Stochastic Bayesian Computation for Autonomous Robot Sensorimotor Systems

Marvin Faix[1] and Jorge Lobo[2] and Raphael Laurent[3] and Dominique Vaufreydaz[4] and Emmanuel Mazer[5]

***Abstract*— This paper presents a stochastic computing implementation of a Bayesian sensorimotor system that performs obstacle avoidance for an autonomous robot. In a previous work we have shown that we are able to automatically design a probabilistic machine which computes inferences on a Bayesian model using stochastic arithmetic. We start from a high level Bayesian model description, then our compiler generates an electronic circuit, corresponding to the probabilistic inference, operating on stochastic bit streams. Our goal in this paper is to show that our compilation toolchain and simulation device work on a classic robotic application, sensor fusion for obstacle avoidance. The novelty is in the way the computations are implemented, opening the way for future low power autonomous robots using such circuits to perform Bayesian Inference.**

I. Introduction

Bayesian approaches have been successfully applied to autonomous robots when they must operate in environments not specially designed for them, where they must handle uncertainty and decide with incomplete data [1], [2], [3]. One of the drawbacks of such approaches can be the demands on the computational power, and insights on this can be found in biology. We want to go beyond the standard current paradigm of exact hardware running software, and explore approximate computing hardware that can support the required computations with enough accuracy.

This work was developed in the scope of the European BAMBI FET project (Bottom-up Approaches to Machines dedicated to Bayesian Inference, www.bambi-fet.eu). BAMBI's goal is to rely on Bayesian theory, and an understanding of computing by living beings, to design a non Von Neumann bio-inspired probabilistic machine. All living systems process information with uncertainty, and this occurs at all levels, from microscopic molecular scale up to complex human perception. An interesting approach to model these processes is Bayesian modelling. This kind of modelling is already used in behaviour prediction and decision making where Bayesian inference proves powerful to carry out such computation [2]. The study of information transmission in unicellular organisms, namely the *Chlamydomonas reinhardtii*, allows us to conjecture that probabilities are coded with a kind of binary telegraphic signals. Using a bitstream representation to encode probabilities, we design a bio-inspired probabilistic machine based on these principles. For an initial version we rely on clocked bit streams to build a machine that takes into account the uncertainty of its inputs and can performs the computations with a small low power circuit. One key benefit is that due to the stochastic representation, the circuit is tolerant to sporadic logic gate faults. This has been increasingly relevant, as the limits of Moore's law are ever imposing, with faults occurring when power is lowered or clock speeds increased in current evermore minute transistors.

In this paper we present our compilation tool chain for a first Bayesian machine based on coding probabilities with stochastic bitstreams. This allows the automatic generation of the circuit implementation. The machine computes an exact inference with approximate computation. A fast high level simulation platform was developed, and the full systems was tested on the robotic classic problem of obstacle avoidance.

In the next two sections we present inspiring works from the literature, and we clarify what is, for us, a probabilistic machine. In section IV we describe our compilation toolchain, that starts from the Bayesian model description, in a high level language programming, and provides the final circuit in VHDL (a hardware description language from which circuits can be synthesised). The following section presents the sensor fusion application as well as simulation and tests on the robot, followed by conclusions.

II. Related Work

Many applications in Robotics already use probabilistic inference and Bayesian theory [1], [2], [3]. In this work we use a specific Programming language named ProBT[4], [5] but many languages dedicated to probabilistic model exist Figaro[6], Blog[7]. In the same way as BAMBI project interests, other teams try to perform probabilistic inference in hardware [8], [9], [10].

Vigoda's architectures [8] use message passing algorithm [11] to propagate uncertainty and compute inferences. Close to the temporal coding of probabilities, Vigoda represents the probabilities with analog values. These architectures are used to perform exact inference. At the opposite end Mansinghka tries to perform Bayesian computation with approximate

*The work leading to these results was partially supported by the European Commission collaborative FET project BAMBI - Bottom-up Approaches to Machines dedicated to Bayesian Inference - FP7-ICT-2013-C, project number 618024 (www.bambi-fet.eu).

