The Complexity of Bayesian Networks Specified by Propositional and Relational Languages

Fabio G. Cozman[a], Denis D. Mauá[b]

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

Abstract

We examine the complexity of inference in Bayesian networks specified by logical languages. We consider representations that range from fragments of propositional logic to function-free first-order logic with equality; in doing so we cover a variety of plate models and of probabilistic relational models. We study the complexity of inferences when network, query and domain are the input (the *inferential* and the *combined* complexity), when the network is fixed and query and domain are the input (the *query/data* complexity), and when the network and query are fixed and the domain is the input (the *domain* complexity). We draw connections with probabilistic databases and liftability results, and obtain complexity classes that range from polynomial to exponential levels; we identify new languages with tractable inference, and we relate our results to languages based on plates and probabilistic relational models.

Keywords: Bayesian networks, Complexity theory, Relational logic, Plate models, Probabilistic relational models

1. Introduction

A Bayesian network can represent any distribution over a given set of random variables [33, 69], and this flexibility has been used to great effect in a variety of applications [107]. Many of these applications contain repetitive patterns of entities and relationships. Thus it is not surprising that practical concerns have led to modeling languages where Bayesian networks are specified using relations, logical variables, and quantifiers [46, 109]. Some of these languages enlarge Bayesian networks with plates [47, 83], while others resort to elements of database schema [44, 58]; some others mix probabilities with logic programming [104, 116] and even with functional programming [85, 89, 100]. The spectrum of tools that specify Bayesian networks by moving beyond propositional sentences is vast, and their applications are remarkable.

Yet most of the existing analysis on the complexity of inference with Bayesian networks focuses on a simplified setting where nodes of a network are associated with categorial variables and distributions are specified by flat tables containing

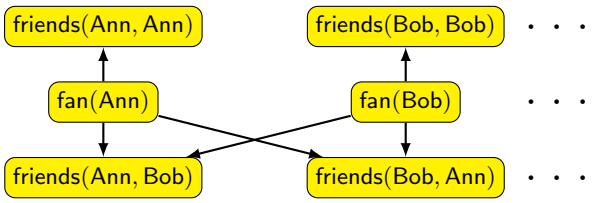


Figure 1: A Bayesian network with a repetitive pattern concerning friendship (only two students are shown; a larger network is obtained for a larger number of students).

probability values [73, 113]. This is certainly unsatisfying: as a point of comparison, consider the topic of *logical* inference, where much is known about the impact of specific constructs on computational complexity — suffice to mention the beautiful and detailed study of inference complexity in description logics [3].

In this paper we examine the complexity of inferences in Bayesian networks as dependent on the *language* that is used to specify the networks. We adopt a simple specification strategy inspired by probabilistic programming [105] and by structural equation models [99], where a Bayesian network over binary variables is specified by a set of logical formulas and a set of independent random variables. As we show in the paper, this abstract specification strategy captures a vast range of modeling languages in the literature. To illustrate the sort of specification we contemplate, consider a short example that will be elaborated later. Suppose we have a population of students, and denote by $\mathsf{fan}(\mathcal{X})$ the fact that student $\mathcal{X}$ is a fan of say a particular band. And write $\mathsf{friends}(\mathcal{X}, y)$ to indicate that $\mathcal{X}$ is a friend of y. Now consider a Bayesian network with a node $\mathsf{fan}(\mathcal{X})$ per student, and a node $\mathsf{friends}(\mathcal{X}, y)$ per pair of students (see Figure 1). Suppose each node $\mathsf{fan}(\mathcal{X})$ is associated with the assessment $\mathbb{P}(\mathsf{fan}(\mathcal{X}) = \mathsf{true}) = 0.2$. And finally suppose that a person is always a friend of herself, and two people are friends if they are fans of the band; that is, for each pair of students, $\mathsf{friends}(\mathcal{X}, y)$ is associated with the formula

$$\mathsf{friends}(\mathcal{X}, y) \leftrightarrow (\mathcal{X} = y) \vee (\mathsf{fan}(\mathcal{X}) \wedge \mathsf{fan}(y)). \tag{1}$$

Now if we have data on some students, we may ask for the probability that some two students are friends, or the probability that a student is a fan. We may wish to consider more sophisticated formulas specifying friendship: how would the complexity of our inferences change, say, if we allowed quantifiers in our formula? Or if we allowed relations of arity higher than two? Such questions are the object of our discussion.

We can thus parameterize computational complexity by the formal language that is allowed in the logical formulas; we can move from sub-Boolean languages to relational ones, in the way producing languages that are similar in power to plate models [47] and to probabilistic relational models [72]. Overall we follow a proven strategy adopted in logical formalisms: we focus on minimal sets of constructs (Boolean operators, quantifiers) that capture the essential connections

between expressivity and complexity, and that can shed light on more sophisticated languages. Our broader goal is to help with the design of knowledge representation formalisms, and in that setting it is important to understand the complexity introduced by language features, however costly those may be.

In this study, we distinguish a few concepts. *Inferential complexity* is the complexity when the network, the query and the domain are given as input. When the specification vocabulary is fixed, inference complexity is akin to *combined complexity* as employed in database theory. *Query complexity* is the complexity when the network is fixed and the input consists of query and domain. The same concept has been often referred to as *data complexity* in the context of probabilistic databases [121], as "query complexity" usually leads to a different direction in databases; we use "query" here as typically done in the Bayesian network literature, and later comment on the connections between these various concepts. Finally, *domain complexity* is the complexity when network and query are fixed, and only the domain is given as input. Query and domain complexity are directly related respectively to *dqe-liftability* and *domain liftability*, concepts that have been used in lifted inference [8, 64]. We make connections with lifted inference and probabilistic databases whenever possible, and benefit from deep results from those fields. One of the contributions of this paper is a framework that can unify these varied research efforts with respect to the analysis of Bayesian networks. We show that many non-trivial complexity classes characterize the cost of inference as induced by various languages, and we make an effort to relate our investigation to various knowledge representation formalisms, from probabilistic description logics to plates to probabilistic relational models.

The paper is organized as follows. Section 2 reviews a few concepts concerning Bayesian networks and computational complexity. Our contributions start in Section 3, where we focus on propositional languages. In Section 4 we extend our framework to relational languages, and review relevant literature on probabilistic databases and lifted inference. In Sections 5 and 6 we study a variety of relational Bayesian network specifications. In Section 7 we connect these specifications to other schemes proposed in the literature. And in Section 8 we relate our results, mostly presented within Wagner's counting hierarchy and its extensions, to Valiant's counting hierarchy and its extensions. Section 9 summarizes our findings and proposes future work.

All proofs are collected in Appendix A.

2. A bit of notation and terminology

We denote by $\mathbb{P}(A)$ the probability of event A. In this paper, every random variable X is a function from a finite sample space (usually a space with finitely many truth assignments or interpretations) to real numbers (usually to $\{0, 1\}$). We refer to an event $\{X = x\}$ as an *assignment*. Say that $\{X = 1\}$ is a *positive* assignment, and $\{X = 0\}$ is a *negative* assignment.

A *graph* consists of a set of *nodes* and a set of *edges* (an edge is specified as a pair of nodes), and we focus on graphs that are directed and acyclic [69]. The parents of a node X, for a given graph, are denoted $\mathrm{pa}(X)$. Suppose we have a

directed acyclic graph $\mathbb{G}$ such that each node is a random variable, and we also have a joint probability distribution $\mathbb{P}$ over these random variables. Say that $\mathbb{G}$ and $\mathbb{P}$ satisfy the Markov condition iff each random variable X is independent with respect to $\mathbb{P}$ of its nondescendants in $\mathbb{G}$ given its parents in $\mathbb{G}$.

A *Bayesian network* is a pair consisting of a directed acyclic graph $\mathbb{G}$ whose nodes are random variables and a joint probability distribution $\mathbb{P}$ over all variables in the graph, such that $\mathbb{G}$ and $\mathbb{P}$ satisfy the Markov condition [92]. For a collection of measurable sets $A_1, \ldots, A_n$, we then have

$$\mathbb{P}(X_1 \in A_1, \ldots, X_n \in A_n) = \prod_{i=1}^{n} \mathbb{P}\Big(X_i \in A_i | \mathrm{pa}(X_i) \in \textstyle\bigcap_{j:X_j \in \mathrm{pa}(X_i)} A_j\Big)$$

whenever the conditional probabilities exist. If all random variables are discrete, then one can specify "local" conditional probabilities $\mathbb{P}(X_i = x_i | \mathrm{pa}(X_i) = \pi_i)$, and the joint probability mass function is the product of these local probabilities:

$$\mathbb{P}(X_1 = x_1, \ldots, X_n = x_n) = \prod_{i=1}^{n} \mathbb{P}(X_i = x_i | \mathrm{pa}(X_i) = \pi_i)\,, \tag{2}$$

where π_i is the projection of $\{x_1, \ldots, x_n\}$ on $\mathrm{pa}(X_i)$, with the understanding that $\mathbb{P}(X_i = x_i | \mathrm{pa}(X_i) = \pi_i)$ stands for $\mathbb{P}(X_i = x_i)$ whenever X_i has no parents.

In this paper we only deal with finite objects, so we can assume that a Bayesian network is fully specified by a finite graph and a local conditional probability distribution per random variable: the local distribution associated with random variable X specifies the probability of X given the parents of X. Often probability values are given in tables (referred to as *conditional probability tables*). Depending on how these tables are encoded, the directed acyclic graph may be redundant; that is, all the information to reconstruct the graph and the joint distribution is already in the tables. Even though we rarely mention the graph $\mathbb{G}$ in our results, graphs are visually useful and we often resort to drawing them in our examples.

A basic computational problem for Bayesian networks is: Given a Bayesian network $\mathbb{B}$, a set of assignments $\mathbf{Q}$ and a set of assignments $\mathbf{E}$, determine whether $\mathbb{P}(\mathbf{Q}|\mathbf{E}) > \gamma$ for some rational number γ. We assume that every probability value is specified as a rational number. Hence $\mathbb{P}(\mathbf{Q}|\mathbf{E}) = \mathbb{P}(\mathbf{Q}, \mathbf{E}) / \mathbb{P}(\mathbf{E})$ is a rational number, as $\mathbb{P}(\mathbf{Q}, \mathbf{E})$ and $\mathbb{P}(\mathbf{E})$ are computed by summing through products given by Expression (2).

We adopt basic terminology and notation from computational complexity [97]. A *language* is a set of strings. A language defines a *decision problem*; that is, the problem of deciding whether an input string is in the language. A *complexity class* is a set of languages; we use well-known complexity classes P, NP, PSPACE, and EXP. We also use ETIME, the class of languages that can be decided by a deterministic Turing machine in time $\mathcal{O}(2^{cN})$ when the input is of size N, for some constant c; similarly, NETIME is the class of languages that can be decided by a nondeterministic Turing machine in time $\mathcal{O}(2^{cN})$. The complexity class PP consists of those languages $\mathcal{L}$ that satisfy the following

property: there is a polynomial-time nondeterministic Turing machine M such that $\ell \in \mathcal{L}$ iff more than half of the computations of M on input ℓ end up accepting. Analogously, we have PEXP, consisting of those languages $\mathcal{L}$ with the following property: there is an exponential-time nondeterministic Turing machine M such that $\ell \in \mathcal{L}$ iff more than half of the computations of M on input ℓ end up accepting [14].

To proceed, we need to define oracles and related complexity classes. 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 class of languages/functions A is defined by a set of Turing machines $\mathcal{M}$ (that is, the languages/functions are decided/computed by these machines), then define $\mathsf{A}^{\mathcal{L}}$ to be the set of languages/functions that are decided/computed by $\{M^{\mathcal{L}} : M \in \mathcal{M}\}$. For a function f, an oracle Turing machine M^f can be similarly defined, and for any class A we have A^f. If A and B are classes of languages/functions, $\mathsf{A}^{\mathsf{B}} = \cup_{x \in \mathsf{B}} \mathsf{A}^x$. For instance, the *polynomial hierarchy* consists of classes $\Sigma_i^{\mathsf{P}} = \mathsf{NP}^{\Sigma_{i-1}^{\mathsf{P}}}$ and $\Pi_i^{\mathsf{P}} = \mathsf{co}\Sigma_i^{\mathsf{P}}$, with $\Sigma_0^{\mathsf{P}} = \mathsf{P}$ (and PH is the union $\cup_i \Pi_i^{\mathsf{P}} = \cup_i \Sigma_i^{\mathsf{P}}$).

Wagner's *polynomial counting hierarchy* is the smallest set of classes containing P and, recursively, for any class C in the polynomial counting hierarchy, the classes PP^{C}, NP^{C}, and $\mathsf{coNP}^{\mathsf{C}}$ (this characterization based on oracles comes from results by Wagner [136, Theorem 4] and Toran [127, Theorem 4.1]). The polynomial hierarchy is included in Wagner's counting polynomial hierarchy.

A distinct counting hierarchy is Valiant's [129]. We examine this hierarchy further in Section 8; for now suffice to say that $\#\mathsf{P}$ is the class of functions such that $f \in \#\mathsf{P}$ iff $f(\ell)$ is the number of computation paths that accept ℓ for some polynomial-time nondeterministic Turing machine. It is as if we had a special machine, called by Valiant a *counting* Turing machine, that on input ℓ prints on a special tape the number of computations that accept ℓ.

We will also use the class PP_1, defined as the set of languages in PP that have a single symbol as input vocabulary. We can take this symbol to be 1, so the input is just a sequence of 1s. One can interpret this input as a non-negative integer written in unary notation. This is the counterpart of Valiant's class $\#\mathsf{P}_1$ that consists of the functions in $\#\mathsf{P}$ that have a single symbol as input vocabulary [130].

We focus on many-one reductions: such a reduction from $\mathcal{L}$ to $\mathcal{L}'$ is a polynomial-time algorithm that takes the input to decision problem $\mathcal{L}$ and transforms it into the input to decision problem $\mathcal{L}'$ such that $\mathcal{L}'$ has the same output as $\mathcal{L}$. A *Turing reduction* from $\mathcal{L}$ to $\mathcal{L}'$ is a polynomial-time algorithm that decides $\mathcal{L}$ using $\mathcal{L}'$ as an oracle. For a complexity class C, a decision problem $\mathcal{L}$ is C-hard with respect to many-one reductions if each decision problem in C can be reduced to $\mathcal{L}$ with many-one reductions. A decision problem is then C-complete with respect to many-one reductions if it is in C and it is C-hard with respect to many-one reductions. Similar definitions of hardness and completeness are obtained when "many-one reductions" are replaced by "Turing reductions".

An important PP-complete (with respect to many-one reductions) decision

problem is MAJSAT: the input is a propositional sentence ϕ and the decision is whether or not the majority of assignments to the propositions in ϕ make ϕ true [48]. Another PP-complete problem (with respect to many-one reductions) is deciding whether $\#\phi > k$ [117]; we use $\#\phi$ to denote the number of satisfying assignments for a formula ϕ. In fact this problem is still PP-complete with respect to many-one reductions even if ϕ is monotone [51]. Recall that a sentence is *monotone* if it contains no negation.

A formula is in kCNF iff it is in Conjunctive Normal Form with k literals per clause (if there is no restriction on k, we just write CNF). MAJSAT is PP-complete with respect to many-one reductions even if the input is restricted to be in CNF; however, it is not known whether MAJSAT is still PP-complete with respect to many-one reductions if the sentence ϕ is in 3CNF. Hence we will resort in proofs to a slightly different decision problem, following results by Bailey et al. [6]. The problem #3SAT($>$) gets as input a propositional sentence ϕ in 3CNF and an integer k, and the decision is whether $\#\phi > k$. We will also use, in the proof of Theorem 2, the following decision problem. Say that a truth assignment to the propositions in a sentence in 3CNF *respects* the 1-in-3 rule if at most one literal per clause is assigned true. Denote by $\#(\text{1-in-3})\phi$ the number of satisfying assignments for ϕ that also respects the 1-in-3 rule. The decision problem #(1-in-3)SAT($>$) gets as input a propositional sentence ϕ in 3CNF and an integer k, and decides whether $\#(\text{1-in-3})\phi > k$. We have:

Proposition 1. *Both* #3SAT($>$) *and #(1-in-3)*SAT($>$) *are* PP*-complete with respect to many-one reductions.*

3. Propositional languages: Inferential and query complexity

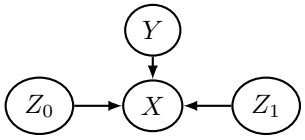
In this section we focus on propositional languages, so as to present our proposed framework in the most accessible manner. Recall that we wish to parameterize the complexity of inferences by the language used in specifying local distributions.

3.1. A specification framework

We are interested in specifying Bayesian networks over binary variables $X_1, \ldots, X_n$, where each random variable X_i is the indicator function of a proposition A_i. That is, consider the space Ω consisting of all truth assignments for these variables (there are 2^n such truth assignments); then X_i yields 1 for a truth assignment that satisfies A_i, and X_i yields 0 for a truth assignment that does not satisfy A_i. *We will often use the same letter to refer to a proposition and the random variable that is the indicator function of the proposition.*

We adopt a specification strategy that moves away from tables of probability values, a specification strategy that we borrow from probabilistic programming [101, 116] and structural models [99]. A *Bayesian network specification* associates with each proposition X_i either

- a logical equivalence $X_i \leftrightarrow \ell_i$, or



$$\mathbb{P}(Y = 1) = 1/3,$$
$$\mathbb{P}(Z_0 = 1) = 1/5, \quad \mathbb{P}(Z_1 = 1) = 7/10,$$
$$X \equiv (Y \wedge Z_1) \vee (\neg Y \wedge Z_0).$$

Figure 2: A Bayesian network specified with logical equivalences and unconditional probabilistic assessments.

- a probabilistic assessment $\mathbb{P}(X_i = 1) = \alpha$,

where ℓ_i is a formula in a propositional language $\mathcal{L}$, such that the only extralogical symbols in ℓ_i are propositions in $\{X_1, \ldots, X_n\}$, and α is a rational number in the interval $[0, 1]$.

We refer to each logical equivalence $X_i \leftrightarrow \ell_i$ as a *definition axiom*, borrowing terminology from description logics [3]. We refer to ℓ_i as the *body* of the definition axiom. In order to avoid confusion between the leftmost symbol $\leftrightarrow$ and possible logical equivalences within ℓ_i, we write a definition axiom as in description logics:

$$X_i \equiv \ell_i,$$

and we emphasize that $\equiv$ is just syntactic sugar for logical equivalence $\leftrightarrow$.

A Bayesian network specification induces a directed graph where the nodes are the random variables $X_1, \ldots, X_n$, and X_j is a parent of X_i if and only if the definition axiom for X_i contains X_j. If this graph is acyclic, as we assume in this paper, then the Bayesian network specification does define a Bayesian network.

Figure 2 depicts a Bayesian network specified this way.

Note that we avoid direct assessments of conditional probability, because one can essentially create negation through $\mathbb{P}(X = 1|Y = 1) = \mathbb{P}(X = 0|Y = 0) = 0$. In our framework, the use of negation is a decision about the language. We will see that negation does make a difference when complexity is analyzed.

Any distribution over binary variables given by a Bayesian network can be equivalently specified using definition axioms, as long as definitions are allowed to contain negation and conjunction (and then disjunction is syntactic sugar). To see that, consider a conditional distribution for X given Y_1 and Y_2; we can specify this distribution using the definition axiom

$$\begin{aligned} X \quad \equiv \quad & (\neg Y_1 \wedge \neg Y_2 \wedge Z_{00}) \vee (\neg Y_1 \wedge Y_2 \wedge Z_{01}) \vee \\ & (Y_1 \wedge \neg Y_2 \wedge Z_{10}) \vee (Y_1 \wedge Y_2 \wedge Z_{11}), \end{aligned}$$

where Z_{ab} are fresh binary variables (that do not appear anywhere else), associated with assessments $\mathbb{P}(Z_{ab} = 1) = \mathbb{P}(X = 1|Y_1 = a, Y_2 = b)$. This sort of encoding can be extended to any set $Y_1, \ldots, Y_m$ of parents, demanding the same space as the corresponding conditional probability table.

Example 1. Consider a simple Bayesian network with random variables X and Y, where Y is the sole parent of X, and where:

$$\mathbb{P}(Y = 1) = 1/3, \quad \mathbb{P}(X = 1|Y = 0) = 1/5, \quad \mathbb{P}(X = 1|Y = 1) = 7/10.$$

Then Figure 2 presents an equivalent specification for this network, in the sense that both specifications have the same marginal distribution over (X, Y). □

Note that definition axioms can exploit structures that conditional probability tables cannot; for instance, to create a Noisy-Or gate [98], we simply write $X \equiv (Y_1 \wedge W_1) \vee (Y_2 \wedge W_2)$, where W_1 and W_2 are inhibitor variables.

3.2. The complexity of propositional languages

Now consider a language $\mathsf{INF}[\mathcal{L}]$ that consists of the strings $(\mathbb{B}, \mathbf{Q}, \mathbf{E}, \gamma)$ for which $\mathbb{P}(\mathbf{Q}|\mathbf{E}) > \gamma$, where

- $\mathbb{P}$ is the distribution encoded by a Bayesian network specification $\mathbb{B}$ with definition axioms whose bodies are formulas in $\mathcal{L}$,
- $\mathbf{Q}$ and $\mathbf{E}$ are sets of assignments (the *query*),
- and γ is a rational number in $[0, 1]$.

In *all* definitions and results in this paper it is assumed that if $\mathbb{P}(\mathbf{E}) = 0$ then the answer to the question "Is $\mathbb{P}(\mathbf{Q}|\mathbf{E}) > \gamma$?" is simply "no", meaning "there is no such probability that is larger than γ".[1]

To start, denote by $\mathsf{Prop}(\wedge, \neg)$ the language of propositional formulas containing conjunction and negation. Then $\mathsf{INF}[\mathsf{Prop}(\wedge, \neg)]$ is the language that decides the probability of a query for networks specified with definition axioms containing conjunction and negation. As every Bayesian network over binary variables can be specified with such definition axioms, $\mathsf{INF}[\mathsf{Prop}(\wedge, \neg)]$ is in fact a PP-complete language [33, Theorems 11.3 and 11.5].

There is obvious interest in finding simple languages $\mathcal{L}$ such that deciding $\mathsf{INF}[\mathcal{L}]$ is a tractable problem, so as to facilitate elicitation, decision-making and learning [31, 37, 62, 106, 114]. And there are indeed propositional languages that generate tractable Bayesian networks: for instance, it is well known that *Noisy-Or* networks display polynomial inference when the query consists of negative assignments [57]. Recall that a Noisy-Or network has a *bipartite* graph with edges pointing from nodes in one set to nodes in the other set, and the latter nodes are associated with Noisy-Or gates.

One might think that tractability can only be attained by imposing some structural conditions on graphs, given results that connect complexity and graph properties [74]. However, it is possible to attain tractability without restrictions on graph topology. Consider the following result, where we use $\mathsf{Prop}(\wedge)$ and $\mathsf{Prop}(\vee)$ to indicate propositional languages respectively restricted to conjunction and restricted to disjunction:

[1] That is, the question is always "Is $\mathbb{P}(\mathbf{E}) > 0$ and $\mathbb{P}(\mathbf{Q}|\mathbf{E}) > \gamma$?"; a different possibility would be to ask whether "If $\mathbb{P}(\mathbf{E}) > 0$ then $\mathbb{P}(\mathbf{Q}|\mathbf{E}) > \gamma$".

Theorem 1. $\mathsf{INF}[\mathsf{Prop}(\wedge)]$ *is in* P *when the query* $(\mathbf{Q}, \mathbf{E})$ *contains only positive assignments, and* $\mathsf{INF}[\mathsf{Prop}(\vee)]$ *is in to* P *when the query contains only negative assignments.*

As the proof of this result shows (in Appendix A), only polynomial effort is needed to compute probabilities for positive queries in networks specified with $\mathsf{Prop}(\wedge)$, *even* if one allows root nodes to be negated (that is, the variables that appear in probabilistic assessments can appear negated in the body of definition axioms).

Alas, even small movements away from the conditions in Theorem 1 takes us to PP-completeness:

Theorem 2. $\mathsf{INF}[\mathsf{Prop}(\wedge)]$ *and* $\mathsf{INF}[\mathsf{Prop}(\vee)]$ *are* PP*-complete with respect to many-one reductions.*

One might try to concoct additional propositional languages by using logical forms in the literature [34]. We leave this to future work; instead of pursuing various possible sub-Boolean languages, we wish to examine the *query complexity* of Bayesian networks, and then move to relational languages.

A digression on reductions. The proof of Theorem 2 is somewhat long because it uses many-one reductions. In Appendix A we show that much simpler proofs for PP-completeness of $\mathsf{INF}[\mathsf{Prop}(\wedge)]$ and $\mathsf{INF}[\mathsf{Prop}(\vee)]$ are possible if one uses Turing reductions. A Turing reduction does give some valuable information: if a problem is PP-complete with Turing reductions, then it is unlikely to be polynomial (for if it were polynomial, then P^{PP} would equal P, a highly unlikely result given current assumptions in complexity theory [125]). However, Turing reductions tend to blur some significant distinctions. For instance, for Turing reductions it does not matter whether $\mathbf{Q}$ is a singleton or not: one can ask for $\mathbb{P}(Q_1|\mathbf{E}_1)$, $\mathbb{P}(Q_2|\mathbf{E}_2)$, and so on, and then obtain $\mathbb{P}(Q_1, Q_2, \ldots|\mathbf{E})$ as the product of the intermediate computations. Hence many-one reductions yield stronger results, so we emphasize them throughout this papper.

3.3. Query complexity

We have so far considered that the input is a string encoding a Bayesian network specification $\mathbb{B}$, a query $(\mathbf{Q}, \mathbf{E})$, and a rational number γ. However in practice one may face a situation where the Bayesian network is fixed, and the input is a string consisting of the pair $(\mathbf{Q}, \mathbf{E})$ and a rational number γ; the goal is to determine whether $\mathbb{P}(\mathbf{Q}|\mathbf{E}) > \gamma$ with respect to the fixed Bayesian network.

