Design of Stochastic Machines Dedicated to Approximate Bayesian inferences

Marvin Faix, Raphä El Laurent, Pierre Bessière, Emmanuel Mazer, Jacques Droulez

Design of Stochastic Machines Dedicated to Approximate Bayesian Inferences

Marvin FAIX , Raphaël LAURENT, Pierre BESSIÈRE, Emmanuel MAZER and Jacques DROULEZ

Abstract—We present an architecture and a compilation toolchain for stochastic machines dedicated to Bayesian inferences. These machines are not Von Neumann and code information with stochastic bitstreams instead of using floating point representations. They only rely on stochastic arithmetic and on Gibbs sampling to perform approximate inferences. They use banks of binary random generators which capture the prior knowledge on which the inference is built. The output of the machine is devised to continuously sample the joint probability distribution of interest. While the method is explained on a simple example, we show that our machine computes a good approximation of the solution to a problem intractable in exact inference.

Index Terms—Stochastic computing, Bayesian Programming.

1 INTRODUCTION

This project is inspired by E.T. Jaynes. His book, "Probability as Logic" [1], proposes probability as an extension of logic to reason with incomplete and uncertain information. Since the beginning of our work in the 90s [2], [3], it has always been our goal to build a machine able to perform this type of reasoning on specialized hardware. We started by designing a method to automate probabilistic reasoning called *Bayesian Programming* [4] and an associated programming langage *ProBT*[1] to specify and perform Bayesian inferences on standard computers. The formalism and the inference engine of ProBT have been used to develop many academic [5], [6], and industrial applications[2].

Recently the European project BAMBI[3] gave us the opportunity to pursue our project. We started this project by designing and implementing machines performing exact inference with stochastic arithmetics on an FPGA. While these non conventional machines could already be used to synchronize a pair of Linear Feedback Shift Registers [7] or to control a robot [8], they could not handle any problem of reasonable complexity. Here we describe a second generation of machines based on sampling which allow to approximate the solution of otherwise intractable problems.

The paper is organized as follows. Related works with similar goals and approaches are described next. Section 3 presents stochastic bitstreams and explains why we made the choice to sample binary probabilistic variables. In particular, it introduces the two only building blocks required to build the stochastic machine: Conditional Distribution Elements (*CDEs*) and Extended C-Elements (*ECEs*). *CDEs* are used to generate programmable and addressable stochastic bitstreams. *ECEs* extend Muller C elements and are used for inference. In the following section we recall the Gibbs algorithm in the case of binary variables and show how it may be implemented by combining our previously defined building blocks. Then, in Section 5 we describe our compilation toolchain. It starts from a Bayesian program on discrete variables, transforms it into an equivalent program on binary variables and finally compiles it into a VHDL circuit specification. Finally we use a simulator to demonstrate the performance of the obtained circuits performing approximate inference on an intractable problem.

2 RELATED WORK

The design of unconventional or non Von Neumann machines is essentially driven by the idea of reducing the power consumption of computing devices. It is also related to the need to process larger amounts of data while being closer as ever to the limits imposed by physical laws on current processors and the end of Moore's law. The need for more data processing with less energy comes from the common availability of extremely large data bases and from applications related to sensor fusion and interpretation.

Massively parallel and multi-core architectures are the current response to these challenges but several projects tend to investigate alternate paths. Projects like SpiNNaker[4] and TrueNorth[5] are still standing on the side of the multi-core approach while being bio-inspired. The global architecture they propose changes to comply with the artificial neural network inspiration, but the core computations are still made with Von Neuman processor architectures.

The PCMOS (Probabilistic-CMOS) project [9] modifies the structure of processors to build resilient and energy efficient machines. The key idea is to lower the nominal voltage at which standard CMOS circuits operate and to compensate errors with a robust design. An algebra formalizes how errors are propagated along a chain of logical

- *M.Faix, P. Bessière, E. Mazer and J. Droulez are with CNRS.*
- *R. Laurent is with ProBaYes SAS.*

Manuscript received September 1st, 2015: revised, April 10th 2016.

1. A free version of ProBT is available at http://www.probayes.com/fr/Bayesian-Programming-Book/downloads/.
2. http://probayes.com
3. http://www.bambi-fet.eu/
4. http://apt.cs.manchester.ac.uk/projects/SpiNNaker/
5. http://www.research.ibm.com/articles/brain-chip.shtml

operations, leading to circuits computing the correct result with a specified probability. Contrary to our approach, in which we design new theory and components to compute inference, PCMOS manages errors on classic CMOS arithmetic operators to perform approximate computation with compromises between circuit size, voltage, and precision.

Closer from our own work are the approaches taken by Vigoda, Masinkha and Jonas at MIT. In all cases, probabilistic programming is seen as a way to program/specify the future machines and they all avoid the Von Neuman architecture as well as the use of FPUs. Vigoda [10] was the first to initiate, in 2003, the design of non conventional machines dedicated to Bayesian inference. The track chosen was to represent probability distributions on binary variables with analog electrical values. He focused on a well known algorithm for exact inference: the message passing algorithm, and devised the necessary analog components to perform the message passing to obtain the circuit performing the desired inference. While the approach has many practical applications it will not handle complex inference problems.