[1]Marvin Faix is a CNRS PhD student, Computer Science, Grenoble, France marvin.faix@inria.fr

[2]Jorge Lobo is a Researcher at the Institute of Systems and Robotics, and Professor at the Electrical and Computer Engineering Department of the University of Coimbra, Portugal jlobo@isr.uc.pt

[3]Raphael Laurent is a Research engineer, computer science, from Probayes Company, Grenoble, France. raphael.laurent@probayes.com

[4]Dominique Vaufreydaz is an Associate Professor working on multimodal perception from University Grenoble-Alpes, LIG Laboratory and Inria, Grenoble, France dominique.vaufreydaz@inria.fr

[5]Emmanuel Mazer is a Researcher in computer science from CNRS, Grenoble, France emmanuel.mazer@inria.fr

inference using a sampling method [9]. Jonas [10] followed this idea and designed samplers based on Markov Chain Monte Carlo algorithms to represent probability distributions.

In our approach, we are focused on the design of a probabilistic machine which computes exact inference with approximate computation, it means less precision. This is due to the stochastic arithmetic used, based on stochastic time coding to represent probabilities. The previous work of Von Neumann [12] in 1956 and Gaines [13] showed the interest of a such coding. Indeed the simplicity of basic operators is the main strength of this approach. However the computation time and the low accuracy have not allowed this way to compute to cope with the emergence of faster and better processors as the number of transistors on a chip grew at an astounding rate. The new interest of the community in these works is due to the current constraints with higher numbers of transistors on a chip, where fault tolerance, low power solutions, and robustness to noise are now important factors (see for example [14], [15]).

The TrueNorth project [16] from IBM is also interested in designing a non Von Neumann bio-inspired architecture. It proposes a neuromorphic system for neuronal networks and uses fixed-point arithmetic units to compute the output of a neuron knowing its inputs. In this point we differ with this approach because we think that the use of probabilities is a natural way to handle the uncertainty which shapes the world [17].

III. What is a Bayesian Machine ?

In this part we specify what is a probabilistic machine dedicated to Bayesian inference in our context. We are focused on soft evidences (as opposed to hard evidences) which are probability distributions over an evidence. A Bayesian machine is a machine which resolves an inference problem. This machine takes soft evidences as inputs and outputs a probability distribution over the searched variables.

A. Soft Evidences

In the context of Bayesian model hard evidence could be defined as a deterministic observation whereas soft evidence represents uncertainty over known variables. In this case a soft evidence is represented by a probability distribution over the corresponding evidence variable. In our robotic sensor fusion application, soft evidences describe the precision of the sensor reading.

B. Structure of A Bayesian Machine

A Bayesian machine takes soft evidences as inputs. The outputs are probability distributions over searched variables. With this representation we ensure homogeneity between inputs and outputs; both are probability distributions. The third set of signals are model parameters of the joint distribution which are constant probability values. Each probability value is coded as a bitstream. Because bitstreams are drawn from a Bernoulli sequence, we use biased PRNG (Pseudo Random Number Generator) to generate the constant parameters of the model, a fast biased TRNG (True Random Number Generator) [18] could be used to have the randomness in the circuit.

Inside the probabilistic machine there are two different computation blocks which correspond to the sums and products to be computed for the exact inference. Figure 1 shows the structure of a probabilistic machine for a very simple example. We infer over variable M knowing two soft evidences $\tilde{P}(D_1)$ and $\tilde{P}(D_2)$, $P(M)$ is the prior of M, $P(D_1)$ and $P(D_2)$ are the constant parameters of the model.