Denote by $\mathsf{QINF}[\mathbb{B}]$, where $\mathbb{B}$ is a Bayesian network specification, the language consisting of each string $(\mathbf{Q}, \mathbf{E}, \gamma)$ for which $\mathbb{P}(\mathbf{Q}|\mathbf{E}) > \gamma$ with respect to $\mathbb{B}$. And denote by $\mathsf{QINF}[\mathcal{L}]$ the set of languages $\mathsf{QINF}[\mathbb{B}]$ where $\mathbb{B}$ is a Bayesian network specification with definition axioms whose bodies are formulas in $\mathcal{L}$.

Definition 1. *Let* $\mathcal{L}$ *be a propositional language and* C *be a complexity class. The* query complexity *of* $\mathcal{L}$ *is in* C *iff every language in* $\mathsf{QINF}[\mathcal{L}]$ *is in* C.

The fact that query complexity may differ from inferential complexity was initially raised by Darwiche and Provan [31], and has led to a number of techniques emphasizing compilation of a fixed Bayesian network [22, 32]. Indeed the expression "query complexity" seems to have been coined by Darwiche [33, Section 6.9], without the formal definition presented here.

The original work by Darwiche and Provan [31] shows how to transform a fixed Bayesian network into a *Query-DAG* such that $\mathbb{P}(\mathbf{Q}|\mathbf{E}) > \gamma$ can be decided in linear time. That is:

Theorem 3 (Darwiche and Provan [31])**.** $\mathsf{QINF}[\mathsf{Prop}(\wedge, \neg)]$ *is in* P.

Results on query complexity become more interesting when we move to relational languages, as we do at once.

4. Relational Languages: Inferential, query, and domain complexity

In this section we extend our specification framework so as to deal with relational languages. Such languages have been used in a variety of applications with repetitive entities and relationships [46, 109], as we have alluded to in Section 1. They have roots in general probabilistic logics that mix deterministic knowledge and uncertain reasoning in very flexible, but sometimes too complicated, ways [56]. The more focused interest in extending Bayesian networks with relational constructs has produced an array of practical languages, as we summarize later in Section 7: plates, PRMs, probabilistic logic programs. It is not easy to extract a "common denominator" from these languages. However, with some reflection we see that they all parameterize random variables using relations; they all allow for logical definitions to be mixed with probabilistic assessments; they all resort to the semantics of first-order logic to interpret relations using sets (domains). We capture these common features in Section 4.1; to do so, we use "parvariables" and related techniques introduced by Poole [103], using as much syntax and semantics as possible from first-order logic and in particular from lifted inference techniques [90] (and borrowing some terminology from description logics and logic programming as appropriate). Throughout we resort to the same idea advocated in Section 3: that is, that a model can be specified by a set of probability assessments and a set of logical definitions that belong to a selected logical language. This is exactly the situation in acyclic probabilistic logic programs and many other existing languages, as shown later in Section 7.

After introducing our modeling framework in Section 4.1, in Section 4.2 we describe the complexity questions we intend to answer, and we offer a few comments on lifted inference and probabilistic databases.

4.1. Relational Bayesian network specifications

A *parameterized random variable*, abbreviated *parvariable*, is a function that yields, for each combination of its input parameters, a random variable. For instance, parvariable X yields a random variable $X(a)$ for each selected a. In

what follows, parvariables and their parameters will correspond to relations and their logical variables.

We use a *vocabulary* consisting of names of relations. Every relation r is associated with a non-negative integer called its *arity*. A logical variable is referred to as a *logvar*. A vector of logvars $[\mathcal{X}_1, \ldots, \mathcal{X}_k]$ is denoted $\vec{\mathcal{X}}$; then $r(\vec{\mathcal{X}})$ is an *atom*. A *domain* is a set; in this paper every domain is finite. When the logvars in an atom are replaced by elements of the domain, we obtain $r(a_1, \ldots, a_k)$, a *ground atom*, often referred to as a *grounding* of relation X. An *interpretation* $\mathbb{I}$ is a function that assigns to each relation X of arity k a relation on $\mathcal{D}^k$. An interpretation can be viewed as a function that assigns true or false to each grounding $r(\vec{a})$, where $\vec{a}$ is a tuple of elements of the domain. Typically in logical languages there is a distinction between *constants* and elements of a domain, but we avoid constants altogether in our discussion: as argued by Bacchus, if constants are used within a probabilistic logic, some sort of additional non-trivial *rigidity* assumption must be used [4]. It is possible that by adding constants to a language we increase its expressivity in ways that cannot be captured with standard probabilities [12, 13]; we leave an analysis of languages with constants to future work.

Given a domain $\mathcal{D}$, we can associate with each grounding $r(\vec{a})$ a random variable $X(\vec{a})$ over the set of all possible interpretations, such that $X(\vec{a})(\mathbb{I}) = 1$ if interpretation $\mathbb{I}$ assigns true to $r(\vec{a})$, and $X(\vec{a})(\mathbb{I}) = 0$ otherwise. Similarly, we can associate with a relation r a parvariable X that yields, once a domain is given, a random variable $X(\vec{a})$ for each grounding $r(\vec{a})$. To simplify matters, we use the same symbol for a grounding $r(\vec{a})$ and its associated random variable $X(\vec{a})$, much as we did with propositions and their associated random variables. Similarly, we use the same symbol for a relation r and its associated parvariable X. We can then write down logical formulas over relations/parvariables, and we can assess probabilities for relations/parvariables. The next example clarifies the dual use of symbols for relations/parvariables.

Example 2. Consider a model of friendship built on top of the example in Section 1. Two people are friends if they are both fans of the same band, or if they are linked in some other unmodeled way, and a person is always a friend of herself. Take relations $\mathsf{friends}$, fan, and linked. Given a domain, say $\mathcal{D} = \{a, b\}$, we have the grounding $\mathsf{friends}(a, b)$, whose intended interpretation is that a and b are friends; we take friendship to be asymmetric so $\mathsf{friends}(a, b)$ may hold while $\mathsf{friends}(b, a)$ may not hold. We also have groundings $\mathsf{fan}(a)$, $\mathsf{linked}(b, a)$, and so on. Each one of these groundings corresponds to a random variable that yields 1 or 0 when the grounding is respectively true or false is an interpretation.

The stated facts about friendship might be encoded by a variant of Formula (1):

$$\mathsf{friends}(\mathcal{X}, y) \;\equiv\; (\mathcal{X} = y) \vee (\mathsf{fan}(\mathcal{X}) \wedge \mathsf{fan}(y)) \vee \mathsf{linked}(\mathcal{X}, y). \tag{3}$$

We can draw a directed graph indicating the dependence of $\mathsf{friends}$ on the other relations, as in Figure 3. Suppose we believe 0.2 is the probability that an

Figure 3: Representing dependences amongst relations in Example 2.

element of the domain is a fan, and 0.1 is the probability that any two people are linked for some other reason. To express these assesssments we might write

$$\mathbb{P}(\mathsf{fan}(\mathcal{X}) = 1) = 0.2 \qquad \text{and} \qquad \mathbb{P}\big(\mathsf{linked}(\mathcal{X}, \mathcal{Y}) = 1\big) = 0.1, \tag{4}$$

with implicit outer universal quantification. □

Given a formula and a domain, we can produce all groundings of the formula by replacing its logvars by elements of the domain in every possible way (as usual when grounding first-order formulas). We can similarly ground probabilistic assessments by grounding the affected relations.

Example 3. In Example 2, we can produce the following groundings from domain $\mathcal{D} = \{a, b\}$ and Formula (3):

$$\begin{aligned}
\mathsf{friends}(a,a) &\equiv (a = a) \vee (\mathsf{fan}(a) \wedge \mathsf{fan}(a)) \vee \mathsf{linked}(a,a),\\
\mathsf{friends}(a,b) &\equiv (a = b) \vee (\mathsf{fan}(a) \wedge \mathsf{fan}(b)) \vee \mathsf{linked}(a,b),\\
\mathsf{friends}(b,a) &\equiv (b = a) \vee (\mathsf{fan}(b) \wedge \mathsf{fan}(a)) \vee \mathsf{linked}(b,a),\\
\mathsf{friends}(b,b) &\equiv (b = b) \vee (\mathsf{fan}(b) \wedge \mathsf{fan}(b)) \vee \mathsf{linked}(b,b),
\end{aligned}$$

Similarly, we get:

$$\begin{aligned}
&\mathbb{P}(\mathsf{fan}(a) = 1) = 0.2, && \mathbb{P}(\mathsf{fan}(b) = 1) = 0.2,\\
&\mathbb{P}(\mathsf{linked}(a,a) = 1) = 0.1, && \mathbb{P}(\mathsf{linked}(a,b) = 1) = 0.1,\\
&\mathbb{P}(\mathsf{linked}(b,a) = 1) = 0.1, && \mathbb{P}(\mathsf{linked}(b,b) = 1) = 0.1,
\end{aligned}$$

by grounding assessments in Expression (4). □

We wish to extend our propositional framework by specifying Bayesian networks using both parameterized probabilistic assessments and first-order definitions. So, suppose we have a finite set of parvariables, each one of them corresponding to a relation in a vocabulary. A *relational Bayesian network specification* associates, with each parvariable X_i, either

- a *definition axiom* $X_i(\vec{\mathcal{X}}) \equiv \ell_i(\vec{\mathcal{X}}, Y_1, \ldots, Y_m)$, or
- a *probabilistic assessment* $\mathbb{P}(X(\vec{\mathcal{X}}) = 1) = \alpha$,

where

- ℓ_i is a well-formed formula in a language $\mathcal{L}$, containing relations $Y_1, \ldots, Y_m$ and free logvars $\vec{\mathcal{X}}$ (and possibly additional logvars bound to quantifiers),
- and α is a nonnegative rational in the interval $[0, 1]$.

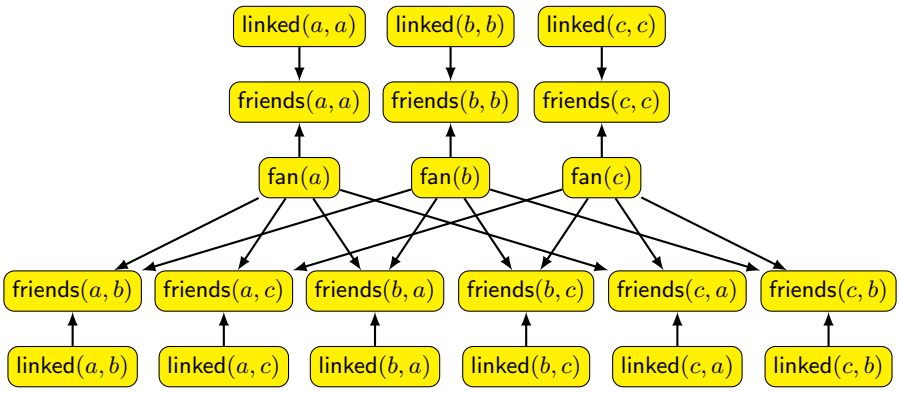


Figure 4: The grounding (on domain $\{a, b, c\}$) of the relational Bayesian network specification in Example 2.

The formula ℓ_i is the *body* of the corresponding definition axiom. The parvariables that appear in ℓ_i are the *parents* of parvariable X_i, and are denoted by $\mathrm{pa}(X_i)$. Clearly the definition axioms induce a directed graph where the nodes are the parvariables and the parents of a parvariable (in the graph) are exactly $\mathrm{pa}(X_i)$. This is the *parvariable graph* of the relational Bayesian network specification (this sort of graph is called a *template dependency graph* by Koller and Friedman [69, Definition 6.13]). For instance, Figure 3 depicts the parvariable graph for Example 2.

When the parvariable graph of a relational Bayesian network specification is acyclic, we say the specification itself is acyclic. *In this paper we assume that relational Bayesian network specifications are acyclic, and we do not even mention this anymore.*

The *grounding* of a relational Bayesian network specification $\mathbb{S}$ on a domain $\mathcal{D}$ is defined as follows. First, produce all groundings of all definition axioms. Then, for each parameterized probabilistic assessment $\mathbb{P}(X(\vec{\chi}) = 1) = \alpha$, produce its ground probabilistic assessments

$$\mathbb{P}(X(\vec{a_1}) = 1) = \alpha, \quad \mathbb{P}(X(\vec{a_2}) = 1) = \alpha, \quad \text{and so on,}$$

for all appropriate tuples $\vec{a_j}$ built from the domain. The grounded relations, definitions and assessments specify a propositional Bayesian network that is then the semantics of $\mathbb{S}$ with respect to domain $\mathcal{D}$.

Example 4. Consider Example 2. For a domain $\{a, b\}$, the relational Bayesian network specification given by Expressions (3) and (4) is grounded into the sentences and assessments in Example 3. By repeating this process for a larger domain $\{a, b, c\}$, we obtain a larger Bayesian network whose graph is depicted in Figure 4. □

Note that logical inference might be used to simplify grounded definitions; for instance, in the previous example, one might note that $\mathsf{friends}(a, a)$ is simply

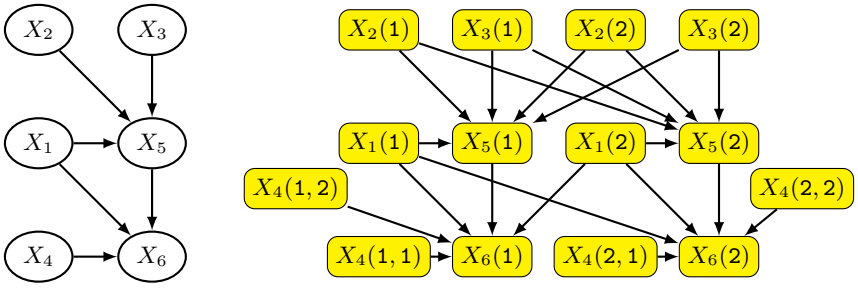


Figure 5: The parvariable graph of the relational Bayesian Network specification in Example 5, and its grounding on domain $\mathcal{D} = \{1, 2\}$.

true. Note also that the grounding of a formula with quantifiers turns, as usual, an existential quantifier into a disjunction, and a universal quantifier into a conjunction.

Example 5. Take the following relational Bayesian network specification (with no particular meaning, just to illustrate a few possibilities):

$$\begin{gathered}\mathbb{P}(X_1(\mathcal{X}) = 1) = 2/3, \qquad \mathbb{P}(X_2(\mathcal{X}) = 1) = 1/10,\\ \mathbb{P}(X_3(\mathcal{X}) = 1) = 4/5, \qquad \mathbb{P}\big(X_4(\mathcal{X}, y) = 1\big) = 1/2,\\ X_5(\mathcal{X}) \;\equiv\!\equiv\; \exists y : \forall z : \neg X_1(\mathcal{X}) \vee X_2(y) \vee X_3(z),\\ X_6(\mathcal{X}) \;\equiv\!\equiv\; X_5(\mathcal{X}) \wedge \exists y : X_4(\mathcal{X}, y) \wedge X_1(y),\end{gathered}$$

Take a domain $\mathcal{D} = \{1, 2\}$; the grounded definition of $X_5(1)$ is

$$\begin{aligned}X_5(1) \;\equiv\!\equiv\; &((\neg X_1(1) \vee X_2(1) \vee X_3(1)) \wedge (\neg X_1(1) \vee X_2(1) \vee X_3(2))) \vee\\ &((\neg X_1(1) \vee X_2(2) \vee X_3(1)) \wedge (\neg X_1(1) \vee X_2(2) \vee X_3(2)))\,.\end{aligned}$$

Figure 5 depicts the parvariable graph and the grounding of this relational Bayesian network specification. □

In order to study complexity questions we must decide how to encode any given domain. Note that there is no need to find special names for the elements of the domain, so we take that the domain is always the set of numbers $\{1, 2, \ldots, N\}$. Now if this list is explicitly given as input, then the size of the input is of order N. However, if only the number N is given as input, then the size of the input is either of order N when N is encoded in unary notation, or of order $\log N$ when N is encoded in binary notation. The distinction between unary and binary notation for input numbers is often used in description logics [3]. We will see later that the encoding of input N has significant impact on complexity.

The conceptual difference between unary and binary encodings of domain size can be captured by the following analogy. Suppose we are interested in the inhabitants of a city: the probabilities that they study, that they marry, that they vote, and so on. Suppose the behavior of these inhabitants is modeled by

a relational Bayesian network specification, and we observe evidence on a few people. If we then take our input N to be in unary notation, we are implicitly assuming that we have a directory, say a mailing list, with the names of all inhabitants; even if we do not care about their specific names, each one of them exists concretely in our modeled reality. But if we take our input N to be in binary notation, we are just focusing on the impact of city size on probabilities, without any regard for the actual inhabitants; we may say that N is a thousand, a million, or maybe even fifty million (and perhaps none of these numbers are close to actual city size).

4.2. Inferential, combined, query and domain complexity

To repeat, we are interested in the relationship between the language $\mathcal{L}$ that is employed in definition axioms and the complexity of inferences. While in the propositional setting we distinguished between inferential and query complexity, here we have an additional distinction to make. Consider the following definitions, where $\mathbb{S}$ is a relational Bayesian network specification, N is the domain size, $\mathbf{Q}$ and $\mathbf{E}$ are sets of assignments for ground atoms, γ is a rational number in $[0, 1]$, and C is a complexity class. Recall that the inequality $\mathbb{P}(\mathbf{Q}|\mathbf{E}) > \gamma$" is to be understood as an abbreviation for "$\mathbb{P}(\mathbf{E}) > 0$ and $\mathbb{P}(\mathbf{Q}|\mathbf{E}) > \gamma$".

Definition 2. *Denote by* $\mathsf{INF}[\mathcal{L}]$ *the language consisting of strings* $(\mathbb{S}, N, \mathbf{Q}, \mathbf{E}, \gamma)$ *for which* $\mathbb{P}(\mathbf{Q}|\mathbf{E}) > \gamma$ *with respect to the grounding of* $\mathbb{S}$ *on domain of size* N*, where* $\mathbb{S}$ *contains definition axioms whose bodies are formulas in* $\mathcal{L}$*. The* inferential complexity *of* $\mathcal{L}$ *is in* C *iff* $\mathsf{INF}[\mathcal{L}]$ *is in* C*; moreover, the inferential complexity is* C*-hard with respect to a reduction iff* $\mathsf{INF}[\mathcal{L}]$ *is* C*-hard with respect to the reduction, and it is* C*-complete with respect to a reduction iff it is in* C *and it is* C*-hard with respect to the reduction.*

Definition 3. *Denote by* $\mathsf{QINF}[\mathbb{S}]$ *the language consisting of strings* $(N, \mathbf{Q}, \mathbf{E}, \gamma)$ *for which* $\mathbb{P}(\mathbf{Q}|\mathbf{E}) > \gamma$ *with respect to the grounding of* $\mathbb{S}$ *on domain of size* N*. Denote by* $\mathsf{QINF}[\mathcal{L}]$ *the set of languages* $\mathsf{QINF}[\mathbb{S}]$ *for* $\mathbb{S}$ *whose bodies of definition axioms are formulas in* $\mathcal{L}$*. The* query complexity *of* $\mathcal{L}$ *is in* C *iff every language in* $\mathsf{QINF}[\mathcal{L}]$ *is in* C*; moreover, the query complexity is* C*-hard with respect to a reduction iff some language in* $\mathsf{QINF}[\mathcal{L}]$ *is* $\mathcal{C}$*-hard with respect to the reduction, and it is* C*-complete with respect to a redution iff it is in* C *and it is* C*-hard with respect to the reduction.*

Definition 4. *Denote by* $\mathsf{DINF}[\mathbb{S}, \mathbf{Q}, \mathbf{E}]$ *the language consisting of strings* (N, γ) *for which* $\mathbb{P}(\mathbf{Q}|\mathbf{E}) > \gamma$ *with respect to the grounding of* $\mathbb{S}$ *on domain of size* N*. Denote by* $\mathsf{DINF}[\mathcal{L}]$ *the set of languages* $\mathsf{DINF}[\mathbb{S}, \mathbf{Q}, \mathbf{E}]$ *for* $\mathbb{S}$ *whose bodies of definition axioms are formulas in* $\mathcal{L}$*, and where* $\mathbf{Q}$ *and* $\mathbf{E}$ *are sets of assignments. The* domain complexity *of* $\mathcal{L}$ *is in* C *iff every language in* $\mathsf{DINF}[\mathcal{L}]$ *is in* C*; moreover, the domain complexity is* C*-hard with respect to a reduction iff some language in* $\mathsf{DINF}[\mathcal{L}]$ *is* $\mathcal{C}$*-hard with respect to the reduction, and it is* C*-complete with respect to a redution iff it is in* C *and it is* C*-hard with respect to the reduction.*

We conclude this section with a number of observations.

Combined complexity. The definition of inferential complexity imposes no restriction on the vocabulary; later we will impose bounds on relation arity. We might instead assume that the vocabulary is fixed; in this case we might use the term *combined complexity*, as this is the term used in finite model theory and database theory to refer to the complexity of model checking when both the formula and the model are given as input, but the vocabulary is fixed [78].

Lifted inference. We note that query and domain complexities are related respectively to *dqe-liftability* and *domain-liftability*, as defined in the study of lifted inference [64, 63].

The term "lifted inference" is usually attached to algorithms that try to compute inferences involving parvariables without actually producing groundings [66, 90, 103]. A formal definition of lifted inference has been proposed by Van den Broeck [132]: an algorithm is *domain lifted* iff inference runs in polynomial-time with respect to N, for fixed model and query. This definition assumes that N is given in unary notation; if N is given in binary notation, the input is of size $\log N$, and a domain lifted algorithm may take exponential time. Domain liftability has been extended to *dqe-liftability*, where the inference must run in polynomial-time with respect to N and the query, for fixed model [64].

In short, dqe-liftability means that query complexity is polynomial, while domain-liftability means that domain complexity is polynomial. Deep results have been obtained both on the limits of liftability [64, 63], and on algorithms that attain liftability [8, 134, 65, 94, 122]. We will use several of these results in our later proofs; in particular, the inferential/query/domain complexity of relational Bayesian network specifications based on function-free first-order logic with equality, and its bounded variable fragments, can be extracted from previous results.

We feel that dqe-liftability and domain-liftability are important concepts but they focus only on a binary choice (polynomial versus non-polynomial); our goal here is to map languages and complexities in more detail. As we have mentioned in Section 1, our main goal is to grasp the complexity, however high, of language features.

Probabilistic databases. Highly relevant material has been produced in the study of probabilistic databases; that is, databases where data may be associated with probabilities [28, 68, 118, 137, 139]. As an example of probabilistic database, the Trio system lets the user indicate that Amy drives an Acura with probability 0.8 [9]. As another example, the NELL system scans text from the web and builds a database of facts, each associated with a number between zero and one [91].

Here we just summarize the framework described by Suciu et al. [121]. Thus consider a set of relations, each implemented as a table. Each tuple in a table may be associated with a probability, and these probabilistic tuples are assumed independent (as dependent tuples can be modeled from independent ones [121, Section 2.7.1]). A probabilistic database management system receives a logical formula $\phi(\vec{\chi})$ and must determine, using data and probabilities in the tables,

the probability $\mathbb{P}(\phi(\vec{a}))$ for tuples $\vec{a}$. The logical formula $\phi(\vec{\chi})$ is referred to as the *query*; for example, ϕ may be a *Union of Conjunctive Queries* (a first-order formula with equality, conjunction, disjunction and existential quantification). Note that the word "query" is not used with the meaning usually adopted in the context of Bayesian networks; in probabilistic databases, a query is a formula whose probability is to be computed.

Suppose that all tuples in the table for relation $X(\vec{\chi})$ receive identical probability value α. This table can be viewed as the grounding of a parvariable $X(\vec{\chi})$ that is associated with a single assessment $\mathbb{P}(X(\vec{\chi}) = 1) = \alpha$. Beame et al. say that the probabilistic database is *symmetric* iff each table can be associated with a parvariable and a single probabilistic assessment [8]. It should be clear that a symmetric probabilistic database and a query expressed as a logical formula ϕ map directly to our relational Bayesian network specification; hence results on both topics can be transferred with little effort. This is plesant because several deep results have been derived on probabilistic databases; as we discuss later, the inferential and query complexity of first-order logic and of some bounded variable fragments have been derived, together with deep results on domain complexity for those logical languages. Research on probabilistic databases has also identified classes of queries (referred to as *safe queries*) that are associated with dichotomy theorems: that is, either a query belongs to such a class and is tractable, or it is not tractable. We return to these results at the end of Section 5.

A distinguishing characteristic of research on probabilistic databases is the intricate search for languages that lead to tractable inferences. As we have already indicated in our previous discussion of lifted inference, our main goal here is to understand the connection between features of a knowledge representation formalism and the complexity of conditional probability calculations. We are not so focused on finding tractable cases that can take upon large volumes of data, even though we are obviously looking for them; indeed we later present tractability results for the $\mathsf{DLLite}^{\mathsf{nf}}$ language, results that we take to be are one of the main contributions of this paper.

Query or data complexity?. The definition of query complexity (Definition 3) reminds one of *data complexity* as adopted in finite model theory and in database theory [78]. It is thus not surprising that research on probabilistic databases has used the term "data complexity" to mean the complexity when the database is the only input, leaving "query" to its meaning in database theory [121].

In the context of Bayesian networks, usually a "query" is a pair $(\mathbf{Q}, \mathbf{E})$ of assignments. If $\mathbf{Q}$ and $\mathbf{E}$ contain all assignments that are present in the input, then there is no real difference between "query" and "data". Indeed, we might have adopted the term "data complexity" throughout this paper as we only discuss queries that contain all available data.[2]

However we feel that there are situations where the "query" is not equal to the

[2]In fact we have used the term *data complexity* in previous work [25].

"data". For instance, in *probabilistic relational models* one often uses auxiliary grounded relations to indicate which groundings are parents of a given grounding (we return to this in Section 7). And in *probabilistic logic programming* one can use *probabilistic facts* to associate probabilities with specific groundings [41, 101, 116]. In these cases one deals with a "query" $(\mathbf{Q}, \mathbf{E})$ and separate "data" that regulate parts of the grounded Bayesian network.

Another possible distinction between "query" and "data" complexities can be found in probabilistic databases. Suppose we have a relational Bayesian network specification, and a formula ϕ whose probability $\mathbb{P}(\phi)$ is to be computed, and data given by tables where each cell is associated with a probability. One might then either fix the specification and vary the formula and the data (we might call this the "query" complexity), or fix the specification and the formula ϕ and vary only the data (this would clearly be the "data" complexity). This sort of distinction actually parallels distinctions found in description logics [19].

