Design of Stochastic Machines Dedicated to Approximate Bayesian inferences
Marvin Faix, Raphä Laurent, Pierre Bessière, Emmanuel Mazer, Jacques Droulez

To cite this version:
Marvin Faix, Raphä Laurent, Pierre Bessière, Emmanuel Mazer, Jacques Droulez. Design of Stochastic Machines Dedicated to Approximate Bayesian inferences. IEEE Transactions on Emerging Topics in Computing, Institute of Electrical and Electronics Engineers, 2016, <10.1109/TETC.2016.2609926>. <hal-01374906>

HAL Id: hal-01374906
https://hal.archives-ouvertes.fr/hal-01374906
Submitted on 2 Oct 2016

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
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 language ProBT [5] 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 [7].

Recently the European project BAMBI [8] 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 [9] and TrueNorth [10] 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
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 nanoscale 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 \( 1 - p \).

\[
RMSE = \sqrt{\frac{1}{2} < (p - p')^2 >}\.
\]

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 \( p = \sum_{i=1}^{m} x_i 2^{-i} \) 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.](image)

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 \( Z_{prev} \)
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 \( Z_n \to Z_{n+1} \).

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.

<table>
<thead>
<tr>
<th>( X )</th>
<th>( Y )</th>
<th>( Z )</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>shift C left, set rightmost bit of C to 0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>( Z_{prev} )</td>
<td>rightmost bit of C to 1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>( Z_{prev} )</td>
<td>rightmost bit of C to 0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>shift C left, set rightmost bit of C to 1</td>
</tr>
</tbody>
</table>

**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.

**4 GIBBS ALGORITHM AND PRODUCT OF ODDS**

**4.1 Gibbs algorithm**

Let us consider a set \( X \) of \( |X| \) binary variables \( X = \{B_1, \ldots, B_{|X|}\} \). Let us consider a subset \( K = \{K_1, \ldots, K_{|K|}\} \) of \( X \) made of binary variables with known values \( \{k_1, \ldots, k_{|K|}\} \). Let us consider a subset \( I = \{I_1, \ldots, I_{|I|}\} \) 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, \( |X| = |K| + |I| + |F| \). The purpose of the Gibbs algorithm is to sample the distribution \( P(I_1, \ldots, I_{|I|} | k_1, \ldots, k_{|K|}) \). 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** \( \text{TRUE} \) **do**

**for** \( B_i \in I \cup F \) **do**

Draw \( b_i \) according to \( P(B_i | b_1 \ldots b_{i-1}, b_{i+1}, \ldots, b_{|X|}) \)

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 \( P(B_i | b_1 \ldots b_{i-1}, b_{i+1}, \ldots, b_{|X|}) \).

**4.2 Sampling from** \( P(B_i | b_1 \ldots b_{i-1}, b_{i+1}, \ldots, b_{|X|}) \)

Any joint probability distribution on \( |X| \) binary variables can be written as:

\[
P(B_1, B_2, \ldots, B_{|X|}) = \prod_{n=1}^{|X|} \left[ P(B_n | B_1, \ldots, B_{n-1}) \right].
\]

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

\[
P(B_i | b_1, \ldots, b_{i-1}, b_{i+1}, \ldots, b_{|X|}) \propto \prod_{n=1}^{i-1} \left[ P(b_n | b_1, \ldots, b_{n-1}) \right] \\
\prod_{n=i+1}^{|X|} \left[ P(b_n | b_1, \ldots, B_i, \ldots, b_{n-1}) \right].
\]

To compute the odds of \( B_i \), we get:
\[ O \left( B_i | b_1, \ldots, b_i-1, b_{i+1}, \ldots, b_X \right) = \prod_{n=i}^{i-1} \left[ \frac{P(b_n | b_1, \ldots, b_{n-1})}{P(b_n | b_1, \ldots, b_{n-1})} \cdot \frac{P(b_{i+1} | b_n, \ldots, b_{i-1})}{P(b_{i+1} | b_n, \ldots, b_{i-1})} \cdot \frac{P(b_X | b_{i+1}, \ldots, b_X)}{P(b_X | b_{i+1}, \ldots, b_X)} \cdot \frac{P(b_1 | b_X)}{P(b_1 | b_X)} \right]. \quad (3) \]