To address this scalability issue, V. Mansinghka [11] and E. Jonas [12] are promoting a sampling approach based on stochastic calculus: Gibbs samplers or MCMC methods (Metropolis-Hastings) to perform approximate probabilistic inferences and return samples of the posterior distribution. These methods are already established in probabilistic languages such as Church [13] or Venture [14], but also exist inside inference engines supporting approximate inference such as ProBT [15]. To obtain a hardware implementation they proposed a set of transition circuits to sample quantized or continuous variables. In [16] the circuits are based on building blocks performing operations (for example normalisation) by evaluating mathematical expressions with dedicated hardware or by using precomputed tables. Our goal was to propose simpler building blocks, and the architecture we propose only uses two types of probabilistic gates: the CDE (similar to the CPT gate of Jonas) and the ECE gate which has no equivalent in the previously proposed architectures.

Our project is also strongly related to the study of nano scale devices to perform Bayesian inference, which motivates our architecture choices. These devices are nanoscale samplers for Poisson distributions which could eventually replace the current source of entropy (PRNG for pseudo random number generator) of the presented architecture, as in the work of the Computing Fabrics Laboratory of the University of Massachusetts Amherst [17] which is presenting an unconventional hardware architecture and nanoscale technology implementation of Bayesian inference.

3 Stochastic Bitstreams and Operators

3.1 Definition

A stream of bits may be considered as encoding the probability p of a binary variable: $p = n_1/(n_1 + n_0)$ where n_1 is the number of 1 and n_0 the number of 0 in the stream. It may also be considered as coding the odds $o = p/(1-p)$ of this binary variable. The odds ratio is then given by $o = n_1/n_0$.

Conversely, given a probability p (or an odds ratio o) we may use software or hardware stochastic processes to generate a bitstream $B(p)$ (respectively $B(o)$) that approximately encodes the probability p (respectively the odds ratio o).

Operators on bitstreams may then be used to perform probabilistic calculus on binary distributions. A central point of this paper is to show that using only two simple operators called *CDE* and *ECE* we can solve complex probabilistic problems.

As these operators compute with an approximate representation of probabilities, we will use the root of the mean square error (RMSE) to measure the precision of an operator: if p' is the observed result given by an operator and p the desired result, we compute the RMSE by considering several experiments leading to several p' and averaging the square error on p and $1-p$.

$$RMSE = \sqrt[2]{2 < (p-p')^2 >}\,.$$

3.2 *CDE*: Conditional Distribution Element

Gupta & Kumaresan [18], [19], proposed a circuit able to generate a stochastic bitstream $B(p)$ encoding a probability p stored in a fixed point format $p = \sum_{i=1}^{i=m} x_i 2^{-i}$ in a buffer X (see red part of Figure 1).

We propose to complement this circuit by an addressable memory (see blue part of Figure 1). Given a value in Y the content of the memory at this address (a given probability p_Y) is written in X. The generated stochastic bitstream $B(p_Y)$ is then conditioned by the value Y.

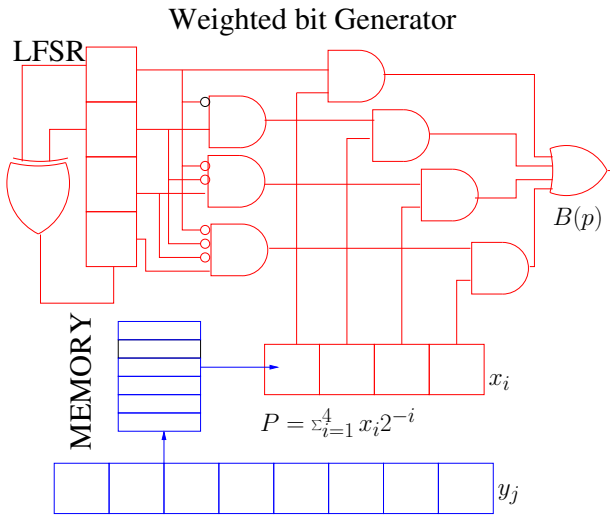

Fig. 1. Hardware implementation of the Conditional Distribution Element.

3.3 *ECE*: Extended C-Element

As in Gaines [19], [20] we use operators to combine stochastic signals and perform arithmetic operations. For example, assuming $B(p_1)$ and $B(p_2)$ are two independent stochastic bitstreams we may use a simple AND gate to obtain $B(p_1.p_2)$. As we will see in the next section, what we really need is the product of two odds: a way to generate $B(o_1.o_2)$ from two independent bitstreams $B(o_1)$ and $B(o_2)$. For that purpose we propose to use an extension of Muller C-elements using more memory resources.

The Muller C-element truth table is given in Table 1. It uses a one bit memory to provide the previous value Z_{prev}

X	Y	Z
0	0	0
0	1	Z_{prev}
1	0	Z_{prev}
1	1	1

TABLE 1
Truth table of a Muller C element with inputs X and Y, and output Z.

as output when the inputs are either $(0,1)$ or $(1,0)$. When two independent and not autocorrelated bitstreams $B(o_1)$ and $B(o_2)$ are given as inputs to a C-element, the output $B(o_1.o_2)$ is an unbiased estimator of the product of the odds o_1 and o_2. This result can be obtained by applying the detailed balance principle to the Markov chain $Z_n \rightarrow Z_{n+1}$.