It is possible that differences between "query" and "data" are not found to be of practical value in future work. For now we prefer to keep open the possibility of a fine-grained analysis of complexity, so we use the term "query complexity" even though our queries are simply sets of assignments containing all available data.

5. The complexity of relational Bayesian network specifications

We start with *function-free first-order logic with equality*, a language we denote by FFFO. One might guess that such a powerful language leads to exponentially hard inference problems. Indeed:

Theorem 4. INF[FFFO] *is* PEXP*-complete with respect to many-one reductions, regardless of whether the domain is specified in unary or binary notation.*

We note that Grove, Halpern and Koller have already argued that counting the number of suitably defined distinct interpretations of monadic first-order logic is hard for the class of languages decided by exponential-time counting Turing machines [54, Theorem 4.14]. As they do not present a proof of their counting result (and no similar proof seems to be available in the literature), and as we need some of the reasoning to address query complexity later, we present a detailed proof of Theorem 4 in Appendix A.

We emphasize that when the domain is specified in binary notation the proof of Theorem 4 only requires relations of arity one. One might hope to find lower complexity classes for fragments of FFFO that go beyond monadic logic but restrict quantification. For instance, in description logics one often restricts relations to have arity two, and existential quantification to follow the pattern $\exists y : X(x, y) \wedge Y(y)$, in many cases obtaining interesting complexity classes [3]. Such a restriction on quantification will not do in our setting. To understand why, note that if $\mathbb{P}\big(X(x, y) = 1\big) = 1$, then

$$\exists y : X(x, y) \wedge Y(y) \qquad \text{is equivalent to} \qquad \exists y : Y(y) \tag{5}$$

with probability one. Similarly, we can impose $\exists y : X(\chi, y)$ with probability one. Now these quantification patterns, together with negation, are sufficient to build the entire proof of Theorem 4. Thus PEXP-completeness is reached again.

We might try an even more stringent restriction. So, consider the following language, inspired by the popular $\mathcal{ALC}$ description logic (to be discussed in more detail in Section 6). Here we have restricted quantification and added the restriction that only two logvars can be used and reused.

Definition 5. *The language* ALC *consists of all formulas recursively defined so that $X(\chi)$ and $X(y)$ are formulas where X is a unary relation, $\neg\phi$ is a formula when ϕ is a formula, $\phi \wedge \varphi$ is a formula when both ϕ and φ are formulas, and $\exists y : X(\chi, y) \wedge Y(y)$ and $\exists \chi : X(y, \chi) \wedge Y(\chi)$ are formulas when X is a binary relation and Y is a unary relation, with the restriction that only χ and y can appear as logvars in formulas.*

Because logical inference with $\mathcal{ALC}$ is a PSPACE-complete problem [3], one might hope that probabilistic inference might be solved within polynomial space. Alas, ALC does not move us below PEXP when domain size is given in binary notation (later we show that unary notation leads to better behavior):

Theorem 5. INF[ALC] *is* PEXP*-complete with respect to many-one reductions when domain size is given in binary notation.*

Now returning to full FFFO, consider its query complexity. We divide the analysis in two parts, as the related proofs are quite different:[3]

Theorem 6. QINF[FFFO] *is* PEXP*-complete with respect to many-one reductions when domain size is specified in binary notation.*

Theorem 7. QINF[FFFO] *is* PP*-complete with respect to many-one reductions when domain size is specified in unary notation.*

As far as domain complexity is concerned, it seems very hard to establish a completeness result for FFFO when domain size is given in binary notation.[4] We simply rephrase an ingenious argument by Jaeger [63] to establish:

Theorem 8. *Suppose* NETIME $\neq$ ETIME. *Then* DINF[FFFO] *is not solved in deterministic exponential time when domain size is given in binary notation.*

And for domain size in unary notation:

Theorem 9. DINF[FFFO] *is* PP_1*-complete with respect to many-one reductions when domain size is given in unary notation.*

[3] The query complexity of *monadic* FFFO seems to be open, both for domain in binary and in unary notation; proofs of Theorems 6 and 7 need relations of arity two.

[4] One might think that, when domain size is given in binary notation, some small change in the proof of Theorem 9 would show that DINF[FFFO] is complete for a suitable subset of PEXP. Alas, it does not seem easy to define a complexity class that can convey the complexity of DINF[FFFO] when domain size is in binary notation. Finding the precise complexity class of DINF[FFFO] is an open problem.

Theorem 9 is in essence implied by arguments in a major result by Beame et al. [8, Lemma 3.9]: they show that counting the number of interpretations for formulas in the *three-variable* fragment FFFO^3 is $\#\mathsf{P}_1$-complete. The fragment FFFO^k consists of the formulas in FFFO that employ at most k logvars; note that logvar symbols may be reused within a formula, but there is a bounded supply of such symbols [78, Chapter 111]. For instance, ALC belongs to FFFO^2. The proof by Beame et al. is rather involved as they restrict themselves to three logvars; in Appendix A we show that Theorem 9 admits a much simpler proof, a small contribution that may be useful to researchers.

It is apparent from Theorems 4, 5, 6 and 8 that we are bound to obtain exponential complexity when domain size is given in binary notation. Hence, from now on we work with domain sizes in unary notation, unless explicitly indicated.

Of course, a domain size in unary notation cannot by itself avoid exponential behavior (consider for instance Theorem 4). In particular, an exponentially large number of groundings can be generated by increasing arity: even a domain with two individuals leads to 2^k groundings for a relation with arity k. Hence, we often assume that our relations have bounded arity. We might instead assume that the vocabulary is fixed, as done in finite model theory when studying combined complexity. We prefer the more general strategy where we bound arity; clearly a fixed vocabulary implies a fixed maximum arity.

With such additional assumptions, we obtain PSPACE-completeness of inferential complexity:

Theorem 10. $\mathsf{INF}[\mathsf{FFFO}]$ *is* PSPACE*-complete with respect to many-one reductions when relations have bounded arity and domain size is given in unary notation.*

With a few differences, this result is implied by results by Beame et al. in their important paper [8, Theorem 4.1]: they show that counting interpretations with a fixed vocabulary is PSPACE-complete (that is, they focus on combined complexity and avoid conditioning assignments). We present a short proof of Theorem 10 within our framework in Appendix A.

Note that the proof of Theorem 7 is already restricted to arity 2, hence $\mathsf{QINF}[\mathsf{FFFO}]$ is PP-complete with respect to many-one reductions when relations have bounded arity (larger than one) and the domain is given in unary notation.

We now turn to FFFO^k. As we have already noted, this sort of language has been studied already, again by Beame et al., who have derived their domain and combined complexity [8]. In Appendix A we present a short proof of the next result, to emphasize that it follows by a simple adaptation of the proof of Theorem 7:

Theorem 11. $\mathsf{INF}[\mathsf{FFFO}^k]$ *is* PP*-complete with respect to many-one reductions for all* $k \geq 0$ *when domain size is given in unary notation.*

Query complexity also follows directly from arguments in the proofs of pre-

vious results, as is clear from the proof of the next theorem in Appendix A:[5]

Theorem 12. $\mathsf{QINF}[\mathsf{FFFO}^k]$ *is* PP*-complete with respect to many-one reductions for all* $k \geq 2$ *when domain size is given in unary notation.*

Now consider domain complexity for the bounded variable fragment; previous results in the literature establish this complexity [8, 132, 134]. In fact, the case $k > 2$ is based on a result by Beame et al. that we have already alluded to; in Appendix A we present a simplified argument for this result.

Theorem 13. $\mathsf{DINF}[\mathsf{FFFO}^k]$ *is* PP_1*-complete with respect to many-one reductions for* $k > 2$ *and polynomial for* $k \leq 2$ *when domain size is given in unary notation.*

There are important knowledge representation formalisms within bounded-variable fragments of FFFO. For instance, many description logics belong to FFFO^2 [3]; one example is the logic $\mathcal{ALC}$ that we have discussed before. In the next section we examine description logics in more detail.

As a different exercise, we now consider the quantifier-free fragment of FFFO. In such a language, every logvar in the body of a definition axiom must appear in the defined relation, as no logvar is bound to any quantifier. Denote this language by QF; in Section 7 we show the close connection between QF and *plate models*. We have:

Theorem 14. *Suppose relations have bounded arity.* $\mathsf{INF}[\mathsf{QF}]$ *and* $\mathsf{QINF}[\mathsf{QF}]$ *are* PP*-complete with respect to many-one reductions, and* $\mathsf{DINF}[\mathsf{QF}]$ *requires constant computational effort. These results hold* even if domain size is given in binary notation.

As we have discussed at the end of Section 4, the literature on lifted inference and on probabilistic databases has produced deep results on query and domain complexity. One example is the definition of *safe queries*, a large class of formulas with tractable query complexity [29]; similar classes of formulas have been studied for symmetric probabilistic databases [53]. Note that even though a formula can be decided in polynomial time to be safe or not, computational support is needed for such a decision. As we have here focused on languages whose complexity can be determined directly from their syntactic features, we leave to future work the study of relational Bayesian network specifications that are based on safe queries and related languages.

6. Specifications based on description logics

The term "description logic" encompasses a large family of formal languages that can encode terminologies and assertions about individuals. Those languages are now fundamental knowledge representation tools, as they have solid

[5] The case $k = 1$ seems to be open; when $k = 1$, query complexity is polynomial when inference is solely on unary relations [133, 134]. When $k = 0$ we obtain propositional networks and then query complexity is polynomial by Theorem 3.

semantics and computational guarantees concerning reasoning tasks [3]. Given the favorable properties of description logics, much effort has been spent in mixing them with probabilities [81].

Relational Bayesian network specifications based on description logics can benefit from well tested tools and results, and offer a natural path to encode probabilistic ontologies. We have already examined the description logic $\mathcal{ALC}$ in the previous section, and we continue that study here.

Typically a description logic deals with *individuals*, *concepts*, and *roles*. An individual like John corresponds to a constant in first-order logic; a concept like researcher corresponds to a unary relation in first-order logic; and a role like buysFrom corresponds to a binary relation in first-order logic. A vocabulary contains a set of individuals plus some *primitve concepts* and some *primitive roles*. From these primitive concepts and roles one can define other concepts and roles using a set of operators. For instance, one may allow for concept *intersection*: then $C \sqcap D$ denotes the intersection of concepts C and D. Likewise, $C \sqcup D$ denotes the *union* of C and D, and $\neg C$ denotes the *complement* of C. For a role r and a concept C, a common construct is $\forall r.C$, called a *value restriction*. Another common construct is $\exists r.C$, an *existential restriction*. Description logics often define additional constructs such as intersection/union/complement of roles, composition of roles, and inverse of roles. For instance, usually r^- denotes the *inverse* of role r.

The semantics of description logics typically resorts to domains and interpretations. A *domain* $\mathcal{D}$ is a set. An *interpretation* $\mathbb{I}$ maps each individual to an element of the domain, each primitive concept to a subset of the domain, and each role to a set of pairs of elements of the domain. And then the semantics of $C \sqcap D$ is fixed by $\mathbb{I}(C \sqcap D) = \mathbb{I}(C) \cap \mathbb{I}(D)$. Similarly, $\mathbb{I}(C \sqcup D) = \mathbb{I}(C) \cup \mathbb{I}(D)$ and $\mathbb{I}(\neg C) = \mathcal{D} \backslash \mathbb{I}(C)$. And for the restricted quantifiers, we have $\mathbb{I}(\forall r.C) = \{x \in \mathcal{D} : \forall y : (x, y) \in \mathbb{I}(r) \rightarrow y \in \mathbb{I}(C)\}$ and $\mathbb{I}(\exists r.C) = \{x \in \mathcal{D} : \exists y : (x, y) \in \mathbb{I}(r) \wedge y \in \mathbb{I}(C)\}$. The semantics of the inverse role r^- is, unsurprisingly, given by $\mathbb{I}(r^-) = \{(x, y) \in \mathcal{D} \times \mathcal{D} : (y, x) \in \mathbb{I}(r)\}$.

We can translate this syntax and semantics to their counterparts in first-order logic. Thus $C \sqcap D$ can be read as $C(x) \wedge D(x)$, $C \sqcup D$ as $C(x) \vee D(x)$, and $\neg C$ as $\neg C(x)$. Moreover, $\forall r.C$ translates to $\forall y : r(x, y) \rightarrow C(y)$ and $\exists r.C$ translates to $\exists y : r(x, y) \wedge C(y)$.

The description logic $\mathcal{ALC}$ adopts intersection, complement, and existential restriction (union and value restrictions are obtained from those constructs). Definition 5 introduced the language ALC that corresponds to $\mathcal{ALC}$, and Theorem 5 examined its inferential complexity when domain size is given in binary notation. For domain size in unary notation, we have:

Theorem 15. *Suppose the domain size is specified in unary notation. Then* INF[ALC] *and* QINF[ALC] *are* PP*-complete with respect to many-one reductions, and* DINF[ALC] *is in* P*.*

Of course, we might go much further than ALC in expressivity and still be within the two-variable fragment of FFFO; for instance, we might allow for Boolean operations on roles, role composition and role inverses.

Conversely, we can also contemplate description logics that are less expressive than $\mathcal{ALC}$ in an attempt to obtain tractability. Indeed, some description logics combine selected Boolean operators with restricted quantification to obtain polynomial complexity of logical inferences. Two notable such description logics are $\mathcal{EL}$ and DL-Lite.[6]

6.1. Specifications based on the description logic $\mathcal{EL}$

Consider the description logic $\mathcal{EL}$, where the only allowed operators are intersection and existential restrictions [2]. Define:

Definition 6. *The language* EL *consists of all formulas recursively defined so that* $X(\chi)$ *and* $X(y)$ *are formulas when* X *is a unary relation,* $\phi \wedge \varphi$ *is a formula when both* ϕ *and* φ *are formulas, and* $\exists y : X(\chi, y) \wedge Y(y)$ *and* $\exists \chi : X(y, \chi) \wedge Y(\chi)$ *are formulas when* X *is a binary relation and* Y *is a unary relation, with the restriction that only the symbols* χ *and* y *can appear as logvars in formulas.*

That is, EL is the negation-free fragment of ALC. We note that $\mathcal{EL}$ also includes the *top concept* $\top$ that is interpreted as the whole domain; in our setting we essentially have it by introducing $\mathbb{P}(\top = 1) = 1$.

Because EL contains conjunction, we easily have that $\mathsf{INF}[\mathsf{EL}]$ is PP-hard by Theorem 2. And domain complexity is polynomial as implied by $\mathsf{DINF}[\mathsf{ALC}]$. Query complexity requires significant additional work as discussed in the proof of the next theorem:

Theorem 16. *Suppose the domain size is specified in unary notation. Then* $\mathsf{INF}[\mathsf{EL}]$ *and* $\mathsf{QINF}[\mathsf{EL}]$ *are* PP*-complete with respect to many-one reductions, even if the query contains only positive assignments, and* $\mathsf{DINF}[\mathsf{EL}]$ *is in* P*.*

6.2. Specifications based on the description logic DL-Lite

The description logic DL-Lite is particularly interesting because it captures central features of ER or UML diagrams, and yet common inference services have polynomial complexity [1, 18].

The simplicity and computational efficiency of the DL-Lite language have led many researchers to mix them with probabilities. For instance, D'Amato et al. [30] propose a variant of DL-Lite where the interpretation of each sentence is conditional on a context that is specified by a Bayesian network. A similar approach was taken by Ceylan and Peñalosa [21], with minor semantic differences. A different approach is to extend the syntax of DL-Lite sentences with probabilistic subsumption connectives, as in the *Probabilistic DL-Lite* [111]. Differently from our focus here, none of those proposals employ DL-Lite to specify Bayesian networks.

[6] Due to their favorable balance between expressivity and complexity, $\mathcal{EL}$ and DL-Lite are the basis for standard profiles of the OWL knowledge representation language, as explained at http://www.w3.org/TR/owl2-profiles/.

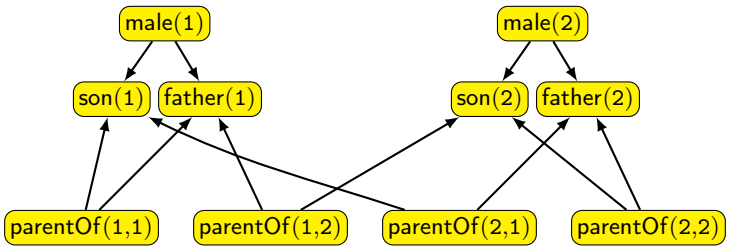


Figure 6: Grounding of relational Bayesian network specification from Example 6 on domain $\mathcal{D} = \{1, 2\}$.

In DL-Lite one has primitive concepts as before, and also *basic concepts*: a basic concept is either a primitive concept, or $\exists r$ for a role r, or $\exists r^-$ for a role r. Again, r^- denotes the inverse of r. And then a concept in DL-Lite is either a basic concept, or $\neg C$ when C is a basic concept, or $C \sqcap D$ when C and D are concepts. The semantics of r^-, $\neg C$ and $C \sqcap D$ are as before, and the semantics of $\exists r$ is, unsurprisingly, given by $\mathbb{I}(\exists r) = \{x \in \mathcal{D} : \exists y : (x, y) \in \mathbb{I}(r)\}$.

We focus on the negation-free fragment of DL-Lite; that is, we consider:

Definition 7. *The language* $\mathsf{DLLite}^{\mathsf{nf}}$ *consists of all formulas recursively defined so that* $X(\mathcal{X})$ *and* $X(y)$ *are formulas when* X *is a unary relation,* $\phi \wedge \varphi$ *is a formula when both* ϕ *and* φ *are formulas, and* $\exists y : X(\mathcal{X}, y)$, $\exists y : X(y, \mathcal{X})$, $\exists \mathcal{X} : X(y, \mathcal{X})$, *and* $\exists \mathcal{X} : X(\mathcal{X}, y)$ *are formulas when* X *is a binary relation, with the restriction that only* $\mathcal{X}$ *and* y *can appear as logvars in formulas.*

Example 6. Consider a short relational specification:

$$\begin{aligned} \mathbb{P}(\mathsf{male}(\mathcal{X}) = 1) &= 1/2, \\ \mathsf{father}(\mathcal{X}) &\equiv \mathsf{male}(\mathcal{X}) \wedge \exists y : \mathsf{parentOf}(\mathcal{X}, y), \\ \mathsf{son}(\mathcal{X}) &\equiv \mathsf{male}(\mathcal{X}) \wedge \exists y : \mathsf{parentOf}(y, \mathcal{X}), \end{aligned}$$

For domain $\mathcal{D} = \{1, 2\}$, this relational Bayesian network specification is grounded into the Bayesian network in Figure 6. □

Note that $\mathsf{INF}[\mathsf{DLLite}^{\mathsf{nf}}]$ is PP-hard by Theorem 2; moreover, as $\mathsf{DLLite}^{\mathsf{nf}}$ belongs to FFFO^2, $\mathsf{INF}[\mathsf{DLLite}^{\mathsf{nf}}]$ is in PP by Theorem 11 and $\mathsf{DINF}[\mathsf{DLLite}^{\mathsf{nf}}]$ is in P [132, 134].[7] More interestingly, we obtain tractability if we restrict queries to positive assignments:

Theorem 17. *Suppose the domain size is specified in unary notation. If in addition the query* $(\mathbf{Q}, \mathbf{E})$ *contains only positive assignments, then* $\mathsf{INF}[\mathsf{DLLite}^{\mathsf{nf}}]$, $\mathsf{QINF}[\mathsf{DLLite}^{\mathsf{nf}}]$, *and* $\mathsf{DINF}[\mathsf{DLLite}^{\mathsf{nf}}]$ *are in* P.

In proving this result (in Appendix A) we show that an inference with a positive query can be reduced to a particular weighted edge cover counting problem

[7] The exact complexity of $\mathsf{QINF}[\mathsf{DLLite}^{\mathsf{nf}}]$ is an open question.

of independent interest. And using these edge cover counting techniques we can also show that a related problem, namely finding the most probable explanation, is polynomial for relational Bayesian network specifications based on $\mathsf{DLLite}^{\mathsf{nf}}$. To state this result precisely, consider a relational Bayesian network specification $\mathbb{S}$ based on $\mathsf{DLLite}^{\mathsf{nf}}$, a set of assignments $\mathbf{E}$ for ground atoms, and a domain size N. Denote by $\mathbf{X}$ the set of random variables that correspond to groundings of relations in $\mathbb{S}$. Now there is at least one set of assignments $\mathbf{M}$ such that: (i) $\mathbf{M}$ contains assignments to all random variables in $\mathbf{X}$ that are not mentioned in $\mathbf{E}$; and (ii) $\mathbb{P}(\mathbf{M}, \mathbf{E})$ is maximum over all such sets of assignments. Denote by $\mathsf{MLE}(\mathbb{S}, \mathbf{E}, N)$ the problem that consists of finding such a set of assignments $\mathbf{M}$.

Theorem 18. *Given a relational Bayesian network $\mathbb{S}$ based on $\mathsf{DLLite}^{\mathsf{nf}}$, a set of positive assignments to grounded relations $\mathbf{E}$, and a domain size N in unary notation, $\mathsf{MLE}(\mathbb{S}, \mathbf{E}, N)$ can be solved in polynomial time.*

These results on $\mathsf{DLLite}^{\mathsf{nf}}$ can be directly extended in some important ways. For example, suppose we allow negative groundings of binary relations in the query. Then most of the proof of Theorem 17 follows through, but we must resort to approximations for weighted edge cover counting [80] so as to develop a fully polynomial-time approximation scheme (FPTAS) for inference. Moreover, the $\mathsf{MLE}(\mathbb{S}, \mathbf{E}, N)$ problem remains polynomial. Similarly, we could allow for different groundings of the same relation to be associated with different probabilities; the proofs given in Appendix A can be modified to develop a FPTAS for inference.

We have so far presented results for a number of languages. Table 1 (both table and caption) summarizes most of our findings; as noted previously, several results on FFFO with bounded relation arity and on FFFO^k have been in essence derived by Beame et al. [8].

7. Plates, probabilistic relational models, and related specification languages

In this paper we have followed a strategy that has long been cherished in the study of formal languages; that is, we have focused on languages that are based on minimal sets of constructs borrowed from logic. Clearly this plan succeeds only to the extent that results can be transfered to practical specification languages. In this section we examine some important cases where our strategy pays off.

Consider, for instance, *plate models*, a rather popular specification formalism. Plate models have been extensively used in statistical practice [82] since they were introduced in the BUGS project [47, 83]. In machine learning, they have been often used since their first appearance [16].

There seems to be no standard formalization for plate models, so we adapt some of our previous concepts as needed. A plate model consists of a set of parvariables, a directed acyclic graph where each node is a parvariable, and a set of *template conditional probability distributions*. Parvariables are *typed*: each

Language (N in unary notation)	Inferential	Query	Domain
Prop($\wedge$), positive query	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
QF with bound on relation arity	PP	PP	P
ALC	PP	PP	P
EL	PP	PP	P
EL, positive query	PP	PP	P
$\mathsf{DLLite}^{\mathsf{nf}}$, positive query	P	P	P

Table 1: Inferential, query and domain complexity for relational Bayesian networks based on various logical languages with domain size given in unary notation. All cells indicate completeness with respect to many-one reductions. On top of these results, note that when domain size is given in binary notation we have, with respect to many-one reductions: INF[FFFO] is PEXP-complete (even when restricted to relations of arity 1), QINF[FFFO] is PEXP-complete (even when restricted to relations of arity 2), and INF[ALC] is PEXP-complete.

parameter of a parvariable is associated with a set, the *domain* of the parameter. All parvariables that share a domain are said to belong to a *plate*. The central constraint on "standard" plate models is that the domains that appear in the parents of a parvariable must appear in the parvariable. For a given parvariable X, its corresponding template conditional probability distribution associates a probability value to each value of X given each configuration of parents of X.

Even though plate models often specify discrete and continuous random variables [82, 120], here we focus on parvariables with values 0 and 1, that can thus be put to correspondence with relations. Dealing with binary random variables is enough to prove PP-hardness in Theorem 19; the proof of this theorem shows that membership to PP is obtained even if parvariables have a bounded number of values that is larger than two.

We can use the same semantics as before to interpret plate models, with a small change: now the groundings of a relation are produced by running only over the domains of its associated logvars.

Example 7. Suppose we are interested in a "University World" containing a population of students and a population of courses [44]. Parvariable $\mathsf{Failed}?(\chi, y)$ yields the final status of student y in course χ; $\mathsf{Difficult}?(\chi)$ is a parvariable indicating the difficulty of a course χ, and $\mathsf{Committed}?(y)$ is a parvariable indicating the commitment of student y.

A plate model is drawn in Figure 7, where plates are rectangles. Each parvariable is associated with a template conditional probability distribution:

$$\mathbb{P}(\mathsf{Difficult}?(\chi) = 1) = 0.3, \qquad \mathbb{P}\big(\mathsf{Committed}?(y) = 1\big) = 0.7,$$

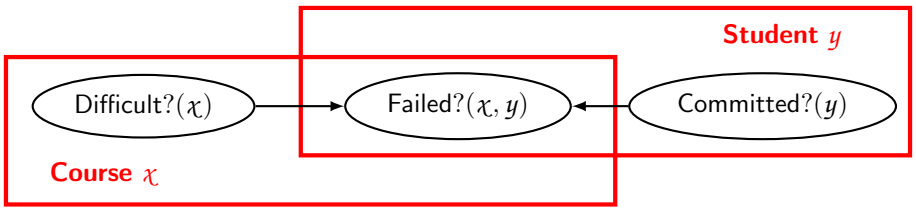


Figure 7: Plate model for the University World. We show logvars explicitly, even though they are not always depicted in plate models.

$$\mathbb{P}\Big(\mathsf{Failed?}(x,y)=1 \,\Big|\, \begin{array}{c}\mathsf{Difficult?}(x)=d,\\ \mathsf{Committed?}(y)=c\end{array}\Big) = \begin{cases} 0.6 & \text{if } d=0, c=0;\\ 0.2 & \text{if } d=0, c=1;\\ 0.9 & \text{if } d=1, c=0;\\ 0.5 & \text{if } d=1, c=1. \end{cases} \quad \square$$

Note that plate models can always be specified using definition axioms in the quantifier-free fragment of FFFO, given that the logvars of a relation appear in its parent relations. For instance, the table in Example 7 can be encoded as follows:

$$\mathsf{Failed?}(x,y) \equiv \left(\begin{array}{c}\big(\neg\mathsf{Difficult?}(x)\wedge\neg\mathsf{Committed?}(y)\wedge \mathsf{A}_1(x,y)\big)\vee\\ \big(\neg\mathsf{Difficult?}(x)\wedge \mathsf{Committed?}(y)\wedge \mathsf{A}_2(x,y)\big)\vee\\ \big(\mathsf{Difficult?}(x)\wedge\neg\mathsf{Committed?}(y)\wedge \mathsf{A}_3(x,y)\big)\vee\\ \big(\mathsf{Difficult?}(x)\wedge \mathsf{Committed?}(y)\wedge \mathsf{A}_4(x,y)\big)\end{array}\right), \quad (6)$$

where we introduced four auxiliary parvariables with associated assessments $\mathbb{P}\big(\mathsf{A}_1(x,y)=1\big) = 0.6$, $\mathbb{P}\big(\mathsf{A}_2(x,y)=1\big) = 0.2$, $\mathbb{P}\big(\mathsf{A}_3(x,y)=1\big) = 0.9$, and $\mathbb{P}\big(\mathsf{A}_4(x,y)=1\big) = 0.5$.