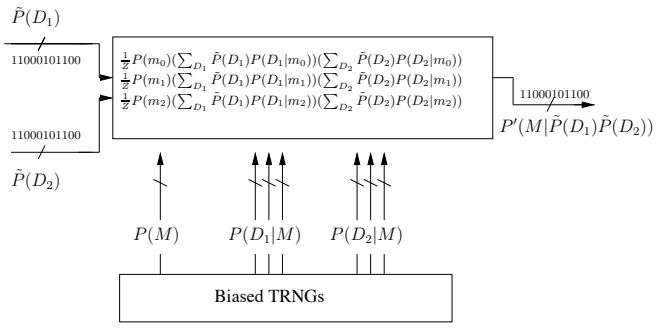

Fig. 1. The probabilistic machine: its inputs are the soft evidences (arrows incoming from the left) and its output is the probability distribution over the variable of interest, in this case M. The probability values of the internal parameters of the terms of the joint distribution are set by a set of TRNGs (True Random Number Generators)

IV. The Compilation ToolChain

In this part we explicit the compilation tool chain which starts from a Bayesian model described in a Bayesian programming language, ProBT [5], and automatically designs the probabilistic machine which implements the inference computation over the model described. The electronic design is a VHDL file which can be used both with our simulation platform or to implement the circuit on a FPGA (Field Programmable Gate Array, a reconfigurable logic device).

A. Bayesian Program

A Bayesian machine can be defined using properties, notation and formalism of Bayesian programming [5]. To design our machine we need to define the Bayesian model, the soft evidence inputs, the constant parameters of the model and the inference to compute. A Bayesian program is a program which describes the probabilistic link between variables. A general model is a joint probability distribution over a set of discrete and finite variables: $P(S \wedge K \wedge F)$ where S, K and F are respectively *Searched*, *Known* and *Free* variables of our model. Each variable could be a conjunction of variables, for example $K = K_1 \wedge ... \wedge K_i$. The inputs of the machine are soft evidences, so we define the soft evidences over the *Known* variables K_i, $\tilde{P}(K_i)$ which is a probability distribution over K_i. To do an inference, as for instance computing $P(S|\tilde{P}(K_1) \wedge ... \tilde{P}(K_i))$, is equal to compute the following expression, where Z is the normalisation constant:

$$P(S|\tilde{P}(K_1) \wedge ... \wedge \tilde{P}(K_i)) = \frac{1}{Z}\sum_{K_1}\tilde{P}(K_1)\ldots\sum_{K_i}\tilde{P}(K_i)\sum_{F}P(S \wedge K \wedge F), \quad (1)$$

$$Z = \sum_{S}\left(\sum_{K_1}\tilde{P}(K_1)\ldots\sum_{K_i}\tilde{P}(K_i)\sum_{F}P(S \wedge K \wedge F)\right) \quad (2)$$

ProBT generates a computation tree which is simplified thanks to the Successive Reduction Algorithm (SRA) [4], itself similar to the sum-product algorithm [11]. The simplification is possible thanks to the dependencies description between variables in the model defined in ProBT. The simplified computation tree is given as input to the compiler which will design automatically the machine.

B. Compilation

The compiler starts with the Bayesian model description written in ProBT and it generates the corresponding VHDL file which describes the dedicated probabilistic computer. In exact inference the expression to compute is composed of many sums and products. This is the fundamental point of using the temporal coding of probabilities. Sum and product operators are easy to implement and need few components using this kind of coding. In this case the terms stochastic sum and stochastic product are used. Every probability in our model is coded with a bitstream. Assume that two bitstreams BS_1 and BS_2, encoding the probability values P_1 and P_2, are decorrelated, the result of these two bitstreams in inputs of an AND gate is a bitstream BS_p which encodes the product $P_1 \times P_2$. In the same way, just using an OR gate extended with a memory we can sum probabilities. These operation are non-normalised. Normalisation can be implemented thanks to a JK flip flop. A schema of these components is given in figure 2. More information about these components and their utilisation is available in [19].

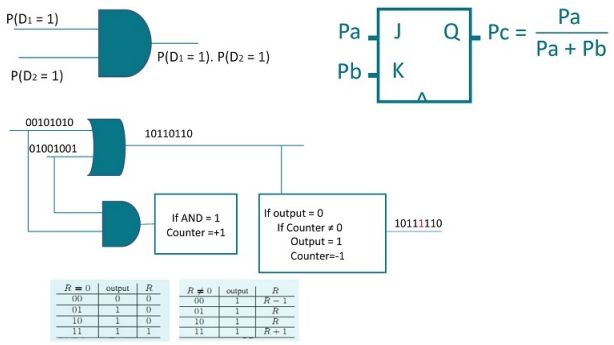

Fig. 2. The three components for product, sum and division in hardware using bitstream representation

