distribution types when needed.

In addition to distributions, Anglican provides random processes, which define sequences of random variables that are not independent and identically distributed. Random processes are defined using the `defproc` macro and implement the `anglican.runtime/random-process` protocol. This protocol has two methods:

- `produce`, which returns the distribution on the next random variable in the sequence;
- `absorb`, which incorporates the value for the next random variable and returns an updated random process.

Venture (MIT BCS/CSAIL). Venture is essentially a Lisp-like higher-order language augmented with two novel abstractions:

- Probabilistic execution traces (PETs or abbreviated as “traces”) are a first-class object that represents the sequence of random choices that a probabilistic program makes. Each program subcomputation that yields a result corresponds to a random variable. PETs serve as the only native form of mutable storage in Venture, and map dynamic “addresses” assigned over the course of program execution to the manifest values taken by the program at those addresses;

- Stochastic procedures (SPs). SPs are used to encapsulate simple probability distributions, as well as user-space VentureScript programs and foreign probabilistic objects. An SP consists of a linked collection of programs and meta-programs that collectively describe aspects of a probabilistic program that are important for its use in modeling and inference. SPs are designed to allow simple probability distributions, user-space VentureScript, and foreign probabilistic programs to be treated uniformly as building blocks of complex probabilistic computations ;

The authors state that Venture uses the usual scalar and symbolic data types from the programming language Scheme. Also in Venture there is support for collections and additional datatypes corresponding to a primitive object from the probability theory and statistics. There is support for the stochastic procedure datatype for using compound procedures returned by `lambda`.

Here is a list of the most important values:

- Atoms – discrete items with no internal structure or ordering;
- Numbers – data types like as integer, rational, real, and complex;
- Collections – vectors, which map numbers to values and support $O(1)$ random access, and maps (map values to values) with support $O(1)$ amortized random access;

- Stochastics procedures – standard library components and can also be created by `Lambda` and others stochastic procedures.

Infer.NET (Microsoft Research). Infer.NET framework for running Bayesian inference in graphical models. Infer.NET provides the state-of-the-art message-passing algorithms and statistical routines needed to perform inference for a wide variety of applications.

In Infer.NET it is possible to create three types of variables: random (values are unknown and whose posterior distributions can be calculated during inference), constant (fixed values), observed (values not specified when the model is constructed, but are given before performing inference).

Infer.NET is used to create variables other than simple data types, such as `bool`, `double`, `int`, `enum`, `string`, `char`. `Vector` and `PositiveDefiniteMatrix` are used as vector and matrix types for creation of probabilistic sets. In addition, all of them and also `TDomain []`, `ISparseList <>`, `IList <>` can be used for discrete, continuous, multivariate and sequence distributions.

For greater convenience and possible simplicity, the developers provided methods for creating random variables with various distribution factors. It can pass in random variables as arguments e.g. `Variable <bool>` instead of `int`. In [11] you can see examples of such usage, as well as with the description and syntax on Infer.NET. Built-in functionality allows you to use different types of data parameters. For example, with discrete distribution.

TensorFlow (Google), library TensorFlow Distributions (Google, Columbia University) and Edward (Columbia University). TensorFlow is based on use of so-called tensors. We will give small definition about tensors. Tensors are simply mathematical objects that can be used to describe physical properties, just like scalars and vectors. In fact tensors are merely a generalization of scalars and vectors; a scalar is a zero rank tensor, and a vector is a first rank tensor [18].

The rank (or order) of a tensor is defined by the number of directions (and hence the dimensionality of the array) required to describe it. For example, properties that require one direction (first rank) can be fully described by a 3×1 column vector, and properties that require two directions (second rank tensors), can be described by 9 numbers, as a 3×3 matrix. As such, in general an n th rank tensor can be described by 3^n coefficients.

Tensors are used to represent the data structure in programs written in TensorFlow. Using tensors,

TensorFlow represents the probability space is an N-dimensional array or list. The tensor has a static type and a dynamic dimension.

TensorFlow provides several possibilities for creating so-called random tensors with different distributions. In this case, after each call and calculation, new random values are created.

Tensors can be of such data types: bool, half, float, float64, uint8, int8, int16, int32, int64, complex64, complex128, string. But you can also use standard data types with Python. For example, as bool, str, list or tuple.