Denote by INF[PLATE] the language consisting of inference problems as in Definition 2, where relational Bayesian network specifications are restricted to satisfy the constraints of plate models. Adopt QINF[PLATE] and DINF[PLATE] similarly. We can reuse arguments in the proof of Theorem 14 to show that:

Theorem 19. INF[PLATE] *and* QINF[PLATE] *are* PP*-complete with respect to many-one reductions, and* DINF[PLATE] *requires constant computational effort. These results hold* even if the domain size is given in binary notation.

One can find extended versions of plate models in the literature, where a node can have children in other plates. See for instance the *smoothed Latent Dirichlet Allocation* (sLDA) model [10] depicted in Figure 8: here $\phi(z)$ has a child $W(x,y)$. In such extended plates a template conditional probability distribution can refer to logvars from plates that are not enclosing the parvariable. If definition axioms are used to specify such template distributions, one gets as before INF[FFFO], QINF[FFFO], etc; that is, results obtained in previous sections apply.

Besides plates, several other languages can encode repetitive Bayesian networks. Early proposals resorted to object orientation [71, 84], to frames [72], and

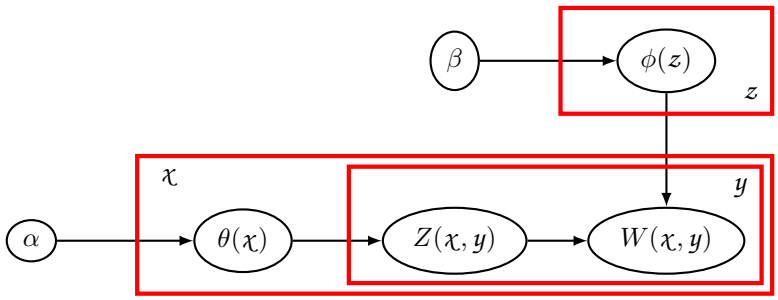


Figure 8: Smoothed Latent Dirichlet Allocation.

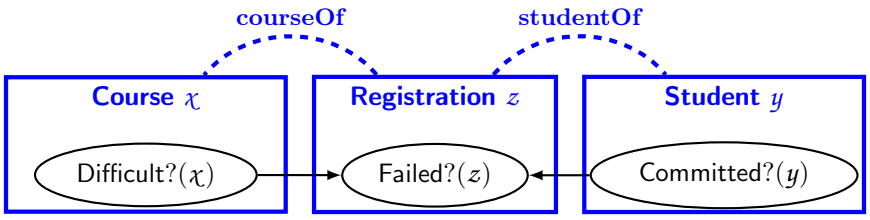


Figure 9: PRM for the University World. We show logvars explicitly, even though they are not always depicted in PRMs. Associations appear as dashed edges [44, 119].

to rules-based statements [5, 49, 55], all inspired by knowledge-based model construction [59, 50, 138]. Some of these proposals coalesced into a family of models loosely grouped under the name of *Probabilistic Relational Models* (PRMs) [43]. We here adopt the definition of PRMs by Getoor et al. [44]; again, to simplify matters, we focus on parvariables that correspond to relations.

Similarly to a plate model, a PRM contains typed parvariables and domains. A domain is now called a *class*; each class appears as a box containing parvariables. For instance, Figure 9 depicts a PRM for the University World: edges between parvariables indicate probabilistic dependence, and dashed edges between classes indicate *associations* between elements of the classes. In Figure 9 we have classes **Course**, **Student**, and **Registration**, with associations between them. Consider association **studentOf**: the idea is that **studentOf**(χ, z) holds when χ is the student in registration z. Following terminology by Koller and Friedman [69], we say that relations that encode classes and associations, such as **Course** and **studentOf**, are *guard parvariables*.

A *relational skeleton* for a PRM is an explicit specification of elements in each class, plus the explicit specification of pairs of objects that are associated. That is, the relational skeleton specifies the groundings of the guard parvariables.

Each parvariable X in a PRM is then associated with a *template probability distribution* that specifies probabilities for the parvariable X given a selected set of other parvariables. The latter are the *parents* of X, again denoted by $\mathrm{pa}(X)$. In the University World of Figure 9, we must associate with Failed? the template probabilities for $\mathbb{P}\big(\mathsf{Failed?}(z) | \mathsf{Difficult?}(\chi), \mathsf{Committed?}(y)\big)$. But differently from

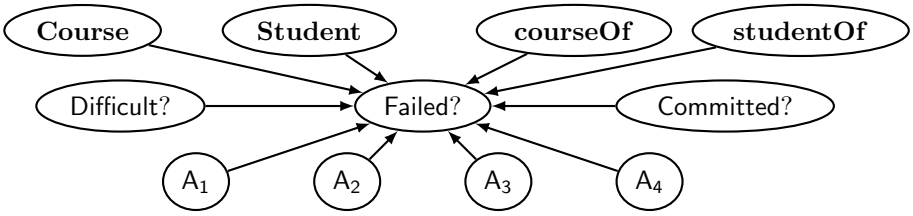


Figure 10: Parvariable graph for the University World PRM.

plate models, here the parents of a particular grounding are determined by going through associations: for instance, to find the parents of $\mathsf{Failed?}(r)$, we must find the course c and the student s such that $\mathbf{courseOf}(c, r)$ and $\mathbf{studentOf}(s, r)$ hold, and then we have parents $\mathsf{Difficult?}(c)$ and $\mathsf{Committed?}(s)$.

All of these types and associations can, of course, be encoded using first-order logic, as long as all parvariables correspond to relations. For instance, here is a definition axiom that captures the PRM for the University World:

$$\mathsf{Failed?}(z) \quad \equiv \quad \forall \mathcal{X} : \forall y : \left(\begin{array}{c} \mathbf{Course}(\mathcal{X}) \wedge \mathbf{Student}(y) \wedge \\ \mathbf{courseOf}(\mathcal{X}, z) \wedge \mathbf{studentOf}(y, z) \end{array} \right) \rightarrow \left(\begin{array}{c} \big(\neg \mathsf{Difficult?}(\mathcal{X}) \wedge \neg \mathsf{Committed?}(y) \wedge \mathsf{A}_1(\mathcal{X}, y)\big) \vee \\ \big(\neg \mathsf{Difficult?}(\mathcal{X}) \wedge \mathsf{Committed?}(y) \wedge \mathsf{A}_2(\mathcal{X}, y)\big) \vee \\ \big(\mathsf{Difficult?}(\mathcal{X}) \wedge \neg \mathsf{Committed?}(y) \wedge \mathsf{A}_3(\mathcal{X}, y)\big) \vee \\ \big(\mathsf{Difficult?}(\mathcal{X}) \wedge \mathsf{Committed?}(y) \wedge \mathsf{A}_4(\mathcal{X}, y)\big) \end{array} \right),$$

using the same auxiliary parvariables employed in Expression (6). The parvariable graph for the resulting specification is depicted in Figure 10.

Thus we can take a PRM and translate it into a relational Bayesian network specification $\mathbb{S}$. As long as the parvariable graph is acyclic, results in the previous sections apply. To see this, note that a skeleton is simply an assignment for all groundings of the guard parvariables. Thus a skeleton can be encoded into a set of assignments $\mathbf{S}$, and our inferences should focus on deciding $\mathbb{P}(\mathbf{Q}|\mathbf{E}, \mathbf{S}) > \gamma$ with respect to $\mathbb{S}$ and a domain that is the union of all classes of the PRM.

Now suppose we have a fixed PRM and we receive as input a skeleton and a query $(\mathbf{Q}, \mathbf{E})$, and we wish to decide whether $\mathbb{P}(\mathbf{Q}|\mathbf{E}) > \gamma$: if the template probability distributions are specified with FFFO, and the parvariable graph is acyclic, we have that query complexity is PP-complete. We can replay our previous results on inferential and query complexity this way. The concept of domain complexity seems less meaningful when PRMs are considered: the larger the domain, the more data on guard parvariables are needed, so we cannot really fix the domain in isolation.

We conclude this section with a few observations.

Cyclic parvariable graphs. Our results assume acyclicity of parvariable graphs, but this is not a condition that is typically imposed on PRMs. A cyclic parvariable graph may still produce an acyclic grounding, depending on the given

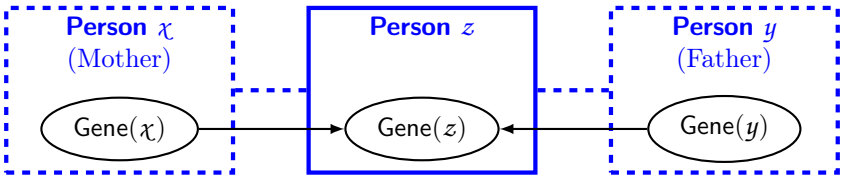


Figure 11: A PRM for the blood-type model, adapted from a proposal by Getoor et al. [44]. Dashed boxes stand for repeated classes; Getoor et al. suggest that some associations may be constrained to be "guaranteed acyclic" so that the whole model is consistent for any skeleton that satisfies the constraints.

skeleton. For instance, one might want to model blood-type inheritance, where a **Person** inherits a genetic predisposition from another **Person**. This creates a loop around the class **Person**, even though we do not expect a cycle in any valid grounding of the PRM. The literature has proposed languages that allow cycles [44, 58]; one example is shown in Figure 11. The challenge then is to guarantee that a given skeleton will lead to an acyclic grounded Bayesian network; future work on cyclic parvariable graphs must deal with such a consistency problem [35].

Other specification languages. There are several other languages that specify PRMs and related formalisms; such languages can be subjected to the same analysis we have explored in this paper. A notable formalism is the Probabilistic Relational Language (PRL) [45]. In PRL a logic program is used to specify a PRM; the specification is divided into logical background (that is, guard parvariables), probabilistic background, and probabilistic dependencies. Two other examples of textual formalisms that can be used to encode PRMs are Logical Bayesian Networks (LBNs) [40, 39] and Bayesian Logic Programs (BLPs) [67, 110]. Both distinguish between *logical* predicates that constrain the grounding into graphs (that is, guard parvariables), and *probabilistic* or *Bayesian* predicates that encode probabilistic assessments [93].

A more visual language, based on Entity-Relationship Diagrams, is DAPER [58]. Figure 12 shows a DAPER diagram for the University World and a DAPER diagram for the blood-type model. Another diagrammatic language is given by Multi-Entity Bayesian Networks (MEBNs), a graphical representation for arbitrary first-order sentences [76]. Several other graphical languages mix probabilities with description logics [20, 23, 36, 70], as mentioned in Section 6.

There are other formalisms in the literature that focus on textual specifications. For instance, Jaeger's *Relational Bayesian Networks* [60, 61] offer a solid representation language where the user can directly specify and manipulate probability values. As a short example, one can specify a Relational Bayesian

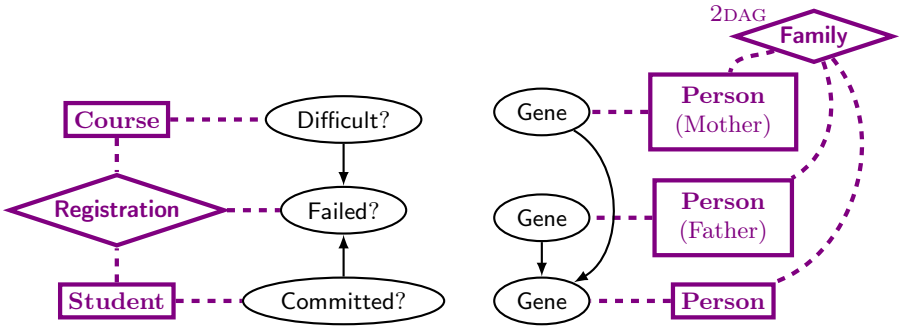


Figure 12: Left: A DAPER diagram for the University World. Right: A DAPER diagram for the blood-type model, as proposed by Heckerman et al. [58]; note the constraint 2DAG, meaning that each child of the node has at most two parents and cannot be its own ancestor.

Network as follows

$$\begin{aligned} \mathsf{burglary}(\mathcal{X}) &= 0.005; \\ \mathsf{alarm}(\mathcal{X}) &= 0.95\mathsf{burglary}(\mathcal{X}) + 0.01(1 - \mathsf{burglary}(\mathcal{X})); \\ \mathsf{cityAlarm} &= \textsc{NoisyOr}\{0.8\mathsf{alarm}(\mathcal{X})|\mathcal{X}; \mathcal{X} = \mathcal{X}\}; \end{aligned}$$

The first sentence specifies, in our framework, $\mathbb{P}(\mathsf{burglary}(\mathcal{X}) = 1) = 0.005$. The second sentence yields the probability of $\mathsf{alarm}(\mathcal{X})$ as a combination of probabilities and values of $\mathsf{burglary}(\mathcal{X})$ — this sentence actually yields a conditional probability. Finally the third sentence uses some special syntax to build a Noisy-Or gate by essentially quantifying over all possible values of logvar $\mathcal{X}$. We have examined the complexity of Relational Bayesian Networks elsewhere [86]; some results and proofs, but not all of them, are similar to the ones presented here.

There are also languages that encode repetitive Bayesian networks using functional programming [85, 89, 100, 123] or logic programming [24, 41, 101, 102, 112, 115], As an example, the functional language Venture allows statements such as $\mathsf{flip}(0.4)$ that assigns probability 0.4 to a particular computation step [85]. Probabilistic logic languages are particularly close to our framework as they have a declarative style, where rules convey the deterministic operations that are mixed with probabilities. Consider for instance ProbLog [41], a popular language that adopts syntactic and semantic conventions usually referred to as "Sato's distribution semantics" [115]. In ProbLog one can write

$$0.005 :: \mathsf{burglary}(X).$$

to mean, in our framework, $\mathbb{P}(\mathsf{burglary}(\mathcal{X}) = 1) = 0.005$. And one can write a rule such as

$$\mathsf{alarm} :- \mathsf{eneryOn}, \mathsf{burglary}(X).$$

to mean

$$\mathsf{alarm} \equiv \mathsf{energyOn} \wedge \exists \mathcal{X} : \mathsf{burglary}(\mathcal{X}).$$

We have examined the complexity of probabilistic logic programming elsewhere [26]; again, some results and proofs, but not all of them, are similar to the ones presented here. Finally, we note that there are other languages that look for relational counterparts of Markov networks, such as relational Markov networks or Markov logic networks [46]. These formalisms are not directly connected to our efforts here, but one can usually translate complexity results on models that employ directed models into results on models that employ undirected models [69]. We leave the pursuit of this path to future work.

8. A detour into Valiant's counting hierarchy

We have so far focused on inferences that compare a conditional probability with a given rational number. However one might argue that the real purpose of a probabilistic inference is to compute a probability value. While most previous work on probabilistic databases and lifted inference has looked at the computation of probability values and has produced results using Valiant's counting hierarchy, in this paper we have so far focused on Wagner's counting hierarchy. In this section we justify our strategy and adapt our results to Valiant's hierarchy.

Valiant defines, for complexity class A, the class $\#\mathsf{A}$ to be $\cup_{\mathcal{L}\in\mathsf{A}}(\#\mathsf{P})^{\mathcal{L}}$, where $(\#\mathsf{P})^{\mathcal{L}}$ is the class of functions counting the accepting paths of nondeterministic polynomial-time Turing machines with $\mathcal{L}$ as oracle [129]. Valiant declares function f to be $\#\mathsf{P}$-hard when $\#\mathsf{P} \subseteq \mathsf{FP}^f$; that is, f is $\#\mathsf{P}$-hard if any function f' in $\#\mathsf{P}$ can be reduced to f by the analogue of a polynomial-time Turing reduction (recall that FP is the class of functions that can be computed in polynomial-time by a deterministic Turing machine). Valiant's is a very loose notion of hardness; as shown by Toda and Watanabe [126], any function in $\#\mathsf{PH}$ can be reduced to a function in $\#\mathsf{P}$ via a Turing reduction (where $\#\mathsf{PH}$ is a counting class with the whole polynomial hierarchy as oracle). In fact their result is stronger as the reduction can be made to compute the function in $\#\mathsf{P}$ a single time (that is, a one-Turing reduction). Thus a Turing reduction is too weak to distinguish classes of functions within $\#\mathsf{PH}$. For this reason, other reductions have been considered for counting problems [7, 38].

A somewhat stringent strategy is to say that f is $\#\mathsf{P}$-hard if any function f' in $\#\mathsf{P}$ can be produced from f by a *parsimonious* reduction; that is, $f'(\ell)$ is computed by applying a polynomial-time function g to ℓ and then computing $f(g(\ell))$ [117]. However, such a strategy is inadequate for our purposes: counting classes such as $\#\mathsf{P}$ produce integers, and we cannot produce integers by computing probabilities.

A sensible strategy is to adopt a reduction that allows for multiplication by a polynomial function. This has been done both in the context of probabilistic inference with "reductions modulo normalization" [73] and in the context of probabilistic databases [121]. We adopt the reductions proposed by Bulatov et al. in their study of weighted constraint satisfaction problems [15]. They define *weighted reductions* as follows (Bulatov et al. consider functions into the

algebraic numbers, but for our purposes we can restrict the weighted reductions to rational numbers):

Definition 8. *Consider functions f_1 and f_2 from an input language $\mathcal{L}$ to rational numbers $\mathbb{Q}$. A* weighted reduction *from f_1 to f_2 is a pair of polynomial-time functions $g_1 : \mathcal{L} \to \mathbb{Q}$ and $g_2 : \mathcal{L} \to \mathcal{L}$ such that $f_1(\ell) = g_1(\ell) f_2(g_2(\ell))$ for all ℓ.*

We say a function f is #P-hard with respect to weighted reductions if any function in #P can be reduced to f via a weighted reduction.

Having decided how to define hardness, we must look at membership. As counting problems generate integers, we cannot really say that probabilistic inference problems belong to any class in Valiant's counting hierarchy. In fact, in his seminal work on the complexity of Bayesian networks, Roth notes that "strictly speaking the problem of computing the degree of belief is not in #P, but easily seem equivalent to a problem in this class" [113]. The challenge is to formalize such an equivalence.

Grove, Halpern, and Koller quantify the complexity of probabilistic inference by allowing polynomial computations to occur after counting [54, Definition 4.12]. Their strategy is to say that f is #P-*easy* if there exists $f' \in \#\mathsf{P}$ and $f'' \in \mathsf{FP}$ such that for all ℓ we have $f(\ell) = f''(f'(\ell))$. Similarly, Campos, Stamoulis and Weyland take f to be #P[1]-*equivalent* if f is #P-hard (in Valiant's sense) and belongs to $\mathsf{FP}^{\#\mathsf{P}[1]}$. Here the superscript #P[1] means that the oracle #P can be called only once. It is certainly a good idea to resort to a new term ("equivalence") in this context; however one must feel that membership to $\mathsf{FP}^{\#\mathsf{P}[1]}$ is too weak a requirement given Toda and Watanabe's theorem [126]: any function in #PH can be produced within $\mathsf{FP}^{\#\mathsf{P}[1]}$.

We adopt a stronger notion of equivalence: a function f is #P-*equivalent* if it is #P-hard with respect to weighted reductions and $g \cdot f$ is in #P for some polynomial-time function g.

Also, we need to define a class of functions that corresponds to the complexity class PEXP. We might extend Valiant's definitions and take #EXP to be $\cup_{\mathcal{L} \in \mathsf{EXP}} (\#\mathsf{P})^{\mathcal{L}}$. However functions in such a class produce numbers whose size is at most polynomial on the size of the input, as the number of accepting paths of a polynomial-time nondeterministic Turing machine on input ℓ is bounded by $2^{p(|\ell|)}$ where p is polynomial and $|\ell|$ is the length of ℓ. This is not appropriate for our purposes, as even simple specifications may lead to exponentially long output (for instance, take $\mathbb{P}(X(\chi) = 1) = 1/2$ and compute $\mathbb{P}(\exists \chi : X(\chi))$: we must be able to write the answer $1 - 1/2^N$ using N bits, and this number of bits is exponential on the input if the domain size is given in binary notation). For this reason, we take #EXP to be the class of functions that can be computed by counting Turing machines of exponential-time complexity [96]. We say that a function f is #EXP-equivalent if f is #EXP-hard with respect to reductions that follow exactly Definition 8, except for the fact that g_1 may be an exponential-time function, and $g \cdot f$ is in #EXP for some exponential-time function g.

Now consider polynomial bounds on space. We will use the class $\natural$PSPACE defined by Ladner [75], consisting of those functions that can be computed by

counting Turing machines with a polynomial-space bound *and* a polynomial bound on the number of nondeterministic moves. This class is actually equal to FPSPACE[poly], the class of functions computable in polynomial space whose outputs are strings encoding numbers in binary notation, and bounded in length by a polynomial [75, Theorem 2]. We say that a function is ♮PSPACE-equivalent if f is ♮PSPACE-hard with respect to weighted reductions (as in Definition 8), and $g \cdot f$ is in #PSPACE for some polynomial-space function g from the input language to the rational numbers. Of course we might have used "FPSPACE[poly]-equivalent" instead, but we have decided to follow Ladner's original notation.

There is one more word of caution when it comes to adopting Valiant's counting Turing machines. It is actually likely that conditional probabilities $\mathbb{P}(\mathbf{Q}|\mathbf{E})$ cannot be produced by counting Turing machines, as classes in Valiant's counting hierarchy are not likely to be closed under division even by polynomial-time computable functions [95]. Thus we must focus on inferences of the form $\mathbb{P}(\mathbf{Q})$; indeed this is the sort of computation that is analyzed in probabilistic databases [121].

Thus the drawback of Valiant's hierarchy is that a significant amount of adaptation is needed before that hierarchy can be applied to probabilistic inference; hence our preference for Wagner's hierarchy. But after all this preliminary work, we can easily convert our previous results accordingly. For instance, we have the following inferential complexity results:

Theorem 20. *Consider the class of functions that gets as input a relational Bayesian network specification based on* FFFO*, a domain size N (in binary or unary notation), and a set of assignments $\mathbf{Q}$, and returns $\mathbb{P}(\mathbf{Q})$. This class of functions is* #EXP*-equivalent.*

Theorem 21. *Consider the class of functions that gets as input a relational Bayesian network specification based on* FFFO *with relations of bounded arity, a domain size N in unary notation, and a set of assignments $\mathbf{Q}$, and returns $\mathbb{P}(\mathbf{Q})$. This class of functions is* ♮PSPACE*-equivalent.*

Theorem 22. *Consider the class of functions that gets as input a relational Bayesian network specification based on* FFFO^k *for $k \geq 2$, a domain size N in unary notation, and a set of assignments $\mathbf{Q}$, and returns $\mathbb{P}(\mathbf{Q})$. This class of functions is* #P*-equivalent.*

Theorem 23. *Consider the class of functions that get as input a plate model based on* FFFO*, a domain size N (either in binary or unary notation), and a set of assignments $\mathbf{Q}$, and returns $\mathbb{P}(\mathbf{Q})$. This class of functions is* #P*-equivalent.*

9. Conclusion

We have presented a framework for specification and analysis of Bayesian networks, particularly networks containing repetitive patterns that can be captured using function-free first-order logic. Our specification framework is based on previous work on probabilistic programs and structural equation models; our

analysis is based on notions of complexity (inferential, combined, query, data, domain) that are similar to concepts used in lifted inference and in probabilistic databases.

Our emphasis was on knowledge representation; in particular we wanted to understand how features of the specification language affect the complexity of inferences. Thus we devoted some effort to connect probabilistic modeling with knowledge representation formalisms, particularly description logics. We hope that we have produced here a sensible framework that unifies several disparate efforts, a contribution that may lead to further insight into probabilistic modeling.

Another contribution of this work is a collection of results on complexity of inferences, as summarized by Table 1 and related commentary. We have also introduced relational Bayesian network specifications based on the $\mathsf{DLLite}^{\mathsf{nf}}$ logic, a language that can be used to specify probabilistic ontologies and a sizeable class of probabilistic entity-relationship diagrams. In proving results about $\mathsf{DLLite}^{\mathsf{nf}}$, we have identified a class of model counting problems with tractability guarantees. Finally, we have shown how to transfer our results into plate models and PRMs, and in doing so we have presented a much needed analysis of these popular specification formalisms.

There are several avenues open for future work. Ultimately, we must understand the relationship between expressivity and complexity of Bayesian networks specifications as thoroughly as we understand this relationship for logical languages. To do so, we may consider Bayesian networks specified by operators from various description and modal logics, or look at languages that allow random variables to have more than two values. In a different direction, we must look at parameterized counting classes [42], so as to refine the analysis even further. There are also several problems that go beyond the inferences discussed in this paper: for example, the computation of Maximum a Posteriori (MAP) configurations, and the verification that a possibly cyclic PRM is consistent for every possible skeleton [35]. There are also models that encode structural uncertainty, say about the presence of edges [69]; novel techniques must be developed to investigate the complexity of such models.

Acknowledgements

The first author is partially supported by CNPq (grant 308433/2014-9) and received financial support from FAPESP (grant 2016/18841-0); the second author is partially supported by CNPq (grants 303920/2016-5 and 420669/2016-7), and received financial support from FAPESP (2016/01055-1).