However, there is some autocorrelation in the output stream (a bit at time t is not independent of a bit at time $t-1$) due to the effect of memory. This autocorrelation induces troubles when C-elements are cascaded [21]. To reduce the autocorrelation of the output, we extend the memory capacity using a buffer C of size D bits. This toroidal buffer works as follows. When the inputs have the same value (X=Y=v) the content of the buffer is shifted to the left, its rightmost bit is set to v and we provide v as output. When the inputs have different values, we provide the rightmost value of C as output and rotate the buffer content to the right. If D=1 we have a regular Muller C element. The truth table of the *ECE* is presented in Table 2.

X	Y	Z	Action
0	0	0	shift C left, set rightmost bit of C to 0
0	1	rightmost bit of C	Rotate C right
1	0	rightmost bit of C	Rotate C right
1	1	1	shift C left, set rightmost bit of C to 1

TABLE 2
Truth table of an *ECE* with inputs X and Y, and output Z.

Several methods using some memory to mitigate the bitstream autocorrelation issue have been compared in [22] where the authors conclude that Edge Memories provide the best performances and speed on their decoding application. The toroidal buffer of the *ECE* serves the same purpose, but without the need of generating a random address, thus reducing the circuit area and power needs. Figure 2 presents the experimental RMSE for a cascade of two products of odds computed either with *ECEs* or with Edge Memories. As the buffer size is increased, the error decrease (until the precision threshold determined by the bitstream size N is reached) is faster for *ECEs* than for Edge Memories.

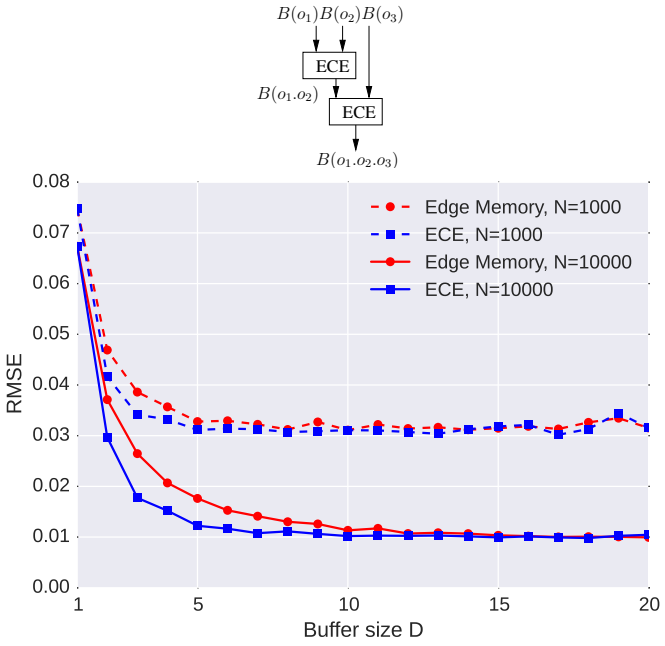

Fig. 2. The bitstream $B(o_1.o_2.o_3)$ is computed, the odds of which are the product of the three input stream odds o_1, o_2 and o_3, which correspond to probability values uniformly drawn between 0 and 1. The RMSE is experimentally computed for buffer sizes ranging from 1 to 20, over 10000 runs. As usual with stochastic computing, the precision depends on the size N of the bitstream (here 1000 or 10000) and goes as $\frac{1}{\sqrt{N}}$.

4 Gibbs Algorithm and Product of Odds

4.1 Gibbs algorithm

Let us consider a set X of $\lfloor X \rfloor$ binary variables $X = \{B_1, \cdots, B_{\lfloor X \rfloor}\}$. Let us consider a subset $K = \{K_1, \cdots, K_{\lfloor K \rfloor}\}$ of X made of binary variables with known values $\{k_1, \cdots, k_{\lfloor K \rfloor}\}$. Let us consider a subset $I = \{I_1, \cdots, I_{\lfloor I \rfloor}\}$ of X made of binary variables of interest. Finally, let us consider a complementary subset F of free variables. We have, $X = K \cup I \cup F$ and, of course, $\lfloor X \rfloor = \lfloor K \rfloor + \lfloor I \rfloor + \lfloor F \rfloor$. The purpose of the Gibbs algorithm is to sample the distribution $P(I_1, \cdots, I_{\lfloor I \rfloor} | k_1, \cdots, k_{\lfloor K \rfloor})$. It operates as follows:

for $B_i \in K$ **do**
  Set B_i to the corresponding value k_j
end for
for $B_i \in I \cup F$ **do**
  Draw a value b_i at random.
end for{Initialization}
while TRUE **do**
  for $B_i \in I \cup F$ **do**
    Draw b_i according to $P(B_i | b_1 \dots b_{i-1}, b_{i+1}, \dots, b_{\lfloor X \rfloor})$
    Output the binary values of the variables in I
  end for
end while

We see that this algorithm only needs to be able to sample from a distribution $P(B_i | b_1 \dots b_{i-1}, b_{i+1}, \dots, b_{\lfloor X \rfloor})$.

