International Journal of Approximate Reasoning 110 (2019) 107–126

Contents lists available at ScienceDirect

International Journal of Approximate Reasoning

www.elsevier.com/locate/ijar

The finite model theory of Bayesian network specifications: Descriptive complexity and zero/one laws

Fabio Gagliardi Cozman [a,*], Denis Deratani Mauá [b]

[a] Escola Politécnica, Universidade de São Paulo, Brazil
[b] Instituto de Matemática e Estatística, Universidade de São Paulo, Brazil

ARTICLE INFO

Article history:
Received 22 December 2017
Received in revised form 11 April 2019
Accepted 12 April 2019
Available online 18 April 2019

Keywords:
Bayesian networks
Finite model theory
Probabilistic relational models
Descriptive complexity
Zero/one laws
Probabilistic logic

ABSTRACT

This paper studies specification languages that describe Bayesian networks using predicates and other logical constructs. First, we adopt an abstract syntax for relational Bayesian network specifications, and review definability and complexity results. We then propose a novel framework to study the descriptive complexity of relational Bayesian network specifications, and show that specifications based on function-free first-order logic capture the complexity class PP; we also exhibit specification languages, based on second-order quantification, that capture the hierarchy of complexity classes $\mathrm{PP}^{\mathrm{NP}^{\cdot^{\cdot^{\cdot^{\mathrm{NP}}}}}}$, a result that does not seem to have equivalent in the literature. Finally, we derive zero/one laws for Bayesian network specifications based on function-free first-order logic, indicating their value in definability analysis.

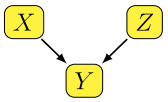
1. Introduction

One can find a variety of specification languages that resort to predicates and other logical constructs to describe Bayesian networks [22,23,34]. In a (propositional) Bayesian network, an "inference" usually refers to the computation of the probability of some random variables taking on some value given some observation of other random variables. Relational Bayesian network specifications offer more sophisticated inference facilities, as now the input also describes the Bayesian network itself. One should thus expect relational specifications to have significant more expressive power than propositional ones, much as first-order logic goes beyond propositional logic.

It is only natural to ask what is the *expressivity* of relational Bayesian network specifications. This question is the focus of this paper, and it can be approached from many directions, following parallel work in finite model theory [25,42,59]. For instance, one approach is to compare syntactically distinct specification languages by examining whether one specification language can, or cannot, specify all models that are specified by the other. Another approach is to investigate what happens to probabilities when the domain size goes to infinity, producing convergence theorems referred to as zero/one laws. Techniques along these lines have actually been examined in pioneering work by Jaeger [49,51,52]. Another approach taken in finite model theory is to show which concepts can, and which cannot, be expressed with particular specification languages. The goal is to obtain "inexpressibility proofs" that show the expressivity boundaries of languages. Yet another strategy is to relate expressivity and complexity; there are two ways to do so.

* Corresponding author.
E-mail address: fgcozman@usp.br (F.G. Cozman).

https://doi.org/10.1016/j.ijar.2019.04.003
0888-613X/



$$\begin{aligned} \mathbb{P}(X=1)=0.2, \quad & \mathbb{P}(Z=1)=0.9, \\ \mathbb{P}(Y=1|X=0,Z=0)=0.3, \quad & \mathbb{P}(Y=1|X=0,Z=1)=0.4, \\ \mathbb{P}(Y=1|X=1,Z=0)=0.5, \quad & \mathbb{P}(Y=1|X=1,Z=1)=0.6. \end{aligned}$$

Fig. 1. A very simple Bayesian network with three binary random variables.

One possibility is to focus on the computational cost of answering specific questions. For instance: How expensive is it to verify that some probability is larger than half when Bayesian networks are described using a particular specification language? The goal then is to show the problem of interest to be complete for some complexity class, say NP or PP.

The other possibility is to show that Bayesian networks described using some specification language $\mathcal{L}$ can solve all and only those problems in some complexity class $\mathcal{C}$. Roughly, that means that a problem in $\mathcal{C}$ can be exactly translated into a fixed Bayesian network specification; an input to the problem is accepted if and only if a corresponding inference in the specification is "accepted" in some appropriate sense. We then say that specification language $\mathcal{L}$ *captures* complexity class $\mathcal{C}$. Such an analysis is related to descriptive complexity theory, a central branch of finite model theory that investigates how complexity classes are "captured" by logical languages [25,42]. The question is, in short: What sort of Turing machine can one precisely describe with such and such logical language?

As a whole, this paper discusses various facets of a *finite model theory of Bayesian networks*, from definability and complexity of inferences to descriptive complexity and zero/one laws. We adopt an abstract specification language, based on logical constructs; we then investigate what can and what cannot be modeled by varying the features of the abstract specification language. Because the topic is novel, most of this paper consists of building a framework in which to operate. In particular, it does not seem that the descriptive complexity of Bayesian networks has been investigated in previous work.

The rest of the paper is organized as follows. In Section 2 we define precisely what we mean by "Bayesian network specification". In Section 3 we briefly discuss definability and inexpressibility proofs, mostly to provide some context to later developments. In Section 4 we present necessary background on complexity theory, including known results on complexity of inferences. We devote considerable space to these sections so as to present our finite model theory from a broad perspective, hopefully exhibiting the opportunities afforded by our framework.

After reviewing the main points of descriptive complexity in Section 5, we move to novel contributions in Section 6. We show that Bayesian network specifications based on function-free first-order logic capture the complexity class PP. That is, a language is in PP if and only if its strings encode valid inferences in a Bayesian network specified with predicates and first-order quantifiers. Note that this is a different statement than PP-completeness, as it provides a characterization of PP that does not use a model of computation (i.e., a Turing machine). And then we look at specifications that allow quantification over predicates, and show that such a restricted version of "second-order" Bayesian networks capture the complexity classes $\mathsf{PP}^{\mathsf{NP}}$, $\mathsf{PP}^{\mathsf{NP}^{\mathsf{NP}}}$, and so on. It does not seem that previous research on descriptive complexity theory has reached these complexity classes.

In Section 7 we bring the machinery of zero/one laws to relational Bayesian network specifications, and present connections with definability issues. Section 8 summarizes our contributions and suggests some avenues for future investigation.

2. Relational Bayesian network specifications

We start off with a long, but well known, definition [20,55]. A *Bayesian network* is a pair consisting of a directed acyclic graph $\mathbb{G}$, where each node is a random variable, and a probability distribution $\mathbb{P}$ over all those random variables, such that the graph and the distribution satisfy the *Markov condition*: each random variable X in $\mathbb{G}$ is conditionally independent, with respect to $\mathbb{P}$, of its nondescendants nonparents given its parents.

Because we are interested in finite objects in this paper, we assume that a Bayesian network has finitely many random variables, and that all random variables have finitely many values, so the set of configurations for all random variables is a finite set. In that case, the distribution $\mathbb{P}$ associated with a Bayesian network always factorizes as follows:

$$\mathbb{P}(X_1=x_1,\ldots,X_n=x_n)=\prod_{i=1}^{n}\mathbb{P}(X_i=x_i \mid \mathrm{pa}(X_i)=\pi_i)\,,$$

where $\mathrm{pa}(X_i)$ refers to those random variables that are parents of X_i, and π_i is the projection of values $\{x_1,\ldots,x_n\}$ on $\mathrm{pa}(X_i)$. Whenever $\mathrm{pa}(X_i)$ is empty, we take $\mathbb{P}(X_i=x_i|\mathrm{pa}(X_i)=\pi_i)$ to mean $\mathbb{P}(X_i=x_i)$. The probability values $\mathbb{P}(X_i=x_i|\mathrm{pa}(X_i)=\pi_i)$ define the *conditional probability table* associated with X_i; that is, a table indicating the probability of each value x_i of X_i given each configuration π_i of the parents of X_i. A very simple Bayesian network is depicted in Fig. 1.

It is often the case that a Bayesian network must encode repetitive patterns over large numbers of variables. We present here a simple but illuminating example in some detail, so as to justify later definitions. Readers who are convinced of the ubiquity of such repetitive patterns may choose to move quickly to Definition 1.

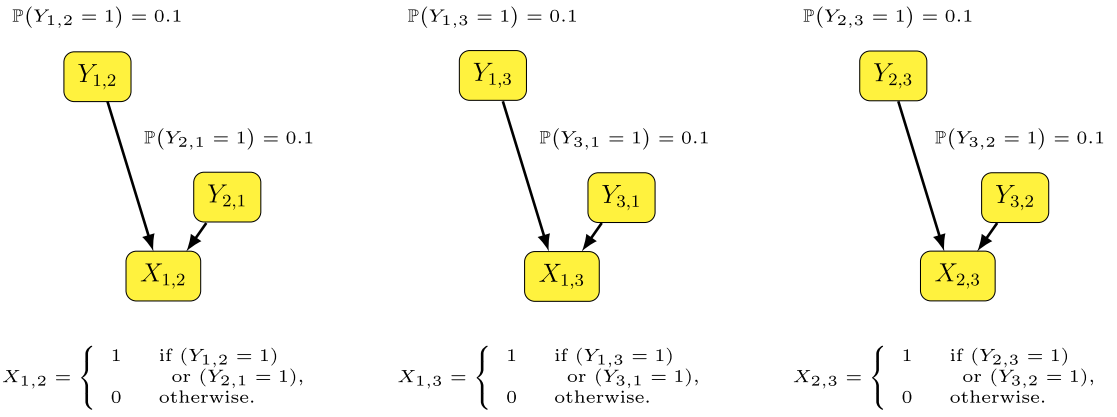


Fig. 2. A Bayesian network representing a $G(3, 0.19)$ Gilbert-style random graph. Not all nodes are shown: three missing nodes $Y_{i,i}$ are associated with conditional probabilities $\mathbb{P}(Y_{i,i}=1)=0.1$, three nodes $X_{i,i}$ are identically zero, and three nodes $X_{i,j}$ for $i>j$ have the same specification as $X_{j,i}$. Note that $\mathbb{P}(X_{1,2}=1)=\mathbb{P}(Y_{1,2}=1)+\mathbb{P}(Y_{2,1}=1)-\mathbb{P}(Y_{1,2}=1)\,\mathbb{P}(Y_{2,1}=1)=0.1+0.1-0.01=0.19$, as desired.

Example 1. Consider a Gilbert-style random undirected graph with parameters N and p, usually referred to as $G(N, p)$ [35]. There we have sites $1, \ldots, N$, and for each pair of distinct sites, there is an undirected link between these two sites with probability p. That is:

$$\mathbb{P}(\text{ there is an undirected link between distinct nodes } i \text{ and } j\ ) = p. \tag{1}$$

Consider specifying a Gilbert-style random graph using, for each pair (i, j), a binary random variable $X_{i,j}$ to indicate a link between i and j. Since the graph is undirected, $X_{i,j}=1$ must imply that $X_{j,i}=1$. A possible specification is to use auxiliary random variables $Y_{i,j}$, for each pair of sites (i, j), with:

$$\mathbb{P}(Y_{i,j}=1) = 1-\sqrt{1-p}. \tag{2}$$

Then specify $\mathbb{P}(X_{i,j}|Y_{i,j}, Y_{j,i})$ such that:

$$X_{i,j} = \begin{cases} 1 & \text{if } (i\neq j) \text{ and } \big((Y_{i,j}=1) \text{ or } (Y_{j,i}=1)\big), \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

Expression (3) implies $X_{i,j}=X_{j,i}$. Moreover, Expressions (2) and (3) imply

$$\begin{aligned} \mathbb{P}(X_{i,j}=1) &= \mathbb{P}(Y_{i,j}=1)+\mathbb{P}(Y_{j,i}=1)-\mathbb{P}(Y_{i,j}=1)\,\mathbb{P}(Y_{j,i}=1) \\ &= 2(1-\sqrt{1-p})-(1-\sqrt{1-p})^2 \\ &= p, \end{aligned}$$

as required by Expression (1). Hence Expressions (2) and (3) have the desired effect. Fig. 2 depicts part of the Bayesian network for the Gilbert-style random graph $G(3, 0.19)$ (the missing part contains specifications for $X_{i,j}$ with $i\geq j$, and for $Y_{i,i}$).

