Towards Quantum Inference on Higher-Order Bayesian Networks

Jérôme Evrard Claudia Faggian Giacomo Gatti Gabriele Vanoni

1 Introduction

A fascinating connection has recently been exposed [10, 1] between quantum circuits and Bayesian Networks [12], the latter being a prominent tool for probabilistic reasoning. Such a line of research exploits the power of quantum computing to accelerate Bayesian inference over Bayesian Networks. To do so, Bayesian Networks are encoded into *quantum circuits*, where each boolean random variable is represented by a qubit. A quantum version of the rejection sampling algorithm [11] is obtained by *amplitude amplification* [2, 8] (a generalization of Grover's search algorithm [7]), yielding a quadratic speed-up.

Is it possible to exploit this body of research to accelerate inference in (higher-order) probabilistic programs ?

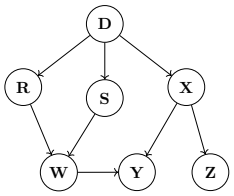
We present the first steps in the development of an end-to-end methodology that would allow users to write stochastic models in a functional higher-order probabilistic programming language (PPL), and that produces as output the quantum circuit ready to be used for quantum rejection sampling. The goal is to leverage both the expressivity of a fully-fledged probabilistic language and the speedup brought by the quantum technology.

The main challenge is that the target of the compilation—a quantum circuit— can only handle a finite amount of information. While this holds true for any low-level language, a critical aspect specific to the quantum setting is that qubits are a scarce resource, limited by physical and engineering constraints. To deal with this issue, integrating resource-awarness in the compilation scheme, we rely on proof-theoretic techniques rooted in Girard's geometry of interaction (GoI) [5] and abstract machines. In this abstract we report on preliminary work, where the probabilistic programming language is *first-order*. However, our goal is eventually to have the full expressivity of an *higher-order* language, building on the approach developed in [3]. Abstract machines could allow one to *linearize* and *defunctionalize* (probabilistic) programs, while building on-the-fly the corresponding (quantum) circuit, in the stile of [4, 3, 9].

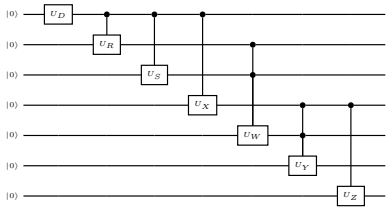
Bayesian Reasoning in a Nutshell. Bayesian methods provide a formalism for reasoning about partial beliefs under conditions of uncertainty. *Random variables* (r.v.s) represent features of the system being modeled. The system is modeled as *a joint probability distribution* on all possible values of the variables of interest – each instantiation representing a possible state of the system. We adopt the standard convention of capital letters (e.g. X, Y) denoting random variables, while lowercase letters (e.g. x, y) are particular fixed *values* of those variables, *i.e.* x being an instantiation of the r.v. X. As standard, $\Pr(x)$ stands for $\Pr(X = x)$. Bold letters $\mathbf{X}, \dots$ denote sets of r.v.s , and $\mathbf{x}, \mathbf{z} \dots$ their instantiations. For simplicity, random variables are here binary, with $\mathsf{Val}(X) = \{\mathtt{t}, \mathtt{f}\}$.

The key operation is *inference*. Given a probabilistic model which is expressed by a probability distribution $\Pr(\mathbf{X})$, and some variable of interest $\mathbf{Y} \subseteq \mathbf{X}$, the marginal probability $\Pr(\mathbf{Y})$ is obtained by *summing out* the irrelevant r.v.s ; this marginal is called the *prior* (probability distribution) of $\mathbf{Y}$. A typical *query* to the model is the *posterior* distribution of $\mathbf{Y}$ given some *evidence* $\mathbf{e}$ for $\mathbf{E} \subseteq \mathbf{X}$ (the observed values being $\mathbf{e}$): $\Pr(\mathbf{Y}|\mathbf{E} = \mathbf{e})$.

Bayesian Networks A Bayesian Network (BN), as the one depicted in Fig. 1, represents a joint probability distribution in a compact way, via a *factorized representation*. A Bayesian Network $\mathcal{B}$ over the set of r.v.s $\mathbf{X} = \{X_1, \dots, X_n\}$ is a directed acyclic graph (DAG) where the set of nodes is $\mathbf{X}$ and the edges represent the dependencies between the r.v.s. To quantify the dependencies, to each node X in the DAG is associated a *conditional probability table (CPT)*. To a Bayesian Network $\mathcal{B}$ over $X_1, \dots, X_n$ is associated a (unique) probability distribution $\Pr(X_1, \dots, X_n)$, noted $[\![\mathcal{B}]\!]$.