C. First Bayesian Machines

We have previously validated our approach on two examples, see [19]. The first one is a very simple example to explicit the corresponding VHDL file from the computational tree given by ProBT. Let a joint probability distribution $P(M \wedge D_1 \wedge D_2)$ over three variables, the searched variable of the *Model* M and two known *Data* D_1 and D_2. The inference to compute the expression $P(M|P(D_1)P(D_2))$:

$$P(M) = \frac{1}{Z} P(M)(\sum_{D_1} \tilde{P}(D_1)P(D_1|M))(\sum_{D_2} \tilde{P}(D_2)P(D_2|M)), \quad (3)$$

where $\tilde{P}(D_1)$ and $\tilde{P}(D_2)$ are the soft evidences. By nature of Bayesian inference, the circuit is duplicated several times. Indeed only the inputs and constants model signals change. Figure 3 shows a subblock of RTL (Register Transfer Level) description generated synthesising the VHDL code to compute a subpart of the expression 3 which is $\sum_{D1} \tilde{P}(D_1)P(D_1|m)$.

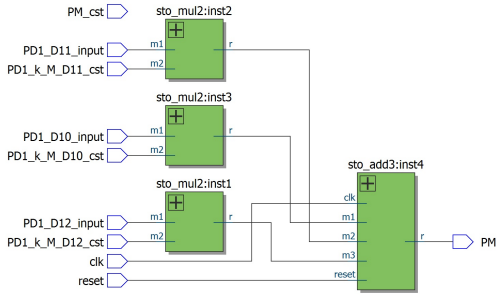

Fig. 3. RTL (Register Transfer Level) description for the stochastic circuit computing $\sum_{D1} \tilde{P}(D_1)P(D_1|m)$

The second example is a Bayesian filter as an application in telecommunication. The goal was to synchronise two Linear Feedback Shift Registers (LFSRs). The first LFSR emits bits corresponding to its state and the second one receives these bits through a noisy transmission chain. A mathematical study shows that inference of the emitter state is done by computing the following expression:

$$P(S_T = s|\tilde{P_T}(O_T)) = \frac{1}{Z} P'(a(s))P_{Out(s)}(O_T) \ , \quad (4)$$

where s is the state, $\tilde{P_T}(O_T)$ is the soft evidence over the received bit, $a(s)$ is the ancestor state of s (see [19] for more detail). Figure 4 shows the corresponding architecture for the LFSR synchronisation. We have shown in [19] that the LFSR synchronisation circuit works with our architecture with transmission error less than 30%, table I.

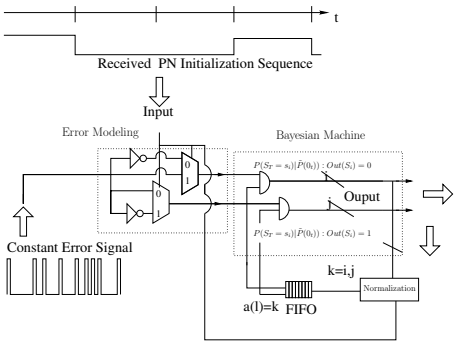

Fig. 4. The architecture of a circuit implementation of a Bayesian filter using stochastic arithmetic for LFSR synchronisation.

The mathematical study of the LFSR synchronisation application drastically reduced the number of components needed to compute the inference. In the following section we present out target robotic application that provides a bigger challenge concerning the size of the Bayesian machine.

Transmission Error	0%	10%	20%	30%
Correctly Recognized States	198	188	184	177
Iterations Before Synchronization	2	57	59	64

TABLE I
RESULTS LFSR SYNCHRONISATION

V. AN AUTONOMOUS SENSORIMOTOR SYSTEM

We chose a classic robotic application to show our solution on a problem with a bigger number of variables and components. We next present the implemented autonomous sensorimotor system, going over the the objectives of the sensor fusion, defining the sensor model, the actual implementation of such an architecture and the results obtained.