More recently, for TensorFlow, another library of adaptation of the vision of probability theory to the modern deep-learning paradigm of end-to-end differentiable computation. It is called TensorFlow Distributions [1]. It is constructed on such two abstractions: Distributions and Bijectors. The first provides a collection of approximately 60 distributions with fast, numerically stable methods for sampling, log density, and many statistics. The second one allows composable volumetracking transformations with automatic caching. Together these enable modular construction of high dimensional distributions and transformations not possible with previous libraries.

Also, this year was presented Edward [15] – deep probabilistic programming library, which expands deep-learning research by enabling new forms of experimentation, faster iteration cycles, and improved reproducibility. Edward provides a language of random variables to construct a broad class of models: directed graphical models, stochastic neural networks, and programs with stochastic control flow. In Edward, random variable is an object parameterized by tensors. For Edward, the TensorFlow Distributions library has a backend.

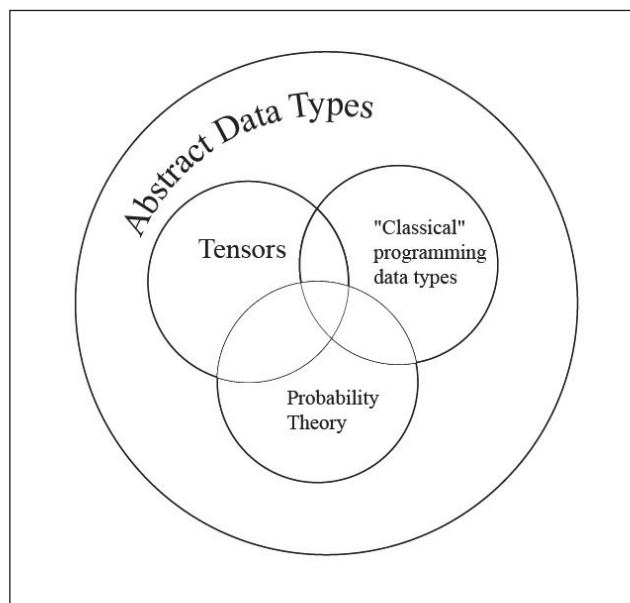


Fig 2. The fields of use of the modern abstract data types in probabilistic programming languages

Discussion and Conclusion. We will write short outputs about each of the probable languages selected by us. The Church language is quite simple in understanding because any computation by means of embedded functions allows to receive random variable. And language uses abstract data types from functional language Scheme.

Anglican uses data types from the language of Clojure, which in turn, takes them with Java. And as it seems to us, it is a direct continuation of the ideas that were laid in the Church, but with the use of modern frameworks.

Venture is another representative of Lisp's-like language. But adds two novel abstractions: probabilistic execution traces and stochastic procedures. The first are necessary for sequence of random choices, whereas the second – to allow simple probability distributions. The data types of the Scheme language are used.

By means of Infer.NET it is possible to create various variables, including random. At the same time using discrete, continuous, multivariate or sequence distributions. Also Infer.NET can be used in other .NET languages, for example, in C++/CLI, F#, IronPython and others.

Tensors are the main mathematical objects for the TensorFlow language. They allow to create multidimensional structures of random variables with different distributions given for calculation. For a task of sets used as the types which are built in language, that and with Python. This year two libraries which I expand functionality of language have been presented. In TensorFlow Distributions, two abstractions are added, which together enable modular construction of high dimensional distributions and transformations. The second library – Edward – allows develop difficult probabilistic models and their algorithms.

We will select several highlights. The first is that all the languages we choose use the data types of their “parent” programming languages. The second is that for the use of distributions and random variables, the built-in functions or methods in each of the languages are used. And the list of these distributions can be different. Depending on various factors (development experience, knowledge in the field of probability theory, etc.), the development of own probabilistic concepts can cause confusion.

In Fig. 2, we tried to depict the main areas of use of modern probabilistic abstract data types in probabilistic programming languages. Modern probabilistic programming languages are used as data types of “parent” languages, and new ones are introduced to describe and interact with probability theory. In future works, we will try to move away from the abstract data types of the main programming paradigms (imperative, declarative), thereby suggesting the use of probabilistic types.

