The semantics of Subroutines and Iteration in the Bayesian Programming language ProBT

R. LAURENT*, K. MEKHNACHA*, E. MAZER† and P. BESSIÈRE‡
*ProbaYes S.A.S, Grenoble, France
†University of Grenoble-Alpes, CNRS/LIG, Grenoble, France
‡University Pierre et Marie Curie, CNRS/ISIR, Paris, France

***Abstract*—Bayesian models are tools of choice when solving problems with incomplete information. Bayesian networks provide a first but limited approach to address such problems. For real world applications, additional semantics is needed to construct more complex models, especially those with repetitive structures or substructures. ProBT, a Bayesian a programming language, provides a set of constructs for developing and applying complex models with substructures and repetitive structures. The goal of this paper is to present and discuss the semantics associated to these constructs.**

***Index Terms*—Probabilistic Programming semantics , Bayesian Programming, ProBT**

I. INTRODUCTION

ProBT [1] was designed to translate the ideas of E.T. Jaynes [2] into an actual programming language. ProBT was defined around the year 2000 [3], [4] and it has been maintained and further developed since then[1]. ProBT is one of the first probabilistic programming languages (as *e.g.* [5], [6]), designed to specify stochastic inferences as Bayesian programs. It includes an exact inference engine as well as several MCMC methods for approximate inferences [4]. The ProbaYes company has been using ProBT for 13 years in industrial applications ranging from driving assistance to optimization of the energy consumption in buildings. With the same goals as [7], recent advances include prototyping dedicated hardware [8]–[10] to interpret ProBT programs.

Early robotic experiments [11] revealed the need for probabilistic submodels; they also pointed out the importance of correctly specifying Bayesian filters. In this paper we present the semantics of two constructs found in ProBT to define stochastic submodels and filters.

II. A BRIEF OVERVIEW OF PROBT

We define a *Bayesian program* as a means of specifying some knowledge structured as a probabilistic model, to which the programmer asks a question (or probabilistic query) to solve a task, which will be answered to by means of probabilistic inference. The constituent elements of a Bayesian program are presented figure 1.

[1]A free version of ProBT is available for academic use at http://www.probayes.com/fr/Bayesian-Programming-Book/downloads/.

$$\begin{cases} Description \begin{cases} Specification\ (\pi) \begin{cases} Variables \\ Decomposition \\ Forms \begin{cases} Parametric \\ Program \end{cases} \end{cases} \\ Identification\ (based\ on\ \delta) \end{cases} \\ Question \end{cases}$$

Figure 1. A Bayesian program is constructed from a *Description* and a *Question*. The *Description* is given by the programmer who writes a *Specification* of a model π and an *Identification* of its parameter values, which can be set in the program or obtained through a learning process from a data set δ. The *Specification* is constructed from a set of relevant variables, a decomposition of the joint probability distribution over these variables and a set of parametric forms (mathematical models) for the terms of this decomposition.

A. Description

The purpose of a description is to specify an effective method for computing a joint probability distribution on a set of N variables $\{X_1, X_2, \cdots, X_N\}$ given a set of experimental data δ and some specification π. This joint distribution is denoted as: $P(X_1 \wedge X_2 \wedge \cdots \wedge X_N | \delta \wedge \pi)$.

B. Specification

To specify preliminary knowledge, the programmer must go through the following three steps.

1) Define the set of relevant variables $\{X_1, X_2, \cdots, X_N\}$ on which the joint probability distribution is defined.
2) Decompose the joint probability distribution. The programmer defines the structure of a complex probabilistic model by combining simpler terms: priors and conditional probability distributions describing the probabilistic dependences between the considered variables.
   Formally, the programmer chooses a partition of $\{X_1, X_2, \cdots, X_N\}$ into K subsets, defined by the K variables $L_1, \cdots, L_K$ where each variable L_k is the conjunction of the variables $\{X_{k_1}, X_{k_2}, \cdots\}$ belonging to the k^{th} subset. The conjunction rule then leads to the following decomposition:

$$\begin{aligned} & P(X_1 \wedge X_2 \wedge \cdots \wedge X_N | \delta \wedge \pi) \\ = \; & P(L_1 | \delta \wedge \pi) \times P(L_2 | L_1 \wedge \delta \wedge \pi) \\ & \times \cdots \times P(L_K | L_{K-1} \wedge \cdots \wedge L_1 \wedge \delta \wedge \pi) \end{aligned} \quad (1)$$

3) Define the forms:

For each term of the decomposition (equation 1), the programmer chooses a parametric form (i.e., a mathematical function $f_\mu(L_k)$) or defines it as a question to another Bayesian program. In general, μ is a vector of parameters that may depend on the conditioning variables, on δ, or both. Learning takes place when some of these parameters are computed using the data set δ.

C. Questions

Given a description (*i.e.*, $P(X_1 \wedge X_2 \wedge \cdots \wedge X_N | \delta \wedge \pi)$), the programmer defines a question by partitioning $\{X_1, X_2, \cdots, X_N\}$ into three sets: the searched variables, the known variables, and the free variables. The variables $Searched$, $Known$, and $Free$ are defined as the conjunctions of the variables belonging to these sets. The question is defined as the set of distributions

$$P(Searched | Known \wedge \delta \wedge \pi) \ , \tag{2}$$

which comprises as many "instantiated questions" as there are possible values $known$ for the conjunction $Known$, each instantiated question being the probability distribution

$$P(Searched | known \wedge \delta \wedge \pi) \ . \tag{3}$$

III. Bayesian Subroutines

ProBT allows to build incrementally more and more sophisticated probabilistic models by combining several Bayesian programs. Indeed, some of the terms of the decomposition of a Bayesian program can be defined as calls to Bayesian subroutines (*i.e.* other Bayesian programs). It follows that, as in standard programming, it is possible to use existing probabilistic models to build more complex ones and to further structure the definition of complex descriptions by reusing some previously defined models.