We thank Cassio Polpo de Campos for valuable insights concerning complexity proofs.

L_1	L_2	L_3	B_1	B_2	B_3	B_4	B_5
0	0	1	0	0	1	0	0
0	1	0	0	0	0	0	1
0	1	1	0	0	0	1	1
1	0	0	0	1	0	0	1
1	0	1	0	1	0	1	1
1	1	0	1	0	0	0	0
1	1	1	1	0	0	1	0

Table A.2: Assignments that satisfy the four clauses in Expression (A.1).

Appendix A. Proofs

Appendix A.1. Propositional languages (Section 3)

Proposition 1. *Both* #3SAT(>) *and* #*(1-in-3)*SAT(>) *are* PP*-complete with respect to many-one reductions.*

Proof. Consider first #3SAT(>). It belongs to PP because deciding $\#\phi > k$, for propositional sentence ϕ, is in PP [117, Theorem 4.4]. And it is PP-hard because it is already PP-complete when the input is $k = 2^{n/2} - 1$ [6, Proposition 1].

Now consider #(1-in-3)SAT(>). Suppose the input is a propositional sentence ϕ in 3CNF with propositions $A_1, \ldots, A_n$ and m clauses. Turn ϕ into another sentence φ in 3CNF by turning each clause $L_1 \vee L_2 \vee L_3$ in ϕ into a set of four clauses:

$$\neg L_1 \vee B_1 \vee B_2, \quad L_2 \vee B_2 \vee B_3, \quad \neg L_3 \vee B_3 \vee B_4, \quad B_1 \vee B_3 \vee B_5, \qquad \text{(A.1)}$$

where the B_j are fresh propositions not in ϕ. We claim that $\#\varphi = \#(\text{1-in-3})\phi$; that is, $\#(\text{1-in-3})\phi > k$ is equivalent to $\#\phi > k$, proving the desired hardness. To prove this claim, note that for each clause $L_1 \vee L_2 \vee L_3$ in ϕ, and for each assignment to (L_1, L_2, L_3) that satisfies the clause, there is only one assignment to $(B_1, B_2, B_3, B_4, B_5)$ that satisfies the clauses in Expression (A.1). To prove this, Table A.2 presents the set of *all* assignments that satisfy the latter four clauses. □

Theorem 1. INF[Prop($\wedge$)] *is in* P *when the query* $(\mathbf{Q}, \mathbf{E})$ *contains only positive assignments, and* INF[Prop($\vee$)] *is in to* P *when the query contains only negative assignments.*

Proof. Consider first INF[Prop($\wedge$)]. To run inference with positive assignments $(\mathbf{Q}, \mathbf{E})$, just run d-separation to collect the set of root variables that must be true given the assignments (note that as soon as a node is set to true, its parents must be true, and so on recursively). Then the probability of the conjunction of assignments in $\mathbf{Q}$ and in $\mathbf{E}$ is just the product of probabilities for these latter atomic propositions to be true, and these probabilities are given in the network

specification. Thus we obtain $\mathbb{P}(\mathbf{Q}, \mathbf{E})$. Now repeat the same polynomial computation only using assignments in $\mathbf{E}$, to obtain $\mathbb{P}(\mathbf{E})$, and determine whether $\mathbb{P}(\mathbf{Q}, \mathbf{E}) / \mathbb{P}(\mathbf{E}) > \gamma$ or not.

Now consider $\mathsf{INF}[\mathsf{Prop}(\vee)]$. For any input network specification, we can easily build a network specification in $\mathsf{INF}[\mathsf{Prop}(\wedge)]$ by turning every variable X into a new variable X' such that $X = \neg X'$. Then the root node associated with assessment $\mathbb{P}(X = 1) = \alpha$ is turned into a root node associated with $\mathbb{P}(X' = 1) = 1 - \alpha$, and a definition axiom $X \equiv\!\equiv \vee_i Y_i$ is turned into a definition axiom $X' \equiv\!\equiv \wedge_i Y_i'$. Any negative evidence is then turned into positive evidence, and the reasoning in the previous paragraph applies. □

Theorem 2. $\mathsf{INF}[\mathsf{Prop}(\wedge)]$ *and* $\mathsf{INF}[\mathsf{Prop}(\vee)]$ *are* PP*-complete with respect to many-one reductions.*

Proof. Membership follows from the fact that $\mathsf{INF}[\mathsf{Prop}(\wedge, \neg)] \in \mathsf{PP}$. We therefore focus on hardness.

Consider first $\mathsf{INF}[\mathsf{Prop}(\wedge)]$. We present a parsimonious reduction from $\#(\text{1-in-3})\mathsf{SAT}(>)$ to $\mathsf{INF}[\mathsf{Prop}(\wedge)]$, following a strategy by Mauá et al. [87].

Take a sentence ϕ in 3CNF with propositions $A_1, \ldots, A_n$ and m clauses. If there is a clause containing three times the same literal (for instance, $(A_1 \vee A_1 \vee A_1)$), then the 1-in-3 rule cannot be respected; we assume that such problems are detected and solved at once, hence we deal with formulas where each clause does not have three identical literals.

For each literal in ϕ, introduce a random variable X_{ij}, where i refers to the ith clause, and j refers to the jth literal (note: $j \in \{1, 2, 3\}$). The set of all such random variables is $\mathbf{L}$.

For instance, suppose we have the sentence $(A_1 \vee A_2 \vee A_3) \wedge (A_4 \vee \neg A_1 \vee A_3)$. We then make the correspondences: $X_{11} \rightsquigarrow A_1$, $X_{12} \rightsquigarrow A_2$, $X_{13} \rightsquigarrow A_3$, $X_{21} \rightsquigarrow A_4$, $X_{22} \rightsquigarrow \neg A_1$, $X_{23} \rightsquigarrow A_3$.

Note that $\{X_{ij} = 1\}$ indicates an assignment of true to the corresponding literal. Say that a configuration of $\mathbf{L}$ is *gratifying* if $X_{i1} + X_{i2} + X_{i3} \geq 1$ for every clause (without necessarily respecting the 1-in-3 rule). Say that a configuration is *respectful* if is respects the 1-in-3 rule; that is, if $X_{i1} + X_{i2} + X_{i3} \leq 1$ for every clause. And say that a configuration is *sensible* if two variables that correspond to the same literal have the same value, and two variables that correspond to a literal and its negation have distinct values (in the example in the last paragraph, both $\{X_{11} = 1, X_{22} = 1\}$ and $\{X_{13} = 1, X_{23} = 0\}$ fail to produce a sensible configuration).

For each random variable X_{ij}, introduce the assessment $\mathbb{P}(X_{ij} = 1) = 1 - \varepsilon$, where ε is a rational number to be determined later. Our strategy is to introduce definition axioms so that only the gratifying-respectful-sensible configurations of $\mathbf{L}$ get high probability, while the remaining configurations have low probability. The main challenge is to do so without negation.

Let $\mathbf{Q}$ be an initially empty set of assignments. We first eliminate the configurations that do not respect the 1-in-3 rule. To do so, for $i = 1, \ldots, m$ include

definition axioms

$$Y_{i1i2} \equiv\equiv X_{i1} \wedge X_{i2}, \qquad Y_{i1i3} \equiv\equiv X_{i1} \wedge X_{i3}, \qquad Y_{i3i2} \equiv\equiv X_{i2} \wedge X_{i3}, \quad \text{(A.2)}$$

and add $\{Y_{i1i2} = 0, Y_{i1i3} = 0, Y_{i2i3} = 0\}$ to $\mathbf{Q}$. This guarantees that configurations of $\mathbf{L}$ that fail to be respectful are incompatible with $\mathbf{Q}$.

We now eliminate gratifying-respectful configurations that are not sensible. We focus on gratifying and respectful configurations because, as we show later, ungratifying configurations compatible with $\mathbf{Q}$ are assigned low probability.

- Suppose first that we have a clause where the same proposition appears more than once. There are three possibilities (recall that clauses with three identical literals are not possible at this point). If the ith clause can be written as $(A \vee A \vee L)$ for some proposition A and some literal L (that may even be $\neg A$), then add $\{X_{i3} = 1\}$ to $\mathbf{Q}$. If instead the ith clause can be written as $(\neg A \vee \neg A \vee L)$ for some proposition A and some literal L (that may even be A), then add $\{X_{i3} = 1\}$ to $\mathbf{Q}$. Finally, if the ith clause can be written as $(A \vee \neg A \vee L)$ for some proposition A and some literal L that is unrelated to A, then add $\{X_{i3} = 0\}$ to $\mathbf{Q}$.

- Suppose we have two distinct clauses $(A \vee L_{i2} \vee L_{i3})$ and $(\neg A \vee L_{j2} \vee L_{j3})$, where A is a proposition and the L_{uv} are literals (possibly referring more than once to the same propositions). Suppose the six literals in these two clauses correspond to variables (X_{i1}, X_{i2}, X_{i3}) and (X_{j1}, X_{j2}, X_{j3}), in this order. We must have $X_{i1} = 1 - X_{j1}$. To encode this relationship, we take two steps. First, introduce the definition axiom

  $$Y_{i1j1} \equiv\equiv X_{i1} \wedge X_{j1},$$

  and add $\{Y_{i1j1} = 0\}$ to $\mathbf{Q}$: at most one of X_{i1} and X_{j1} is equal to 1, but there may still be gratifying-respectful configurations where $X_{i1} = X_{j1} = 0$. Thus the second step is enforce the sentence $\theta = \neg(L_{i2} \vee L_{i3}) \vee \neg(L_{j2} \vee L_{j3})$, as this forbids $X_{i1} = X_{j1} = 0$. Note that θ is equivalent to $\neg(L_{i2} \wedge L_{j2}) \wedge \neg(L_{i2} \wedge L_{j3}) \wedge \neg(L_{i3} \wedge L_{j2}) \wedge \neg(L_{i3} \wedge L_{j3})$, so introduce the definition axiom

  $$Y_{iujv} \equiv\equiv X_{iu} \wedge X_{jv}$$

  and add $\{Y_{iujv} = 0\}$ to $\mathbf{Q}$, for each $u \in \{2, 3\}$ and $v \in \{2, 3\}$. Proceed similarly if the literals of interest appear in other positions in the clauses.

- Now we focus on any two distinct clauses that share a literal. Say we have $(A \vee L_{i2} \vee L_{i3})$ and $(A \vee L_{j2} \vee L_{j3})$ where the symbols are as in the previous bullet, and where the literals are again paired with variables (X_{i1}, X_{i2}, X_{i3}) and (X_{j1}, X_{j2}, X_{j3}). If $X_{i1} = 1$, then we must have $X_{j1} = 1$, and to guarantee this in a gratifying-respectful configuration we introduce

  $$Y_{i1j2} \equiv\equiv X_{i1} \wedge X_{j2}, \qquad Y_{i1j3} \equiv\equiv X_{i1} \wedge X_{j3},$$

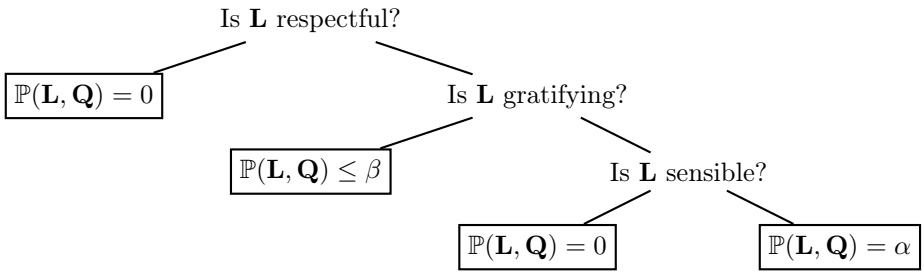


Figure A.13: Decision tree of the probability assigned to configurations of the network constructed in the proof of Theorem 2. Right branch always follows "yes", while left branch always follows "no".

and add $\{Y_{i1j2}=0, Y_{i1j3}=0\}$ to $\mathbf{Q}$. Similarly, if $X_{j1}=1$, we must have $X_{i1}=1$, so introduce

$$Y_{i2j1} \equiv\equiv X_{i2} \wedge X_{j1}, \qquad Y_{i3j1} \equiv\equiv X_{i3} \wedge X_{j1},$$

and add $\{Y_{i2j1}=0, Y_{i3j1}=0\}$ to $\mathbf{Q}$. Again, proceed similarly if the literals of interest appear in other positions in the clauses.

Consider a configuration $x_{11},\ldots,x_{m3}$ of $\mathbf{L}$. If this is a gratifying-respectful-sensible configuration, we have that

$$\mathbb{P}(X_{11}=x_{11},\ldots,X_{m3}=x_{m3}) = (1-\varepsilon)^m\varepsilon^{2m} = \alpha\,.$$

If the configuration is respectful but *not* gratifying, then

$$\mathbb{P}(X_{11}=x_{11},\ldots,X_{m3}=x_{m3}) \leq (1-\varepsilon)^{m-1}\varepsilon^{2m+1} = \beta\,.$$

The number of respectful configurations is at most 4^m, since for each i there are 4 ways to assign values to (X_{i1},X_{i2},X_{i3}) such that $X_{i1}+X_{i2}+X_{i3}\leq 1$.

The whole reasoning is illustrated in the decision tree in Figure A.13.

If the number of solutions to the original problem is strictly greater than k then $\mathbb{P}(\mathbf{Q}) \geq (k+1)\alpha$. And if the number of solutions is smaller or equal than k then $\mathbb{P}(\mathbf{Q}) \leq k\alpha + 4^m\beta$. Now we must choose ε so that $(k+1)\alpha > k\alpha + 4^m\beta$, so that we can differentiate between the two cases. We do so by choosing $\varepsilon < 1/(1+4^m)$. Then $(\varphi, k+1)$ is in the language #(1-in-3)$\mathsf{SAT}(>)$ iff $\mathbb{P}(\mathbf{Q}) > (k+1)\alpha$.

The whole construction is polynomial: the number of definition axioms is at most quadratic in the number of literals of φ, and ε can be encoded with $\mathcal{O}(m+n)$ bits.

Because the construction just described is somewhat complicated, we present an example. Consider again the sentence $(A_1 \vee A_2 \vee A_3) \wedge (A_4 \vee \neg A_1 \vee A_3)$ and the related variables X_{ij}. We introduce definitions enforcing the 1-in-3 rule:

$$\begin{array}{lll} Y_{1112} \equiv\equiv X_{11} \wedge X_{12} & Y_{1113} \equiv\equiv X_{11} \wedge X_{13} & Y_{1213} \equiv\equiv X_{12} \wedge X_{13}\,, \\ Y_{2122} \equiv\equiv X_{21} \wedge X_{22} & Y_{2123} \equiv\equiv X_{21} \wedge X_{23} & Y_{2223} \equiv\equiv X_{22} \wedge X_{23}\,, \end{array}$$

and appropriate assignments in $\mathbf{Q}$. We then guarantee that at most one of A_1 and $\neg A_1$ is true, by introducing $Y_{1122} \equiv X_{11} \wedge X_{22}$, and by adding $\{Y_{1122} = 0\}$ to $\mathbf{Q}$. Note that there are configurations that are not sensible but that satisfy the previous constraints: for instance, $\{L_{13} = L_{23} = 1, L_{11} = L_{12} = L_{21} = L_{22} = 0\}$ is not sensible and has probability $\alpha = (1-\varepsilon)^2\varepsilon^4$. To remove those configurations that are not sensible but that have "high" probability, we introduce:

$$
\begin{aligned}
Y_{1221} &\equiv X_{12} \wedge X_{21}\,, & Y_{1223} &\equiv X_{12} \wedge X_{23}\,,\\
Y_{1321} &\equiv X_{13} \wedge X_{21}\,, & Y_{1323} &\equiv X_{13} \wedge X_{23}\,,\\
Y_{1321} &\equiv X_{13} \wedge X_{21}\,, & Y_{1322} &\equiv X_{13} \wedge X_{22}\,,\\
Y_{1123} &\equiv X_{11} \wedge X_{23}\,, & Y_{1223} &\equiv X_{12} \wedge X_{23}\,,
\end{aligned}
$$

and we add $\{E_{1221} = 0, E_{1223} = 0, E_{1321} = 0, E_{1323} = 0, E_{1321} = 0, E_{1322} = 0, E_{1123} = 0, E_{1223} = 0\}$ to $\mathbf{Q}$. There are $2^6 = 64$ configurations of $X_{11}, \ldots, X_{23}$, and 15 of them have $X_{i1} = X_{i2} = X_{i3} = 0$ for $i = 1$ or $i = 2$ (or both). Among these ungratifying configurations, 8 do not respect the 1-in-3 rule; the remaining 7 that respect the 1-in-3 rule are assigned at most probability β. Among the 49 gratifying configurations (i.e., those that assign $X_{ij} = 1$ for some j for $i = 1, 2$), 40 do not respect the 1-in-3 rule. Of the remaining 9 configurations, 7 are not sensible. The last 2 configurations are assigned probability α. We thus have that

$$\mathbb{P}(\mathbf{Q}) = \sum_{x_{11},\ldots,x_{23}} \mathbb{P}(X_{11} = x_{11}, \ldots, X_{23} = x_{23}, \mathbf{Q}) \leq 2\alpha + 7\beta,$$

which implies that $(\varphi, 3)$ is not in $\#(\text{1-in-3})\mathsf{SAT}(>)$; indeed, there are $2 < 3$ assignments to A_1, A_2, A_3, A_4 that satisfy φ and respect the 1-in-3 rule.

This concludes our discussion of $\mathsf{INF}[\mathsf{Prop}(\wedge)]$, so we move to $\mathsf{INF}[\mathsf{Prop}(\vee)]$. To prove its PP-completeness, we must do almost exactly the same construction described before, with a few changes that we enumerate.

First, we associate each literal with a random variable X_{ij} as before, but now X_{ij} stands for a *negated* literal. That is, if the literal corresponding to X_{ij} is A and A is true, then $\{X_{ij} = 0\}$. Thus we must associate each X_{ij} with the assessment $\mathbb{P}(X_{ij} = 1) = \varepsilon$. Definitions must change accordingly: a configuration is now gratifying if $X_{i1} + X_{i2} + X_{i3} < 3$.

Second, the previous construction used a number of definition axioms of the form

$$Y_{iujv} \equiv X_{iu} \wedge X'_{jv},$$

with associated assessment $\{Y_{iujv} = 0\}$. We must replace each such pair by a definition axiom

$$Y_{iujv} \equiv X_{iu} \vee X'_{jv}$$

and an assessment $\{Y_{iujv} = 1\}$; recall that X_{iu} is just the negation of the variable used in the previous construction, so the overall effect of the constraints is the same.

All other arguments carry, so we obtain the desired hardness. □

It is instructive to look at a proof of Theorem 2 that uses Turing reductions, as it is much shorter than the previous proof:

Proof. To prove hardness of INF[Prop($\vee$)], we use the fact that the function #MON2SAT is #P-complete with respect to Turing reductions [130, Theorem 1]. Recall that #MON2SAT is the function that counts the number of satisfying assignments of a monotone sentence in 2CNF. So, we can take any MAJSAT problem where the input is sentence ϕ and produce (with polynomial effort) another sentence ϕ' in 2CNF such that $\#\phi$ is obtained from $\#\phi'$ (again with polynomial effort). And we can compute $\#\phi'$ using INF[Prop($\vee$)], as follows. Write ϕ' as $\bigwedge_{i=1}^{m}(A_{i_1} \vee A_{i_2})$, where each A_{i_j} is a proposition in $A_1, \ldots, A_n$. Introduce fresh propositions/variables C_i and definition axioms $C_i \equiv A_{i_1} \vee A_{i_2}$. Also introduce $\mathbb{P}(A_i = 1) = 1/2$ for each A_i, and consider the query $\mathbf{Q} = \{C_1, \ldots, C_m\}$. Note that $\mathbb{P}(\mathbf{Q}) > \gamma$ if and only if $\#\phi' = 2^n\mathbb{P}(\mathbf{Q}) > 2^n\gamma$, so we can bracket $\#\phi'$. From $\#\phi'$ we obtain $\#\phi$ and we can decide whether $\#\phi > 2^{n-1}$, thus solving the original MAJSAT problem.

To prove hardness of INF[Prop($\wedge$)], note that the number of satisfying assignments of ϕ' in 2CNF is equal to the number of satisfying assignments of $\bigwedge_{i=1}^{m}(\neg A_{i_1} \vee \neg A_{i_2})$, because one can take each satisfying assignment for the latter sentence and create a satisfying assignment for ϕ' by interchanging true and false, and likewise for the unsatisfying assignments. Introduce fresh propositions/variables C_i and definition axioms $C_i \equiv A_{i_1} \wedge A_{i_2}$. Also introduce $\mathbb{P}(A_i = 1) = 1/2$ for each A_i, and consider the query where $\mathbf{Q} = \{\neg C_1, \ldots, \neg C_m\}$. Again we can bracket the number of assignments that satisfy ϕ', and thus we can solve any MAJSAT problem by using INF[Prop($\wedge$)] and appropriate auxiliary polynomial computations. □

Appendix A.2. Relational languages (Section 5)

Theorem 4. INF[FFFO] *is* PEXP*-complete with respect to many-one reductions, regardless of whether the domain is specified in unary or binary notation.*

Proof. To prove membership, note that a relational Bayesian network specification based on FFFO can be grounded into an exponentially large Bayesian network, and inference can be carried out in that network using a counting Turing machine with an exponential bound on time. This is true even if we have unbounded arity of relations: even if we have domain size 2^N and maximum arity k, grounding each relation generates up to 2^{kN} nodes, still an exponential quantity in the input.

To prove hardness, we focus on binary domain size N as this simplifies the notation. Clearly if N is given in unary, then an exponential number of groundings can be produced by increasing the arity of relations (even if the domain is of size 2, an arity k leads to 2^k groundings).

Given the scarcity of PEXP-complete problems in the literature, we have to work directly from Turing machines. Start by taking any language $\mathcal{L}$ such that $\ell \in \mathcal{L}$ if and only ℓ is accepted by more than half of the computation paths of a nondeterministic Turing machine $\mathbb{M}$ within time $2^{p(|\ell|)}$ where p is a polynomial

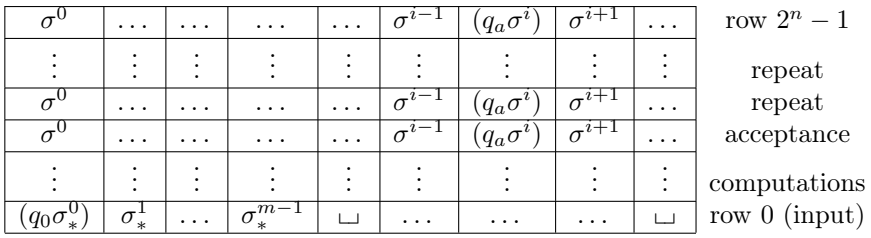


Figure A.14: An accepting computation.

and $|\ell|$ denotes the size of ℓ. To simplify matters, denote $p(|\ell|)$ by n. The Turing machine is defined by its alphabet, its states, and its transition function.

Denote by σ a symbol in $\mathbb{M}$'s alphabet, and by q a state of $\mathbb{M}$. A configuration of $\mathbb{M}$ can be described by a string $\sigma^0\sigma^1\ldots\sigma^{i-1}(q\sigma^i)\sigma^{i+1}\ldots\sigma^{2^n-1}$, where each σ^j is a symbol in the tape, $(q\sigma^i)$ indicates both the state q and the position of the head at cell i with symbol σ^i. The initial configuration is $(q_0\sigma^0)\sigma^1_*\ldots\sigma^{m-1}_*$ followed by 2^n-m blanks, where q_0 is the initial state. There are also states q_a and q_r that respectively indicate acceptance or rejection of the input string $\sigma^0_*\ldots\sigma^{m-1}_*$. We assume that if q_a or q_r appear in some configuration, then the configuration is not modified anymore (that is, the transition moves from this configuration to itself). This is necessary to guarantee that the number of accepting computations is equal to the number of ways in which we can fill in a matrix of computation. For instance, a particular accepting computation could be depicted as a $2^n\times 2^n$ matrix as in Figure A.14, where ␣ denotes the blank, and where we complete the rows of the matrix after the acceptance by repeating the accepting row.

The transition function δ of $\mathbb{M}$ takes a pair (q,σ) consisting of a state and a symbol in the machine's tape, and returns a triple (q',σ',m): the next state q', the symbol σ' to be written in the tape (we assume that a blank is never written by the machine), and an integer m in $\{-1,0,1\}$. Here -1 means that the head is to move left, 0 means that the head is to stay in the current cell, and 1 means that the head is to move right.

We now encode this Turing machine using monadic logic, mixing some ideas by Lewis [77] and by Tobies [124].

Take a domain of size 2^{2n}. The idea is that each $\mathcal{X}$ is a cell in the computation matrix. From now on, a "point" is a cell in that matrix. Introduce parvariables $X_0(\mathcal{X}),\ldots,X_{n-1}(\mathcal{X})$ and $Y_0(\mathcal{X}),\ldots,Y_{n-1}(\mathcal{X})$ to encode the index of the column and the row of point $\mathcal{X}$. Impose, for $0\le i\le n-1$, the assessments $\mathbb{P}(X_i(\mathcal{X})=1)=\mathbb{P}(Y_i(\mathcal{X})=1)=1/2$.