Bayesian Network $\mathcal{B}$ (assuming given the CPT's).



Encoding of $\mathcal{B}$ as a quantum circuit.

Figure 1: Example from [10].

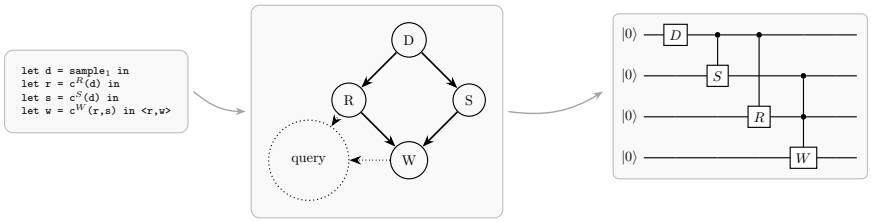


Figure 2: Compilation scheme.

From Programs To Bayesian Networks. PPLs are far more expressive than Bayesian networks. Indeed, programming with Bayesian networks has been compared to the task of programming using logical circuits. Instead, PPLs have all the features that one could expect from a high-level programming language. Remarkably, many of *first-order* PPLs *compile* programs into Bayesian networks first, and then perform the inference task on the network [13]. Recently, it has been shown how to extract a Bayesian network from a typed *higher-order* program, using *semantical* methods [3], in particular *intersection types*. That study paves the way to the quest for an *effective procedure* that could *actually* compile the program down to a Bayesian network. Our goal here is exactly to develop an *effective* methodology to compile a program into a BN, moreover taking into account the issues which are specific to the further encoding into a quantum circuit, which is ready to be used for the quantum rejection sampling algorithm.

2 The Bayesian Abstract Machine

The language we adopt is a basic probabilistic call-by-value calculus, corresponding to the *first-order* fragment of [3]. Our language is typed, but here we omit the description of the types, which is completely standard. Let $\mathcal{V}$ be a countable set of variables. Terms are defined by the following grammar:

$$
\begin{array}{llll}
\textsc{Terms} & t, u & ::= & v \mid \mathtt{sample}_d \mid \mathtt{case}\ v\ \mathtt{of}\ \{v_i \Rightarrow \mathtt{sample}_{d_i}\} \mid \mathtt{obs}(x = b) \\
 & & & \mathtt{let}\ x = u\ \mathtt{in}\ t \mid \mathtt{let}\ \langle x, y\rangle = v\ \mathtt{in}\ t \\
\textsc{Values} & v, w & ::= & x \in \mathcal{V} \mid \langle v, w\rangle \mid b \\
\textsc{Booleans} & b & ::= & \mathtt{t} \mid \mathtt{f} \\
\textsc{Contexts} & C & ::= & [\cdot] \mid \mathtt{let}\ x = C\ \mathtt{in}\ t \mid \mathtt{let}\ x = u\ \mathtt{in}\ C \mid \mathtt{let}\ \langle x, y\rangle = C\ \mathtt{in}\ t \mid \\
 & & & \mathtt{let}\ \langle x, y\rangle = v\ \mathtt{in}\ C \mid \langle v, C\rangle \mid \langle C, w\rangle
\end{array}
$$

The probabilistic primitive $\mathtt{sample}_d$ samples a boolean value from a (Bernoulli) distribution d. The `case` construct is just a generalized if/then/else—please notice that the `case` expression is restricted, because we reserve it to the encoding of CPTs. Observed data (the *evidence*) are specified syntactically using an *observe* construct—for example $\mathtt{obs}(\mathtt{w} = \mathtt{t})$ states that the variable `w` has been observed to be true. For brevity, we will shorten a `case` expression depending on n variables $x_1, \ldots, x_n$ as follows:

$$\mathtt{c}_{\langle x_1,\ldots,x_n\rangle} := \mathtt{case}\ \langle x_1, \ldots, x_n\rangle\ \mathtt{of}\ \{v_i \Rightarrow \mathtt{sample}_{d_i}\}_{v_i \in \{\mathtt{t},\mathtt{f}\}^n}$$

This language is easily seen to be able to encode all standard Bayesian Networks, see Fig. 2 for an example, and [6] for a brief tutorial.