Suppose now that we are interested in the probability that a random graph is fully connected. That is, we want to compute the probability that for each pair (i, j) with $i\neq j$ we have $\{X_{i,j}=1\}$. We might introduce a random variable Z to indicate the latter event:

$$Z = \begin{cases} 1 & \text{if for all } i, j : (i\neq j) \text{ implies } X_{i,j}=1, \\ 0 & \text{otherwise.} \end{cases} \tag{4}$$

Fig. 3 (left) sketches the Bayesian network that is obtained for $N=750$ and $p=0.005$. In this figure, and later ones, we omit probability values. Fig. 3 (right) depicts an instantiated graph obtained by sampling from the Bayesian network. □

There are several tools that can be used to specify a repetitive Bayesian network. A popular strategy is to use *plates*, as introduced in the BUGS project [36,60]. To summarize, a plate represents a set of indexed random variables, and is often drawn as a rectangle enclosing the indexed random variables. For instance, a plate representation for Gilbert-style random graphs is presented in Fig. 4 (left). Plates usually do not allow the child of a node in a plate to be outside that plate, but "extended" plates in the literature allow it [8]. For instance, the plate model in Fig. 4 (right) leaves Z outside of the plates, with a parent inside the plates. Several other visual formalisms that mix entity-relationship diagrams with Bayesian networks are referred to as *Probabilistic Relational Models (PRMs)* [33,46,58]. There are also textual specification languages;

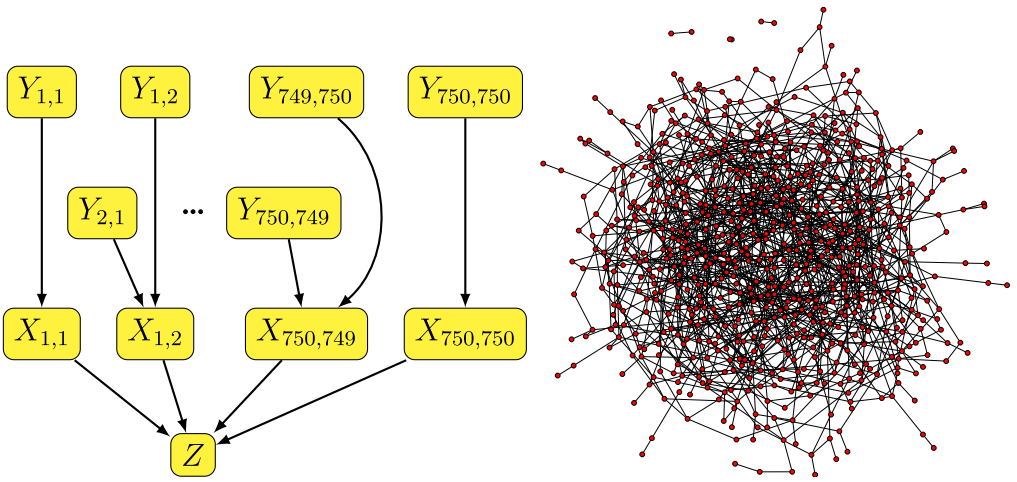


Fig. 3. Left: The Bayesian network representing a $G(750, 0.005)$ random graph and an indicator Z of full connectedness. Right: A graph generated by sampling the random variables in the Bayesian network, with $N = 750$ (each link in the graph corresponds to a realization of a random variable $X_{i,j}$). The graph has 1347 links (the expected number of links is $0.005 \times 750 \times 749/2 = 1404.38$); 18 sites are not linked to other sites, and are not shown in the figure. The average degree is 3.68, and no site has degree larger than 11.

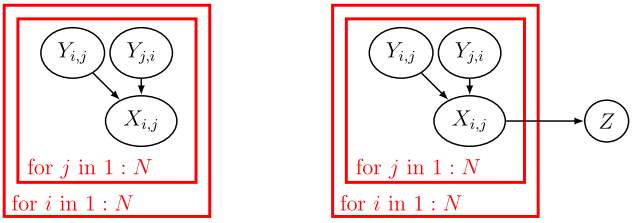


Fig. 4. Plates for Gilbert-style random graphs (left) and for Gilbert-style random graphs with detector of full connectedness (right).

for instance, proposals based on knowledge-based model construction [38,47,85], on object orientation [57,61], on rule-based descriptions [6,44], and on logic programming [71,77]. Some of these specification languages are rather powerful, such as Jaeger's Relational Bayesian Networks [48,50]. Quite a few recent specification languages explore probabilistic logic programming [30,74], and a surge of interest on probabilistic programming has emphasized both functional programming [40,62,64,69,81,86] and combinations of functional and procedural programming [13,39,70]. Another particularly interesting mix of probabilities and Bayesian networks has emerged in connection with probabilistic description logics [14,15,26,56], where one tries to benefit both from the decidability of description logics [4] and the modularity of specifications based on graphs.

This flood of specification languages demands a corresponding toolset for their analysis and synthesis. That is, one must have tools to analyze the complexity of specific language features, and to examine what can/cannot be expressed within a particular feature set. And one must have an understanding of expressivity and complexity so as to design new specification languages that can match practical needs. This is the ultimate goal of our research program.

Each of the specification languages we have mentioned previously offer a different feature set; to proceed with a reasonably general investigation, we need to distill their essential elements into an abstract core. A similar situation is found in the theory of computer programming languages: much is learned by focusing on a few logical constructs that carry most meaning. Here we also resort to logics; before we move on, we briefly review some relevant concepts.

First, we always assume that there is a vocabulary containing predicates. Each predicate r has a nonnegative arity (a predicate of zero arity behaves like a proposition). Formulas may contain the logical connectives: negation ($\neg$), conjunction ($\wedge$), disjunction ($\vee$), implication ($\Rightarrow$), equivalence ($\Leftrightarrow$); besides, formulas may contain logical variables such as x, y, and both existential quantifiers ($\exists$) and universal quantifiers ($\forall$); finally, formulas may contain equality ($=$). The set of well-formed formulas in first-order logic that employ these connectives, logical variables and quantifiers, equality, and arbitrary predicates, is referred to as FFFO.

We now describe the abstract specifications that we use in the remainder of the paper. The idea is simple, and certainly not new [16,19]; as summarized by Poole, we take "a probabilistic model as a deterministic system with stochastic inputs" [73].

Definition 1. A *relational Bayesian network specification*, abbreviated RELBN, consists of two parts. First, we have a finite vocabulary $\mathcal{V}$ containing names of predicates, each with arity specified. Then we have, for each predicate in $\mathcal{V}$, *either* a probability assessment *or* a logical definition, specified as follows:

- A *probabilistic assessment* for predicate r is written
$$\mathbb{P}(r) = \alpha, \tag{5}$$
where α is a rational number in $[0, 1]$.
- A *logical definition* for predicate r with arity k is written
$$r(x_1, \ldots, x_k) \equiv\!\equiv \phi(x_1, \ldots, x_k), \tag{6}$$
where $\phi(x_1, \ldots, x_k)$ is a formula in FFFO whose extralogical symbols are only names in $\mathcal{V}$, or free logical variables in $\{x_1, \ldots, x_k\}$, or possibly other logical variables bound to quantifiers in $\phi(x_1, \ldots, x_k)$. We say that r is *defined* by $\phi(x_1, \ldots, x_k)$. □

We restrict ourselves to rational numbers so as to avoid difficulties with computing real numbers.

If a predicate r is associated with a probabilistic assessment $\mathbb{P}(r) = \alpha$, then r is a *root* predicate; otherwise, if r is associated with a logical definition, then r is a *non-root* predicate. The *dependency graph* of a RELBN is a directed graph where each node is a predicate of the vocabulary, and where there is an edge from relation s to relation r if and only if s appears in the logical definition of r. Thus a root predicate is a root node in the dependency graph. We assume in this paper that the dependency graph is acyclic, unless explicitly indicated (we discuss cyclic graphs in Section 8). That is, no recursive definitions are allowed in specifications.

Consider a simple example based on Expressions (2), (3), and (4):

Example 2. Take a vocabulary $\mathcal{V} = \{\mathsf{dirlink}, \mathsf{link}, \mathsf{fullyConnected}\}$. Predicates dirlink and link have arity two, and $\mathsf{link}(a, b)$ is to be interpreted as denoting a link between a and b. Predicate fullyConnected has arity zero, just indicating whether the property of full connectedness holds. The following RELBN is well-formed:

$$\begin{aligned} \mathbb{P}(\mathsf{dirlink}) &= \alpha, \\ \mathsf{link}(x, y) &\equiv\!\equiv \neg(x = y) \wedge (\mathsf{dirlink}(x, y) \vee \mathsf{dirlink}(y, x)), \\ \mathsf{fullyConnected} &\equiv\!\equiv \forall x, y : \neg(x = y) \Rightarrow \mathsf{link}(x, y). \end{aligned} \tag{7}$$

Given the semantics to be introduced shortly, this RELBN specifies a Gilbert-style random graph. □

To attach semantics to a given RELBN with vocabulary $\mathcal{V}$, we use *domains* and *interpretations*, similarly to the semantics of first-order logic [27]. A domain $\mathcal{D}$ is a finite set whose elements are called *constants*. For a predicate r of arity k, a *grounding* $r(a_1, \ldots, a_k)$ is obtained by selecting k elements from $\mathcal{D}$. Note that if $k = 0$, there is only one grounding denoted by r itself. An interpretation maps each grounding $r(a_1, \ldots, a_n)$ either to true or to false. Hence if $\mathcal{V}$ contains a single predicate with arity 2, then for a domain of size 5 we have 2^{25} possible interpretations. Denote by $\mathcal{I}_{\mathcal{V},\mathcal{D}}$ the set of all interpretations for predicates in $\mathcal{V}$ with respect to $\mathcal{D}$.

Example 3. Consider Example 2, and domain $\mathcal{D} = \{a, b\}$. There are two interpretations for predicate fullyConnected, and 2^4 interpretations for predicate dirlink (as there are four possible groundings of this predicate) and likewise for link. Hence there are 2^9 possible interpretations. □

The semantics of a RELBN τ with vocabulary $\mathcal{V}$ is a mapping that takes a domain $\mathcal{D}$ and yields a unique probability measure over all interpretations of $\mathcal{V}$ with respect to $\mathcal{D}$, by creating a Bayesian network out of τ and $\mathcal{D}$. This mapping always produces a single measure if the dependency graph of the RELBN is acyclic, as follows.

First, produce all groundings of predicates in $\mathcal{V}$. Now associate with each grounding $r(a_1, \ldots, a_k)$ a binary random variable $\hat{r}(a_1, \ldots, a_k)$, such that $\hat{r}(a_1, \ldots, a_k) = 1$ if $r(a_1, \ldots, a_k)$ is true, and 0 otherwise. These random variables are functions from the set of all interpretations to $\{0, 1\}$; to be precise, $\hat{r}(a_1, \ldots, a_k)$ is the indicator function of the event

$$\{\mathbb{I} \in \mathcal{I}_{\mathcal{V},\mathcal{D}} : \mathbb{I}(r(a_1, \ldots, a_k)) = \mathsf{true}\}.$$

Second, take the random variables that correspond to groundings of root predicates. All of those "root" binary random variables are assumed independent. And if r is associated with a probabilistic assessment $\mathbb{P}(r) = \alpha$, associate each $\hat{r}(a_1, \ldots, a_k)$ with probability $\mathbb{P}\left(\hat{r}(a_1, \ldots, a_k) = 1\right) = \alpha$.

Third, for each random variable $\hat{r}(a_1, \ldots, a_k)$ such that r is a non-root predicate associated with logical definition $r(x_1, \ldots, x_k) \equiv\!\equiv \phi(x_1, \ldots, x_k)$, build a propositional formula $\varphi^r_{a_1,\ldots,a_k}$ as follows. Replace logical variables $x_1, \ldots, x_k$ by constants to obtain $\phi(a_1, \ldots, a_k)$, and then replace each universal quantifier in $\phi(a_1, \ldots, a_k)$ by a conjunction over all instances

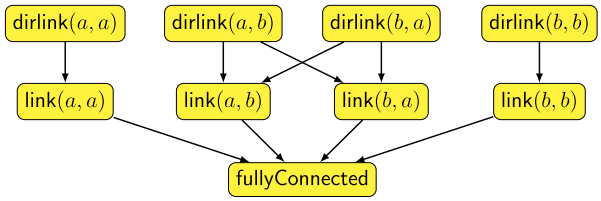


Fig. 5. The Bayesian network that encodes the semantics of the RELBN in Example 2, with domain $\mathcal{D} = \{a, b\}$. The edges out of dirlink(a, a), dirlink(b, b), link(a, a), and link(b, b), are actually unnecessary (they could be removed by logical simplication).

of the bound logical variable, and replace each existential quantifier in $\phi(a_1, \ldots, a_k)$ by a disjunction over all instances of the bound logical variable. The resulting variable-free formula is $\varphi^r_{a_1,\ldots,a_k}$.

Now build a graph where each node is a random variable $\hat{r}(a_1, \ldots, a_k)$, for all groundings. Consider each random variable $\hat{r}(a_1, \ldots, a_k)$ such that r is a non-root predicate; the random variables that correspond to groundings in $\varphi^r_{a_1,\ldots,a_k}$ are referred to as the *parents* of $\hat{r}(a_1, \ldots, a_k)$. The conditional probabilities of $\hat{r}(a_1, \ldots, a_k)$ given its parents are just 0 or 1, depending on $\varphi^r_{a_1,\ldots,a_k}$. Draw a directed edge from each parent of $\hat{r}(a_1, \ldots, a_k)$ to $\hat{r}(a_1, \ldots, a_k)$. By going through all groundings of non-root predicates, we obtain a directed graph that is acyclic if the dependency graph of the RELBN is acyclic. Together with the probabilistic assignments and the logical expressions, we have a Bayesian network that is the promised semantics of the RELBN τ with respect to the domain $\mathcal{D}$.

We denote the Bayesian network just built by $\tau(\mathcal{D})$; if all that is known about the domain is its size N, then we write $\tau(N)$.

We have used special notation to indicate the random variable corresponding to grounding $r(a_1, \ldots, a_k)$; namely, $\hat{r}(a_1, \ldots, a_k)$. But in fact there is no need to burden the notation like this; we can easily use $r(a_1, \ldots, a_k)$ both to mean the grounding *and* the random variable that is the indicator of the grounding over the set of possible interpretations. We adopt such simplified notation in the remainder of this paper.

Example 4. Consider again Example 2, and domain $\mathcal{D} = \{a, b\}$. Thus we have random variables dirlink(a, a), dirlink(a, b), dirlink(b, a), dirlink(b, b), link(a, a), link(a, b), link(b, a), link(b, b), and fullyConnected. We have

$$\mathbb{P}(\text{dirlink}(a, a) = 1) = \beta, \quad \mathbb{P}(\text{dirlink}(a, b) = 1) = \beta,$$
$$\mathbb{P}(\text{dirlink}(b, a) = 1) = \beta, \quad \mathbb{P}(\text{dirlink}(b, b) = 1) = \beta.$$

Also we have

$$\text{link}(a, b) \Leftrightarrow \neg(a = b) \wedge (\text{dirlink}(a, b) \vee \text{dirlink}(b, a)),$$

and likewise for the other groundings of link. Note that we mix here logical notation with groundings/random variables. Clearly some basic reasoning can simplify the latter expression to

$$\text{link}(a, b) \Leftrightarrow \text{dirlink}(a, b) \vee \text{dirlink}(b, a),$$

but we do not require such simplifications to establish a semantics. Finally we have

$$\begin{aligned}\text{fullyConnected} \Leftrightarrow\ & (\neg(a = a) \Rightarrow \text{link}(a, a)) \wedge (\neg(a = b) \Rightarrow \text{link}(a, b)) \\ & \wedge (\neg(b = a) \Rightarrow \text{link}(b, a)) \wedge (\neg(b = b) \Rightarrow \text{link}(b, b)),\end{aligned} \tag{8}$$

or equivalently, with the obvious simplifications:

$$\text{fullyConnected} \Leftrightarrow \text{link}(a, b) \wedge \text{link}(b, a). \tag{9}$$

Fig. 5 shows the directed acyclic graph for the generated Bayesian network. Note that we add all edges from instances of link to fullyConnected, as required by Expression (8), even though not all edges are needed as shown by Expression (9). □

Matters are rather simple when all predicates have arity zero, in which case the whole specification corresponds to some "propositional" Bayesian network. In fact, any Bayesian network with binary random variables can be specified using such predicates, as the next example illustrates.