We need to specify the concept of adjacent points in the computation matrix. To this end we introduce two macros, $\mathsf{EAST}(\mathcal{X},\mathcal{Y})$ and $\mathsf{NORTH}(\mathcal{X},\mathcal{Y})$ (note that we do not actually need binary relations here; these expressions are just syntactic sugar). The meaning of $\mathsf{EAST}(\mathcal{X},\mathcal{Y})$ is that for point $\mathcal{X}$ there is a point $\mathcal{Y}$ that is

immediately to the right of $\mathcal{X}$; the meaning of $\mathsf{NORTH}(\mathcal{X}, y)$ is that for point $\mathcal{X}$ there is a point y that is immediately on top of $\mathcal{X}$ [124].

$$\begin{aligned}\mathsf{EAST}(\mathcal{X}, y) \quad := \quad & \bigwedge_{k=0}^{n-1} (\wedge_{j=0}^{k-1} X_j(\mathcal{X})) \rightarrow (X_k(\mathcal{X}) \leftrightarrow \neg X_k(y)) \\ & \wedge \bigwedge_{k=0}^{n-1} (\vee_{j=0}^{k-1} \neg X_j(\mathcal{X})) \rightarrow (X_k(\mathcal{X}) \leftrightarrow X_k(y)) \\ & \wedge \bigwedge_{k=0}^{n-1} (Y_k(\mathcal{X}) \leftrightarrow Y_k(y)).\end{aligned} \tag{A.3}$$

$$\begin{aligned}\mathsf{NORTH}(\mathcal{X}, y) \quad := \quad & \bigwedge_{k=0}^{n-1} (\wedge_{j=0}^{k-1} Y_j(\mathcal{X})) \rightarrow (Y_k(\mathcal{X}) \leftrightarrow \neg Y_k(y)) \\ & \wedge \bigwedge_{k=0}^{n-1} (\vee_{j=0}^{k-1} \neg Y_j(\mathcal{X})) \rightarrow (Y_k(\mathcal{X}) \leftrightarrow Y_k(y)) \\ & \wedge \bigwedge_{k=0}^{n-1} (X_k(\mathcal{X}) \leftrightarrow X_k(y)).\end{aligned} \tag{A.4}$$

In these expressions an empty conjunction means true and an empty disjunction means false.

Now introduce

$$Z_1 \equiv\equiv \left(\forall \mathcal{X} : \exists y : \mathsf{EAST}(\mathcal{X}, y)\right) \wedge \left(\forall \mathcal{X} : \exists y : \mathsf{NORTH}(\mathcal{X}, y)\right),$$

$$Z_2 \equiv\equiv \exists \mathcal{X} : \bigwedge_{k=0}^{n-1} (\neg X_k(\mathcal{X}) \wedge \neg Y_k(\mathcal{X})).$$

Now if $Z_1 \wedge Z_2$ is true, we "build" a square "board" of size $2^n \times 2^n$ (in fact this is a torus as the top row is followed by the bottom row, and the rightmost column is followed by the leftmost column). The intuition is that Z_2 guarantees the existence of an element in the "origin" and then Z_1 guarantees that neighbors exist to the "east" and to the "north".

Introduce a relation C_j for each triplet (α, β, γ) where each element of the triplet is either a symbol σ or a symbol of the form $(q\sigma)$ for our machine $\mathbb{M}$, and with an additional condition: if (α, β, γ) has β equal to a blank, then γ is a blank. Furthermore, introduce a relation C_j for each triple $(\diamond, \beta, \gamma)$, where β and γ are as before, and $\diamond$ is a new special symbol (these relations are needed later to encode the "left border" of the board). We refer to each C_j as a *tile*, as we are in effect encoding a domino system [77]. For each tile, impose $\mathbb{P}(C_j(\mathcal{X}) = 1) = 1/2$.

Now each point must have one and only one tile:

$$Z_3 \equiv\equiv \forall \mathcal{X} : \left(\bigvee_{j=0}^{c-1} C_j(\mathcal{X})\right) \wedge \left(\bigwedge_{0 \le j \le c-1, 0 \le k \le c-1, j \neq k} \neg (C_j(\mathcal{X}) \wedge C_k(\mathcal{X}))\right).$$

Having defined the tiles, we now define a pair of relations encoding the "horizontal" and "vertical" constraints on tiles, so as to encode the transition function of the Turing machine. Denote by H the relation consisting of pairs of tiles that satisfy the horizontal constraints and by V the relation consisting of pairs of tiles that satisfy the vertical constraints.

The horizontal constraints must enforce the fact that, in a fixed row, a tile (α, β, γ) at column i for $0 \leq i \leq 2^n - 1$ overlaps the tile $(\alpha', \beta', \gamma')$ at column $i + 1$ by satisfying

$$((\alpha, \beta, \gamma), (\alpha', \beta', \gamma')) : \beta = \alpha', \gamma = \beta'.$$

The vertical constraints must encode the possible computations. To do so, consider a tile $t = (\alpha, \beta, \gamma)$ at row j, for $0 \leq j \leq 2^n - 1$, and tile $t' = (\alpha', \beta', \gamma')$ at row $j + 1$, both at the same column. The pair (t, t') is in V if and only if (a) t' can be reached from t given the states in the Turing machine; and (b) if $t = (\diamond, \beta, \gamma)$, then $t' = (\diamond, \beta', \gamma')$ for β' and γ' that follow from the behavior of $\mathbb{M}$.

We distinguish the last row and the last column, as the transition function does not apply to them:

$$D_X(\mathcal{X}) \equiv\equiv \bigwedge_{k=0}^{n-1} X_k(\mathcal{X}), \qquad D_Y(\mathcal{X}) \equiv\equiv \bigwedge_{k=0}^{n-1} Y_k(\mathcal{X}).$$

We can now encode the transition function:

$$Z_4 \equiv\equiv \forall \mathcal{X} : \neg D_X(\mathcal{X}) \rightarrow \left(\bigwedge_{j=0}^{c-1} C_j(\mathcal{X}) \rightarrow (\forall y : \mathsf{EAST}(\mathcal{X}, y) \rightarrow \vee_{k:(j,k)\in H} C_k(y)) \right),$$

$$Z_5 \equiv\equiv \forall \mathcal{X} : \neg D_Y(\mathcal{X}) \rightarrow \left(\bigwedge_{j=0}^{c-1} C_j(\mathcal{X}) \rightarrow (\forall y : \mathsf{NORTH}(\mathcal{X}, y) \rightarrow \vee_{k:(j,k)\in V} C_k(y)) \right).$$

We create a parvariable that signals the accepting state:

$$Z_6 \equiv\equiv \exists \mathcal{X} : \bigvee_{j:C_j \text{contains } q_a} C_j(\mathcal{X}).$$

Finally, we must also impose the initial conditions. Take the tiles in the first row so that symbols in the input of $\mathbb{M}$ are encoded as m tiles, with the first tile $t^0 = (\diamond, (q_0\sigma_*^0), \sigma_*^1)$ and the following ones $t^j = (\sigma_*^{j-1}, \sigma_*^j, \sigma_*^{j+1})$ up to $t^{m-1} = (\sigma_*^{m-2}, \sigma_*^{m-1}, \sqcup)$. So the next tile will be $(\sigma_*^{m-1}, \sqcup, \sqcup)$, and after that all tiles in the first row will contain only blanks. Now take individuals a_i for $i \in \{0, \ldots, m-1\}$ and create an assignment $\{C_i^0(a_i) = 1\}$ for each a_i, where C_i^0 is the ith tile encoding the initial conditions. Denote by $\mathbf{E}$ the set of such assignments.

Now $\mathbb{P}\Big(Z_6 = 1 | \mathbf{E} \wedge \bigwedge_{i=1}^{5} \{Z_i = 1\}\Big) > 1/2$ if and only if the number of correct arrangments of tiles that contain the accepting state is larger than the total

number of possible valid arrangements. Hence an inference with the constructed relational Bayesian network specification decides the language $\mathcal{L}$ we started with, as desired. □

Theorem 5. INF[ALC] *is* PEXP*-complete with respect to many-one reductions when domain size is given in binary notation.*

Proof. Membership follows from Theorem 4.

To prove hardness, we simplify a previous proof by Cozman and Polastro [27], using as much as possible the construction in the proof of Theorem 4. So assume we have the input ℓ and the Turing machine $\mathbb{M}$ as described in that proof, as well as the domain of size 2^{2n}. Assume also that we have the parvariables $X_0(\mathcal{X}), \ldots, X_{n-1}(\mathcal{X}), Y_0(\mathcal{X}), \ldots, Y_{n-1}(\mathcal{X})$, together with their associated assessments $\mathbb{P}(X_i(\mathcal{X}) = 1) = \mathbb{P}(Y_i(\mathcal{X}) = 1) = 1/2$, and the parvariables $C_j(\mathcal{X})$ with their associated assessments $\mathbb{P}(C_j(\mathcal{X}) = 1) = 1/2$. Then note that, using the arguments around Expression (5), we can "create" quantification patterns such as $\exists y : X(\mathcal{X}, y)$ and $\exists y : Y(y)$ with probability one. We can use these patterns to write *all* expressions defining $Z_1, Z_2, \ldots, Z_6$ in the proof of Theorem 4, *except* for the fact that macros EAST and NORTH (respectively Expressions (A.3) and (A.4)) cannot be written directly with ALC. Thus we could, *except* for the macros EAST and NORTH, repeat here the proof of Theorem 4, at the end making our desired decision $\mathbb{P}\Big(Z_6 = 1 | \mathbf{E} \wedge \bigwedge_{i=1}^{5} \{Z_i = 1\}\Big) > 1/2$.

To reproduce the behavior of the macros EAST and NORTH, we resort to techniques developed by Tobies [124]. First we introduce two binary relations, $\mathsf{east}(\mathcal{X}, y)$ and $\mathsf{north}(\mathcal{X}, y)$; these relations will be forced to behave as the missing macros. The groundings of east and north will be set to specific values given the evidence on $Z_1, \ldots, Z_5$, but to complete the specification we just introduce $\mathbb{P}\big(\mathsf{east}(\mathcal{X}, y) = 1\big) = 1/2$ and $\mathbb{P}\big(\mathsf{north}(\mathcal{X}, y) = 1\big) = 1/2$. Then introduce:

$$
\begin{aligned}
Z_7(\mathcal{X}) \quad \equiv \quad & \bigwedge_{k=0}^{n-1} (\wedge_{j=0}^{k-1} X_j(\mathcal{X})) \rightarrow \left(\begin{array}{c} X_k(\mathcal{X}) \rightarrow \forall y : \mathsf{east}(\mathcal{X}, y) \rightarrow \neg X_k(y) \\ \wedge \\ \neg X_k(\mathcal{X}) \rightarrow \forall y : \mathsf{east}(\mathcal{X}, y) \rightarrow X_k(y) \end{array} \right) \\
& \wedge \bigwedge_{k=0}^{n-1} (\vee_{j=0}^{k-1} \neg X_j(\mathcal{X})) \rightarrow \left(\begin{array}{c} X_k(\mathcal{X}) \rightarrow \forall y : \mathsf{east}(\mathcal{X}, y) \rightarrow X_k(y) \\ \wedge \\ \neg X_k(\mathcal{X}) \rightarrow \forall y : \mathsf{east}(\mathcal{X}, y) \rightarrow \neg X_k(y) \end{array} \right) \\
& \wedge \bigwedge_{k=0}^{n-1} \left(\begin{array}{c} Y_k(\mathcal{X}) \rightarrow \forall y : \mathsf{east}(\mathcal{X}, y) \rightarrow Y_k(y) \\ \wedge \\ \neg Y_k(\mathcal{X}) \rightarrow \forall y : \mathsf{east}(\mathcal{X}, y) \rightarrow \neg Y_k(y) \end{array} \right),
\end{aligned}
$$

$$
\begin{aligned}
Z_8(\mathcal{X}) \quad \equiv \quad & \bigwedge_{k=0}^{n-1} (\wedge_{j=0}^{k-1} Y_j(\mathcal{X})) \rightarrow \left(\begin{array}{c} Y_k(\mathcal{X}) \rightarrow \forall y : \mathsf{north}(\mathcal{X}, y) \rightarrow \neg Y_k(y) \\ \wedge \\ \neg Y_k(\mathcal{X}) \rightarrow \forall y : \mathsf{north}(\mathcal{X}, y) \rightarrow Y_k(y) \end{array} \right) \\
& \wedge \bigwedge_{k=0}^{n-1} (\vee_{j=0}^{k-1} \neg Y_j(\mathcal{X})) \rightarrow \left(\begin{array}{c} Y_k(\mathcal{X}) \rightarrow \forall y : \mathsf{north}(\mathcal{X}, y) \rightarrow Y_k(y) \\ \wedge \\ \neg Y_k(\mathcal{X}) \rightarrow \forall y : \mathsf{north}(\mathcal{X}, y) \rightarrow \neg Y_k(y) \end{array} \right) \\
& \wedge \bigwedge_{k=0}^{n-1} \left(\begin{array}{c} X_k(\mathcal{X}) \rightarrow \forall y : \mathsf{north}(\mathcal{X}, y) \rightarrow X_k(y) \\ \wedge \\ \neg X_k(\mathcal{X}) \rightarrow \forall y : \mathsf{north}(\mathcal{X}, y) \rightarrow \neg X_k(y) \end{array} \right).
\end{aligned}
$$

Now replace EAST by east and NORTH by north in the definitions of Z_1, Z_4 and Z_5, and introduce

$$Z_9 = \forall \mathcal{X} : Z_7(\mathcal{X}) \wedge Z_8(\mathcal{X}).$$

If $Z_1 \wedge Z_2 \wedge Z_9$ is true, we 'build" a torus as in the proof of Theorem 4; this is a consequence of the fact that each one of the 2^{2n} cells of the torus must contain at least an element of the domain [124, Lemma 3.5], and the fact that we have exactly 2^{2n} elements in the domain. Hence we produce the desired decision by $\mathbb{P}\Big(Z_6 = 1 | \{Z_9 = 1\} \wedge \mathbf{E} \wedge \bigwedge_{i=1}^{5} \{Z_i = 1\}\Big) > 1/2$. □

Theorem 6. QINF[FFFO] *is* PEXP*-complete with respect to many-one reductions when domain size is specified in binary notation.*

Proof. Membership is obvious as the inferential complexity is already in PEXP. To show hardness, take a Turing machine $\mathbb{M}$ that solves some PEXP-complete problem within 2^n steps, where n is polynomial on the size of the input ℓ (as in the proof of Theorem 4). That is, there is a PEXP-complete language $\mathcal{L}$ such that $\ell \in \mathcal{L}$ if and only if the input string ℓ is accepted by more than half of the computation paths of $\mathbb{M}$ within time 2^n.

Such a Turing machine $\mathbb{M}$ has alphabet, states and transitions as in the proof of Theorem 4. Assume that $\mathbb{M}$ repeats its configuration as soon as it enters into the accepting or the rejecting state, as in the proof of Theorem 4.

To encode $\mathbb{M}$ we resort to a construction by Grädel [52] where relations of arity two are used. We use: (a) for each state q of $\mathbb{M}$, a unary relation X_q; (b) for each symbol σ in the alphabet of $\mathbb{M}$, a binary relation Y_σ; (c) a binary relation Z. The idea is that $X_q(\mathcal{X})$ means that $\mathbb{M}$ is in state q at computation step $\mathcal{X}$, while $Y_\sigma(\mathcal{X}, y)$ means that σ is the symbol at the yth position in the tape at computation step $\mathcal{X}$, and $Z(\mathcal{X}, y)$ means that the machine head is at the yth position in the tape at computation step $\mathcal{X}$. Impose $\mathbb{P}(X_q(\mathcal{X}) = 1) = \mathbb{P}\big(Y_\sigma(\mathcal{X}, y) = 1\big) = \mathbb{P}\big(Z(\mathcal{X}, y) = 1\big) = 1/2$.

We use a distinguished relation $<$, assumed not to be in the vocabulary. This relation is to be a linear order on the domain; to obtain this behavior, just introduce $\mathbb{P}\big(<(\mathcal{X}, y) = 1\big) = 1/2$ and

$$
\begin{aligned}
Z_1 \quad \equiv \quad & (\forall \mathcal{X} : \neg(\mathcal{X} < \mathcal{X})) \wedge \\
& (\forall \mathcal{X} : \forall y : \forall z : (\mathcal{X} < y \wedge y < z) \rightarrow \mathcal{X} < z) \wedge \\
& (\forall \mathcal{X} : \forall y : (\mathcal{X} < y) \vee (y < \mathcal{X}) \vee (\mathcal{X} = y)).
\end{aligned}
$$

We will later set evidence on Z_1 to force $<$ to be a linear order. The important point is that we can assume that a domain of size 2^n is given and all elements of this domain are ordered according to $<$.

Clearly we can define a successor relation using $<$:

$$\mathsf{successor}(\mathcal{x}, y) \equiv (\mathcal{x} < y) \wedge \left(\neg\exists z : (\mathcal{x} < z) \wedge (z < y)\right).$$

Also, we can define a relation that signals the "first" individual:

$$\mathsf{first}(\mathcal{x}) \equiv \neg\exists y : y < \mathcal{x}.$$

We must guarantee that at any given step the machine is in a single state, each cell of the tape has a single symbol, and the head is at a a single position of the tape:

$$Z_2 \equiv \forall \mathcal{x} : \bigvee_q \left(X_q(\mathcal{x}) \wedge \bigwedge_{q' \neq q} \neg X_{q'}(\mathcal{x}) \right),$$

$$Z_3 \equiv \forall \mathcal{x} : \forall y : \bigvee_\sigma \left(Y_\sigma(\mathcal{x}, y) \wedge \bigwedge_{\sigma' \neq \sigma} \neg Y_{\sigma'}(\mathcal{x}, y) \right),$$

$$Z_4 \equiv \forall \mathcal{x} : (\exists y : Z(\mathcal{x}, y) \wedge \forall z : (z \neq y) \rightarrow \neg Z(\mathcal{x}, z)).$$

We also have to guarantee that computations do not change the content of a cell that is not visited by the head:

$$Z_5 \equiv \forall \mathcal{x} : \forall y : \forall z : \bigwedge_\sigma \left(\neg Z(\mathcal{x}, y) \wedge Y_\sigma(\mathcal{x}, y) \wedge \mathsf{successor}(\mathcal{x}, z)\right) \rightarrow Y_\sigma(z, y).$$

We must encode the changes made by the transition function:

$$\begin{aligned} Z_6 \equiv\ & \forall \mathcal{x} : \forall y : \forall z : \bigwedge_{q,\sigma} \left(Z(\mathcal{x}, y) \wedge Y_\sigma(\mathcal{x}, y) \wedge X_q(\mathcal{x}) \wedge \mathsf{successor}(\mathcal{x}, z)\right) \rightarrow \\ & \bigvee_{(q'\sigma',1)\in\delta(q,\sigma)} \left(X_{q'}(z) \wedge Y_{\sigma'}(z, y) \wedge (\forall w : \mathsf{successor}(y, w) \rightarrow Z(z, w))\right) \\ & \vee \bigvee_{(q'\sigma',0)\in\delta(q,\sigma)} \left(X_{q'}(z) \wedge Y_{\sigma'}(z, y) \wedge Z(z, y)\right) \\ & \vee \bigvee_{(q'\sigma',1)\in\delta(q,\sigma)} \left(X_{q'}(z) \wedge Y_{\sigma'}(z, y) \wedge (\forall w : \mathsf{successor}(w, y) \rightarrow Z(z, w))\right). \end{aligned}$$

We must also guarantee that all cells to the right of a blank cell are also blank:

$$Z_7 \equiv \forall \mathcal{x} : \forall y : \forall z : Y_\sqcup(\mathcal{x}, y) \wedge \mathsf{successor}(y, z) \rightarrow Y_\sqcup(\mathcal{x}, z).$$

Finally, we must signal the accepting state:

$$Z_8 \equiv \exists \mathcal{x} : X_{q_a}(\mathcal{x}).$$

We have thus created a set of formulas that encode the behavior of the Turing machine. Now take the input string ℓ, equal to $\sigma_*^0, \sigma_*^1, \ldots, \sigma_*^{m-1}$, and encode it as a query as follows. Start by "creating" the first individual in the ordering by taking the assignment $\{\mathsf{first}(a_0) = 1\}$. Then introduce $\{Z(a_0, a_0) = 1\}$ to initialize the head. Introduce $\{Y_{\sigma_*^0}(a_0, a_0) = 1\}$ to impose the initial condition on the first cell, and for each subsequent initial condition σ_*^i we set $\{Y_{\sigma_*^i}(a_0, a_i) = 1\}$ and $\{\mathsf{successor}(a_{i-1}, a_i) = 1\}$ where a_i is a fresh individual. Finally, set $\{Y_{\sqcup}(a_0, a_m) = 1\}$ and $\{\mathsf{successor}(a_{m-1}, a_m) = 1\}$ and $\{X_{q_0}(a_0) = 1\}$. These assignments are denoted by $\mathbf{E}$.

Now $\mathbb{P}\Big(Z_8 = 1 | \mathbf{E} \wedge \bigwedge_{i=1}^{7} \{Z_i = 1\}\Big) > 1/2$ for a domain of size 2^n if and only if the number of interpretations reaching the accepting state is larger than the total number of possible interpretations encoding computation paths. □

Theorem 7. QINF[FFFO] *is* PP*-complete with respect to many-one reductions when domain size is specified in unary notation.*

Proof. To prove hardness, take a MAJSAT problem where ϕ is in CNF with m clauses and propositions $A_1, \ldots, A_n$. Make sure $m = n$: if $m < n$, then add trivial clauses such as $A_1 \vee \neg A_1$; if instead $n < m$, then add fresh propositions $A_{n+1}, \ldots, A_m$. These changes do not change the output of MAJSAT. Introduce unary relations $\mathsf{sat}(\mathcal{X})$ and impose $\mathbb{P}(\mathsf{sat}(\mathcal{X})) = 1/2$. Take a domain $\{\mathtt{1}, \ldots, \mathtt{n}\}$; the elements of the domain serve a dual purpose, indexing both propositions and clauses. Introduce relations $\mathsf{sat}(\mathcal{X})$, $\mathsf{positiveLit}(\mathcal{X}, y)$ and $\mathsf{negativeLit}(\mathcal{X}, y)$, assessments $\mathbb{P}(\mathsf{sat}(\mathcal{X}) = 1) = \mathbb{P}\big(\mathsf{positiveLit}(\mathcal{X}, y) = 1\big) = \mathbb{P}\big(\mathsf{negativeLit}(\mathcal{X}, y) = 1\big) = 1/2$, and definition axioms:

$$\begin{aligned} \mathsf{clause}(\mathcal{X}) \quad &\equiv \quad \exists y : (\mathsf{positiveLit}(\mathcal{X}, y) \wedge \mathsf{sat}(y)) \vee \\ &\qquad \exists y : (\mathsf{negativeLit}(\mathcal{X}, y) \wedge \neg \mathsf{sat}(y)), \qquad \text{(A.5)} \\ \mathsf{query} \quad &\equiv \quad \forall \mathcal{X} : \mathsf{clause}(\mathcal{X}). \qquad \text{(A.6)} \end{aligned}$$

Take evidence $\mathbf{E}$ as follows. For each clause, run over the literals. Consider the ith clause, and its non-negated literal A_j: set $\mathsf{positiveLit}(i, j)$ to true. And consider negated literal $\neg A_j$: set $\mathsf{negativeLit}(i, j)$ to true. Set all other groundings of $\mathsf{positiveLit}$ and $\mathsf{negativeLit}$ to false. Note that $\mathbb{P}(\mathbf{E}) = 2^{-2n^2} > 0$. Now decide whether $\mathbb{P}(\mathsf{query} = 1 | \mathbf{E}) > 1/2$. If YES, the MAJSAT problem is accepted, if NO, it is not accepted. Hence we have the desired polynomial reduction (the query is quadratic on domain size; all other elements are linear on domain size).

To prove membership in PP, we describe a Turing machine $\mathbb{M}$ that decides whether $\mathbb{P}(Q|\mathbf{E}) > \gamma$. The machine guesses a truth assignment for each one of the polynomially-many grounded root nodes (and writes the guess in the working tape). Note that each grounded root node X is associated with an assessment $\mathbb{P}(X = 1) = c/d$, where c and d are the smallest such integers. Then the machine replicates its computation paths out of the guess on X: there are c paths with identical behavior for guess $\{X = 1\}$, and $d - c$ paths with identical behavior for guess $\{X = 0\}$.

Now the machine verifies whether the set of guessed truth assignment satisfies $\mathbf{E}$; if not, move to state q_1. If yes, then verify whether the guessed truth assignment fails to satisfy $\mathbf{Q}$; if not, move to state q_2. And if yes, then move to state q_3. The key point is that there is a logarithmic space, hence polynomial-time, algorithm that can verifiy whether a set of assignments holds once the root nodes are set [78, Section 6.2].

Suppose that out of N computation paths that $\mathbb{M}$ can take, N_1 of them reach q_1, N_2 reach q_2, and N_3 reach q_3. By construction,

$$N_1/N = 1 - \mathbb{P}(\mathbf{E})\,, \quad N_2/N = \mathbb{P}(\neg\mathbf{Q}, \mathbf{E})\,, \quad N_3/N = \mathbb{P}(\mathbf{Q}, \mathbf{E})\,, \tag{A.7}$$

where we abuse notation by taking $\neg\mathbf{Q}$ to mean that some assignment in $\mathbf{Q}$ is not true. Note that up to this point we do not have any rejecting nor accepting path, so the specification of $\mathbb{M}$ is not complete.