ENVIRONMENTS	e	$::=$	$\epsilon \mid [x{\leftarrow}c] \cdot e$	TOKENS	s	$::=$	$(\underline{t}, e, p, \mathcal{G})$
CLOSURES	c	$::=$	(t, e)	NODES	p	$::=$	$\texttt{query} \mid (t, C)$
INIT. STATE.	q_t	$::=$	$(\{(t, \epsilon, \texttt{query})\}, \emptyset)$				

Term	Env.	Data		Graph		Term	Env.	Data		Graph
$\mathcal{T} \cup \{\ \underline{\texttt{let } x = u \texttt{ in } t}$	e	p	$\}$	$\mathcal{G}$	$\rightarrow_{\mathsf{let}}$	$\mathcal{T} \cup \{\ \underline{t}$	$e{\cdot}[x{\leftarrow}(u,e)]$	p	$\}$	$\mathcal{G}$
$\mathcal{T} \cup \{\ \underline{x}$	$e'{\cdot}[x{\leftarrow}(u,e)]{\cdot}e''$	p	$\}$	$\mathcal{G}$	$\rightarrow_{\mathsf{var}}$	$\mathcal{T} \cup \{\ \underline{u}$	e	p	$\}$	$\mathcal{G}$
$\mathcal{T} \cup \{\ \underline{\langle v, w \rangle}$	e	p	$\}$	$\mathcal{G}$	$\rightarrow_{\mathsf{spawn}}$	$\mathcal{T} \cup \{\ \underline{v}$	e	p	,	
						$\underline{w}$	e	p	$\}$	$\mathcal{G}$
$\mathcal{T} \cup \{\ \underline{\texttt{case } v \texttt{ of } \{v_i \Rightarrow t_i\}}$	e	p	$\}$	(V, E)	$\rightarrow_{\mathsf{cond}}$	$\mathcal{T} \cup \{\ \underline{v}$	e	cp	$\}$	$(V \cup \mathsf{cp}, E \cup \{(\mathsf{cp}, p)\})$
$\mathcal{T} \cup \{\ \underline{\texttt{sample}_d}$	e	p	$\}$	(V, E)	$\rightarrow_{\mathsf{sample}}$	$\mathcal{T}$				$(V \cup \mathsf{cp}, E \cup \{(\mathsf{cp}, p)\})$
$\mathcal{T} \cup \{\ \underline{b}$	e	p	$\}$	(V, E)	$\rightarrow_{\mathsf{const}}$	$\mathcal{T}$				(V, E)

Figure 3: The data structures and (a selection of) the transition rules of the BKAM.

The Compilation Scheme. Fig. 2 illustrates the essence of our *compilation scheme*: the term on the l.h.s is compiled first into a Bayesian Network and then into a quantum circuit. The core of this process is an *abstract machine* which is a variation of the Krivine Abstract Machine. The essential definitions of the Bayesian Krivine Abstract Machine (BKAM) is in Fig. 3. For the sake of readability, we only give the key constructs, omitting the handling of evidence (easily obtained by adding to the token a component that remembers the observation).

The idea behind the machine is that while the term is evaluated, the corresponding (Bayesian network) graph is built. The *state* of the machine is a *set* of tokens $\mathcal{T}$, each one being an independent process, together with a *global* graph that is updated each time a token finds a probabilistic primitive, *i.e.* a construct that generates a new random variable. Random variables are represented as pairs term/context, this way avoiding the need to generate fresh names (the probabilistic constructs encoding CPTs in an immediate way, we not only build the graph but actually the full BN). The macro `cp` returns the current pair term/context, *i.e.* the active position inside the syntax tree of the term. The *initial* state starts from the root of the term, with a distinguished dummy random variable `query` that will have as parents in the final DAG all the random variables appearing in the conclusion. Transition rules act on *one* token at a time, leaving the others unchanged. This way, the machine is non-deterministic, because of the many possible choices of the token which has to make a transition. Rules $\rightarrow_{\mathsf{let}}$ and $\rightarrow_{\mathsf{var}}$ are standard, and just traverse `let`-redexes. Rule $\rightarrow_{\mathsf{spawn}}$ is the only one generating new processes. Indeed, if a random variable has more that one parent, as `W` in our example, both its ancestors have to be explored. When either a $\texttt{sample}_d$ or a constant b is found, the process dies, and in the former case the corresponding random variable is added to the graph, together with the edge connecting it to the previously found random variable/position p, that is stored in the Data field. The same happens when a `case` construct is found. The BKAM is intrinsically parallel, and thus non-deterministic, but we have the following property.