4.2 Sampling from $P(B_i | b_1 \dots b_{i-1}, b_{i+1}, \dots, b_{\lfloor X \rfloor})$

Any joint probability distribution on $\lfloor X \rfloor$ binary variables can be written as:

$$\begin{array}{rl} & P\left(B_1, B_2, \cdots, B_{\lfloor X \rfloor}\right) \\ = & \prod_{n=1}^{\lfloor X \rfloor} \left[P\left(B_n | B_1, \cdots B_{n-1}\right)\right] . \end{array} \quad (1)$$

If we are interested in the probability distribution of one of these binary variables B_i knowing the $\lfloor X \rfloor - 1$ values b_j of all the others, we get:

$$\begin{array}{rl} & P\left(B_i | b_1, \cdots, b_{i-1}, b_{i+1}, \cdots, b_{\lfloor X \rfloor}\right) \\ \propto & \prod_{n=1}^{i-1} \left[P\left(b_n | b_1, \cdots, b_{n-1}\right)\right] \\ & P\left(B_i | b_1, \cdots, b_{i-1}\right) \\ & \prod_{n=i+1}^{\lfloor X \rfloor} \left[P\left(b_n | b_1, \cdots, B_i, \cdots, b_{n-1}\right)\right] . \end{array} \quad (2)$$

To compute the odds of B_i, we get:

$$
\begin{aligned}
& O\left(B_i|b_1,\cdots,b_{i-1},b_{i+1},\cdots,b_{\lfloor X\rfloor}\right) \\
= & \prod_{n=1}^{i-1}\left[\frac{P(b_n|b_1,\cdots,b_{n-1})}{P(b_n|b_1,\cdots,b_{n-1})}\right] \\
& \frac{P(B_i=1|b_1,\cdots,b_{i-1})}{P(B_i=0|b_1,\cdots,b_{i-1})} \\
& \prod_{n=i+1}^{\lfloor X\rfloor}\left[\frac{P(b_n|b_1,\cdots,B_i=1,\cdots,b_{n-1})}{P(b_n|b_1,\cdots,B_i=0,\cdots,b_{n-1})}\right] .
\end{aligned} \tag{3}
$$

The first term vanishes:

$$
\begin{aligned}
& O\left(B_i|b_1,\cdots,b_{i-1},b_{i+1},\cdots,b_{\lfloor X\rfloor}\right) \\
= & \frac{P(B_i=1|b_1,\cdots,b_{i-1})}{P(B_i=0|b_1,\cdots,b_{i-1})} \\
& \prod_{n=i+1}^{\lfloor X\rfloor}\left[\frac{P(b_n|b_1,\cdots,B_i=1,\cdots,b_{n-1})}{P(b_n|b_1,\cdots,B_i=0,\cdots,b_{n-1})}\right] .
\end{aligned} \tag{4}
$$

As some variables may be independent, it may occur that some terms in the remaining products are independent of B_i in which case they also get simplified. Equation 4 shows how to compute the odds of any binary variable knowing the value of all the others as a product of probability ratios. Each term has a likelihood, homogeneous to odds o, which can be sampled by bitstreams generated with probability $p = o/(1+o)$ by *CDEs* described in Section 3.2, and their product can be computed by *ECEs* described in Section 3.3.

4.3 Stochastic Gibbs Circuit

4.3.1 Principle

Let us take an example with three binary variables $X = \{R, S, G\}$ to present the architecture of the stochastic Gibbs circuit. This example is inspired from the classical "sprinkler" Bayes net. R,S,G are binary variables corresponding to the predicates "it has been Raining", "the Sprinkler was turned on" and "the Grass is wet". The joint distribution may be decomposed as $P(R,S,G) = P(R)P(S|R)P(G|R,S)$. We are interested in the probability that it rained knowing that the grass is either wet or dry: $P(R|G=g)$. We have $K = \{G\}$, $I = \{R\}$ and $F = \{S\}$.

According to the Gibbs algorithm we need to sample R and S, which means, according to Equation 4, to compute two odds with the following formulas:

$$
\begin{aligned}
O(R|G=g,S=s) &= \frac{P(R=1)}{P(R=0)}\cdot\frac{P(s|R=1)}{P(s|R=0)}\cdot\frac{P(g|R=1,s)}{P(g|R=0,s)} , \\
O(S|G=g,R=r) &= \frac{P(S=1|r)}{P(S=0|r)}\cdot\frac{P(g|r,S=1)}{P(g|r,S=0)} .
\end{aligned}
$$

These formulas include 13 ratios which are preliminary knowledge specified in the model. We use *CDEs* to store them and to produce the associated bitstreams. We need one *CDE* for each of the 3 terms appearing in the decomposition. One for $P(R)$ to produce when needed a bitstream $B(o_R)$ with $o_R = \frac{P(R=1)}{P(R=0)}$. One for $P(S|R)$ with a 4 slot memory to produce respectively $B(o_S^1)$ with $o_S^1 = \frac{P(S=1|R=0)}{P(S=0|R=0)}$, $B(o_S^2)$ with $o_S^2 = \frac{P(S=1|R=1)}{P(S=0|R=1)}$, $B(o_S^3)$ with $o_S^3 = \frac{P(S=0|R=1)}{P(S=0|R=0)}$ and $B(o_S^4)$ with $o_S^4 = \frac{P(S=1|R=1)}{P(S=1|R=0)}$. Finally, one more for $P(G|R,S)$ with a 8 slot memory.