The remainder of this proof just reproduces a construction by Park in his proof of PP-completeness for propositional Bayesian networks [33, Theorem 11.5]. Park's construction adds rejecting/accepting computation paths emanating from q_1, q_2 and q_3. It uses numbers

$$a = \begin{cases} 1 & \text{if } \gamma < 1/2, \\ 1/(2\gamma) & \text{otherwise,} \end{cases} \qquad b = \begin{cases} (1-2\gamma)/(2-2\gamma) & \text{if } \gamma < 1/2, \\ 0 & \text{otherwise,} \end{cases}$$

and the smallest integers a_1, a_2, b_1, b_2 such that $a = a_1/a_2$ and $b = b_1/b_2$. Now, out of q_1 branch into a_2b_2 computation paths that immediately stop at the accepting state, and a_2b_2 computation paths that immediately stop at the rejecting state.[8] Out of q_2 branch into $2a_2b_1$ paths that immediately stop at the accepting state, and $2(b_2 - b_1)a_2$ paths that immediately stop at the rejecting state. Out of q_3 branch into $2a_1b_2$ paths that immediately stop at the accepting state, and $2(a_2 - a_1)b_2$ paths that immediately stop at the rejecting state. For the whole machine $\mathbb{M}$, the number of computation paths that end up at the accepting state is $a_2b_2N_1 + 2a_2b_1N_2 + 2a_1b_2N_3$, and the total number of computation paths is $a_2b_2N_1 + a_2b_2N_1 + 2b_1a_2N_2 + 2(b_2 - b_1)a_2N_2 + 2a_1b_2N_3 + 2(a_2 - a_1)b_2N_3 = 2a_2b_2N$. Hence the number of accepting paths divided by the total number of paths is $(N_1(1/2) + (b_1/b_2)N_2 + (a_1/a_2)N_3)/N$. By combining this construction with Expression (A.7), we obtain

$$\frac{\frac{N_1}{2} + \frac{b_1N_2}{b_2} + \frac{a_1N_3}{a_2}}{N} > 1/2 \quad \Leftrightarrow \quad \frac{1 - \mathbb{P}(\mathbf{E})}{2} + b\mathbb{P}(\neg\mathbf{Q}, \mathbf{E}) + a\mathbb{P}(\mathbf{Q}, \mathbf{E}) > 1/2$$

$$\Leftrightarrow \quad a\mathbb{P}(\mathbf{Q}, \mathbf{E}) + b\mathbb{P}(\neg\mathbf{Q}, \mathbf{E}) > \mathbb{P}(\mathbf{E})/2 \quad \Leftrightarrow \quad a\mathbb{P}(\mathbf{Q}|\mathbf{E}) + b\mathbb{P}(\neg\mathbf{Q}|\mathbf{E}) > 1/2,$$

[8] The number of created paths may be exponential in the numbers $a2$ and b_2; however it is always possible to construct a polynomial sequence of steps that encodes an exponential number of paths (say the number of paths has B bits; then build B distinct branches, each one of them multiplying alternatives so as to simulate an exponential). This sort of branching scheme is also assumed whenever needed.

as we can assume that $\mathbb{P}(\mathbf{E}) > 0$ (otherwise the number of accepting paths is equal to the number of rejecting paths), and then

$$\begin{cases} \text{if } \gamma < 1/2: & \mathbb{P}(\mathbf{Q}|\mathbf{E}) + \frac{1-2\gamma}{2-2\gamma}(1 - \mathbb{P}(\mathbf{Q}|\mathbf{E})) > 1/2 & \Leftrightarrow \mathbb{P}(\mathbf{Q}|\mathbf{E}) > \gamma; \\ \text{if } \gamma \geq 1/2: & (1/(2\gamma))\mathbb{P}(\mathbf{Q}|\mathbf{E}) > 1/2 & \Leftrightarrow \mathbb{P}(\mathbf{Q}|\mathbf{E}) > \gamma. \end{cases}$$

Hence the number of accepting computation paths of $\mathbb{M}$ is larger than half the total number of computation paths if and only if $\mathbb{P}(\mathbf{Q}|\mathbf{E}) > \gamma$. This completes the proof of membership. □

Theorem 8. *Suppose* NETIME $\neq$ ETIME. *Then* DINF[FFFO] *is not solved in deterministic exponential time when domain size is given in binary notation.*

Proof. Jaeger describes results implying, in case NETIME $\neq$ ETIME, that there is a sentence $\phi \in$ FFFO such that the *spectrum* of ϕ cannot be recognized in deterministic exponential time [63]. Recall: the spectrum of a sentence is a set containing each integer N, in binary notation, such that ϕ has a model whose domain size is N [52]. So take N in binary notation, the relational Bayesian network specification $A \equiv\!\equiv \phi$, and decide whether $\mathbb{P}(A) > 0$ for domain size N; if yes, then N is in the spectrum of ϕ. □

Theorem 9. DINF[FFFO] *is* PP_1*-complete with respect to many-one reductions when domain size is given in unary notation.*

Proof. To prove membership, just consider the Turing machine used in the proof of Theorem 7, now with a fixed query. This is a polynomial-time nondeterministic Turing machine that gets the domain size in unary (that is, as a sequence of 1s) and produces the desired output.

To prove hardness, take a Turing machine with input alphabet consisting of symbol 1, and that solves a PP_1-complete problem in N^m steps for input consisting of N symbols 1. Take the probabilistic assessment and the definition axioms for successor, first, and Z_1 as in the proof of Theorem 6. Now introduce relations X_q, Y_σ and Z as in that proof, with the difference that x is substituted for m logvars x_i, and likewise y is substituted for m logvars y_j. For instance, we now have $Z(x_1, \ldots, x_m, y_1, \ldots, y_m)$. Repeat definition axioms for $Z_2, \ldots, Z_8$ as presented in the proof of Theorem 6, with appropriate changes in the arity of relations. In doing so we have an encoding for the Turing machine where the computation steps are indexed by a vector $[x_1, \ldots, x_m]$, and the tape is indexed by a vector $[y_1, \ldots, y_m]$. The remaining problem is to insert the input. To do so, introduce:

$$\begin{aligned} Z' \quad \equiv\!\equiv \quad & \forall x : \forall y_1 : \ldots \forall y_m : \mathsf{first}(x) \rightarrow \\ & \left(\bigwedge_{i \in \{2,\ldots,m\}} \mathsf{first}(y_i) \rightarrow Y_1(\overbrace{x, \ldots, x}^{m \text{ logvars}}, y_1, \ldots, y_m) \right) \\ & \wedge \left(\neg \bigwedge_{i \in \{2,\ldots,m\}} \mathsf{first}(y_i) \rightarrow Y_\sqcup(\overbrace{x, \ldots, x}^{m \text{ logvars}}, y_1, \ldots, y_m) \right). \end{aligned}$$

Now $\mathbb{P}\Big(Z_8 = 1 | \{Z' = 1\} \wedge \bigwedge_{i=1}^{7} \{Z_i = 1\}\Big) > 1/2$ for a domain of size N if and only if the number of interpretations that set an accepting state to true is larger than half the total number of interpretations encoding computation paths. □

Theorem 10. $\mathsf{INF}[\mathsf{FFFO}]$ *is* PSPACE*-complete with respect to many-one reductions when relations have bounded arity and domain size is given in unary notation.*

Proof. To prove membership, construct a Turing machine that goes over the truth assignments for all of the polynomially-many grounded root nodes. The machine generates an assignment, writes it using polynomial space, and verifies whether $\mathbf{E}$ can be satisfied: there is a polynomial-space algorithm to do this, as we basically need to do model checking in first-order logic [52, Section 3.1.4]. While cycling through truth assignments, keep adding the probabilities of the truth assignments that satisfy $\mathbf{E}$. If the resulting probability for $\mathbf{E}$ is zero, reject; otherwise, again go through every truth assignment of the root nodes, now keeping track of how many of them satisfy $\{\mathbf{Q}, \mathbf{E}\}$, and adding the probabilities for these assignments. Then divide the probability of $\{\mathbf{Q}, \mathbf{E}\}$ by the probability of $\mathbf{E}$, and compare the result with the rational number γ.

To show hardness, consider the definition axiom $Y \equiv\!\equiv Q_1 x_1 : \ldots Q_n x_n : \phi(x_1, \ldots, x_n)$, where each Q_i is a quantifier (either $\forall$ or $\exists$) and ϕ is a quantifier-free formula containing only Boolean operators, a unary relation X, and logvars $x_1, \ldots, x_n$. The relation X is associated with assessment $\mathbb{P}(X(\chi) = 1) = 1/2$. Take domain $\mathcal{D} = \{\mathtt{0}, \mathtt{1}\}$ and evidence $\mathbf{E} = \{X(\mathtt{0}) = 0, X(\mathtt{1}) = 1\}$. Then $\mathbb{P}(Y = 1 | \mathbf{E}) > 1/2$ if and only if $Q_1 x_1 : \ldots Q_n x_n : \phi(x_1, \ldots, x_n)$ is satisfiable. Deciding the latter satisfiability question is in fact equivalent to deciding the satisfiability of a Quantified Boolean Formula, a PSPACE-complete problem [78, Section 6.5]. □

Now consider the bounded variable fragment FFFO^k. It is important to notice that if the body of every definition axiom belongs to FFFO^k for an integer k, then all definition axioms together are equivalent to a single formula in FFFO^k. Hence results on logical inference for FFFO^k can be used to derive inferential, query and domain complexities.

Theorem 11. $\mathsf{INF}[\mathsf{FFFO}^k]$ *is* PP*-complete with respect to many-one reductions for all* $k \geq 0$ *when domain size is given in unary notation.*

Proof. Hardness is trivial: even $\mathsf{Prop}(\wedge, \neg)$ is PP-hard. To prove membership, use the Turing machine described in the proof of membership in Theorem 7, with a small difference: when it is necessary to check whether $\mathbf{E}$ (or $\mathbf{Q} \cup \mathbf{E}$) holds given a guessed assignment for root nodes, use the appropriate model checking algorithm [135], as this verification can be done in polynomial time. □

Theorem 12. $\mathsf{QINF}[\mathsf{FFFO}^k]$ *is* PP*-complete with respect to many-one reductions for all* $k \geq 2$ *when domain size is given in unary notation.*

Proof. To prove membership, note that QINF[FFFO] is in PP by Theorem 7. To prove hardness, note that the proof of hardness in Theorem 7 uses only FFFO^2. □

Theorem 13. $\mathsf{DINF}[\mathsf{FFFO}^k]$ *is* PP_1*-complete with respect to many-one reductions for* $k > 2$ *and polynomial for* $k \leq 2$ *when domain size is given in unary notation.*

Proof. For $k \leq 2$, results in the literature show how to count the number of satisfying models of a formula in polynomial time [132, 134].

For $k > 2$, membership is shown as the proof of Theorem 9. Hardness has been in essence proved by Beame et al. [8, Lemmas 3.8, 3.9]; our only contribution is to simplify their arguments by removing the need to enumerate the counting Turing machines. Take a Turing machine $\mathbb{M}$ that solves a $\#\mathsf{P}_1$-complete problem in N^m steps for an input consisting of N ones. By padding the input, we can always guarantee that $\mathbb{M}$ runs in time linear in the input. To show this, consider that for the input sequence with N ones, we can generate another sequence $S(N)$ consisting of $f(N) = (2N+1)2^{m\lceil \log_2 N \rceil}$ ones. Because $(2^{1+\log_2 N})^m \geq 2^{m\lceil \log_2 N \rceil}$, we have $(2N+1)2^m N^m > f(N)$, and consequently $S(N)$ can be generated in polynomial time. Modify $\mathbb{M}$ so that the new machine proceeds as follows:
(a) it receives $S(N)$;
(b) in linear time it produces the binary representation of $S(N)$, using an auxiliary tape;[9]
(c) it then discards the trailing zeroes to obtain $2N+1$;
(d) it computes N;
(e) it writes N ones in its tape;
(f) and then it runs the original computation in $\mathbb{M}$.
Because $2^{m\lceil \log_2 N \rceil} \geq N^m$, we have $f(N) > N^m$, and consequently the new machine runs in time that is overall linear in the input size $f(N)$, and in space within $f(N)$. Suppose, to be concrete, that the new machine runs in time that is smaller than $Mf(N)$ for some integer M. We just have to encode this machine in FFFO^3, by reproducing a clever construction due to Beame et al. [8]. We now briefly replay that construction in a simplified form.

We use the Turing machine encoding described in the proof of Theorem 8, but instead of using a single relation $Z(\chi, y)$ to indicate the head position at

[9] For instance: go from left to right replacing pairs 11 by new symbols ♣♡; if a blank is reached in the middle of such a pair, then add a 1 at the first blank in the auxiliary tape, and if a blank is reached after such a pair, then add a 0 at the first blank in the auxiliary tape; then mark the current end of the auxiliary tape with a symbol ♠ and return from the end of the main tape, erasing it and adding a 1 to the end of the auxiliary tape for each ♡ in the main tape; now copy the 1s after ♠ from the auxiliary tape to the main tape (and remove these 1s from the auxiliary tape), and repeat. Each iteration has cost smaller than $(U + U + \log U)c$ for some constant c, where U is the number of ones in the main tape; thus the total cost from input of size $f(N)$ is smaller than $3c(f(N) + f(N)/2 + f(N)/4 + \dots) \leq 6cf(N)$.

step $\mathcal{X}$, we use

$$Z^{1,1}(\mathcal{X}, y), \ldots, Z^{M,1}(\mathcal{X}, y), Z^{1,2}(\mathcal{X}, y), \ldots, Z^{M,2}(\mathcal{X}, y),$$

with the understanding that for a fixed $\mathcal{X}$ we have that $Z^{i,j}(\mathcal{X}, y)$ yields the position y of the head in step $\mathcal{X}$ and sub-step i, either in the main tape (tape 1) or in the auxiliary tape (tape 2). So, $Z^{1,j}$ is followed by $Z^{2,j}$ and so on until $Z^{M,j}$ for a fixed step $\mathcal{X}$. Similarly, we use $X_q^t(\mathcal{X})$, $Y_\sigma^{t,1}(\mathcal{X}, y)$ and $Y_\sigma^{t,2}(\mathcal{X}, y)$ for $t \in \{1, \ldots, M\}$. Definition axioms must be changed accordingly; for instance, we have

$$Z_2 \equiv\equiv \forall \mathcal{X} : \bigwedge_t \bigvee_q \left(X_q^t(\mathcal{X}) \wedge \bigwedge_{q' \neq q} \neg X_{q'}^t(\mathcal{X}) \right),$$

and

$$Z_3 \equiv\equiv \forall \mathcal{X} : \bigwedge_t \forall y : \bigwedge_{j \in \{1,2\}} \bigvee_\sigma \left(Y_\sigma^{t,j}(\mathcal{X}, y) \wedge \bigwedge_{\sigma' \neq \sigma} \neg Y_{\sigma'}^{t,j}(\mathcal{X}, y) \right).$$

As another example, we can change Z_4 as follows. First, introduce auxiliary definition axioms:

$$W_1^t(\mathcal{X}) \equiv\equiv \exists y : Z^{t,1}(\mathcal{X}, y) \wedge (\forall z : (z \neq y) \rightarrow \neg Z^{t,1}(\mathcal{X}, z)) \wedge (\forall z : \neg Z^{t,2}(\mathcal{X}, z)),$$

$$W_2^t(\mathcal{X}) \equiv\equiv \exists y : Z^{t,2}(\mathcal{X}, y) \wedge (\forall z : (z \neq y) \rightarrow \neg Z^{t,2}(\mathcal{X}, z)) \wedge (\forall z : \neg Z^{t,1}(\mathcal{X}, z)),$$

and then write:

$$Z_4 \equiv\equiv \forall \mathcal{X} : \bigwedge_t W_1^t(\mathcal{X}) \wedge W_2^t(\mathcal{X}).$$

Similar changes must be made to Z_7:

$$Z_7 \equiv\equiv \forall \mathcal{X} : \bigwedge_t \forall y : \bigwedge_{j \in \{1,2\}} \forall z : Y_\sqcup^{t,j}(\mathcal{X}, y) \wedge \mathsf{successor}(y, z) \rightarrow Y_\sqcup^{t,j}(\mathcal{X}, z).$$

The changes to Z_5 and Z_6 are similar, but require more tedious repetition; we omit the complete expressions but explain the procedure. Basically, Z_5 and Z_6 encode the transitions of the Turing machine. So, instead of just taking the successor of a computation step $\mathcal{X}$, we must operate in substeps: the successor of step $\mathcal{X}$ substep t is $\mathcal{X}$ substep $t + 1$, unless $t = M$ (in which case we must move to the successor of $\mathcal{X}$, substep 1). We can also capture the behavior of the Turing machine with two transition functions, one per tape, and it is necessary to encode each one of them appropriately. It is enough to have M different versions of Z_5 and $2M$ different versions of Z_6, each one of them responsible for one particular substep transition.

We must then encode the initial conditions. Introduce:

$$\mathsf{last}(\mathcal{X}) \equiv\equiv \neg \exists y : \mathcal{X} < y$$

and

$$\begin{aligned} Z_8 \equiv\equiv\ & \Big(\forall \mathcal{x} : \forall y : (\mathsf{first}(\mathcal{x}) \wedge \neg \mathsf{last}(y)) \rightarrow Y_1^{1,1}(\mathcal{x}, y)\Big) \\ & \wedge \big(\forall \mathcal{x} : \forall y : (\mathsf{first}(\mathcal{x}) \wedge \mathsf{last}(y)) \rightarrow Y_{\sqcup}^{1,1}(\mathcal{x}, y)\big) \\ & \wedge \big(\forall \mathcal{x} : \forall y : \mathsf{first}(\mathcal{x}) \rightarrow Y_{\sqcup}^{1,2}(\mathcal{x}, y)\big)\,. \end{aligned}$$

Finally, we must detect acceptance:

$$Z_9 \equiv\equiv \exists \mathcal{x} : \bigvee_t X_{q_a}^t(\mathcal{x}).$$

Now $\mathbb{P}\Big(Z_9 = 1 | \bigwedge_{i=1}^{8} \{Z_i = 1\}\Big) > 1/2$ for a domain of size $f(N) + 1$ if and only if the number of interpretations that set an accepting state to true is larger than half the total number of interpretations encoding computation paths. □

Theorem 14. *Suppose relations have bounded arity.* INF[QF] *and* QINF[QF] *are* PP*-complete with respect to many-one reductions, and* DINF[QF] *requires constant computational effort. These results hold* even if domain size is given in binary notation.

Proof. Consider first INF[QF]. To prove membership, take a relational Bayesian network specification $\mathbb{S}$ with relations $X_1, \ldots, X_n$, all with arity no larger than k. Suppose we ground this specification on a domain of size N. To compute $\mathbb{P}(\mathbf{Q}|\mathbf{E})$, the only relevant groundings are the ones that are ancestors of each of the ground atoms in $\mathbf{Q} \cup \mathbf{E}$. Our strategy will be to bound the number of such relevant groundings. To do that, take a grounding $X_i(a_1, \ldots, a_{k_i})$ in $\mathbf{Q} \cup \mathbf{E}$, and suppose that X_i is not a root node in the parvariable graph. Each parent X_j of X_i in the parvariable graph may appear in several different forms in the definition axiom related to X_i; that is, we may have $X_j(\mathcal{x}_2, \mathcal{x}_3), X_j(\mathcal{x}_9, \mathcal{x}_1), \ldots$, and each one of these combinations leads to a distinct grounding. There are in fact at most $k_i^{k_i}$ ways to select individuals from the grounding $X_i(a_1, \ldots, a_{k_i})$ so as to form groundings of X_j. So for each parent of X_i in the parvariable graph there will be at most k^k relevant groundings. And each parent of these parents will again have at most k^k relevant groundings; hence there are at most $(n-1)k^k$ relevant groundings that are ancestors of $X_i(a_1, \ldots, a_{k_i})$. We can take the union of all groundings that are ancestors of groundings of $\mathbf{Q} \cup \mathbf{E}$, and the number of such groundings is still polynomial in the size of the input. Thus in polynomial time we can build a polynomially-large Bayesian network that is a fragment of the grounded Bayesian network. Then we can run a Bayesian network inference in this smaller network (an effort within PP); note that domain size is actually not important so it can be specified either in unary or binary notation. To prove hardness, note that INF[Prop($\wedge, \neg$)] is PP-hard, and a propositional specification can be reproduced within QF.

Now consider QINF[QF]. To prove membership, note that even INF[QF] is in PP. To prove hardness, take an instance of #3SAT($>$) consisting of a sentence ϕ

in 3CNF, with propositions $A_1, \ldots, A_n$, and an integer k. Consider the relational Bayesian network specification consisting of eight definition axioms:

$$\begin{aligned}
\mathsf{clause0}(\mathcal{x}, y, z) &\equiv \neg\mathsf{sat}(\mathcal{x}) \vee \neg\mathsf{sat}(y) \vee \neg\mathsf{sat}(z), \\
\mathsf{clause1}(\mathcal{x}, y, z) &\equiv \neg\mathsf{sat}(\mathcal{x}) \vee \neg\mathsf{sat}(y) \vee \mathsf{sat}(z), \\
\mathsf{clause2}(\mathcal{x}, y, z) &\equiv \neg\mathsf{sat}(\mathcal{x}) \vee \mathsf{sat}(y) \vee \neg\mathsf{sat}(z), \\
&\vdots \quad \vdots \quad \vdots \\
\mathsf{clause7}(\mathcal{x}, y, z) &\equiv \mathsf{sat}(\mathcal{x}) \vee \mathsf{sat}(y) \vee \mathsf{sat}(z),
\end{aligned}$$

and $\mathbb{P}(\mathsf{sat}(\mathcal{x}) = 1) = 1/2$. Now the query is just a set of assignments $\mathbf{Q}$ ($\mathbf{E}$ is empty) containing an assignment per clause. If a clause is $\neg A_2 \vee A_3 \vee \neg A_1$, then take the corresponding assignment $\{\mathsf{clause2}(a_2, a_3, a_1) = 1\}$, and so on. The #3SAT(>) problem is solved by deciding whether $\mathbb{P}(\mathbf{Q}) > k/2^n$ with domain of size n; hence the desired hardness is proved.

And DINF[QF] requires constant effort: in fact, domain size is not relevant to a fixed inference, as can be seen from the proof of inferential complexity above. □

Appendix A.3. Description logics (Section 6)

Theorem 15. *Suppose the domain size is specified in unary notation. Then* INF[ALC] *and* QINF[ALC] *are* PP*-complete with respect to many-one reductions, and* DINF[ALC] *is in* P.

Proof. Because ALC is in FFFO^2, both INF[ALC] and QINF[ALC] are in PP by Theorems 11 and 12, and DINF[ALC] is in P. A simple proof for PP-hardness of INF[ALC] and QINF[ALC] can be extracted from the proof of Theorem 7: reproduce that proof, except that Expression (A.6) must be discarded and the inference now asks whether $\mathbb{P}(\mathsf{clause}(1) = 1, \ldots, \mathsf{clause}(\mathtt{n}) = 1|\mathbf{E}) > 1/2$. This shows the desired hardness as Expression (A.5) belongs to ALC. □

Theorem 16. *Suppose the domain size is specified in unary notation. Then* INF[EL] *and* QINF[EL] *are* PP*-complete with respect to many-one reductions, even if the query contains only positive assignments, and* DINF[EL] *is in* P.

Proof. INF[EL] belongs to PP by Theorem 11 as EL belongs to FFFO^2. Hardness is obtained from hardness of query complexity.

So, consider QINF[EL]. Membership follows from membership of INF[EL], so we focus on hardness. Our strategy is to reduce #(1-in-3)SAT(>) to QINF[EL], using most of the construction in the proof of Theorem 2. So take a sentence ϕ in 3CNF with propositions $A_1, \ldots, A_n$ and m clauses, and an integer k. The goal is to decide whether $\#(1\text{-in-}3)\phi > k$. We can assume that no clause contains a repeated literal.

We start by adapting several steps in the proof of Theorem 2. First, associate each literal with a random variable X_{ij} (where X_{ij} stands for a *negated* literal). In the present proof we use a parvariable $X(\mathcal{x})$; the idea is that $\mathcal{x}$ is the integer

$3(i-1)+j$ for some $i \in \{1,\ldots,n\}$ and $j \in \{1,2,3\}$ (clearly we can obtain (i,j) from χ and vice-versa). Then associate X with the assessment

$$\mathbb{P}(X(\chi) = 1) = \varepsilon,$$

where ε is exactly as in the proof for $\mathsf{INF}[\mathsf{Prop}(\vee)]$.

The next step in the proof of Theorem 2 is to introduce a number of definition axioms of the form $Y_{iuv} \equiv\!\equiv X_{iu} \vee X_{iv}$, together with assignments $\{Y_{iuv} = 1\}$. There are $3m$ such axioms. Then additional axioms are added to guarantee that configurations are sensible. Note that we can compute in polynomial time the total number of definition axioms that are to be created. We denote this number by N, as we will use it as the size of the domain. In any case, we can easily bound N: first, each clause produces 3 definition axioms as in Expression (A.2); second, to guarantee that configurations are sensible, every time a literal is identical to another literal, or identical to the negation of another literal, four definition axioms are inserted (there are $3m$ literals, and for each one there may be 2 identical/negated literals in the other $m-1$ clauses). Thus we have that $N \leq 3m + 4 \times 3m \times 2(m-1) = 24m^2 - 21m$. Suppose we order these definition axioms from 1 to N by some appropriate scheme.

To encode these N definition axioms, we introduce two other parvariables $Y(\chi)$ and $Z(\chi, y)$, with definition axiom

$$Y(\chi) \equiv\!\equiv \exists y : Z(\chi, y) \wedge X(y)$$

and assessment

$$\mathbb{P}\big(Z(\chi, y) = 1\big) = \eta,$$

for some η to be determined later. The idea is this. We take a domain with size N, and for each χ from 1 to N, we set $Z(\chi, y)$ to 0 if $X(y)$ does not appear in the definition axiom indexed by χ, and we set $Z(\chi, y)$ to 1 if $X(y)$ appears in the definition axiom indexed by χ. We collect all these assignments in a set $\mathbf{E}$. Note that $\mathbf{E}$ in effect "creates" all the desired definition axioms by selecting two instances of X per instance of Y.

Note that if we enforce $\{Y(\chi) = 1\}$ for all χ, we obtain the same construction used in the proof of Theorem 2, we one difference: in that proof we had $3m$ variables X_{ij}, while here we have N variables $X(\chi)$ (note that $N \geq 3m$, and in fact $N > 3m$ for $m > 1$).

Consider grounding this relational Bayesian network specification and computing

$$\mathbb{P}(X(\mathtt{1}) = x_1, \ldots, X(\mathtt{N}) = x_N, Y(\mathtt{1}) = y_1, \ldots, Y(\mathtt{N}) = y_N | \mathbf{E})\,.$$

This distribution is encoded by a Bayesian network with nodes $X(\mathtt{1}), \ldots, X(\mathtt{N})$ and nodes $Y(\mathtt{1}), \ldots, Y(\mathtt{N})$, where all nodes $Z(\chi, y)$ are removed as they are set by $\mathbf{E}$; also, each node $Y(\chi)$ has two parents, and all nodes $X(\mathtt{3m}+\mathtt{1}), \ldots, X(\mathtt{N})$ have no children. Denote by $\mathbf{L}$ a generic configuration of $X(\mathtt{1}), \ldots, X(\mathtt{3m})$, and by $\mathbf{Q}$ a configuration of $Y(\mathtt{1}), \ldots, Y(\mathtt{N})$ where all variables are assigned value

1. As in the proof of Theorem 2, we have $\mathbb{P}(\mathbf{L}) = \alpha$ if $\mathbf{L}$ is gratifying-sensible-respectful, and $\mathbb{P}(\mathbf{L}) \leq \beta$ if $\mathbf{L}$ is respectful but not gratifying. If #(1-in-3)$\phi > k$, then $\mathbb{P}(\mathbf{Q}|\mathbf{E}) = \sum_{\mathbf{L}} \mathbb{P}(\mathbf{L}, \mathbf{Q}) \geq (k+1)\alpha$. And if #(1-in-3)$\phi \leq k$, then $\mathbb{P}(\mathbf{Q}|\mathbf{E}) \leq k\alpha + 4^m\beta$. Define $\delta_1 = (k+1)\alpha$ and $\delta_2 = k\alpha + 4^m\beta$ and choose $\varepsilon < 1/(1+4^m)$ to guarantee that $\delta_1 > \delta_2$, so that we can differentiate between the two cases with an inference.