Proposition. *The BKAM is confluent.*

Because of confluence, the final state is unique. Then, given a notion of semantics $[\![\cdot]\!]$ such as the one given in [3], which assigns a probability distribution on a finite set of random variables to a typed term, we have the following correctness property for the BKAM.

Theorem. *If* $(\{(t, \epsilon, \texttt{query})\}, \emptyset) \rightarrow^* (\emptyset, \mathcal{G})$ *then* $[\![t]\!] = [\![\mathcal{G}]\!]$.

3 Resource-Awareness: Qubits Are a Scarce Commodity

The number of qubits in quantum computers is limited by several physical and engineering constraints, which make it challenging to build and control large-scale quantum systems. Our approach integrates specific techniques to reduce the number of needed qubits. More specifically, in the quantum speedup of Bayesian inference, the *critical resource* is the number of distinct variables in the Bayesian Network, as for each a qubit is allocated in the encoding. We adopt two solutions to help alleviating the need for qubits.

Pruning non-essential variables. When querying a model described by a BN over r.v.s $\mathbf{X}$, we are usually only interested in a subset $\mathbf{Y} \subset \mathbf{X}$ (the irrelevant variables being marginalized). In this scenario,

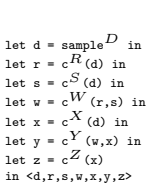


Figure 4: Term u.

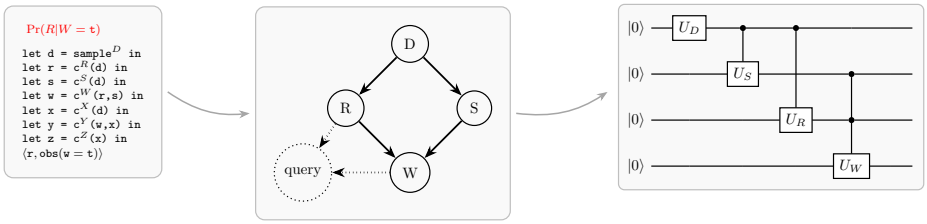


Figure 5: Pruning approach.

a well-known technique to reduce the cost of computation relies on the fact that is often possible to work with a smaller BN without loosing information, by *pruning* nodes corresponding to r.v.s which will not be used. The machine in Fig. 3 integrates this approach and is designed to produce the *minimal* BN needed to compute the query.

Example 3.1 (Pruning, Fig. 5)**.** *Let as fix a term u, given Fig. 4, describing the same probabilistic model over 7 r.v.s (D, R, S, W, X, Y, Z) as in Fig. 1. Our query is* $\Pr(R|W = \mathtt{t})$*. Concretely, we need to compute the marginal* $\Pr(R, W = \mathtt{t})$*, which is described by the term*

$$\mathtt{let}\ \langle d, r, s, w, x, y, z\rangle = u\ \mathtt{in}\ \langle r, \mathtt{obs}(w = \mathtt{t})\rangle$$

As illustrated in Fig. 5, the machine will build the BN over (only) 4 necessary variables, namely D, S, R, W, which will then be compiled in a quantum circuit with (only) 4 qubits.

Decomposition of the target BN in smaller ones. Assume that the machine in Fig. 3 compiles a term t into a Bayesian Network $\mathcal{B}$, with the goal of computing $\Pr(\mathbf{Y}|\mathbf{E} = \mathbf{e})$ ($\mathbf{E}$ being the evidence). We can further refine the machine in order to detect conditional independences among the r.v.s and —possibly— produce as target a set of disjoint smaller Bayesian Networks $\mathcal{B}_1, \ldots, \mathcal{B}_k$, each over a *subset* of the total r.v.s . Each such $\mathcal{B}_i$ can then be encoded a separately (reducing the needed number of qubits) to compute $\Pr(\mathbf{Y}_i|\mathbf{E}_i)$; only at the end the results are put together. The correctness of this operation is yield by conditional independence.