We then need 2 *ECEs* to compute the product of the 3 ratios required to sample $o(R|G=g)$ or the 2 required for $o(S|G=g)$. This leads to the circuit sketched in Figure 3 dedicated to computing the inference $P(R|G=g)$. The input has to be set to the desired value g and the output is a binary bitstream $B(o)$ encoding $O(R|G=g)$.

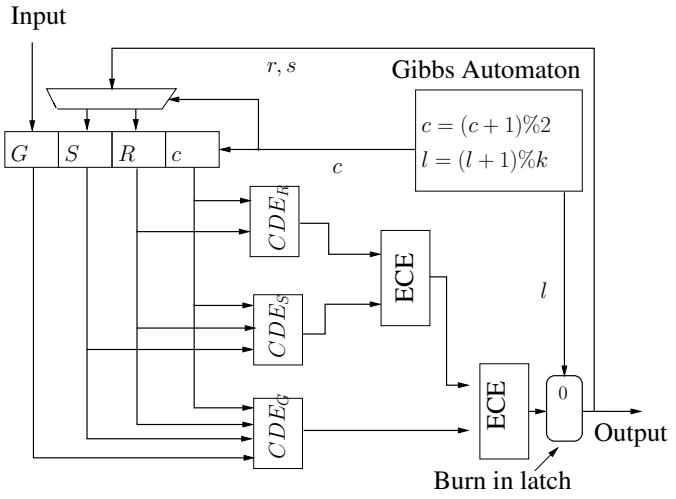

Fig. 3. Stochastic Gibbs Circuit implementation of the Sprinkler example.

4.3.2 Control and subtleties

The Gibbs algorithm requires to scan in sequence all the binary variables in $I \cup F$. This is obtained using a counter c. c is used by the multiplexer at the top left of Figure 3 to select which variable to update. It is also used as an input of each *CDE* to select the appropriate ratio to produce as they depend on the sampled variable being either R or S. For instance, if we sample S ($c = 0$), we do not need the ratio $\frac{P(R=1)}{P(R=0)}$ and the output of the CDE_R should be the bitstream $B(o = 1)$ with equal number of 0 and 1. This bitstream is the neutral element for an *ECE*.

Note that the conditional distributions implemented by the *CDE*s are not necessarily the conditional distributions of the model. For instance, for CDE_S, the compiler will set in memory at address $\{S = 1, R = 1, C = 1\}$ the value $p = \frac{o_S^4}{1+o_S^4}$ with $o_S^4 = \frac{P(S=1|R=1)}{P(S=1|R=0)}$. This is due to the fact that *CDEs* are used not only to store odds but more generally to store probability ratios.

As *ECEs* involve memory buffers, a burning sequence is required for the memory content to be reliable. This is implemented using a counter l which validates the output after a burning sequence of length k.

5 Experimentation and Results

5.1 The compilation toolchain

A compilation toolchain was developed to evaluate the proposed architecture. The input is any Bayesian program using discrete variables written in ProBT (see Figure 4 for an example). The output is a VHDL program specifying the circuit dedicated to the corresponding inference.

5.2 Preprocessing

Any probabilistic program using discrete variables is first transformed into an equivalent specification using only binary variables. For example, a distribution $P(V)$ on a discrete variable with 2^n values will be transformed into one distribution and $n-1$ conditional distributions on binary variables $B_i^V : i \in 1 \ldots n$:

$$P(V) = P(B_1^V, B_2^V, B_3^V, \ldots, B_{n-1}^V)$$
$$= P(B_1^V)P(B_2^V|B_1^V)P(B_3^V|B_1^V B_2^V)\ldots P(B_n^V|B_1^V \ldots B_{n-1}^V)\ .$$

While the "Binary" representation looks less compact, it requires at most the same number of parameters as for representing $P(V)$. Each probability distribution, as for instance "$P(B_2^V|B_1^V)$", is automatically computed using the inference engine of ProBT.

5.3 Compiler

The compiler starts from the previously generated "binary" Bayesian program and generates a VHDL file describing the final circuit. It specifies a circuit implementation of Equation 4 the architecture of which is similar to the one presented in Figure 3, with two extra simplifications. One allowing to reduce the size of the memory of the *CDE*: it is not necessary to store all the terms which cancel out in equation 3. Another one which reduces the number of *ECEs* by only considering informative binary signals: $B(o), o \neq 1$.

5.4 Simulator

The generated VHDL code could easily be simulated at the gate or the transistor levels with the standard EDA tools. However, this type of simulation is time consuming. We have developed our own simulation engine allowing very fast simulation of our virtual stochastic machines.