A. Objectives

The goal is to allow an autonomous robot to adjust its trajectory to avoid obstacles given a distance estimate provided by two types of sensors. The infrared sensors and ultrasonic sensors are fused to increase precision over the distance estimation. The robot is shown in figure **??**. The result of this inference is the probability distribution over the rotation velocity enabling the robot to avoid the obstacle while moving forward.

B. Sensor Model

Each sensor has is own model depending on the distance. We define three levels of distance: close, medium and far. The robot turns by increasing the velocity of the wheel corresponding to the opposite side of the direction wanted. We define five rotation velocities: speed on the left, half speed on the left, null, half speed on the right, speed on the right. We set an uniform distribution over the prior distance variables. Both infrared and ultrasonic sensors have 80% confidence on the reading knowing the distance. We know the rotation velocity knowing the distance model too. A piece of ProBT code which describes this system is given in appendix.

C. Fusion

We define our joint distribution model:

$$\begin{aligned}&P(D_0 \wedge D_1 \wedge D_2 \wedge IR_0 \wedge IR_1 \wedge IR_2 \wedge US_0 \wedge US_1 US_2 \wedge V_{ROT})\\ &= P(V_{ROT}|D_0 D_1 D_2) \times P(D_0) \times P(D_1) \times P(D_2)\\ &P(IR_0|D_0) \times P(IR_1|D_1) \times P(IR_2|D_2)\\ &P(US_0|D_0) \times P(US_1|D_1) \times P(US_2|D_2)\end{aligned} \tag{5}$$

This is the conjunction of variables defined in the sensor model. It means conjunction between the three priors variables, the three conditionals variables over IR (infrared) variable, the three conditionals variables over US (ultrasonic) variable and the conditional variable over rotation velocity knowing distances. Inference over this model corresponds to asking $P(V_{ROT}|\tilde{P}_{IR_0}\tilde{P}_{IR_1}\tilde{P}_{IR_2}\tilde{P}_{US_0}\tilde{P}_{US_1}\tilde{P}_{US_2})$ and computing the following expression 6:

$$\begin{aligned}&P(V_{ROT}|\tilde{P}_{IR_0}\tilde{P}_{IR_1}\tilde{P}_{IR_2}\tilde{P}_{US_0}\tilde{P}_{US_1}\tilde{P}_{US_2})\\ =\ &\sum_{D_2}\Bigg[P(D_2)\sum_{US_2}\tilde{P}(US_2)P(US_2|D_2)\\ &\sum_{D_1}\Bigg(P(D_1)\sum_{IR_1}\tilde{P}(IR_1)P(IR_1|D_1)\\ &\sum_{D_0}\Big(P(D_0)\sum_{US_0}\tilde{P}(US_0)P(US_0|D_0)\\ &\sum_{IR_0}(\tilde{P}(IR_0)P(IR_0|D_0)) \times P(VROT|D_0D_1D_2)\Big)\\ &\sum_{US_1}\tilde{P}(US_1)P(US_1|D_1)\Bigg)\sum_{IR_2}\tilde{P}(IR_2)P(IR_2|D_2)\Bigg]\end{aligned} \tag{6}$$

D. Circuit Implementation for Sensorimotor Fusion

The generated circuit involves about 2500 components and several thousands of signals. The circuit is duplicated five times, indeed the searched variable $P(V_{ROT})$ has five possible values $P(V_{ROT_0})...P(V_{ROT_4})$. So the circuit is fully parallel in terms of the cardinality of the searched variable $P(V_{ROT_i})$. The important thing to observe is the number of stages in the circuit. The time dilution of probabilities, due to the many required products, decreases drastically the probabilistic value. So the bitstream representing the probability value is filled with a lot of 0. The bitstream size has to be very long to evaluate this value because of our temporal coding choice.

To use the robot platform, a simulation c++ code is generated from the VHDL code. It exactly corresponds to the VHDL code in terms of component and signals.