References:

1. Alemi A., Dillon J.V., Langmore I. Tensorflow Distributions. 2017.
2. Anglican Homepage. URL: <https://probprog.github.io/anglican/index.html> (дата звернення 28.11.2017).
3. Anglican Language syntax, URL: <https://probprog.github.io/anglican/index.html> (дата звернення 28.11.2017).
4. Anglican Inference methods, URL: <https://probprog.github.io/anglican/inference/index.html> (дата звернення 28.11.2017).
5. Computation in Church, URL: http://projects.csail.mit.edu/church/wiki/Computation_in_Church (дата звернення 25.11.2017).
6. Constants, Sequences, and Random Values page, URL: https://www.tensorflow.org/api_guides/python/constant_op (дата звернення 28.11.2017).
7. Edward Homepage, URL: <http://edwardlib.org/> (дата звернення 27.11.2017).
8. Developing Custom Random Variables URL: <http://edwardlib.org/api/model-development> (дата звернення 27.11.2017).
9. Gordon A.D., Henzinger T.A., Nori A.V., Rajamani S.K. (2014, May). Probabilistic programming. Proceedings of the on Future of Software Engineering. 2014. P. 167–181.
10. Lu A. Venture: an extensible platform for probabilistic meta-programming: дис. ... канд. техн. наук.
11. Mansinghka V., Selsam D., Perov Y. Venture: a higher-order probabilistic programming platform with programmable inference.
12. Infer.NET 2.6. URL: <http://research.microsoft.com/infernet> (дата звернення 14.11.2017).
13. Probability Theory and The Meaning of Probabilistic Programs. URL: http://projects.csail.mit.edu/church/wiki/Probability_Theory_and_The_Meaning_of_Probabilistic_Programs (дата звернення 14.11.2017).
14. TensorFlow Homepage. URL: <https://www.tensorflow.org/> (дата звернення 27.11.2017).
15. Tolpin D., van de Meent J.W., Yang H. Design and Implementation of Probabilistic Programming Language Anglican. 2014.
16. Tran, D., Kucukelbir, A., Dieng, A. B., Rudolph, M., Liang, D. & Blei, D. M.: Edward: A library for probabilistic modeling, inference, and criticism.
17. Vasilenko V., Shyrii V., Baklan I. Modern programming paradigm – probabilistic programming. In XIII International scientific conference “Intellectual systems of decision-making and problems of computational intelligence”. 2017.
18. Venture Homepage. URL: <http://probcomp.org/venture/> (дата звернення 14.11. 2017).
19. What is a Tensor. URL: https://www.doitpoms.ac.uk/tlplib/tensors/what_is_tensor.php (дата звернення 26.09.2017).

**СИСТЕМНИЙ АНАЛІЗ АБСТРАКТНИХ ТИПІВ ДАНИХ
У ЙМОВІРНІСНИХ МОВАХ ПРОГРАМУВАННЯ**

Сьогодні існує безліч різних ймовірнісних мов програмування, які певною мірою використовують поняття теорії ймовірностей для розрахунків. Але ми хотіли б знати, які типи даних існують для вирішення ймовірнісних завдань. У роботі ми подаємо системний аналіз абстрактних типів даних вибраними мовами ймовірнісного програмування. Ми виявили кілька моментів. Перший момент полягає в тому, що всі вибрані нами мови, використовують типи даних своїх «батьківських» мов програмування. Другий полягає в тому, що для використання дистрибутивів і випадкових величин використовуються вбудовані функції або методи кожної з мов. І написання власного може призвести до деяких труднощів. Ми спробували зобразити основні сфери використання сучасних ймовірнісних абстрактних типів даних у ймовірнісних мовах програмування.

Ключові слова: ймовірнісне програмування, абстрактні типи даних, ймовірнісні мови програмування, мови програмування.

**СИСТЕМНЫЙ АНАЛИЗ АБСТРАКТНЫХ ТИПОВ ДАННЫХ
В ВЕРОЯТНОСТНЫХ ЯЗЫКАХ ПРОГРАММИРОВАНИЯ**

Сегодня существует множество различных вероятностных языков программирования, которые в определенной степени используют понятия теории вероятностей для расчетов. Но мы хотели бы знать, какие типы данных существуют для решения вероятностных задач. В работе мы представляем системный анализ абстрактных типов данных на выбранных языках вероятностного программирования. Мы обнаружили несколько моментов. Первый момент заключается в том, что все выбранные нами языки используют типы данных своих «родительских» языков программирования. Второй заключается в том, что для использования дистрибутивов и случайных величин используются встроенные функции или методы каждого из языков. И написание своих может вызвать некоторые трудности. Мы попытались изобразить основные сферы использования современных вероятностных абстрактных типов данных в вероятностных языках программирования.

Ключевые слова: вероятностное программирование, абстрактные типы данных, вероятностные языки программирования, языки программирования.