5.5 Experiments

We designed a problem with circular dependences between its variables (as is happens in some image processing applications) and chose the parametric forms so that we could compute an analytic solution to be used as reference. We define a joint probability distribution on the n variables $B_i \in 0, 1$ and the n variables $V_j \in \{0, \ldots, 2^m - 1\}$ as:

$$P(V_1, ..., V_n, B_1, ..., B_n) = \left(\prod_{i=1}^{n} P(V_i)\right) \prod_{i=1}^{n} P(B_i|V_i V_{1+i\%n}) \tag{5}$$

with

$$\begin{array}{l} P(V_i = k_1) \propto u(1-u) \\ P(B_i = 1|V_i = k_1 V_{1+i\%n} = k_2) \\ \quad = g(a_i, b_i, c_i, d_i)u^{a_i}(1-u)^{b_i}v^{c_i}(1-v)^{d_i}\ , \end{array} \tag{6}$$

(see the Appendix for the definition of g) where $u = (k_1 + 0.5)2^{-m}$ and $v = (k_2 + 0.5)2^{-m}$. We consider the following inference based on the previous joint distribution:

$$P(V_i|b_1 = 1, \ldots, b_n = 1)\ .$$

In this particular example 2^{m*n} floating point additions would be necessary to compute the exact inference, which is intractable for large values of m and n. However it is possible to get an approximation of the exact solution with the following close form expression:

$$P(V_i = k|b_1 = 1, \ldots, b_n = 1) \approx 2^{-m} Beta((k + 0.5)2^{-m}, \alpha, \beta) \tag{7}$$

where $Beta(u, \alpha, \beta)$ is the probability density function of a beta distribution with two known integer parameters α and β (see the Appendix).

```
from probt import *
V = plArray("V", plIntegerType(0, 1), 10)
B = plArray("B", plIntegerType(0, 255), 10)
model = plComputableObjectList()
b = plValues(B)
for i in range(10) do
  j = (i + 1) mod 256
  model = model*plDistribution(V[i], FV(i))
  model = model*plCndDistribution(B[i], V[i]^V[j], FB(i))
  b[i] = 1 # to perform an inference with all B[i] set to 1
end for
joint = plJointDistribution(model)
joint.ask(V[9],b).instantiate(b).VHDL('Ex.vhd')
```

Fig. 4. The specification of the test example written in ProBT: FV and FB are Python functions implementing Equation 6. This program is compiled to VHDL code, which is simulated to evaluate the result.

5.6 Results

We used a tractable instance of the previously defined inference (with $m = n = 5$) to evaluate the quality of our analytic approximation. ProBT was used to compute $P(V_2|b_1 = 1 \ldots b_5 = 1)$ with exact inference by marginalizing over all unknown variables. The absolute difference with the analytic solution is always lower than 10^{-17}, which validates the quality of the analytic approximation.

Increasing m will increase further the precision (see the Appendix). We now report results for a problem of size $m = 8,\ n = 20$. The exact inference would require 2^{160} operations. We use the analytic approximation to evaluate the results since the exact inference solution is out of reach. The precision depends on:

- The size D of the buffers used in *ECEs* to prevent the output bitstream autocorrelation.
- The burn-in duration BI allowing the *ECEs* to stabilize the output bitstream encoding the product $B(o_1 o_2)$.
- The number N of samples chosen to approximate the output distribution.

Figure 5 shows the histogram resulting from the approximate inference of $P(V_3|b_1 = 1, \ldots, b_{20} = 1)$ with the following parameters: $D = 16$, $BI = 200$, $N = 100000$. The size D of the *ECE* buffers depends on the depth of *ECE* cascades. In the example Figure 2, the precision was evaluated with a two stage product of odds. Here we use *ECEs* with a buffer size D=16, which provides a good compromise between size and precision for this example cascading 3 *ECEs*. (Indeed, we compute the product of 8 odd ratios, by arranging the *ECEs* as a binary tree of depth 3.)

Table 3 shows the impact of the burn-in BI and of the number N of samples on the Kullback-Leibler divergence between the analytic approximation and the results obtained with our stochastic architecture. We observe that the precision increases with N, but slowly. This is unsurprising since as mentionned before the precision of Bernoulli sequences go as $\sqrt[2]{N}$. Increasing the length of the burning sequence after which the samples computed by the tree of ECEs are considered valid has a cost in terms of computation time, but it allows to significantly increase the precision

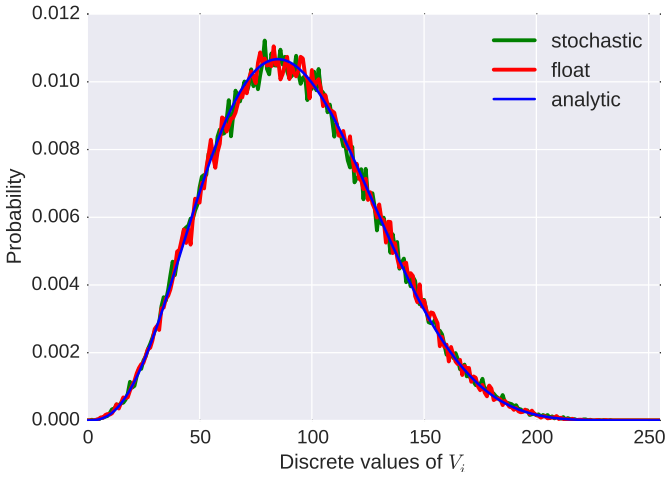

Fig. 5. Comparison of the approximation computed by our stochastic machine (in green) with a floating point implementation of the same sampling algorithm (in red) and with the analytic solution (in blue).

of the results until they are as good as for floating point computations. This suggests a way to use the same circuits with different tradeoffs between latency (and power needs) and the desired precision.