E. Simulation and Results

To simulate our circuit we used the c++ code which takes as inputs the sensors reading, computes the probability distributions and send the order for the rotation velocity. The given linear velocity is $0.2\text{m}s^{-1}$ because of the time computation of our simulator. Indeed for a bitstream size of 5000, the simulator needs $86ms$ to send an order. This computation time is linear with the bitstream length used to the computation in the simulator. The specification for this robot was $100ms$ between two orders with this linear velocity. So if we want to increase the linear velocity we need to make simulations faster or decrease the bitstream size.

These results obviously show that a bigger bitstream size increases the precision over the rotation velocity probability distribution. The result over the variable V_{ROT_0} is volatile because of the low probability value for this variable. A final validation with the robot avoiding various obstacles was done. In an indoor office scenario, the robot successfully avoided obstacles using the Bayesian machine implemented on the simulated circuit. A video is available at `http://www.bambi-fet.eu/a-robot-controller-powered-with-the-bayesian-machine-1/`, and image 5 shows some of the frames where you can see

Bit stream size	Tool	V_{ROT_0}	V_{ROT_1}	V_{ROT_2}	V_{ROT_3}	V_{ROT_4}
without Bit stream	ProBT	0.005	0.012	0.068	0.293	0.622
5000	simulator	0	0	0.052	0.21	0.74
	error	n.a	n.a	24%	28%	19%
10000	simulator	0	0	0.040	0.320	0.640
	error	n.a	n.a	41%	9%	2.9%
100000	simulator	0.004	0.007	0.064	0.331	0.594
	error	25%	4%	6%	13%	4.5%
1000000	simulator	0.003	0.013	0.068	0.289	0.628
	error	41%	8%	0%	1.5%	1%

TABLE II
COMPARISON BETWEEN THEORETICAL RESULT AND SIMULATION FOR DIFFERENT BITSTREAM SIZE

Fig. 5. Robot performing obstacle avoidance using the Bayesian machine.

the robot and the obstacles put in its path, in this case the corridor walls.

VI. CONCLUSIONS

We presented our work on Bayesian machine using stochastic arithmetic to compute exact Bayesian inference. These non Von Neumann architectures face the classic electronics constraints: fault tolerant circuit design, low power, energy efficient and robust circuit, especially because the probability coding intrinsically deals with these constraints. The temporal coding is one of the main strength of our approach, it relies with living beings model computation studied in BAMBI project. For us, a bio-inspired machine is a machine which takes probability distribution in inputs, computes with an arithmetic dedicated to the probabilistic model, transmits information with stochastic binary signals representing temporal coding of probabilities and outputs probability distribution.

Now we are able to transform any Bayesian program into a Bayesian machine, which computes a specific inference, described in VHDL language. This description can both be simulated with our specific high level simulation platform or synthesized and emulated on a FPGA. Our method has been tested in [19] and the automatic compilation toolchain presented was tested on a classic robotic application which is the sensor fusion enabling the robot to know the distance of an obstacle and avoid it. It is possible to reduce the circuit size by the number of states of the searched variable $P(V_{ROT})$ designing multiplexer in inputs of components which need directly a soft input or parameter constant value of the system. In this case multiplexer selects the right soft inputs and parameter constant values of the model for the corresponding $P(V_{ROT_i})$ computed.

The main limitations of this work are the classic limits in exact Bayesian computation, as the number and dimension of variables increases, problems become intractable. Problems intractable with classic computations on standard processors are still intractable with our method. We are now working on an approximate inference compiler to scale some intractable Bayesian problems in exact inference.

APPENDIX

```
# priors
PD0= plUniform(D0)
PD1= plUniform(D1)
PD2= plUniform(D2)
# sensor IR model
PIR =[0.8, 0.1, 0.1,0.1, 0.8, 0.1,0.1, 0.1, 0.8]
PIR0_k_D0=plDistributionTable (IR0, D0, PIR)
PIR1_k_D1=plDistributionTable (IR1, D1, PIR)
PIR2_k_D2=plDistributionTable (IR2, D2, PIR)
# sensor US model
PUS= [0.8, 0.1, 0.1,0.1, 0.8, 0.1,0.1, 0.1, 0.8]
PUS0_k_D0=plDistributionTable (US0, D0, PUS)
PUS1_k_D1=plDistributionTable (US1, D1, PUS)
PUS2_k_D2=plDistributionTable (US2, D2, PUS)
# Velocity rotation model
PVROT_k_D0D1D2
=plDistributionTable(VROT, D0^D1^D2, PVROT)
```

Fig. 6. ProBT code with the sensor specification.