Example 5. Suppose we have three binary random variables X, Y and Z, and the Bayesian network in Fig. 1. We can specify this Bayesian network as follows. First, take X, Y and Z to stand for predicates with zero arity. Then introduce probabilistic assessments $\mathbb{P}(X) = 0.2$ and $\mathbb{P}(Z) = 0.9$. Now specify $\mathbb{P}(Y = 1 | X = x, Z = z)$ for each possible pair (x, z) by introducing four auxiliary random variables $W_{x,z}$ and the associated logical definition

$$Y \equiv\equiv (\neg X \wedge \neg Z \wedge W_{00}) \vee (\neg X \wedge Z \wedge W_{01}) \vee (X \wedge \neg Z \wedge W_{10}) \vee (X \wedge Z \wedge W_{11}),$$

plus auxiliary probabilistic assessments $\mathbb{P}(W_{x,z}) = \mathbb{P}(Y = 1 | X = x, Z = z)$ for each possible (x, z). The marginal probabilities over X, Y, and Z are exactly the probabilities over these random variables as given by the original Bayesian network. □

To conclude this section, we note that rather complex phenomena can be encoded with a RELBN using the constructs we have so far.

Example 6. Gilbert-style random graphs are rather simplistic models. Many other kinds of random graphs have been developed, to account for features of real networks [66]. A common feature of real graphs is that the distribution of degrees of sites follows a power law; such graphs are broadly referred to as *scale-free* networks. It is well-known that scale-free networks appear when a graph is grown by linking a new site to an older site with a probability that depends on how many connections the old site displays (this is a "rich gets richer" type of graph). Another way to generate scale-free networks is to use *preferential attachment*: each site gets a *fitness* value drawn from a probability distribution, and the probability that two sites are linked depends on a function of their fitnesses [11].

Here is one preferential attachment scheme that leads to scale-free behavior [11]. First, assume that the fitness of a site is an integer f between 1 and some M, drawn with probability proportional to $1/f^{\gamma}$ for some γ. Second, assume the probability of a link between two sites to be $f_1 f_2 / M^2$, where f_1 and f_2 are the fitness values of the sites. Fig. 6 depicts a graph produced by preferential attachment. As described in Appendix A, the whole process can in fact be encoded into a RELBN. Indeed, the graph in Fig. 6 was sampled from the Bayesian network produced by grounding the RELBN in Appendix A, with 750 sites, adopting $M = 64$ and $\gamma = 1.6$. Note the presence of hubs (highly connected sites) and the heavy tail of the distribution of degrees; both properties are found in scale-free random graphs. □

3. Definability, inexpressivity, and the like

A systematic study of probabilistic patterns that can/cannot be expressed with RELBNs seems to be missing in the literature, with few exceptions [9,10,63].[1] In this short section we present a few ideas that set the stage for results detailed later.

First, note that inexpressibility proofs are indeed central to finite model theory [41,59], and many techniques developed for logical languages can also shed light on the definability of purely probabilistic patterns. For instance, note that we can specify full connectivity for random graphs with FFFO, as we have shown in Section 2. Now suppose we want to compute the probability that a Gilbert-style model generates a 2-colorable graph. Can we do it? Alas, a predicate that stands for 2-colorability *cannot* be expressed with first-order logic [59]. But we can write, using *existential second-order logic*:

$$\mathsf{colorable} \equiv\equiv \exists \mathsf{red} : \forall \mathcal{X}, y : \mathsf{link}(\mathcal{X}, y) \Rightarrow (\mathsf{red}(\mathcal{X}) \Leftrightarrow \neg \mathsf{red}(y)); \tag{10}$$

and then $\mathbb{P}(\mathsf{colorable} = 1)$ yields the desired probability, once we have a definition for link (Expression (7)). Similarly, a predicate $\mathsf{path}(\mathcal{X}, y)$, indicating existence of a path between sites $\mathcal{X}$ and y, cannot be defined with FFFO, but it can be defined with second-order logic [2]. We return to second-order specification languages in Section 6.2.

One can use results presented later to obtain genuinely probabilistic inexpressibility proofs. To illustrate a probabilistic inexpressibility result, consider Jaeger's suggestion that we should be able to use the "mean" of probability values in specifications [48,50]. For instance, we should be able to specify

$$\mathbb{P}(r(a_1) = 1) = \frac{\mathbb{P}(s(a_1) = 1) + \mathbb{P}(s(a_2) = 1) + \mathbb{P}(s(a_3) = 1)}{3}.$$

To be more precise, consider a simplified and abstracted version of Jaeger's proposal, as follows. For a predicate r with arity k, write

$$r(\mathcal{X}_1, \ldots, \mathcal{X}_k) \sim \mathsf{mean}(\phi(\mathcal{X}_1, \ldots, \mathcal{X}_k, y_1, \ldots, y_m) | \varphi(\mathcal{X}_1, \ldots, \mathcal{X}_k, y_1, \ldots, y_m))$$

where ϕ is a formula containing r and other predicates of the vocabulary and φ is a formula without predicates and without quantifiers (for instance, $\neg(\mathcal{X} = y)$). The predicates in ϕ other than r are understood as parents of r in the dependency graph. The formula ϕ must have $\mathcal{X}_1, \ldots, \mathcal{X}_k$ as free variables, plus other free variables $y_1, \ldots, y_m$ that must all appear in φ. Mimicking Jaeger's original proposal, the semantics of such a formula is obtained in two steps. Consider a grounding $r(a_1, \ldots, a_k)$. First, collect all groundings of $y_1, \ldots, y_m$ that satisfy $\varphi(a_1, \ldots, a_k, y_1, \ldots, y_m)$. Second, assign to $\mathbb{P}(r(a_1, \ldots, a_k))$ the mean of the probability of each such grounding. For instance, take

$$r(\mathcal{X}) \sim \mathsf{mean}(s(\mathcal{X}, y) | \neg(\mathcal{X} = y));$$

[1] Many deep results on the expressivity of Bayesian networks focus on independence relations in propositional settings [24,68,75,78].

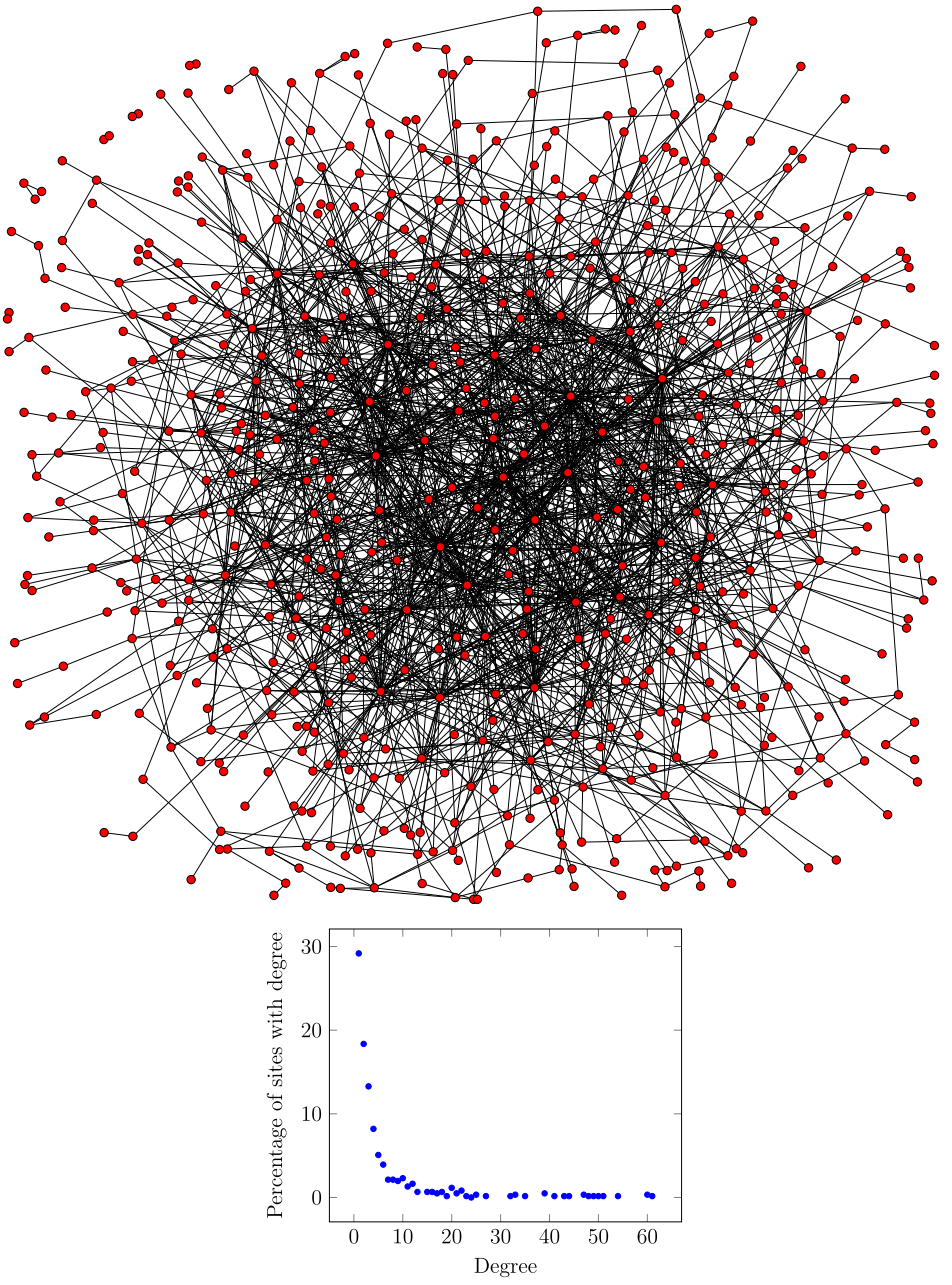


Fig. 6. A graph with 750 sites, generated using the fitness model in Example 6, where fitness is an integer f between 1 and 64, drawn with probability proportional to $1/f^{1.6}$. The graph has 1762 links; 140 sites are not linked to other sites, and are not shown in the figure. The average degree is 5.777, and some sites are connected to many other sites (the hubs can be seen at the center of the graph); one particular site has degree 61. The distribution of degrees is also shown, displaying a long tail and evidence of a power law over degrees.

for a domain $\mathcal{D} = \{a_1, a_2, a_3\}$, we obtain

$$\mathbb{P}(r(a_1)) = (\mathbb{P}(s(a_1, a_2)) + \mathbb{P}(s(a_1, a_3)) + \mathbb{P}(s(a_2, a_3)))/3.$$

Is it possible to encode the mean combination function using a RELBN? One might suspect that producing a mean operation with FFFO is no trivial matter. And in fact, as we show later using probabilistic techniques, it is indeed *impossible* to do so (Theorem 7).

To conclude this section, we note that there are other ways to study expressivity, as indicated in Section 1. We do not pursue every possible strategy in this paper; instead in the next sections we focus on the connection between expressivity and complexity, and on zero/one laws and their effect on definability.

4. Some complexity theory, and the complexity of inferences

We adopt standard concepts from complexity theory [67]. A *string* is a sequence of 0s and 1s. A *language* is a set of strings; a *complexity class* is a set of languages.

4.1. Turing machines and complexity classes

A non-deterministic Turing machine is specified as $(Q, \Sigma, q_0, Q_a, Q_r, \delta)$, a tuple where Q is a set of states; Σ is the alphabet (including 0, 1, and blank; the latter cannot be in the input); $q_0 \in Q$ is the initial state; Q_a is the set of accepting states; Q_r is the set of rejecting states; and δ is the transition function that takes a pair from $Q \times \Sigma$ (that is, current state and current symbol in tape) and returns a subset of $Q \times \Sigma \times \{-1, 0, 1\}$ (that is, next state, symbol to be written in tape, and head movement where -1 means left, 1 means right, and 0 means no motion); if δ always returns a singleton, the machine is in fact *deterministic*. If an input can lead the machine to stop in an accepting state, then the input is accepted; if the input leads the machine to always stop in a rejecting state, then the input is rejected. A language is *decided* by a Turing machine if the machine accepts each string in the language, and rejects each string not in the language. A language implicitly defines a *decision problem* that consists of deciding whether a given input string is in the language. We later use complexity classes such as P, NP, PSPACE, respectively the set of languages that can be decided by a deterministic Turing machine with a polynomial time bound, by a nondeterministic Turing machine with a polynomial time bound, and by a deterministic Turing machine with a polynomial space bound.

If a Turing machine is such that, whenever its transition function maps to a non-singleton set, the transition is selected with uniform probability within that set, then the Turing machine is a *probabilistic Turing machine*. The complexity class PP is the set of languages that are decided by a probabilistic Turing machine in polynomial time, with an error probability strictly less than $1/2$ for all input strings — that is, if the input is in the language, the machine accepts the input with probability strictly larger than half; if the input is not in the language, the machine accepts the input with probability less than or equal to half. This complexity class can be equivalently defined as follows: a language is in PP if and only if there is a polynomial nondeterministic Turing machine such that a string is in the language if and only if more than half of the computation paths of the machine end in accepting states when the string is the input (all other paths end in rejecting states). The class PP_1 is the set of languages in PP that have a single symbol (say 1) in the input vocabulary (so the input is always a sequence of 1s). And the class PEXP is the set of languages that are decided by a probabilistic Turing machine with an exponential time bound.