Number of samples	1000	10000	100000
burn in = 10	1.5293	1.2222	1.15294
burn in = 20	1.3145	1.1482	1.13608
burn in = 50	0.2447	0.1000	0.07979
burn in = 100	0.1414	0.0178	0.00218
burn in = 200	0.1333	0.0155	0.00182
burn in = 400	0.1529	0.0151	0.00188
floating point	0.1458	0.0184	0.00181

TABLE 3
Kullblack divergence from the Closed-Form Solution (in bits).

6 Conclusion

We have presented a compilation toolchain to design automatically stochastic machines that solve approximate Bayesian inference problems using the Gibbs sampling algorithm. This compilation toolchain is generic: it can translate any inference problem expressed as a Bayesian program into a circuit comprised only of two types of stochastic components: the *CDE* and the *ECE*. We implemented a quantitative test to evaluate our architecture on intractable problems. The results are promising: the output distribution of our stochastic machine approximates the analytic solution as well as a floating point implementation of Gibbs sampling.

In the near future we will extend and test our approach on Bayesian filters. In a second step, we will design a generic and programmable machine, which will accept any new Bayesian program without the need for a new circuit synthesis, opening the way to actually programmable Bayesian computing devices.

Acknowledgments

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] E. T. Jaynes, *Probability Theory: the Logic of Science*. Cambridge University Press, 2003.
[2] K. Mekhnacha, E. Mazer, and P. Bessière, "The design and implementation of a bayesian CAD modeler for robotic applications," *Advanced Robotics*, vol. 15, no. 1, pp. 45–69, 2001.
[3] O. Lebeltel, P. Bessière, J. Diard, and E. Mazer, "Bayesian robot programming," *Advanced Robotics*, vol. 16, no. 1, pp. 49–79, 2004.
[4] P. Bessière, E. Mazer, J. M. Ahuactzin, and K. Mekhnacha, *Bayesian programming*. CRC Press, 2013.
[5] P. Bessière, C. Laugier, and R. Siegwart, *Probabilistic Reasoning and Decision Making in Sensory-Motor Systems*. Springer, 2008.
[6] J. Ferreira and J. Dias, *Probabilistic Approaches to Robotic Perception*. Spinger, 2013.
[7] M. Faix, R. Laurent, J.Lobo, and E. Mazer, "Cognitive computation: a bayesian machine case study," in *The 14th IEEE International Conference on Cognitive Informatics and Cognitive Computing*. IEEE, 2015.
[8] M. Faix, J. Lobo, R. Laurent, D. Vaufreydaz, and E. Mazer, "Stochastic bayesian computation for autonomous robot sensorimotor systems," in *Workshop on Unconventional computing for Bayesian inference IROS*. IEEE, 2015.
[9] L. N. Chakrapani, B. E. Akgul, S. Cheemalavagu, P. Korkmaz, K. V. Palem, and B. Seshasayee, "Ultra-efficient (embedded) SOC architectures based on probabilistic CMOS (PCMOS) technology," in *Proceedings of the conference on Design, automation and test in Europe*. European Design and Automation Association, 2006, pp. 1110–1115.
[10] B. Vigoda, "Analog logic: Continuous-time analog circuits for statistical signal processing," Ph.D. dissertation, Massachusetts Institute of Technology, 2003.
[11] V. K. Mansinghka, "Natively probabilistic computation," Ph.D. dissertation, Massachusetts Institute of Technology, 2009.
[12] E. M. Jonas, "Stochastic architectures for probabilistic computation," Ph.D. dissertation, Massachusetts Institute of Technology, 2014.
[13] N. Goodman, V. Mansinghka, D. Roy, K. Bonawitz, and D. Tarlow, "Church: a language for generative models," *arXiv preprint arXiv:1206.3255*, 2012.
[14] R. A. Mansinghka V., "Venture: a general purpose probabilistic programming platform with efficient stochastic inference," *arXiv preprint arXiv:1404:0099*, 2014.
[15] 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.
[16] V. Mansinghka and E. Jonas, "Building fast bayesian computing machines out of intentionally stochastic, digital parts," *arXiv preprint arXiv:1402.4914*, 2014.
[17] S. Khasanvis and all, "Self-similar magneto-electric nanocircuit technology for probabilistic inference engines," *arXiv preprint arXiv:1504.04056*, 2015.
[18] P. K. Gupta and R. Kumaresan, "Binary multiplication with pn sequences," *IEEE Trans. Acoustics Speech Signal Process*, vol. 36, pp. 603–606, 1988.
[19] A. Alaghi and J. P. Hayes, "Survey of stochastic computing," *ACM Transactions on Embedded computing systems (TECS)*, vol. 12, no. 2s, p. 92, 2013.
[20] B. Gaines, "Stochastic computing systems," in *Advances in information systems science*. Springer, 1969, vol. 2, pp. 37–172.
[21] J. Friedman, L. Calvet, P. Bessière, J. Droulez, and D. Querlioz, "Bayesian inference with Muller C-Elements," *IEEE Transactions on Circuits and Systems I*, 2016.
[22] S. S. Tehrani, C. Winstead, W. J. Gross, S. Mannor, S. L. Howard, and V. C. Gaudet, "Relaxation dynamics in stochastic iterative decoders," *IEEE Transactions on Signal Processing*, vol. 58, no. 11, pp. 5955–5961, 2010.