ACKNOWLEDGEMENT

This work was made possible thanks to two grants from CNRS: Nano-Bayes and Defi-Bayes, and thanks to the EU collaborative FET Project BAMBI FP7-ICT-2013-C, project number 618024.

REFERENCES

[1] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics*. MIT Press, 2005.

[2] Pierre Bessière, Christian Laugier, and Roland Siegwart. *Probabilistic reasoning and decision making in sensory-motor systems*, volume 46. Springer Science & Business Media, 2008.

[3] João Filipe Ferreira and Jorge Dias. *Probabilistic approaches to robotic perception*. Springer, 2014.

[4] Kamel Mekhnacha, Juan-Manuel Ahuactzin, Pierre Bessière, Emmanuel Mazer, and Linda Smail. Exact and approximate inference in ProBT. *Revue d'intelligence artificielle*, 21(3):295–331, 2007.

[5] Pierre Bessière, Emmanuel Mazer, Juan Manuel Ahuactzin, and Kamel Mekhnacha. *Bayesian programming*. CRC Press, 2013.

[6] Avi Pfeffer. Practical probabilistic programming. In *Inductive Logic Programming*, pages 2–3. Springer, 2011.

[7] Brian Milch, Bhaskara Marthi, Stuart Russell, David Sontag, Daniel L Ong, and Andrey Kolobov. BLOG: Probabilistic models with unknown objects. *Statistical relational learning*, page 373, 2007.

[8] Benjamin Vigoda. *Analog logic: Continuous-Time analog circuits for statistical signal processing*. PhD thesis, Massachusetts Institute of Technology, 2003.

[9] Vikash Kumar Mansinghka. *Natively probabilistic computation*. PhD thesis, Massachusetts Institute of Technology, 2009.

[10] Eric Michael Jonas. *Stochastic architectures for probabilistic computation*. PhD thesis, Massachusetts Institute of Technology, 2014.

[11] Judea Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann Publishers Inc., 1988.
[12] John Von Neumann. Probabilistic logics and the synthesis of reliable organisms from unreliable components. *Automata studies*, 34:43–98, 1956.
[13] BR Gaines. Stochastic computing systems. In *Advances in information systems science*, volume 2, pages 37–172. Springer, 1969.
[14] Weikang Qian, Xin Li, Marc D Riedel, Kia Bazargan, and David J Lilja. An architecture for fault-tolerant computation with stochastic logic. *IEEE Transactions on Computers*, 60(1):93–105, 2011.
[15] Peng Li, Weikang Qian, and David J Lilja. A stochastic reconfigurable architecture for fault-tolerant computation with sequential logic. In *30th International Conference on Computer Design (ICCD)*, pages 303–308. IEEE, 2012.
[16] Steven K Esser, Alexander Andreopoulos, Rathinakumar Appuswamy, Pallab Datta, Davis Barch, Arnon Amir, John Arthur, Andrew Cassidy, Myron Flickner, Paul Merolla, et al. Cognitive computing systems: Algorithms and applications for networks of neurosynaptic cores. In *Neural Networks (IJCNN), The 2013 International Joint Conference on*, pages 1–10. IEEE, 2013.
[17] Edwin T Jaynes. *Probability theory: the logic of science*. Cambridge university press, 2003.
[18] Abdelkarim Cherkaoui, Viktor Fischer, Laurent Fesquet, and Alain Aubert. A very high speed true random number generator with entropy assessment. In *Cryptographic Hardware and Embedded Systems-CHES 2013*, pages 179–196. Springer, 2013.
[19] Marvin Faix, Emmanuel Mazer, Raphael Laurent, and Jorge Lobo. Cognitive computation: a bayesian machine case study. In *Cognitive Informatics and Cognitive Computing*. IEEE, 2015.