An oracle Turing machine $M^{\mathcal{L}}$, where $\mathcal{L}$ is a language, is a Turing machine with additional tapes, such that it can write a string ℓ to a tape and obtain from the oracle, in unit time, the decision as to whether $\ell \in \mathcal{L}$ or not. If A and B are sets of languages, then $\mathsf{A}^{\mathsf{B}} = \cup_{x \in \mathsf{B}} \mathsf{A}^x$, where $\mathsf{A}^{\mathcal{L}}$ is the set of languages that are decided by oracle Turing machines of the same sort and bound as A, but with additional oracle access to $\mathcal{L}$. For instance, $\mathsf{NP}^{\mathsf{NP}}$ is the class of languages that are decided by polynomial time nondeterministic Turing machines with access to an oracle in NP, while $\mathsf{PP}^{\mathsf{NP}}$ is the class of languages that are decided by polynomial time probabilistic Turing machines with access to an oracle in NP. Whole hierarchies can be built by stacking oracles. For instance, the *polynomial hierarchy* [80] consists of classes $\Sigma^p_{k+1} = \mathsf{NP}^{\Sigma^p_k}$ and its companions $\mathsf{coNP}^{\Sigma^p_k}$, for $k \geq 0$, with $\Sigma^p_0 = \mathsf{P}$. That is, the polynomial hierarchy consists of P and, recursively, NP^{C} and $\mathsf{coNP}^{\mathsf{C}}$, for any class C in the polynomial hierarchy. And Wagner's *counting hierarchy* [84] consists of classes P and, recursively, PP^{C}, NP^{C}, and $\mathsf{coNP}^{\mathsf{C}}$, for any class C in the counting hierarchy [82, Theorem 4.1].

A *polynomial-time many-one reduction* from language $\mathcal{L}$ to language $\mathcal{L}'$ is an algorithm that takes an input string ℓ and transforms it into a string ℓ' such that $\ell \in \mathcal{L}$ if and only if $\ell' \in \mathcal{L}'$. A language *is reduced* to another language if and only if there is a polynomial time many-one reduction from the former to the latter.

4.2. Data, inferential, query, domain complexity

The complexity of satisfiability for a logical language is the complexity of determining whether a given formula is satisfiable. And the complexity of model checking for a logical language is the complexity of determining whether a given formula is true with respect to a given domain and a given interpretation; if the formula is fixed and the pair domain/interpretation to be checked is the input, then one usually speaks of *data complexity*. These and other complexity notions have received detailed scrutiny in finite model theory [25,42].

An analogous study can be pursued with respect to the complexity of inferences in RELBNs. In fact we can distinguish at least three ways to quantify such complexity. One of these concepts (query complexity) resembles data complexity, and will be implicitly used later in our analysis of descriptive complexity.

The first situation is this. Suppose we take as our input a string describing: a RELBN τ specified using a logical language $\mathcal{L}$; a domain of size N; a conjunction of groundings $\mathbf{Q}$; another conjunction of groundings $\mathbf{E}$; and a rational γ. Suppose further that we accept the input string if and only if $\mathbb{P}(\mathbf{E}) > 0$ and $\mathbb{P}(\mathbf{Q}|\mathbf{E}) > \gamma$ with respect to the Bayesian network $\tau(N)$. Then the *inferential complexity* of $\mathcal{L}$ is complete for complexity class $\mathcal{C}$ if and only if the set of accepted input strings is in $\mathcal{C}$, and moreover if every language in $\mathcal{C}$ can be reduced to one such set of accepted strings.

Inferential complexity is not the only possible measure of complexity. Suppose that we fix a RELBN τ described using logical language $\mathcal{L}$, and we have input strings encoding domain of size N, $\mathbf{Q}$, $\mathbf{E}$, and γ. For each fixed τ, the problem is

Table 1
Various patterns of inferential, query and domain complexity for RELBNs. All cells indicate completeness with respect to polynomial-time many-one reductions. As we restrict input $\mathbf{Q}$ and $\mathbf{E}$ to conjunctions of groundings (with no negation), we only allow "positive" evidence [16].

Language	Inferential	Query	Domain
Prop($\wedge$)	P	P	—
Prop($\wedge, \neg$), Prop($\wedge$), Prop($\vee$)	PP	P	—
FFFO	PEXP	PP	PP_1
FFFO with bound on relation arity	PSPACE	PP	PP_1
FFFO^k with $k \geq 3$	PP	PP	PP_1
$\mathsf{DLLite}^{\mathsf{nf}}$	P	P	P

again to decide whether $\mathbb{P}(\mathbf{E}) > 0$ and $\mathbb{P}(\mathbf{Q}|\mathbf{E}) > \gamma$ with respect to $\tau(N)$. We say that the *query complexity* of $\mathcal{L}$ is complete for complexity class $\mathcal{C}$ if and only if the set of accepted input strings is in $\mathcal{C}$ for each τ, and every language in $\mathcal{C}$ can be reduced to the set of accepted strings induced by some τ. We can go one step further and consider *domain complexity*: here we have the same definition as query complexity, but $\mathbf{Q}$ and $\mathbf{E}$ are also fixed, and the input consists only of domain with size N (with γ fixed at $1/2$). Domain complexity conveys the cost of doing a fixed inference as the domain grows.

To see that distinct logical languages display varying levels of complexity, consider Table 1 [7,16,79]. The inferential, query and domain complexities for several languages are shown: Prop($\mathcal{R}$) denotes a propositional language (all predicates with arity zero) restricted to operators in $\mathcal{R}$; FFFO^k is FFFO restricted to k symbols for logical variables; $\mathsf{DLLite}^{\mathsf{nf}}$ corresponds to the negation-free fragment of the description logic DL-Lite [3,12]. In short, $\mathsf{DLLite}^{\mathsf{nf}}$ consists of all logical definitions containing unary predicates, conjunction, and the constructs $\exists x : r(x, y)$ and $\exists y : r(x, y)$ (that is, binary predicates only appear in such restricted existential quantification), additionally restricted to only two symbols for logical variables (x and y). Another point to mention regarding Table 1 is the fact that domain complexity is often related to complexity class PP_1, because the input can be taken simply as a sequence of identical symbols, one per element of the domain.

The analysis of query and domain complexity is closely related, respectively, to *dqe liftability* and *domain liftability*. These two concepts have been proposed to study *lifted* inference; that is, to study inference methods that avoid dealing explicitly with all groundings, rather operating with predicates themselves [72,83]. A specification language is domain-liftable when its domain complexity is polynomial; it is dqe-liftable when its query complexity is polynomial — in both cases the idea is to characterize "liftability" by requiring the complexity of inferences to avoid the blow-up caused by grounding [53,54]. Moreover, query complexity is often analyzed in connection with probabilistic databases [7,79].

There are many ways to enlarge Table 1, as we note in Section 8; however, here we are more interested in using complexity theory to capture expressivity. This is the goal of descriptive complexity, to which we now turn.

5. A bit of descriptive complexity

Descriptive complexity reveals the expressive power of a logical language by indicating precisely the kind of Turing machines that can be described by the language. At the same time, by showing that complexity classes can be described solely by logical means, descriptive complexity demonstrates that Turing machines are not needed to define complexity classes. The central idea is that a logical language may "capture" a complexity class; to formalize this notion, we need a few preliminary concepts.

A pair domain/interpretation is a *structure*. If $\phi(x_1, \ldots, x_k)$ is a first-order formula with free variables $x_1, \ldots, x_k$, then structure $\mathfrak{A}$ is a *model* of $\phi(a_1, \ldots, a_k)$ if and only if $\phi(x_1, \ldots, x_k)$ is true in structure $\mathfrak{A}$ when the logical variables $x_1, \ldots, x_k$ are replaced by elements $a_1, \ldots, a_k$ of the domain.

There is an *isomorphism* between structures $\mathfrak{A}_1$ and $\mathfrak{A}_2$ if and only if there is a bijective mapping g between the domains such that if $r(a_1, \ldots, a_k)$ is true in $\mathfrak{A}_1$, then $r(g(a_1), \ldots, g(a_k))$ is true in $\mathfrak{A}_2$, and moreover if $r(a_1, \ldots, a_k)$ is true in $\mathfrak{A}_2$, then $r(g^{-1}(a_1), \ldots, g^{-1}(a_k))$ is true in $\mathfrak{A}_1$ (g^{-1} denotes the inverse of g). An *isomorphism-closed* set of structures is a set of structures such that, whenever a structure is in the set, all structures that are isomorphic to it are also in the set.

We assume that every structure is given as a string, encoded as follows for a fixed vocabulary where the maximum predicate arity is K [41, Section 3.1.5]. Suppose the domain contains elements $a_1, \ldots, a_N$. To encode interpretations with respect to the domain, we need to order the elements of the domain, say $a_1 < a_2 < \cdots < a_N$; assume an order is selected. The string begins with N symbols 1 followed by $N^K - N + 1$ symbols 0. We take some order for the predicates, $r_1, \ldots, r_n$. We then append, in this order, the encoding of the interpretation of each predicate. Focus on predicate r_i of arity k. Using the linear ordering on the domain, we can enumerate lexicographically all k-tuples over the domain (there are N^k such tuples). We then encode the interpretation of r_i by N^k symbols followed by $N^K - N^k$ symbols 0; the jth first symbol (up to the N^kth symbol) is 1 if the jth k-tuple belongs to the interpretation, and 0 otherwise. Thus we have $1 + (n + 1)N^K$ symbols in the string.

Logical language $\mathcal{L}$ is said to *capture* complexity class $\mathcal{C}$ when exactly two conditions are met. First, the data complexity of $\mathcal{L}$ is in $\mathcal{C}$. Second, for each isomorphism-closed set of finite structures $\mathcal{S}$, for a non-empty vocabulary $\mathcal{V}$, such that these structures are encoded as strings of a language in $\mathcal{C}$, there is a sentence ϕ in $\mathcal{L}$, with vocabulary $\mathcal{V}$, such that $\mathcal{S}$ is exactly

the set of all models of ϕ. Or, more concisely, $\mathcal{L}$ captures $\mathcal{C}$ if and only if: for any isomorphism-closed set $\mathcal{S}$ of finite structures of some non-empty finite vocabulary, $\mathcal{S}$ is in $\mathcal{C}$ if and only if $\mathcal{S}$ is the class of finite models of a sentence in $\mathcal{L}$.

Note that descriptive complexity is distinct from data complexity. For example, the data complexity of FFFO is within polynomial bounds, but it is not the case that FFFO captures polynomial complexity: 2-colorability of a given graph (a graph can be encoded as a structure) can be checked in polynomial time, but 2-colorability cannot be expressed in FFFO.

To continue this example, note that 2-colorability can be expressed with *existential function-free second-order logic* (Expression (10)). A formula ϕ in such logic has the form $\exists r_1 \ldots \exists r_n \phi'$, where ϕ' is a formula of FFFO containing *predicate variables* $r_1, \ldots, r_n$, and possibly other predicate variables. If ϕ has no free predicate variables nor free logical variables, then ϕ is a sentence. A structure $\mathfrak{A}$ is, as before, a pair domain/interpretation. The set of well-formed formulas in existential function-free second-order logic is denoted by ESO.

Possibly the most celebrated result in descriptive complexity is Fagin's theorem: ESO captures NP [29]. Or, in more detail:

Theorem 1. *Let $\mathcal{S}$ be an isomorphism-closed set of finite structures of some non-empty finite vocabulary. Then $\mathcal{S}$ is in complexity class* NP *if and only if $\mathcal{S}$ is the class of finite models of a sentence in* ESO.

Fagin's theorem offers a definition of NP that is not tied to any computational model; rather, it is tied to the expressivity of ESO. It is actually not surprising that NP contains the problem of deciding whether an input structure is a model of a fixed ESO sentence; the surprising part of Fagin's theorem is that every language in NP can be exactly encoded by an ESO sentence.[2]

The proof of Fagin's theorem employs a predicate $<$ to determine when an element of the domain precedes another (elements of the domain are employed as time steps and as positions of a tape). However, this predicate $<$ is not in the input vocabulary; its interpretation as a linear order (over the input domain) cannot be given in the input. An important point is that a linear order can be generated with ESO by introducing $<$ not as a predicate, but rather as a predicate variable that is properly constrained. To guarantee that the predicate variable $<$ indeed behaves as a linear order, call $\phi_<$ the universal closure of $\neg(x < x)$ and $(x < y \wedge y < z) \Rightarrow x < z$ and $(x < y) \vee (y < x) \vee (x = y)$. Finally, introduce the ESO sentence $\exists <: \phi_<$, to guarantee that $<$ is built without the need to enlarge the input vocabulary. We later use a similar argument when we capture various complexity classes.

6. The descriptive complexity of relational Bayesian network specifications

In this section we show that RELBNs capture PP, while second-order extensions of RELBNs capture $\mathsf{PP}^{\Sigma_k^p}$, for each k. Mimicking our previous discussion, an specification language $\mathcal{L}$ is said to *capture* complexity class $\mathcal{C}$ if and only if: for any isomorphism-closed set $\mathcal{S}$ of finite structures of some non-empty finite vocabulary, $\mathcal{S}$ is in $\mathcal{C}$ if and only if $\mathcal{S}$ is the class of strings accepted by an specification in $\mathcal{L}$. For this definition to make sense, a preliminary question is how exactly to accept an input structure when we have a specification τ based on $\mathcal{L}$.