Example 3.2 (Divide-and-Conquer, Fig. 6)**.** *Consider again the term in Fig. 4 describing a model over 7 r.v.s . Assume we are now interested in the query* $\Pr(R, Y, Z|W = \mathtt{t}, D = \mathtt{t})$*, involving several r.v.s . All the variables here are needed in the computation, however the sets $\{R, S, W\}$ and $\{X, Y, Z\}$ are* conditionally independent *given W and D. The refined machine produces the output in Fig. 6, to be contrasted with the BN and the quantum circuit in Fig. 1, which we would obtain by naively compiling u.*

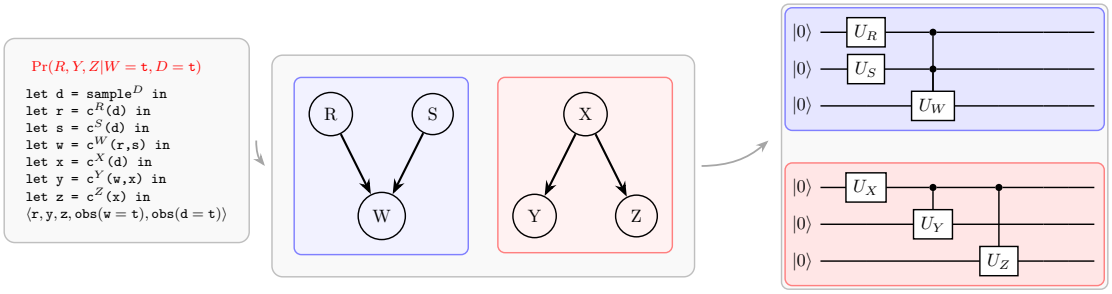


Figure 6: Divide-and-Conquer approach, via *conditional independence*.

References

[1] Sima Esfandiarpour Borujeni, Saideep Nannapaneni, Nam H. Nguyen, Elizabeth C. Behrman, and James Edward Steck. Quantum circuit representation of bayesian networks. *Expert Syst. Appl.*, 176:114768, 2021.

[2] Gilles Brassard and Peter Høyer. An exact quantum polynomial-time algorithm for simon's problem. In *5th ISTCS*, pages 12–23, 1997.

[3] Claudia Faggian, Daniele Pautasso, and Gabriele Vanoni. Higher order bayesian networks, exactly. *Proc. ACM Program. Lang.*, 8(POPL):2514–2546, 2024.

[4] Dan R. Ghica. Geometry of synthesis: a structured approach to VLSI design. In *Proc. of POPL*, pages 363–375. ACM, 2007.

[5] Jean-Yves Girard. Geometry of interaction 1: Interpretation of system f. In *Logic Colloquium '88*, pages 221 – 260. 1989.

[6] Andrew D. Gordon, Thomas A. Henzinger, Aditya V. Nori, and Sriram K. Rajamani. Probabilistic programming. In James D. Herbsleb and Matthew B. Dwyer, editors, *Proceedings of the on Future of Software Engineering, FOSE 2014, Hyderabad, India, May 31 - June 7, 2014*, pages 167–181. ACM, 2014.

[7] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the 28th Symposium on the Theory of Computing (STOC)*, pages 212–219. ACM, 1996.

[8] Lov K. Grover. Quantum computers can search rapidly by using almost any transformation. *Phys. Rev. Lett.*, 80:4329–4332, May 1998.

[9] Ugo Dal Lago, Claudia Faggian, Benoît Valiron, and Akira Yoshimizu. The geometry of parallelism: classical, probabilistic, and quantum effects. In *Proc of POPL*, pages 833–845, 2017.

[10] Guang Hao Low, Theodore J. Yoder, and Isaac L. Chuang. Quantum inference on bayesian networks. *Phys. Rev. A*, 89:062315, Jun 2014.

[11] Maris Ozols, Martin Roetteler, and Jérémie Roland. Quantum rejection sampling. *ACM Trans. Comput. Theory*, 5(3):11:1–11:33, 2013.

[12] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1988.

[13] Jan-Willem van de Meent, Brooks Paige, Hongseok Yang, and Frank Wood. An introduction to probabilistic programming. *CoRR*, abs/1809.10756, 2018.