The first term vanishes:

\[ O \left( B_i | b_1, \ldots, b_i-1, b_{i+1}, \ldots, b_X \right) = \frac{P(b_i | b_1, \ldots, b_{i-1})}{P(b_i | b_1, \ldots, b_{i-1})} \cdot \prod_{n=i+1}^{X} \frac{P(b_n | b_{i+1}, \ldots, b_{n-1})}{P(b_n | b_{i+1}, \ldots, b_{n-1})}. \quad (4) \]

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:

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

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 \( o_R = P(R = 1) \). One for \( P(S|R) \) with a 4 slot memory to produce respectively \( B(o^1_S) \) with \( o^1_S = \frac{P(S = 1 | R = 1)}{P(S = 0 | R = 0)} \), \( B(o^2_S) \) with \( o^2_S = \frac{P(S = 1 | R = 1)}{P(S = 0 | R = 0)} \), \( B(o^3_S) \) with \( o^3_S = \frac{P(S = 0 | R = 1)}{P(S = 0 | R = 0)} \), and \( B(o^4_S) \) with \( o^4_S = \frac{P(S = 0 | R = 1)}{P(S = 0 | R = 0)} \).

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 \( B(a) \) encoding \( O(R | G = g) \).

### 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 \( B^V_i : i \in 1 \ldots n \).
allowing to reduce the size of the memory of the CDE. Equation 4, the architecture of which is similar to the one
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
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
developed our own simulation engine allowing very fast simulation of our virtual stochastic machines.

5.5 Experiments
We designed a problem with circular dependencies between
its variables (as is happens in some image processing appli-
cations) 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:

$$P(V_1, \ldots, V_n, B_1, \ldots, B_n) = \prod_{i=1}^{n} P(V_i) \prod_{i=1}^{n} P(B_i | V_i V_{i+1} \ldots V_n)$$  \hspace{1cm} (5)

with

$$P(V_i = k_i) \propto u(1 - u)$$
$$P(B_i = 1 | V_i = k_i V_{i+1} \ldots V_n = k_2)$$
$$= g(a_i, b_i, c_i, d_i) u^{a_i} (1 - u)^{b_i} v^{c_i} (1 - v)^{d_i}$$  \hspace{1cm} (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 $2^{mn}$ floating point additions
would be necessary to compute the exact inference, which
is intractable for large values of $m$ and $n$. However it is
easy to get an approximation of the exact solution with the
following close form expression:

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

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

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 sta-
  bilize 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 ap-
proximate 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
The precision increases with $N$, but slowly. This is unsur-
prising since as mentioned before the precision of Bernoulli
sequences go as $\sqrt{N}$. Increasing the length of the burning
sequence after which the samples computed by the tree of
ECEs is considered valid has a cost in terms of computa-
tion time, but it allows to significantly increase the precision.

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.

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):
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')
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.

<table>
<thead>
<tr>
<th>Number of samples</th>
<th>burn in = 10</th>
<th>burn in = 20</th>
<th>burn in = 50</th>
<th>burn in = 100</th>
<th>burn in = 200</th>
<th>burn in = 400</th>
<th>floating point</th>
</tr>
</thead>
<tbody>
<tr>
<td>Probability</td>
<td>0.000</td>
<td>0.002</td>
<td>0.004</td>
<td>0.006</td>
<td>0.008</td>
<td>0.010</td>
<td>0.012</td>
</tr>
</tbody>
</table>

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).

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.
Marvin Faix (born in 1990) received the M.Eng degree in Integrated Electronic Systems in 2013 from the Ecole Nationale 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-2013-618024). He works at the Laboratoire Informatique de Grenoble (LIG) and in collaboration with the laboratory TIMA (Techniques de l’Informatique et de la Microélectronique 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 Droz (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 d’Etudes 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.