We adopt the following scheme to accept/reject an input structure $\mathfrak{A}$ of some non-empty vocabulary $\mathcal{V}$ using a specification τ whose vocabulary contains $\mathcal{V}$. First, note that structure $\mathfrak{A}$ defines a domain $\mathcal{D}$ and consequently a probabilistic model $\tau(\mathcal{D})$ out of τ. Denote by $\mathbb{P}_{\tau(\mathcal{D})}$ the probability measure defined by $\tau(\mathcal{D})$. The particular form of $\mathbb{P}_{\tau(\mathcal{D})}$ depends on $\mathcal{L}$; for instance, for a RELBN τ the resulting $\mathbb{P}_{\tau(\mathcal{D})}$ is encoded by a Bayesian network. Second, note also that each grounding of a predicate in $\mathcal{V}$ appears in $\tau(\mathcal{D})$ as a random variable; this random variable is set to true or false by the interpretation in $\mathfrak{A}$. The *evidence* $\mathbf{E}$ induced by $\mathfrak{A}$ is just the set of these assignments to random variables corresponding to groundings of predicates in $\mathcal{V}$. Third, we assume that a specification τ that accepts/rejects input structures contains two distinguished predicates of zero arity, A and B; the predicate A is the *conditioned predicate*, while B is the *conditioning predicate*. Finally, we adopt the following convention: if $\mathbb{P}_{\tau(\mathcal{D})}(A = 1 | B = 1, \mathbf{E}) > 1/2$, then accept $\mathfrak{A}$; otherwise, reject $\mathfrak{A}$.

We can now specialize these definitions to particular specification languages.

6.1. Capturing PP

Consider the class of RELBNs, where definitions are based on formulas in FFFO. We show that this specification language captures PP. The easy part is that the query complexity of RELBNs is in PP. The harder part is to show that any language of structures in PP can be "decided" by some RELBN τ on an extended vocabulary $\mathcal{V}'$ containing the input vocabulary $\mathcal{V}$ plus a conditioned predicate A and a conditioning predicate B.

As already noted, from a given structure $\mathfrak{A}$ we can produce a Bayesian network $\tau(\mathcal{D})$ from a RELBN τ; hence $\mathbb{P}_{\tau(\mathcal{D})}$ is the probability measure encoded by a Bayesian network $\tau(\mathcal{D})$. Note that such a probability distribution is over the set of interpretations; we still have structures as usual but the focus now is on probability distributions over structures (that is, over their interpretations).

Now that we have the proper context, here is the main result:

[2] Note that any string in a language in NP can be turned into a structure for some finite vocabulary [41, Section 3.1.5].

Theorem 2. *Let $\mathcal{S}$ be an isomorphism-closed set of finite structures of some non-empty finite vocabulary. Then $\mathcal{S}$ is in complexity class* PP *if and only if $\mathcal{S}$ is the class of finite structures that are accepted by a fixed* RELBN *with fixed conditioned and conditioning predicates.*

Proof. If $\mathcal{S}$ is the class of structures that are accepted by a fixed RELBN, we must show that $\mathcal{S}$ can be decided by a polynomial time probabilistic Turing machine. This is the same as proving that the query complexity of RELBNs based on FFFO is in PP, a result already in the literature [16, Theorem 7]. The proof of that result builds a nondeterministic Turing machine such that the input structure $\mathfrak{A}$ is accepted by more than half of the computation paths if and only if $\mathbb{P}(A=1|B=1,\mathbf{E}) > 1/2$. To summarize, the machine starts by guessing a truth assignment for all groundings of root nodes (in such a way that a grounding of r, where r is associated with $\mathbb{P}(r)=\alpha$, is assigned true in a fraction α of computation paths). Now each complete truth assignment over groundings of root predicates defines an interpretation. Recall that model checking of a fixed function-free first-order sentence is in P [59]. Then the machine verifies, in polynomial time, whether $\{B=1,\mathbf{E}\}$ holds: if not, it moves to a state q_1; if yes, the machine verifies whether $\{A=1,B=1,\mathbf{E}\}$ holds. If the latter check is negative, the machine moves to a state q_2; if the check is positive, then the machine moves to a state q_3. After reaching q_1 or q_2 or q_3, the machine branches in a particular way to guarantee that more than half of its computation paths are accepted if and only if $\mathbb{P}(A=1|B=1,\mathbf{E}) > 1/2$ (an identical strategy has been used to prove the complexity of inference in Bayesian networks [20, Theorems 11.3 and 11.5]). From q_1, branch into a path that accepts and one that rejects; from q_2, branch into two paths that reject; and from q_3, branch into two paths that accept. Denote by N_i the number of computation paths that reach q_i. The number of accepting computation paths is N_1+2N_3, and the total number of computation paths is $2(N_1+N_2+N_3)$. The input is accepted by this Turing machine iff $(N_1+2N_3)/(2(N_1+N_2+N_3)) > 1/2$, and this is equivalent to $\mathbb{P}(A=1|B=1,\mathbf{E}) > 1/2$ because, by construction, $N_1/(N_1+N_2+N_3)$ is equal to $1-\mathbb{P}(B=1,\mathbf{E})$ and $N_3/(N_1+N_2+N_3)$ is equal to $\mathbb{P}(A=1,B=1,\mathbf{E})$. Thus the machine does decide the input language correctly.

To prove the other direction, we adapt the proof of Fagin's theorem as described by Grädel [41], similarly to work by Saluja et al. [76, Theorem 1]. So, suppose that $\mathcal{S}$ is encoded as a language decided by some probabilistic Turing machine. Equivalently, there is a nondeterministic Turing machine $\mathbb{TM}$ that determines whether the majority of its computation paths accept an input, and the input structure is accepted/rejected accordingly. Then there is a first-order sentence $\phi_{\mathbb{TM}}$, with vocabulary consisting of the vocabulary of the input plus some additional predicates, such that each interpretation of this joint vocabulary is a model of the sentence if and only if it encodes a computation path of the Turing machine, as long as there is an available additional predicate that is guaranteed to be a linear order on the domain. We briefly review Grädel's construction of this sentence $\phi_{\mathbb{TM}}$, only emphasizing the points that are relevant to the present proof.

First, the size of the input string is polynomial on the size N of the domain $\mathcal{D}$ in the input structure $\mathfrak{A}$; as $\mathbb{TM}$ runs for a number of steps that is a known polynomial of the length of the input string, we can assume that $\mathbb{TM}$ stops (with acceptance or rejection) after exactly N^M-1 steps, where M is some integer. So we must only represent N^M-1 time steps, and N^M-1 positions in the machine tape. We create a "time index" by using M logical variables in predicates; there are N^M ways to substitute these logical variables by elements of $\mathcal{D}$, and therefore we have N^M tuples of elements. We refer to such a tuple by $\vec{x}$. Similarly, we index tape positions using M logical variables in predicates; we refer to such a tuple of logical variables by $\vec{y}$.

Second, the encoding of a Turing machine requires a total order over elements of the domain. As discussed after the statement of Fagin's theorem (Theorem 1), we can express all conditions over a linear order $<$ with a formula $\phi_<$ in FFFO. So, introduce a predicate variable $<$, later to be forced to behave as a linear order. Using $<$, it is possible to define with FFFO a successor predicate, a predicate first that indicates the first element, and a predicate last that indicates the last element (Grädel uses constants to represent these elements, but we do not have constants). For instance, the predicate first is associated with definition $\text{first}(x) \equiv \neg\exists y : y < x$.

We use predicates X_q (one per state), Y_σ (one per symbol), and Z: $X_q(\vec{x})$ means that state is q at time step $\vec{x}$; $Y_\sigma(\vec{x},\vec{y})$ means that the symbol in position $\vec{y}$ is σ at time step $\vec{x}$; $Z(\vec{x},\vec{y})$ means that the head is at position $\vec{y}$ at time step $\vec{x}$. It is necessary to guarantee that, at each step, a single state is true, each tape position holds a single symbol, and the head is at a single position; all those conditions can be expressed by a formula ϕ_a in FFFO. For instance, we must impose $\forall x : \bigvee_q X_q(x) \wedge \bigwedge_{q'\neq q} \neg X_{q'}(x)$ to guarantee that a single state is true at each step.

Grädel introduces two formulas, referred to as START and COMPUTE, such that the universal closure of their conjunction encodes the whole behavior of $\mathbb{TM}$ using predicates X_q, Y_σ and Z, plus the predicates in $\mathcal{V}$ and some auxiliary predicates as described previously (we do not repeat these two formulas here). Denote by ϕ_S the universal closure of START, and by ϕ_C the universal closure of COMPUTE. Another formula, referred to as ϕ_E, is true if and only if an accepting state is reached by $\mathbb{TM}$: $\exists x : \bigvee_{q\in Q_a} X_q(x)$. Grädel's argument shows that, given an interpretation of $<$ that satisfies $\phi_<$, each interpretation of $\phi'_{\mathbb{TM}} = \phi_a \wedge \phi_S \wedge \phi_C$ corresponds exactly to a computation sequence of $\mathbb{TM}$; moreover, each interpretation of $\phi_{\mathbb{TM}} = \phi'_{\mathbb{TM}} \wedge \phi_E$ corresponds exactly to a computation sequence of $\mathbb{TM}$ that ends up in acceptance. Grädel's proof of Fagin's theorem is then finished by taking the ESO sentence $\exists <: \exists\{X_q\} : \exists\{Y_\sigma\} : \exists Z : \phi_< \wedge \phi_{\mathbb{TM}}$. This last sentence is not used in our construction; rather, it is replaced by probabilistic constructs.

For our purposes we must impose a uniform probability measure over all interpretations of X_q, Y_σ, and Z, to obtain a uniform probability measure over all possible computations. Assume for a moment that a linear order $<$ is given (satisfying $\phi_<$). Introduce, for all states q and all symbols σ,

$$\mathbb{P}(X_q) = 1/2, \qquad \mathbb{P}(Y_\sigma) = 1/2, \qquad \mathbb{P}(Z) = 1/2, \tag{11}$$

interpreted as if we were specifying a RELBN. Of course not all interpretations of these predicates satisfy $\phi'_{\mathbb{T}\mathbb{M}}$ and $\mathbf{E}$. To guarantee that we are only dealing with interpretations satisfying $\phi'_{\mathbb{T}\mathbb{M}}$ and $\mathbf{E}$, we must condition on the event that both $\phi'_{\mathbb{T}\mathbb{M}}$ and $\mathbf{E}$ hold. Also, we are interested in deciding whether, *among those satisfying interpretations*, more than half of them correspond to accepting computations. So we introduce B through the logical definition $B \equiv\equiv \phi'_{\mathbb{T}\mathbb{M}}$, and introduce A through the logical definition

$$A \equiv\equiv \phi_E; \tag{12}$$

then $\mathbb{P}(A = 1|B = 1, \mathbf{E})$ yields

$$\mathbb{P}\left(\phi_E|\phi'_{\mathbb{T}\mathbb{M}} \wedge \mathbf{E}\right) = \frac{\mathbb{P}\left(\phi_E \wedge \phi'_{\mathbb{T}\mathbb{M}} \wedge \mathbf{E}\right)}{\mathbb{P}\left(\phi'_{\mathbb{T}\mathbb{M}} \wedge \mathbf{E}\right)} = \frac{\mathbb{P}(\phi_{\mathbb{T}\mathbb{M}} \wedge \mathbf{E})}{\mathbb{P}\left(\phi'_{\mathbb{T}\mathbb{M}} \wedge \mathbf{E}\right)}.$$

In the previous paragraph we assumed the interpretation of $<$ to be given as input. In Fagin's theorem, the interpretation of $<$ is "internalized" by the second-order quantification $\exists <$. Here we must instead resort to probabilities. We introduce the probabilistic assessment

$$\mathbb{P}(<) = 1/2, \tag{13}$$

and add the linear order to the definition of B:

$$B \equiv\equiv \phi'_{\mathbb{T}\mathbb{M}} \wedge \phi_<. \tag{14}$$

Take the RELBN τ consisting of Expressions (11), (12), (13), and (14); by producing the Bayesian network $\tau(\mathcal{D})$ where $\mathcal{D}$ is the domain in the input structure $\mathfrak{A}$, we have:

$$\mathbb{P}_{\tau(\mathcal{D})}(A = 1|B = 1, \mathbf{E}) = \frac{\mathbb{P}(\phi_< \wedge \phi_{\mathbb{T}\mathbb{M}} \wedge \mathbf{E})}{\mathbb{P}\left(\phi_< \wedge \phi'_{\mathbb{T}\mathbb{M}} \wedge \mathbf{E}\right)},$$

as desired. Note that there are actually $N!$ linear orders that satisfy $\phi_<$, but for each one of these linear orders we have corresponding interpretations for all other predicates, hence the ratio between accepting computations and all computations is as desired. Thus the input structure is accepted by the majority of computations in $\mathbb{T}\mathbb{M}$ if and only if $\mathbb{P}_{\tau(\mathcal{D})}(A = 1|B = 1, \mathbf{E}) > 1/2$. □

6.2. Moving to second-order

Suppose we have a specification that follows the syntax of RELBN, except for the fact that one logical definition contains, in its right hand side, a formula in ESO without free predicate variables. For instance, consider the specification of a Gilbert-style random graph with an indicator of 2-colorability, produced by putting together Expressions (7) and (10):

$$\begin{aligned} \mathbb{P}(\mathsf{dirlink}) &= \alpha, \\ \mathsf{link}(x, y) &\equiv\equiv \neg(x = y) \wedge (\mathsf{dirlink}(x, y) \vee \mathsf{dirlink}(y, x)), \\ \mathsf{colorable} &\equiv\equiv \exists \mathsf{red} : \forall x, y : \mathsf{link}(x, y) \Rightarrow (\mathsf{red}(x) \Leftrightarrow \neg\mathsf{red}(y)). \end{aligned}$$

We denote this set of specifications by 1ESOBNs. To be more precise, a 1ESOBN consists of three parts. First, we have a finite vocabulary $\mathcal{V}$ containing names of predicates. Some predicates in $\mathcal{V}$ are associated with probabilistic assessments as in RELBNs (such as Expression (5)), while the remaining predicates are associated with logical definitions as in RELBNs (6), except for one predicate r in $\mathcal{V}$. The third part of the specification is a single *second order existential* definition associated with r:

$$r(x_1, \ldots, x_k) \equiv\equiv \exists s_1 \ldots \exists s_m : \phi(x_1, \ldots, x_k, s_1, \ldots, s_m), \tag{15}$$

where $s_1, \ldots, s_m$ are predicate variables, and $\phi(x_1, \ldots, x_k, s_1, \ldots, s_m)$ is a formula of FFFO whose extralogical symbols are only names in $\mathcal{V}$, or predicate variables in $\{s_1, \ldots, s_m\}$, or free logical variables $x_1, \ldots, x_k$, or possibly other logical variables that are bound to quantifiers in $\phi(x_1, \ldots, x_k, s_1, \ldots, s_m)$. Note that at this point we have in essence abandoned the graph-based scheme of Bayesian networks; the syntax of these second-order Bayesian networks is entirely based on textual formulas that are not easily translated into graphs (even though one can certainly create a dependency graph for a second-order Bayesian network).