Marvin FAIX (born in 1990) received the M.Eng degree in Integrated Electronic Systems in 2013 from the Ecole Nationnale Supérieure de Physique, ELectronique and Matériaux (PHELMA), Grenoble, France. He started his PhD in 2014 on the research topic "Probabilistic computing: from theory to hardware implementation", granted by the European FET Project BAMBI – Bottom-up Approaches to Machines dedicated to Bayesian Inference – (FP7-ICT-2013-C 618024). He works at the Laboratoire Informatique de Grenoble (LIG) and in collaboration with the laboratory TIMA (Techniques de l'Informatique et de la Microélectronqiue pour l'Architecture des systèmes intégrés). His work is oriented at probability theory to design new ways to compute Bayesian inference in hardware.

Raphaël LAURENT (born in 1988) received a computer science engineering master degree from ENSIMAG and an Artificial Intelligence master degree from the Joseph Fourier University (Grenoble, France). During his PhD at Grenoble-Alpes University he acquired valuable experience in probabilistic information processing systems as he implemented Bayesian models and algorithms to study cognitive aspects of speech perception, production and learning. He is now working in the ProbaYes R&D group which develops efficient solutions in the fields of machine learning, probabilistic models, optimization and data science for a wide range of industrial applications. His current research interests within the BAMBI project (Bottom-up Approaches to Machines dedicated to Bayesian Inference) are the design and hardware implementation of unconventional computing architectures along with the development of software tools allowing to automatically map probabilistic programs into efficient circuit implementations based on new computation paradigms.

Pierre BESSIÈRE was born in 1958. He received the engineering degree and the Ph.D. degree in computer science from the Institut National Polytechnique de Grenoble (INPG), France, in 1981 and 1983, respectively. He did a Post-Doctorate at SRI International (Stanford Research Institute) working on a project for National Aeronautics and Space Administration (NASA). He then worked for five years in an industrial company as the leader of different artificial intelligence projects. He came back to research in 1989. He has been a senior researcher at Centre National de la Recherche Scientifique (CNRS) since 1992. His main research concerns have been evolutionary algorithms and probabilistic reasoning for perception, inference and action. He leads the Bayesian Programming research group (Bayesian-Programming.org) on these two subjects. Twenty PhD diplomas and numerous international publications are fruits of the activity of this group during the last 15 years. He also leads the BIBA (Bayesian Inspired Brain and Artefact) and BAMBI (Bottom-up Approaches to Machines dedicated to Bayesian Inference) European projects and was a Partner in BACS (Bayesian Approach to Cognitive Systems). He is a co-founder and scientific adviser of the ProbaYes Company, which develops and sells Bayesian solutions for the industry.

Emmanuel MAZER (born in 1953) is a senior researcher at CNRS. He defended two doctoral dissertations in Robotics in 1981 and 1989. He stayed at the Artificial Intelligence Department of the University of Edinburgh during his post-doc and was a fellow researcher at the Massachusetts Institute of Technology for 3 years. He is a founder of several high tech companies in France and abroad (including ProbaYes). He was part of the European projects BIBA (Bayesian Inspired Brain and artefact), BACS (Bayesian Approach to Cognitive Systems) and BAMBI (Bottom-up Approaches to Machines dedicated to Bayesian Inference). Within ProbaYes, he contributed to several industrial projects involving computational geometry and Bayesian inference techniques.

Jacques DROULEZ (born in 1950) received a mathematical and engineer training (Ecole Polytechnique, Paris), a complete medical training (MD: Lariboisire - St Louis Hospital, Paris), a master in biochemistry and a Habilitation to supervise research in cognitive sciences. He has got a fellowship from the Centre National dEtudes Spatiales (1978-1982). He is now Emeritus Director of Research at the Centre National de la Recherche Scientifique (CNRS), and head of the research team active perception and probabilistic approach at ISIR laboratory of UPMC-Sorbonne university. His main research themes are the perception of 3D motion and objects, the theoretical study of models for multi-sensory interactions and the adaptive motor control. He has about 90 publications in international journals including one in PNAS on sensory-motor integration model and one in Nature on object perception during self-motion. He is involved in several European and national research programs and in multidisciplinary scientific networks.