There are several advantages to Bayesian programming subroutine calls: (i) as in standard programming, subroutines make the code more compact and easier to read; (ii) as in standard programming the use of a submodel allows to hide the details regarding the definition of this model; (iii) calling subroutines gives the ability to use Bayesian programs that have been specified and learned by others; (iv) contrary to standard function calls, since Bayesian subroutines are probabilistic, they do not transmit to the calling code a single value but a whole probability distribution.

IV. Iterations in Generic Bayesian Filters

Often a sequence of measurements helps to better characterize the state of a system. Bayesian filters serve this purpose. They may be seen as special cases of dynamic Bayesian networks and are often used to process time series of sensor readings. The following program (Equation 4) defines a generic Bayesian filter, for which the inference solving the question will iterate a simple computation on two building blocks: a transition model $P\left(S^t \mid S^{t-1}\right)$ and a sensor model $P\left(O^t \mid S^t\right)$.

$$Pr \begin{cases} Ds \begin{cases} Sp \begin{cases} Va : \begin{cases} S^t, \forall t \in [0, \ldots, T] : S^t \in D_S \\ O^t, \forall t \in [1, \ldots, T] : O^t \in D_O \end{cases} \\ Dc : \begin{cases} P\left(S^0 \wedge O^1, \ldots S^t \wedge O^t \ldots S^T \wedge O^T\right) = \\ P\left(S^0\right) \prod^{t \in [1 \ldots T]} \left(P\left(S^t \mid S^{t-1}\right) P\left(O^t \mid S^t\right)\right) \end{cases} \\ Fo : \begin{cases} P\left(S^0\right) = \text{Initial condition} \\ P\left(S^t \mid S^{t-1}\right) = \text{Transition Model} \\ P\left(O^t \mid S^t\right) = \text{Sensor Model} \end{cases} \\ Id : \text{learning from } \delta \end{cases} \\ Qu : P\left(S^T \mid o^1 \ldots o^T\right) \end{cases} \tag{4}$$

The question of the Bayesian program shown in Equation 4 can be adapted to address several interesting tasks:

- Filtering: $P\left(S^T \mid o^1 \ldots o^T\right)$: the purpose of this question is to infer the current state according to the past sequence of measurements.
- Smoothing: $P\left(S^k \mid o^1 \ldots o^k \ldots o^T\right)$ estimates a past state of the system by taking into account more recent measurements ($k < T$).
- Forecasting: $P\left(S^T \mid o^1 \ldots o^k\right)$ estimates the state of the system in the future based on the current measurements ($T > k$).

V. Applications

To illustrate the semantics of the proposed constructs we will detail two robotics applications. How to program a complex task with Bayesian subroutines: the night watchman task for a mobile robot; and how to program classical Hidden Markov Models (HMM) for motion recognition with a generic Bayesian filter.

References

[1] P. Bessière, E. Mazer, J. M. Ahuactzin, and K. Mekhnacha, *Bayesian programming*. CRC Press, 2013.

[2] E. T. Jaynes, *Probability Theory: The Logic of Science*. Cambridge University Press, Apr. 2003.

[3] O. Lebeltel, "Programmation bayésienne des robots," Thèse de doctorat, Institut National Polytechnique de Grenoble, Grenoble, France, 1999.

[4] K. Mekhnacha, J.-M. Ahuactzin, P. Bessière, E. Mazer, and L. Smail, "Exact and approximate inference in ProBT," *Revue d'Intelligence Artificielle*, vol. 21/3, pp. 295–332, 2007.

[5] N. Goodman, V. Mansinghka, D. Roy, K. Bonawitz, and J. Tenenbaum, "Church: A language for generative models," in *Proceedings of the 24th Conference on Uncertainty in Artificial Intelligence (UAI)*, 2008, pp. 220–229.

[6] V. Mansinghka, D. Selsam, and Y. Perov, "Venture: a higher-order probabilistic programming platform with programmable inference," *arXiv preprint arXiv:1404.0099*, 2014.

[7] E. M. Jonas, "Stochastic architectures for probabilistic computation," Ph.D. dissertation, Massachusetts Institute of Technology, 2014.

[8] M. Faix, E. Mazer, R. Laurent, M. Othman Abdallah, R. Le Hy, and J. Lobo, "Cognitive computation: A bayesian machine case study," in *Cognitive Informatics Cognitive Computing (ICCI*CC), 2015 IEEE 14th International Conference on*, 2015, pp. 67–75.

[9] A. Coninx, R. Laurent, M. A. Aslam, P. Bessière, J. Lobo, E. Mazer, and J. Droulez, "Bayesian Sensor Fusion with Fast and Low Power Stochastic Circuits," in *IEEE 1st International Conference on Rebooting Computing*, 2016.

[10] M. Faix, R. Laurent, P. Bessière, E. Mazer, and J. Droulez, "Design of Stochastic Machines Dedicated to Approximate Bayesian Inferences," *Transactions on Emerging Topics in Computing*, 2016.

[11] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005.