What kind of complexity class is captured by 1ESOBNs? We adopt a strategy for acceptance/rejection that mimicks the strategy for acceptance/rejection we adopted for RELBNs. That is, we have an input structure on a vocabulary $\mathcal{V}$, and we assume that there is a conditioned predicate A and a conditioning predicate B in the 1ESOBN; we also assume that quantified predicates are not in $\mathcal{V}$. The decision as to whether to accept or reject the input structure is based on whether the probability of A given B and $\mathbf{E}$ (the evidence conveyed by the input structure) is larger than $1/2$ or not.

We have[3]:

Theorem 3. *Let $\mathcal{S}$ be an isomorphism-closed set of finite structures of some non-empty finite vocabulary. Then $\mathcal{S}$ is in complexity class* $\mathsf{PP}^{\mathsf{NP}}$ *if and only if $\mathcal{S}$ is the class of finite structures that are accepted by a fixed* 1ESOBN *with fixed conditioned and conditioning predicates.*

Proof. To prove that a class of finite structures that are accepted by a fixed 1ESOBN can be decided within $\mathsf{PP}^{\mathsf{NP}}$, put together the argument in the first paragraph of the proof of Theorem 2 and the fact that an ESO sentence can be evaluated with an NP oracle [80]. That is, there is a "base" Turing machine that gets an input structure and proceeds as the machine in the proof of Theorem 2, with the proviso that model checking is not in P anymore; rather, one must resort to an oracle to do the model check. Because an NP oracle suffices to model check an ESO sentence, the desired query complexity is proved.

To prove the other direction, the proof of Theorem 2 must be modified. We use again the same predicates X_q, Y_σ and Z for the base machine, plus predicates Y'_σ and Z' that control the content of the auxiliary tape where strings are written to and received from the oracle. All of those predicates are root predicates associated with probability $1/2$, to guarantee again that all interpretations have identical probability. The predicates are used in constraints following the machinery of Grädel's proof of Fagin's theorem [41]. That is, again we have a linear order and predicates that define a first element and a successor relation. The elements of the domain are used both to mark the time steps and to indicate tape positions. Formulas ϕ_S and ϕ_E are just as before, respectively setting up all predicates, and detecting acceptance. And the COMPUTE formula has the same behavior as before, *except* that now it must deal with transitions from the base Turing machine to the oracle Turing machine.

So, there must be a distinguished state q' such that, once the machine is in this state, its next transition depends on the result of the oracle machine. The behavior of the oracle machine is then encoded with a single ESO formula with quantified predicate variables X^o_q, Y^o_σ, Z^o (the superscript o refers to the "oracle"), with access to Y'_σ and Z'. This formula is exactly the formula built in Grädel's proof of Fagin's theorem, so we avoid repeating it here.

In the proof of Theorem 2 it was necessary to introduce logical variables that "mark the steps" of the Turing machine (these steps are ordered by the introduced linear order), and similarly to refer to the position in the tape. Here we also need a set of logical variables to mark the steps, but we also need logical variables that mark the steps of the oracle machine; also we need two sets of logical variables, one to refer to the position in the base tape, the other to refer to the position in the oracle tape. And of course a linear order over the domain must also be built as in the proof of Theorem 2. Again, an input structure is accepted by the majority of computation paths in the (base) Turing machine if and only if we have $\mathbb{P}(A = 1 | B = 1, \mathbf{E}) > 1/2$, where A, B and $\mathbf{E}$ are as in the proof of Theorem 2. Note that the number of possible structures satisfying the second-order sentence that describes the oracle machine may be very large, but this number is not counted (only the computations in the base machine are counted). □

Note that by allowing an increasing number of second-order quantifier blocks, we can capture other complexity classes in the counting hierarchy. For instance, we may allow k logical definitions that use ESO, using negation to produce universal quantification whenever necessary. For this scheme to increase expressivity, we must allow logical definitions to have free predicate variables; these predicate variables must be bound at some level in the definitions, and if included in the dependency graph they cannot create a cycle. Alternatively, we may allow a single logical definition to use a formula from second-order logic that starts with k alternating quantifier blocks. In both cases we say that we have a specification in kESOBN. With such a specification we can write down formulas in Σ^1_k; that is, formulas that consist of a FFFO formula preceded by k blocks of distinct quantifiers, each block either existential or universal, and the first block existential. In fact, Σ^1_k captures the complexity class Σ^p_k in the polynomial hierarchy [80], so we can capture various classes in the counting polynomial hierarchy:

Theorem 4. *Let $\mathcal{S}$ be an isomorphism-closed set of finite structures of some non-empty finite vocabulary. Then $\mathcal{S}$ is in complexity class* $\mathsf{PP}^{\Sigma^p_k}$ *if and only if $\mathcal{S}$ is the class of finite structures that are accepted by a fixed k*ESOBN *with fixed conditioned and conditioning predicates.*

Proof. The proof of this theorem is similar to the proof of Theorem 3; hence we only indicate the necessary changes.

To prove that a class of finite structures that are accepted by a fixed kESOBN can be decided within $\mathsf{PP}^{\Sigma^p_k}$, take a "base" Turing machine that gets an input structure and proceeds as the machine in the proof of Theorem 2, with the proviso that model checking is not in P anymore; rather, one must resort to an oracle to do the model check. Because a Σ^p_k oracle suffices to model check an sentence in Σ^1_k [80], the desired query complexity is proved.

[3] In a previous publication we argued that second-order Bayesian network specifications capture $\mathsf{PP}^{\mathsf{NP}}$, failing to state the necessary restriction to a single logical definition based on a second-order formula [17]. We correct that mistake here.

To prove the other direction, use again the same predicates X_q, Y_σ and Z for the base machine, plus predicates Y'_σ and Z' that control the content of the auxiliary tape where strings are written to and received from the oracle. Again, all of those predicates are root predicates associated with probability 1/2, and the predicates are used in constraints as in Grädel's proof of Fagin's theorem; there is a linear order and associated predicates, and the elements of the domain mark both the time steps and tape positions (for the base and the oracle machines). Formulas ϕ_S and ϕ_E are built as before; also the COMPUTE formula deals with transitions, including transitions between base and oracle machines. And there must be a distinguished state q' such that, once the machine is in this state, its next transition depends on the result of the oracle machine. The behavior of the oracle machine is then encoded with a single Σ^1_k formula with access to Y'_σ and Z'; this formula can be built as Σ^1_k captures Σ^p_k [80]. Again, an input structure is accepted by the majority of computation paths in the (base) Turing machine if and only if we have $\mathbb{P}(A = 1|B = 1, \mathbf{E}) > 1/2$, where A, B and $\mathbf{E}$ are as before. □

6.3. A comment

These results on descriptive complexity can be interpreted intuitively as follows: suppose we have a (physical, social, economic) phenomenon that can be simulated by a probabilistic Turing machine in polynomial time: given an input, the machine runs for a number of steps that is polynomial in the length of the input, and the machine produces an output probabilistically, distributed exactly as the corresponding output of the phenomenon. Our first theorem shows that this phenomenon *can* be modeled by a relational Bayesian network specification in the sense that, given the input as evidence, then an inference with the network will manipulate the same probabilities as the phenomenon.

But what happens if the phenomenon is so complex that it requires even more computational power to be simulated? For instance, what happens if the phenomenon is so complicated as to require, for its simulation, a polynomial time probabilistic Turing machine with another nondeterministic Turing machine as oracle? This phenomenon cannot be modeled by a "first-order" Bayesian network specification, unless widely accepted assumptions about complexity classes collapse. However, our second theorem shows that this phenomenon *can* be modeled by a suitable "second-order" Bayesian network specification.

7. Zero/one laws for relational Bayesian network specifications

Suppose we have a vocabulary and that, for each domain, we assign a uniform probability measure over all interpretations. Take the probability of a function-free first-order sentence ϕ to be simply the probability of the set of interpretations that satisfy ϕ. Amazingly, the probability of ϕ converges, as the domain grows, either to 0 or to 1 under rather general circumstances. To be more precise, Fagin's zero/one law states that if ϕ is *any* sentence in FFFO with predicates of positive arity,[4] then $\mathbb{P}(\phi)$ converges, if we adopt uniform probabilities over all interpretations, either to 0 or to 1 [29,37]. Similar convergence properties have been proved for many logics; indeed the search for zero/one laws is a major topic in finite model theory [25,42].

One must be surprised: Why do these results appear in finite model theory at all, when they clearly belong to finite *probabilistic* model theory? In any case, little work is needed to transfer the zero/one law for FFFO into a zero/one law for relational Bayesian network specifications. We do it at once.

For a RELBN τ, where $\tau(N)$ is the Bayesian network produced by a domain of size N, denote by $\mathbb{P}_{\tau(N)}$ the corresponding probability measure. We are interested in the probability of some sentence ϕ, denoted by $\mathbb{P}_{\tau(N)}(\phi)$; that is, the probability, with respect to $\tau(N)$, of the set of interpretations that satisfy ϕ.

We have:

Theorem 5. *If τ is a* RELBN *containing only predicates of positive arity, then* $\lim_{N\to\infty} \mathbb{P}_{\tau(N)}(\phi)$ *exists and is either* 0 *or* 1.

Proof. The proof has two parts. First, we show that τ can be turned into another RELBN τ' such that all interpretations receive identical probability, and that we can build a sentence φ containing only the root predicates of the latter RELBN, such that $\mathbb{P}_{\tau'(N)}(\varphi) = \mathbb{P}_{\tau(N)}(\phi)$. Second, we invoke the zero/one law for FFFO to obtain the desired limit.

So, start with the input τ. If all root predicates are associated with probabilities 1/2, then all interpretations have identical probabilities and we make $\tau' = \tau$. So suppose some root predicates are not associated with probability 1/2. Take a root predicate r associated with a probabilistic assessment $\mathbb{P}(r) = \alpha$ where $\alpha \neq 1/2$. Suppose $\alpha = a/b$ where a and b are the smallest possible nonnegative integers, both encoded in binary notation with less than B bits. Introduce B predicates, each one representing a bit, and each one of them associated with probability 1/2. Thus a realization of these predicates is a nonnegative integer c with B bits. To generate a sample of r, it is then enough to decide whether $a/b < c$; if yes, then the sample assigns true to r; if no, then the sample assigns false to r. Deciding $a/b < c$ can be done by deciding $a < bc$; to do so, we need to multiply two binary numbers b and c and compare the result with a binary number a. These operations can be encoded with digital circuits, hence they can be translated into first-order logic (this sort of encoding is used in Appendix A

[4] Hence ϕ has no free variables and no functions, and consequently no constants, and also no propositions.

to produce samples). We can therefore transform τ into an enlarged RELBN τ' where all root predicates are associated with probability 1/2. Hence in τ' the probability measure over all grounded root predicates is uniform for any domain.

To compute $\mathbb{P}_{\tau(N)}(\phi)$, we can recursively replace each non-root predicate of τ' that appears in ϕ by its definition, until we only have root predicates. Call the resulting sentence φ; clearly $\mathbb{P}_{\tau'(N)}(\varphi) = \mathbb{P}_{\tau(N)}(\phi)$.

So, we have a uniform probability measure over the interpretations of the root predicates, and a single first-order sentence φ whose probability is just $\mathbb{P}_{\tau'(N)}(\varphi)$, for whatever N. The zero/one law for FFFO yields the desired convergence result. □

A simple example is given by the RELBN τ in Expression (7), by considering the probability of full connectedness:

$$\lim_{N\to\infty} \mathbb{P}_{\tau(N)}(\forall \mathcal{X}, y : \neg(\mathcal{X}=y) \Rightarrow \mathsf{link}(\mathcal{X}, y)) = \begin{cases} \lim_{N\to\infty} \alpha^{N(N-1)/2} = 0 & \text{if } \alpha < 1, \\ \lim_{N\to\infty} 1^{N(N-1)/2} = 1 & \text{if } \alpha = 1. \end{cases}$$

In fact the same construction described in the proof of Theorem 5 can be used to handle a sentence ϕ containing constants; in that case the basic convergence theorem by Glebskii et al. [37] implies convergence, but not necessarily convergence to 0 or 1.

A previous result on the convergence properties of Jaeger's relational Bayesian networks [49] similarly establishes convergence of probabilities (not necessarily to 0 or 1). Jaeger imposes some conditions on combination functions and focuses on the probability of groundings, adapting steps of Fagin's proof of the zero/one law as needed. Here we have a simpler specification language and therefore we can directly use Fagin's zero/one law in our proof.

Another byproduct of the construction in the proof of Theorem 5 is that we can easily pinpoint the complexity of computing the limiting value of a probability given a RELBN.

Theorem 6. *Consider a decision problem where the input is a* RELBN τ*, containing only predicates of positive and bounded arity, and a sentence* ϕ *on that vocabulary, and such that this input is accepted if and only if* $\lim_{N\to\infty} \mathbb{P}_{\tau(N)}(\phi) = 1$*. This decision problem is* PSPACE*-complete.*

Proof. Use the construction in the proof of Theorem 5, and then apply Grandjean's complexity analysis [43]. □

This theorem shows that the complexity of limiting inference with RELBNs with bounded arity is the same as their inferential complexity (indicated in Table 1).

These results just scratch the surface of what can be explored concerning the convergence properties of RELBNs. For instance, we have not discussed the convergence of conditional probabilities. Also, one can consider various other specification languages, based both on fragments and extensions of FFFO, so as to map which specification languages guarantee convergence. Finally, one can embark on a classification of specification languages concerning the complexity of computing limiting probabilities: for instance, what is this complexity for RELBNs restricted to the two-variable fragment of FFFO?

Instead of pursuing such possibilities, we want to end this section by establishing a connection with the previous discussion of definability. Indeed, zero/one laws are already used to prove logical inexpressibility; an example suffices to illustrate the method. Suppose we want to define a concept even, that is true exactly when the size of the domain is even. But $\mathbb{P}_{\tau(N)}(\mathsf{even})$ is 1 if the domain size is even, and 0 otherwise; there is no limiting probability, hence even cannot be defined with FFFO [59]. We can use the same reasoning to extract inexpressibility results from the zero/one law for RELBNs. For instance, consider the combination function mean described in Section 3.

Theorem 7. *It is impossible to define the combination function* mean *with a* RELBN *(that is, with a specification based on* FFFO*).*

Proof. Suppose we could in fact define mean within a RELBN. Then consider a RELBN as follows. First, take auxiliary predicate one, of arity one, associated with probabilistic assessment $\mathbb{P}(\mathsf{one}) = 1$, and add a logical definition for a predicate equal:

$$\mathsf{equal}(\mathcal{X}, y) \equiv\equiv (\mathcal{X} = y) \wedge \mathsf{one}(y).$$

Now introduce a dirlink predicate, associated with a mean combination function:

$$\mathsf{dirlink}(\mathcal{X}, y) \sim \mathsf{mean}\,(\mathsf{equal}(\mathcal{X}, z) | (z = z))\,.$$

The reading of this assertion is this: for a substitution $\mathcal{X}\backslash a, y\backslash b$, where a and b are elements of the domain, all possible groundings for z satisfy $(z = z)$; thus we have to collect the probability of $\mathsf{equal}(a, c)$ for each element c of the domain. When c is just equal to a, we get probability 1; otherwise we get probability 0. Hence the result of mean is probability $1/N$ where N is the size of the domain. As in Expression (7), define predicate link:

$$\mathsf{link}(\mathcal{X}, y) \equiv\equiv \neg(\mathcal{X} = y) \wedge (\mathsf{dirlink}(\mathcal{X}, y) \vee \mathsf{dirlink}(y, \mathcal{X})),$$

so that the probability of link between two elements of the domain is $(2/N) - (1/N^2)$. We can interpret, as before, each element of the domain as a site and each grounding $\mathsf{link}(a, b)$ as a link between sites a and b.

Finally, consider a sentence ϕ that is true if and only if the graph consisting of these sites and links contains a triangle:

$$\exists x, y, z : \neg(x = y) \wedge \neg(y = z) \wedge \neg(z = x) \wedge \mathsf{link}(x, y) \wedge \mathsf{link}(y, z) \wedge \mathsf{link}(z, x).$$

By a well-known result from the theory of random graphs we obtain $\lim_{N\to\infty} \mathbb{P}_{\tau(N)}(\phi) = 1 - e^{-2^3/6}$ (this limit was originally obtained for the evolution of the Erdös-Rényi graph with N sites and $N - 1$ links [28, Theorem 3a]; this random graph is asymptotically equivalent to $G(N, 2/N - 1/N^2)$). Because the limit is neither zero nor one, we conclude that mean cannot be defined using FFFO. □

8. Conclusion: a finite model theory of Bayesian networks?

We hope this paper is useful not only in presenting some novel results on descriptive complexity and zero/one laws for relational Bayesian network specifications, but also as a chart for a finite model theory for Bayesian networks. Thus we have included a relatively long presentation of RELBNs, and also a discussion of definability issues and known results about complexity of inferences.

We have introduced a theory of descriptive complexity for Bayesian networks, a topic that does not seem to have received due attention so far. To summarize, we have shown that RELBNs capture PP, and we have indicated how we can go beyond PP in our modeling tools. Specifically, we added existential second-order quantification to capture the complexity classes $\mathsf{PP}^{\Sigma_k^p}$.

We have also indicated how zero/one laws can be applied to RELBNs so as to obtain useful insights, as we have demonstrated in our inexpressibility proof for the mean combination function.

Altogether these results should offer some understanding about the expressivity of relational languages that specify Bayesian networks.

Our contributions can also be appreciated from two broad perspectives. First, there has been, for decades, significant study of model theory for probabilistic logics [1,5,31,32,45]. By dealing with domains of arbitrary cardinality, and with logics that include too many constructs (for instance, functions) and that exclude valuable tools (for instance, independence relations), these previous investigations arrive at results that are often too weak — for instance, almost always obtaining undecidability or very high computational complexity. By focusing on modular tools such as Bayesian networks, and by focusing on finite domains, we are able to obtain much sharper results, nailing down specific complexity classes such as PP and $\mathsf{PP}^{\mathsf{NP}}$. From a second perspective, we note that despite significant interest in relational variants of Bayesian networks during the last twenty years [34,57], there has been surprisingly little systematic study of their theoretical properties, with the exception of Jaeger's efforts [51]. By importing ideas and tools from finite model theory, we can explore several fruitful research questions that clarify the expressive power of these relational specification languages.

Our results are also interesting from a point of view centered on complexity theory. There has been little work on capturing counting/probabilistic classes; the most significant previous results capture #P using counting [76]. We offer a more concrete modeling language that captures PP, and we move into the counting hierarchy — we are not aware of any similar result in the literature. Our results show how classes in the counting hierarchy can be tied to the expressivity of modeling tools, not to any particular computational model (much as Fagin's theorem does for NP).

Much work is yet to be done to build a complete finite model theory of Bayesian networks. In Section 3 we have sketched a possible research program concerning inexpressibility of combination functions and other probabilistic patterns; such a program is largely open. Our discussion of complexity (both the complexity of inferences, and descriptive complexity) only dealt with some fragments of FFFO and ESO. There are many other possible fragments of these logical languages, and also many other extensions. For instance, finite model theory pays considerable attention to fixed-point logics, as those languages offer recursion and can be related to logic programming [25,59]; the addition of fixed-point operators to RELBNs can build a bridge to probabilistic logic programming [18]. In a different direction, counting quantifiers are very useful, for instance in description logics [4], and they should be investigated in depth. It would be particularly interesting to consider description logics that are fragments of first-order logic, so as to understand the expressivity and complexity of their probabilistic counterparts.

The analysis of complexity (in various forms) can be enlarged not only by investigating more specification languages, but also by contemplating other decision problems. For instance, one might check the complexity of Most Probable Explanations (MPE), or the expressivity that can be attained by computing Maximum-A-Posteriori configurations (MAP). Such decision problems are called "elementary problems" by Jaeger [51], as they are defined with respect to a particular grounding of a Bayesian network specification.

There are "non-elementary inference problems" that deserve attention as well. For instance, Jaeger's relational Bayesian networks may fail to produce a grounded Bayesian network; deciding whether this is the case for a particular relational specification is a *global semantics* problem. Even though there are algorithms that solve the global semantics problem in special cases [33,65], the general problem has been conjectured to be undecidable by Jaeger [51]. A precise translation of the global semantics problem into first-order logic with an added transitive closure operator has been produced recently

[21], but decidability remains an open problem. In this paper we have avoided questions of global semantics by requiring the dependency graph of any RELBN to be acyclic; with cycles, decisions about global semantics may be nontrivial.

Finally, we have only glanced over the theory of zero/one laws; there are many possible laws to be found by examining various specification languages, and it would also be important to investigate the computational complexity of computing limiting probabilities in those cases. Note that cycles in definitions, once incorporated in the study, allow one to specify dynamic models; in that context zero/one laws would be very interesting as they would be connected to convergence of state trajectories over time.[5]

Conflict of interest statement

There is no conflict of interest.

Acknowledgements

The first author is partially supported by CNPq grant #308433/2014-9; the second author is partially supported by CNPq grants #303920/2016-5 and #420669/2016-7. The work was partially supported by São Paulo Research Foundation (FAPESP) grants 2016/18841-0 and 2015/21880-4, and by the *Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil* (CAPES) Finance Code 001. Opinions and conclusions in this paper are sole responsibility of the authors and do not necessarily reflect FAPESP, CNPq or CAPES positions.

Appendix A. A relational Bayesian network specification for scale-free random graphs

Caldarelli et al. [11] show that the following abstract procedure generates graphs with scale-free behavior, with input N (number of sites) and γ. First, to each site χ assign a fitness value $f(\chi)$ with probability proportional to $f(\chi)^{-\gamma}$. Then, for each pair of sites (χ, y), introduce a link between them with probability $f(\chi)f(y)/F^2$, where F is the largest fitness value. Here we instantiate this general method by taking any fitness value to be an integer between 1 and some integer M that is a power of 2, and by taking the probability of a link between sites χ and y to be $f(\chi)f(y)/M^2$.

To create a relational Bayesian network specification that can reproduce these steps, we must encode various arithmetic operations using logical constructs. Hence we can only operate with binary numbers.

Our strategy is, for each site, to generate a uniformly distributed integer in binary notation from 0 to $2^B - 1$, for a selected number of bits B; then we use this integer to generate a fitness value with appropriate distribution; we then generate another uniformly distributed integer from 0 to $M^2 - 1$, and use this second integer and the fitness value to produce a link probability. We now explain each one of these steps in more detail.

To produce a uniformly distributed integer (in binary notation) from 0 to $2^B - 1$, we simply introduce predicates $\mathsf{bit}_1(\chi)$ to $\mathsf{bit}_B(\chi)$, where χ stands for a site. For each site, a configuration of the corresponding B random variables can be read as a binary number. Now associate the assessment $\mathbb{P}(\mathsf{bit}_i) = 1/2$ with each predicate bit_i. Once we fix a domain (that is, a set of sites), we have a uniformly distributed random integer per site. Of course we are not interested in these random integers themselves; we wish to generate a fitness value per site. But once we have a uniformly distributed random integer W, we can generate a fitness value with a desired distribution, using some pre-computed numbers. For instance, suppose we have parameter $\gamma = 3$ and $M = 4$ (that is, possible fitness values are 1, 2, 3, 4). The probability that the fitness of a node is equal to f is $(1/f^3)\nu$, where ν is a normalization constant such that $\nu^{-1} = \sum_{j=1}^{4}(1/j^3) = 2035/1728$. Now suppose $B = 13$ and we have an integer w that is a realization of a uniformly distributed random integer between 0 and $2^{13} - 1$. We must compare w with $2^{13}\sum_{j=1}^{k}(1/j^3)\nu$ for increasing values of k, until we find k such that $w \leq k$; at that point we stop and declare $f(\chi)$ to be k. In general we can pre-compute numbers $2^B\sum_{j=1}^{k}(1/j^\gamma)\nu$ for k from 1 to M and $\nu^{-1} = \sum_{j=1}^{M}(1/j^\gamma)$, so the necessary operations are comparisons with fixed numbers, and an encoding of the result of these comparisons can be produced with $\log_2 M$ predicates. It is actually simple to encode such operations with a digital circuit (parameterized by χ), performing comparisons and some additional binary encoding of the fitness value. Clearly a digital circuit can be encoded through logical expressions that are all parameterized by χ.

The final step is to multiply the binary numbers encoding $f(\chi)$ and $f(y)$ for distinct sites χ and y, and then to compare the result with a random integer w' between 0 and $M^2 - 1$ (w' can be produced just as w, by a series of predicates similar to the bit_i predicates; hence our requirement that M is a power of 2). If, and only if, $w' \leq f(\chi)f(y)$, the corresponding grounding of a predicate $\mathsf{link}(\chi, y)$ must be set to true. Again, all of those operations (binary multiplication and comparison) can be done with digital circuits and thus encoded with logical constructs parameterized by χ and y.

The graph presented in Fig. 6 was produced by selecting $\gamma = 1.6$, $M = 64$, and $B = 13$, and by taking 750 sites. These parameters were selected so as to produce a reasonably large graph that could still be reasonably printed and viewed.

[5] We thank a reviewer for bringing this possibility to our attention.

References

[1] Martín Abadi, Joseph Y. Halpern, Decidability and expressiveness for first-order logics of probability, Inf. Comput. 112 (1) (1994) 1–36.
[2] Miklos Ajtai, Ronald Fagin, Reachability is harder for directed than for undirected finite graphs, J. Symb. Log. 55 (1) (1990) 113–150.
[3] Alessandro Artale, Diego Calvanese, Roman Kontchakov, Michael Zakharyashev, The DL-Lite family and relations, J. Artif. Intell. Res. 36 (2009) 1–69.
[4] Franz Baader, Werner Nutt, Basic description logics, in: Description Logic Handbook, Cambridge University Press, 2002, pp. 47–100.
[5] Fahiem Bacchus, Representing and Reasoning With Probabilistic Knowledge: A Logical Approach, MIT Press, Cambridge, 1990.
[6] Fahiem Bacchus, Using first-order probability logic for the construction of Bayesian networks, in: Conference on Uncertainty in Artificial Intelligence, 1993, pp. 219–226.
[7] Paul Beame, Guy Van den Broeck, Eric Gribkoff, Dan Suciu, Symmetric weighted first-order model counting, in: ACM Symposium on Principles of Database Systems (PODS), 2015.
[8] David M. Blei, Andrew Y. Ng, Michael I. Jordan, Latent Dirichlet allocation, J. Mach. Learn. Res. 3 (2003) 993–1022.
[9] David Buchman, David Poole, Negative probabilities in probabilistic logic programs, Int. J. Approx. Reason. 83 (2017) 43–59.
[10] David Buchman, David Poole, Why rules are complex: real-valued probabilistic logic programs are not fully expressive, in: Conference on Uncertainty in Artificial Intelligence, 2017.
[11] Guido Caldarelli, Andrea Capocci, Paolo De Los Rios, Miguel A. Muñoz, Scale-free networks from varying vertex intrinsic fitness, Phys. Rev. Lett. 89 (25) (2002) 258702.
[12] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, Riccardo Rosati, DL-Lite: tractable description logics for ontologies, in: AAAI, 2005, pp. 602–607.
[13] Bob Carpenter, Andrew Gelman, Matthew D. Hoffman, Daniel Lee, Ben Goodrich, Michael Betancourt, Marcus Brubaker, Jiqiang Guo, Peter Li, Allen Riddell Stan, A probabilistic programming language, J. Stat. Softw. 76 (1) (2017).
[14] Rommel N. Carvalho, Kathryn B. Laskey, Paulo C. Costa, PR-OWL 2.0 — bridging the gap to OWL semantics, in: URSW 2008-2010/UniDL 2010, in: LNAI, vol. 7123, 2013, pp. 1–18.
[15] Paulo C.G. Costa, Kathryn B. Laskey, PR-OWL: a framework for probabilistic ontologies, in: Conference on Formal Ontology in Information Systems, 2006.
[16] Fabio G. Cozman, Denis D. Mauá, The complexity of Bayesian networks specified by propositional and relational languages, Artif. Intell. J. 262 (2018) 96–141.
[17] Fabio G. Cozman, Denis D. Mauá, The descriptive complexity of Bayesian network specifications, in: Symbolic and Quantitative Approaches to Reasoning With Uncertainty, Springer, 2017, pp. 93–103.
[18] Fabio G. Cozman, Denis D. Mauá, On the semantics and complexity of probabilistic logic programs, J. Artif. Intell. Res. 60 (2017) 221–262.
[19] Fabio G. Cozman, Denis D. Mauá, The complexity of plate probabilistic models, in: Scalable Uncertainty Management, in: LNCS, vol. 9310, Springer, 2015, pp. 36–49.
[20] Adnan Darwiche, Modeling and Reasoning With Bayesian Networks, Cambridge University Press, 2009.
[21] Glauber de Bona, Fabio G. Cozman, Encoding the consistency of relational Bayesian networks, in: Encontro Nacional de Inteligência Artificial e Computacional, Uberlândia, Brasil, 2017.
[22] Luc de Raedt, Logical and Relational Learning, Springer, 2008.
[23] Luc de Raedt, Paolo Frasconi, Kristian Kersting, Stephen Muggleton, Probabilistic Inductive Logic Programming, Springer, 2010.
[24] Peter R. de Waal, Marginals of DAG-isomorphic independence models, in: Symbolic and Quantitative Approaches to Reasoning With Uncertainty, in: Lecture Notes in Computer Science, vol. 5590, 2009, pp. 192–203.
[25] Heinz-Dieter Ebbinghaus, J. Flum, Finite Model Theory, Springer-Verlag, 1995.
[26] Zhongli Ding, Yun Peng, Rong Pan, BayesOWL: uncertainty modeling in semantic web ontologies, in: Soft Computing in Ontologies and Semantic Web, in: Studies in Fuzziness and Soft Computing, vol. 204, Springer, Berlin/Heidelberg, 2006, pp. 3–29.
[27] Herbert B. Enderton, A Mathematical Introduction to Logic, Academic Press, 1972.
[28] Paul Erdös, Alfréd Rényi, On the evolution of random graphs, Publ. Math. Inst. Hung. Acad. Sci. 5 (1) (1960) 17–60.
[29] Ronald Fagin, Probabilities on finite models, J. Symb. Log. 41 (1) (1976) 50–58.
[30] Daan Fierens, Guy Van den Broeck, Joris Renkens, Dimitar Shrerionov, Bernd Gutmann, Gerda Janssens, Luc De Raedt, Inference and learning in probabilistic logic programs using weighted Boolean formulas, Theory Pract. Log. Program. 15 (3) (2014) 358–401.
[31] Haim Gaifman, Concerning measures on first-order calculi, Isr. J. Math. 2 (1964) 1–18.
[32] Haim Gaifman, Marc Snir, Probabilities over rich languages, testing and randomness, J. Symb. Log. 47 (3) (1982) 495–548.
[33] Lise Getoor, Nir Friedman, Daphne Koller, Avi Pfeffer, Ben Taskar, Probabilistic relational models, in: Introduction to Statistical Relational Learning, 2007.
[34] Lise Getoor, Ben Taskar, Introduction to Statistical Relational Learning, MIT Press, 2007.
[35] Edgar Nelson Gilbert, Random graphs, Ann. Math. Stat. 30 (4) (1959) 1141–1144.
[36] Walter R. Gilks, Andrew Thomas, David Spiegelhalter, A language and program for complex Bayesian modelling, Statistician 43 (1993) 169–178.
[37] Yu V. Glebskii, D.I. Kogan, M.I. Liogon'kii, V.A. Talanov, Range and degree of realizability of formulas in the restricted predicate calculus, Kibernetika 2 (5) (1969) 17–27.
[38] Robert P. Goldman, Eugene Charniak, Dynamic construction of belief networks, in: Conference of Uncertainty in Artificial Intelligence, 1990, pp. 90–97.
[39] Noah Goodman, Andreas Stuhlmuller, The design and implementation of probabilistic programming languages, http://dippl.org, 2014. (Accessed 23 November 2017).
[40] Noah D. Goodman, Vikash K. Mansinghka, Daniel Roy, Keith Bonawitz, Joshua B. Tenenbaum, Church: a language for generative models, in: Conference on Uncertainty in Artificial Intelligence, 2008, pp. 220–229.
[41] Erich Grädel, Finite model theory and descriptive complexity, in: Finite Model Theory and Its Applications, Springer, 2007, pp. 125–229.
[42] Erich E. Grädel, Phokion G. Kolaitis, Leonid Libkin, Maarten Marx, Joel Spencer, Moshe Y. Vardi, Yde Venema, Scott Weinstein, Finite Model Theory and Its Applications, Springer, 2007.
[43] Etienne Grandjean, Complexity of the first-order theory of almost all finite structures, Inf. Control 57 (1983) 180–204.
[44] Peter Haddawy, Generating Bayesian networks from probability logic knowledge, in: Conference on Uncertainty in Artificial Intelligence, 1994, pp. 262–269.
[45] Joseph Y. Halpern, Reasoning About Uncertainty, MIT Press, Cambridge, Massachusetts, 2003.
[46] David Heckerman, Chris Meek, Daphne Koller, Probabilistic entity-relationship models, PRMs, and plate models, in: Lise Getoor, Ben Taskar (Eds.), Introduction to Statistical Relational Learning, MIT Press, 2007, pp. 201–238.
[47] Michael C. Horsch, David Poole, A dynamic approach to probabilistic inference using Bayesian networks, in: Conference of Uncertainty in Artificial Intelligence, 1990, pp. 155–161.
[48] Manfred Jaeger, Relational Bayesian networks, in: Dan Geiger, Prakash Pundalik Shenoy (Eds.), Conference on Uncertainty in Artificial Intelligence, Morgan Kaufmann, San Francisco, California, 1997, pp. 266–273.
[49] Manfred Jaeger, Convergence results for relational Bayesian networks, in: Symposium on Logic in Computer Science (LICS), 1998.

[50] Manfred Jaeger, Complex probabilistic modeling with recursive relational Bayesian networks, Ann. Math. Artif. Intell. 32 (2001) 179–220.
[51] Manfred Jaeger, Relational Bayesian Networks: A Survey, Linkoping Electronic Articles in Computer and Information Science, vol. 6, 2002.
[52] Manfred Jaeger, Model-theoretic expressivity analysis, in: Luc De Raedt, Paolo Frasconi, Kristian Kersting, Stephen Muggleton (Eds.), Probabilistic Inductive Logic Programming, Springer, 2008.
[53] Manfred Jaeger, Lower complexity bounds for lifted inference, Theory Pract. Log. Program. 15 (2) (2014) 246–264.
[54] Manfred Jaeger, Guy Van Den Broeck, Liftability of probabilistic inference: upper and lower bounds, in: 2nd Statistical Relational AI (StaRAI-12) Workshop, 2012.
[55] Daphne Koller, Nir Friedman, Probabilistic Graphical Models: Principles and Techniques, MIT Press, 2009.
[56] Daphne Koller, Alon Y. Levy, Avi Pfeffer, P-CLASSIC: a tractable probabilistic description logic, in: AAAI, 1997, pp. 390–397.
[57] Daphne Koller, Avi Pfeffer, Object-oriented Bayesian networks, in: Conference on Uncertainty in Artificial Intelligence, 1997, pp. 302–313.
[58] Daphne Koller, Avi Pfeffer, Probabilistic frame-based systems, in: National Conference on Artificial Intelligence (AAAI), 1998, pp. 580–587.
[59] Leonid Libkin, Elements of Finite Model Theory, Springer, 2004.
[60] David Lunn, David Spiegelhalter, Andrew Thomas, Nicky Best, The BUGS project: evolution, critique and future directions, Stat. Med. 28 (2009) 3049–3067.
[61] Suzanne Mahoney, Kathryn B. Laskey, Network engineering for complex belief networks, in: Conference on Uncertainty in Artificial Intelligence, 1996.
[62] Vikash K. Mansinghka, Alexey Radul, CoreVenture: a high level, reflective machine language for probabilistic programming, in: NIPS Workshop on Probabilistic Programming, 2014.
[63] Wannes Meert, Nima Taghipour, Hendrik Blockeel, First-order Bayes-ball, in: European Conference on Machine Learning and Knowledge Discovery in Databases, 2010, pp. 369–384.
[64] Brian Milch, Bhaskara Marthi, Stuart Russell, David Sontag, Daniel L. Ong, Andrey Kolobov, BLOG: probabilistic models with unknown objects, in: IJCAI, 2005.
[65] Brian Milch, Bhaskara Marthi, David Sontag, Stuart Russell, Daniel L. Ong, Andrey Kolobov, Approximate inference for infinite contingent Bayesian networks, in: Artificial Intelligence and Statistics, 2005.
[66] Mark E.J. Newman, Networks: An Introduction, Oxford University Press, 2010.
[67] Christos H. Papadimitriou, Computational Complexity, Addison-Wesley Publishing, 1994.
[68] Judea Pearl, Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference, Morgan Kaufmann, San Mateo, California, 1988.
[69] Avi Pfeffer, IBAL: a probabilistic rational programming language, in: International Joint Conference on Artificial Intelligence, 2001, pp. 733–740.
[70] Avi Pfeffer, Practical Probabilistic Programming, Manning Publications, 2016.
[71] David Poole, Probabilistic Horn abduction and Bayesian networks, Artif. Intell. 64 (1993) 81–129.
[72] David Poole, First-order probabilistic inference, in: International Joint Conference on Artificial Intelligence (IJCAI), 2003, pp. 985–991.
[73] David Poole, Probabilistic programming languages: independent choices and deterministic systems, in: Rina Dechter, Hector Geffner, Joseph Y. Halpern (Eds.), Heuristics, Probability and Causality — A Tribute to Judea Pearl, College Publications, 2010, pp. 253–269.
[74] Fabrizio Riguzzi, The distribution semantics is well-defined for all normal programs, in: Fabrizio Riguzzi, Joost Vennekens (Eds.), International Workshop on Probabilistic Logic Programming, in: CEUR Workshop Proceedings, vol. 1413, 2015, pp. 69–84.
[75] Kayvan Sadegui, Steffen Lauritzen, Markov properties for mixed graphs, Bernoulli 20 (2) (2014) 676–696.
[76] Sanjeev Saluja, K.V. Subrahmanyam, Madhukar N. Thakur, Descriptive complexity of #P functions, J. Comput. Syst. Sci. 50 (1995) 493–505.
[77] Taisuke Sato, A statistical learning method for logic programs with distribution semantics, in: Int. Conference on Logic Programming, 1995, pp. 715–729.
[78] Milan Studeny, Probabilistic Conditional Independence Structures, Springer-Verlag, London, 2005.
[79] Dan Suciu, Dan Oiteanu, Christopher Ré, Christoph Koch, Probabilistic Databases, Morgan & Claypool Publishers, 2011.
[80] Larry J. Stockmeyer, The polynomial-time hierarchy, Theor. Comput. Sci. 3 (1977) 1–22.
[81] Moritz Tenorth, Michael Beetz, KnowRob: a knowledge processing infrastructure for cognition-enabled robots, Int. J. Robot. Res. 32 (5) (2013) 566–590.
[82] Jacobo Tóran, Complexity classes defined by counting quantifiers, J. ACM 38 (3) (1991) 753–774.
[83] Guy Van den Broeck, On the completeness of first-order knowledge compilation for lifted probabilistic inference, in: Neural Processing Information Systems, 2011, pp. 1386–1394.
[84] Klaus W. Wagner, The complexity of combinatorial problems with succinct input representation, Acta Inform. 23 (1986) 325–356.
[85] Michael P. Wellman, John S. Breese, Robert P. Goldman, From knowledge bases to decision models, Knowl. Eng. Rev. 7 (1) (1992) 35–53.
[86] Frank Wood, Jan Willem van de Meent, Vikash K. Mansinghka, A new approach to probabilistic programming inference, in: Conference on Artificial Intelligence and Statistics, 2014, pp. 1024–1032.