We have thus solved our original problem using a fixed Bayesian network specification plus a query $(\mathbf{Q}, \mathbf{E})$. Hence PP-hardness of QINF[EL] is obtained. However, note that $\mathbf{Q}$ contains only positive assignments, but $\mathbf{E}$ contains both positive and negative assignments. We now constrain ourselves to positive assignments.

Denote by $\mathbf{E}_1$ the assignments of the form $\{Z(\chi, y) = 1\}$ in $\mathbf{E}$, and denote by $\mathbf{E}_0$ the assignments of the form $\{Z(\chi, y) = 0\}$ in $\mathbf{E}$. Consider:

$$\mathbb{P}(\mathbf{Q}|\mathbf{E}_1) = \mathbb{P}(\mathbf{Q}|\mathbf{E}_0, \mathbf{E}_1)\,\mathbb{P}(\mathbf{E}_0|\mathbf{E}_1) + \mathbb{P}(\mathbf{Q}|\mathbf{E}_0^c, \mathbf{E}_1)\,\mathbb{P}(\mathbf{E}_0^c|\mathbf{E}_1)\,,$$

where $\mathbf{E}_0^c$ is the event consisting of configurations of those variables that appear in $\mathbf{E}_0$ such that at least one of these variables is assigned 1 (of course, such variables are assigned 0 in $\mathbf{E}_0$).

We have that $\mathbb{P}(\mathbf{Q}|\mathbf{E}_0, \mathbf{E}_1) = \mathbb{P}(\mathbf{Q}|\mathbf{E})$ by definition. And variables in $\mathbf{E}_0$ and $\mathbf{E}_1$ are independent, hence $\mathbb{P}(\mathbf{E}_0|\mathbf{E}_1) = \mathbb{P}(\mathbf{E}_0) = (1-\eta)^M$ where M is the number of variables in $\mathbf{E}_0$ (so $M \leq N^2$). Consequently, $\mathbb{P}(\mathbf{E}_0^c|\mathbf{E}_1) = 1-(1-\eta)^M$. Thus we reach:

$$\mathbb{P}(\mathbf{Q}|\mathbf{E}_1) = (1-\eta)^M\mathbb{P}(\mathbf{Q}|\mathbf{E}) + (1-(1-\eta)^M)\mathbb{P}(\mathbf{Q}|\mathbf{E}_0^c, \mathbf{E}_1)\,.$$

Now reason as follows. If #(1-in-3)$\phi > k$, then $\mathbb{P}(\mathbf{Q}|\mathbf{E}_1) \geq (1-\eta)^M\delta_1$. And if #(1-in-3)$\phi \leq k$, then $\mathbb{P}(\mathbf{Q}|\mathbf{E}_1) \leq (1-(1-\eta)^M)+(1-\eta)^M\delta_2$. To guarantee that $(1-\eta)^M\delta_1 > (1-(1-\eta)^M)+(1-\eta)^M\delta_2$, we must have $(1-\eta)^M > 1/(1+\delta_1-\delta_2)$. We do so by selecting η appropriately. Denote $1/(1+\delta_1-\delta_2)$ by δ_3, and note that $\delta_3 \in (0,1)$ by our choice of ε. We must select η so that $1-\eta > \delta_3^{1/M}$; to guarantee this, we find a quantity δ_4 that is guaranteed to be larger than $\delta_3^{1/M}$, and impose $1-\eta > \delta_4$. Note that $1+(x-1)/M > x^{1/M}$ for any $x \in (0,1)$, so take $\delta_4 = 1+(\delta_3-1)/M$ and impose

$$1-\eta > \delta_4 = 1+(\delta_3-1)/M = 1+\left(\frac{1}{1+\delta_1-\delta_2}-1\right)/M;$$

that is,

$$\eta < 1-\delta_4 = \left(1-\frac{1}{1+\delta_1-\delta_2}\right)/M.$$

By doing so, we can differentiate between the two cases with an inference, so the desired hardness is proved.

Domain complexity is polynomial because EL is in FFFO^2 [132, 134]. □

Theorem 17. *Suppose the domain size is specified in unary notation. If in addition the query* $(\mathbf{Q}, \mathbf{E})$ *contains only positive assignments, then* INF[DLLite$^{\mathsf{nf}}$] *and* QINF[DLLite$^{\mathsf{nf}}$] *and* DINF[DLLite$^{\mathsf{nf}}$] *are in* P.

Proof. We prove the polynomial complexity of $\mathsf{INF}[\mathsf{DLLite}^{\mathsf{nf}}]$ with positive queries by a quadratic-time reduction to multiple problems of counting weighted edge covers with uniform weights in a particular class of graphs. Each one of these counting problems can be in turn solved in quadratic time; thus the whole effort is polynomial.

We describe techniques to compute the probability of a set of positive assignments. One must first compute the probability of all assignments in $(\mathbf{Q}, \mathbf{E})$, then the probability of assignments in $\mathbf{E}$, and finally divide both numbers. From now on $\mathbf{Q}$ refers to a set of positive assignments whose probability is of interest; this set may include or be restricted to $\mathbf{E}$ as needed.

We first simplify a bit the notation, as follows. Introduce $\mathsf{e}_\mathsf{r}(\mathcal{X}) \equiv \exists y : \mathsf{r}(\mathcal{X}, y)$ and $\mathsf{e}^-_\mathsf{r}(\mathcal{X}) \equiv \exists y : \mathsf{r}(y, \mathcal{X})$. Then replace each appearance of the formula $\exists y : \mathsf{r}(\mathcal{X}, y)$ by $\mathsf{e}_\mathsf{r}(\mathcal{X})$ (or $\exists \mathcal{X} : \mathsf{r}(y, \mathcal{X})$ by $\mathsf{e}_\mathsf{r}(y)$), and each appearance of $\exists y : \mathsf{r}(y, \mathcal{X})$ by $\mathsf{e}^-_\mathsf{r}(\mathcal{X})$ (or $\exists \mathcal{X} : \mathsf{r}(\mathcal{X}, y)$ by $\mathsf{e}^-_\mathsf{r}(y)$). This transformation allows us to easily refer to groundings of existentially quantified formulas as groundings of e_r and e^-_r, respectively.

Observe that only the nodes with assignments in $\mathbf{Q}$ and their ancestors are relevant for the computation of $\mathbb{P}(\mathbf{Q})$, as every other node in the Bayesian network is barren [33]. Hence, we can assume without loss of generality that $\mathbf{Q}$ contains only leaves of the network. If $\mathbf{Q}$ contains only root nodes, then $\mathbb{P}(\mathbf{Q})$ can be computed trivially as the product of marginal probabilities which are readily available from the specification. Thus assume that $\mathbf{Q}$ assigns a positive value to at least one non-root leaf grounding $\mathsf{s}(a)$, where a is some individual in the domain. Suppose $\mathsf{s}(a)$ is associated with a logical sentence $X_1 \wedge \cdots \wedge X_k$, where each X_i is a grounding of a unary relation in a. It follows that $\mathbb{P}(\mathbf{Q}) = \mathbb{P}(\mathsf{s}(a) = 1 | X_1 = 1, \ldots, X_k = 1)\, \mathbb{P}(\mathbf{Q}') = \mathbb{P}(\mathbf{Q}')$, where $\mathbf{Q}'$ is $\mathbf{Q}$ after removing the assignment $\{\mathsf{s}(a) = 1\}$ and adding the assignments $\{X_1 = 1, \ldots, X_k = 1\}$. The problem of computing $\mathbb{P}(\mathbf{Q})$ boils down to computing $\mathbb{P}(\mathbf{Q}')$. By repeating this procedure for all non-root nodes which are not groundings of e_r or e^-_r, we end up with a set $\mathbf{A}$ containing positive assignments of groundings of relations, including of auxiliary relations e_r and e^-_r. Each root node is is marginally independent from all other groundings in $\mathbf{A}$; hence $\mathbb{P}(\mathbf{A}) = \mathbb{P}(\mathbf{B}|\mathbf{C}) \prod_i \mathbb{P}(A_i)$, where each A_i is an assignment to a root node, $\mathbf{B}$ are (positive) assignments to groundings of relations e_r and e^-_r for relations r, and $\mathbf{C} \subseteq \{A_1, A_2, \ldots\}$ are groundings of binary relations (if $\mathbf{C}$ is empty then assume it expresses a tautology). Because the marginal probabilities $\mathbb{P}(A_i)$ are available from the specification, $\prod_i \mathbb{P}(A_i)$ can be computed in linear time from the input. We thus focus on computing $\mathbb{P}(\mathbf{B}|\mathbf{C})$. To recap, $\mathbf{B}$ is a set of assignments $\mathsf{e}_\mathsf{r}(a) = 1$ and $\mathsf{e}^-_\mathsf{r}(b) = 1$ and $\mathbf{C}$ is a set of assignments $\mathsf{r}(c, d) = 1$ for arbitrary binary relations r and individuals a, b, c and d. Note that if $\mathbf{B}$ is empty, we are done.

For a binary relation r, let $\mathcal{D}_\mathsf{r}$ be the set of individuals $a \in \mathcal{D}$ such that $\mathsf{e}_\mathsf{r}(a) = 1$ is in $\mathbf{B}$, and let $\mathcal{D}^-_\mathsf{r}$ be the set of individuals $a \in \mathcal{D}$ such that $\mathbf{B}$ contains $\mathsf{e}^-_\mathsf{r}(a) = 1$. Let $\mathsf{gr}(\mathsf{r})$ be the set of all groundings of relation r, and let $\mathsf{r}_1, \ldots, \mathsf{r}_k$ be the binary relations in the (relational) network. By the factorization property of Bayesian networks it follows that

$$\mathbb{P}(\mathbf{B}|\mathbf{C}) = \sum_{\mathsf{gr}(\mathsf{r}_1)} \cdots \sum_{\mathsf{gr}(\mathsf{r}_k)} \prod_{i=1}^{k} \prod_{a \in \mathcal{D}_{\mathsf{r}_i}} \mathbb{P}(\mathsf{e}_{\mathsf{r}_i}(a) = 1 | \mathrm{pa}(\mathsf{e}_{\mathsf{r}_i}(a)), \mathbf{C}) \times \prod_{a \in \mathcal{D}_{\mathsf{r}_i}^-} \mathbb{P}\big(\mathsf{e}_{\mathsf{r}_i}^-(a) = 1 | \mathrm{pa}(\mathsf{e}_{\mathsf{r}_i}^-(a)), \mathbf{C}\big)\, \mathbb{P}(\mathsf{gr}(\mathsf{r}_i)|\mathbf{C}) \ ,$$

noting that we only write down the groundings from $\mathsf{gr}(\mathsf{r}_i)$ that affect groundings $\mathsf{e}_{\mathsf{r}_i}$ and $\mathsf{e}_{\mathsf{r}_i}^-$ (using the Markov condition on the grounded Bayesian network).

By distributing the products over sums in the last expression, we obtain:

$$\mathbb{P}(\mathbf{B}|\mathbf{C}) = \prod_{i=1}^{k} \sum_{\mathsf{gr}(\mathsf{r}_i)} \prod_{a \in \mathcal{D}_{\mathsf{r}_i}} \mathbb{P}(\mathsf{e}_{\mathsf{r}_i}(a) = 1 | \mathrm{pa}(\mathsf{e}_{\mathsf{r}_i}(a)), \mathbf{C}) \times \prod_{a \in \mathcal{D}_{\mathsf{r}_i}^-} \mathbb{P}\big(\mathsf{e}_{\mathsf{r}_i}^-(a) = 1 | \mathrm{pa}(\mathsf{e}_{\mathsf{r}_i}^-(a)), \mathbf{C}\big)\, \mathbb{P}(\mathsf{gr}(\mathsf{r}_i)|\mathbf{C}) \ .$$

Now consider an assignment $\mathsf{r}(a, b) = 1$ in $\mathbf{C}$. By construction, the children of the grounding $\mathsf{r}(a, b)$ are $\mathsf{e}_{\mathsf{r}}(a)$ and $\mathsf{e}_{\mathsf{r}}^-(b)$. Moreover, the assignment $\mathsf{r}(a, b) = 1$ implies that $\mathbb{P}(\mathsf{e}_{\mathsf{r}}(a) = 1|\mathrm{pa}(\mathsf{e}_{\mathsf{r}}(a)), \mathbf{C}) = 1$ (for any assignment to the other parents) and $\mathbb{P}(\mathsf{e}_{\mathsf{r}}^-(b) = 1|\mathrm{pa}(\mathsf{e}_{\mathsf{r}}(a)), \mathbf{C}) = 1$ (for any assignment to the other parents). This is equivalent in the factorization above to removing $\mathsf{r}(a, b)$ from $\mathbf{C}$ (as it is independent of all other groundings), and removing individuals a from $\mathcal{D}_{\mathsf{r}}$ and b from $\mathcal{D}_{\mathsf{r}}^-$. So repeat this procedure for every grounding in $\mathbf{C}$ until this set is empty (this can be done in polynomial time). The inference problem becomes one of computing

$$\gamma(\mathsf{r}_i) = \sum_{\mathsf{gr}(\mathsf{r}_i)} \prod_{a \in \mathcal{D}_{\mathsf{r}_i}} \mathbb{P}(\mathsf{e}_{\mathsf{r}_i}(a) = 1 | \mathrm{pa}(\mathsf{e}_{\mathsf{r}_i}(a))) \prod_{a \in \mathcal{D}_{\mathsf{r}_i}^-} \mathbb{P}\big(\mathsf{e}_{\mathsf{r}_i}^-(a) = 1 | \mathrm{pa}(\mathsf{e}_{\mathsf{r}_i}^-(a))\big)\, \mathbb{P}(\mathsf{gr}(\mathsf{r}_i))$$

for every relation r_i, $i = 1, \ldots, k$, with properly adapted sets $\mathcal{D}_{\mathsf{r}_i}$ and $\mathcal{D}_{\mathsf{r}_i}^-$. We will show that this problem can be reduced to a tractable instance of counting weighted edge covers. To this end, we need to introduce some notation and terminology that we also use in the next section.

A *black-and-white graph* (bw-graph) is a triple $G = (V, E, \chi)$ where (V, E) is a simple undirected graph and $\chi : V \to \{0, 1\}$ is a binary-valued function on the node set (assume 0 means white and 1 means black). Denote by $E_G(u)$ the set of edges incident on a node $u \in V$, and by $N_G(u)$ the open neighborhood of u (i.e., not including u). An *edge cover* of a bw-graph is a subset of the edges $C \subseteq E$ such that for each black node u we have that $E_G(u) \cap C \neq \emptyset$ (i.e., there is *at least one* edge incident in each black node). Denote by $\mathsf{EC}(G)$ the set of edge covers of G. For any $C \in \mathsf{EC}(G)$ and real $\lambda > 0$, we say that $\lambda^{|C|}$ is the weight of cover C. The *partition function* of a bw-graph G is the total sum of cover weights (a.k.a. the weighted edge cover counting): $Z(G, \lambda) = \sum_{C \in \mathsf{EC}(G)} \lambda^{|C|}$. Note that $Z(G, 1) = |\mathsf{EC}(G)|$ counts the number of edge covers.

We recast marginal inference in DL-Lite Bayesian networks as the computation of the partition function of the bw-graph $G_r = (V_1 \cup V_2 \cup V_3 \cup V_4, E, \chi)$ such that:

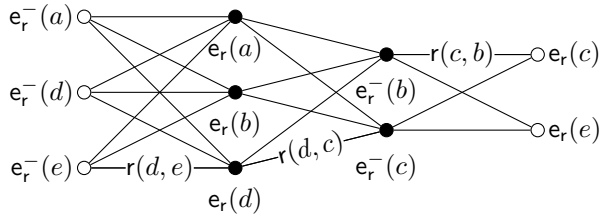


Figure A.15: Representing assignments by graphs. Columns of nodes correspond respectively to V_1, V_2, V_3 and V_4; black nodes are in $\mathbf{B}$, and white nodes are not in $\mathbf{B}$. Each edge corresponds to a grounding of r; for clarity's sake, we label only three edges.

- $V_1 = \{\mathsf{e}_\mathsf{r}^-(a) : a \in \mathcal{D} \setminus \mathcal{D}_\mathsf{r}^-\}$ and $\chi(v) = 0$ for all $v \in V_1$,
- $V_2 = \{\mathsf{e}_\mathsf{r}(a) : a \in \mathcal{D}_\mathsf{r}\}$ and $\chi(v) = 1$ for all $v \in V_2$,
- $V_3 = \{\mathsf{e}_\mathsf{r}^-(a) : a \in \mathcal{D}_\mathsf{r}^-\}$ and $\chi(v) = 1$ for all $v \in V_3$,
- $V_4 = \{\mathsf{e}_\mathsf{r}(a) : a \in \mathcal{D} \setminus \mathcal{D}_\mathsf{r}\}$ and $\chi(v) = 0$ for all $v \in V_4$,
- for $i = 1, 2, 3$ the subgraph with nodes $V_i \cup V_{i+1}$ is bipartite complete (i.e., every node in V_i is connect to a node in V_{i+1}, and no two nodes in V_i or in V_{i+1} are connected).

The graph G_r is called the *intersection graph* of $\mathbf{B}$ with respect to r and $\mathcal{D}$. In this graph an edge with endpoints $\mathsf{e}_\mathsf{r}(a)$ and $\mathsf{e}_\mathsf{r}^-(b)$ represents the grounding $\mathsf{r}(a, b)$; the parents of a node correspond exactly to the parents of the related node in the Bayesian network. For example, the graph in Figure A.15 represents the assignments $\mathbf{B} = \{\mathsf{e}_\mathsf{r}(a) = 1, \mathsf{e}_\mathsf{r}(b) = 1, \mathsf{e}_\mathsf{r}(d) = 1, \mathsf{e}_\mathsf{r}^-(b) = 1, \mathsf{e}_\mathsf{r}^-(c) = 1\}$, with respect to domain $\mathcal{D} = \{a, b, c, d, e\}$. The black nodes (resp., white nodes) represent groundings in (resp., not in) $\mathbf{B}$.

The following result connects marginal inference and weighted edge cover counting:

Lemma 1. *Let $G_r = (V_1, V_2, V_3, V_4, E)$ be the intersection graph of $\mathbf{B}$ with respect to a relation r and domain $\mathcal{D}$. Then $\gamma(\mathsf{r}) = Z(G_r, \alpha/(1-\alpha))/(1-\alpha)^{|E|}$, where $\alpha = \mathbb{P}\big(\mathsf{r}(\mathcal{x}, \mathcal{y})\big)$.*

Proof of Lemma 1. Consider an edge cover C of the graph. The assignment that sets to true all groundings $\mathsf{r}(a, b)$ corresponding to edges in C, and sets to false the remaining groundings of r makes

$$\mathbb{P}(\mathsf{e}_\mathsf{r}(a) = 1 | \mathrm{pa}(\mathsf{e}_\mathsf{r}(a))) = \mathbb{P}\big(\mathsf{e}_\mathsf{r}^-(b) = 1 | \mathrm{pa}(\mathsf{e}_\mathsf{r}^-(b))\big) = 1$$

for every $a \in \mathcal{D}_\mathsf{r}$ and $b \in \mathcal{D}_\mathsf{r}^-$; it makes $\mathbb{P}(\mathsf{gr}(\mathsf{r})) = \mathbb{P}(\mathsf{r})^{|C|}\,(1 - \mathbb{P}(\mathsf{r}))^{|E|-|C|} = (1-\alpha)^{|E|}\alpha^{|C|}/(1-\alpha)^{|C|}$, which is the weight of the cover C scaled by $(1-\alpha)^{|E|}$. Now consider a set of edges C which is not an edge cover and that yields an assignment to groundings $\mathsf{gr}(\mathsf{r})$ as before. There is at least one node in $V_2 \cup V_3$

that does not contain any incident edges in C. Assume that node is $\mathsf{e}(a)$; then all parents of $\mathsf{e}(a)$ are assigned false, which implies that $\mathbb{P}(\mathsf{e}_\mathsf{r}(a) = 1|\mathrm{pa}(\mathsf{e}_\mathsf{r}(a))) = 0$. The same is true if the node not covered is a grounding $\mathsf{e}^-(a)$. Hence, for each edge cover C the probability of the corresponding assignment equals its weight up to the factor $(1-\alpha)^{|E|}$. And for each edge set C which is not an edge cover its corresponding assignment has probability zero. □

We have thus established that, if a particular class of weighted edge cover counting problems is polynomial, then marginal inference in DL-Lite Bayesian networks is also polynomial for positive assignments in the input. Because the problem of weighted edge cover counting is of independent interest, we deal with it in the next subsection; by describing there a polynomial counting algorithm, we finish the proof of Theorem 17. □

Theorem 18. *Given a relational Bayesian network* $\mathbb{S}$ *based on* $\mathsf{DLLite}^{\mathsf{nf}}$*, a set of positive assignments to grounded relations* $\mathbf{E}$*, and a domain size* N *in unary notation,* $\mathsf{MLE}(\mathbb{S}, \mathbf{E}, N)$ *can be solved in polynomial time.*

Proof. In this theorem we are interested in finding an assignment $\mathbf{X}$ to all groundings that maximizes $\mathbb{P}(\mathbf{X} \wedge \mathbf{E})$, where $\mathbf{E}$ is a set of positive assignments. Perform the substitution of formulas $\exists y : \mathsf{r}(\chi, y)$ and $\exists y : \mathsf{r}(y, \chi)$ by logically equivalent concepts e_r and e_r^- as in the proof of Theorem 17. Consider a non-root grounding $\mathsf{s}(a)$ in $\mathbf{E}$ which is not the grounding of e_r or e_r^-; by construction, $\mathsf{s}(a)$ is logically equivalent to a conjunction $X_1 \wedge \cdots \wedge X_k$, where $X_1, \ldots, X_k$ are unary groundings. Because $\mathsf{s}(a)$ is assigned to true, any assignment $\mathbf{X}$ with nonzero probability assigns $X_1, \ldots, X_k$ to true. Moreover, since $\mathsf{s}(a)$ is a non-root node, its corresponding probability is one. Hence, if we include all the assignments $\{X_i = 1\}$ to its parents in $\mathbf{E}$, the MPE value does not change. Assume we repeate this procedure until $\mathbf{E}$ contains all ancestors of the original groundings that are groundings of unary relations. Note that at this point we only need to assign values to nodes that are either not ancestors of any node in the original set $\mathbf{E}$, and to groundings of (collapsed) binary relations r.

Consider the groundings of primitive unary relations r that are not ancestors of any grounding in $\mathbf{E}$. Setting their value to maximize the marginal probability does not introduce any inconsistency with respect to $\mathbf{E}$. Moreover, for any assignment to these groundings, we can find a consistent assignment to the remaining groundings (which are internal nodes and not ancestors of $\mathbf{E}$), that is, an assignment which assigns positive probability. Because this is the maximum probability we can obtain for these groundings, this is a partial optimum assignment.

We are thus only left with the problem of assigning values to the groundings of relations r that are ancestors of $\mathbf{E}$. Consider a relation r such that $\mathbb{P}(\mathsf{r}) \geq 1/2$. Then assigning all groundings of r to true maximizes their marginal probability and satisfies the logical equivalences of all groundings in $\mathbf{E}$. Hence, this is a maximum assignment (and its value can be computed efficiently). So assume there is a relation r with $\mathbb{P}(\mathsf{r}) < 1/2$ such that a grounding of e_r or e_r^- appear in $\mathbf{E}$. In this case, the greedy assignment sets every grounding of r; however,

such an assignment is inconsistent with the logical equivalence of e_r and e_r^-, hence it gets probability zero. Now consider an assignment that assigns exactly one grounding $\mathsf{r}(a,b)$ to true and all the other to false. This assignment is consistent with $\mathsf{e}_\mathsf{r}(a)$ and $\mathsf{e}_\mathsf{r}(b)$, and maximizes the probability; any assignment that sets more groundings to true has a lower probability since it replaces a term $1-\mathbb{P}(\mathsf{r}) \geq 1/2$ with a term $\mathbb{P}(\mathsf{r}) < 1/2$ in the joint probability. More generally, to maximize the joint probability we need to assign to true as few groundings $\mathsf{r}(a,b)$ which are ancestors of $\mathbf{E}$ as possible. This is equivalent to a minimum cardinality edge covering problem as follows.

For every relation r in the relational network, construct the bipartite complete graph $G_\mathsf{r} = (V_1, V_2, E)$ such that V_1 is the set of groundings $\mathsf{e}_\mathsf{r}(a)$ that appears and has no parent $\mathsf{r}(a,b)$ in $\mathbf{E}$, and V_2 is the set of groundings $\mathsf{e}_\mathsf{r}^-(a)$ that appears and has no parents in $\mathbf{E}$. We identify an edge connecting $\mathsf{e}_\mathsf{r}(a)$ and $\mathsf{e}_\mathsf{r}^-(b)$ with the grounding $\mathsf{r}(a,b)$. For any set $C \subseteq E$, construct an assignment by attaching true to the groundings $\mathsf{r}(a,b)$ in C and false to every other grounding $\mathsf{r}(a,b)$. This assignment is consistent with $\mathbf{E}$ if and only if C is an edge cover; hence the minimum cardinality edge cover maximizes the joint probability (it is consistent with $\mathbf{E}$ and attaches true to the least number of groundings of $\mathbf{r}$). This concludes the proof of Theorem 18. □

Appendix A.4. Counting edge covers in polynomial time

This section focuses on weighted edge cover counting as needed in the proof of Theorem 17. Even though there has been significant work in connecting model counting with graph-theoretical representations of formulas [11, 108, 128, 131, 140], the class of problems discussed here seems to have escaped these previous efforts.

Recall the notation and terminology introduced right before Lemma 1: a black-and-white graph (bw-graph, for short) is a triple $G = (V, E, \chi)$, where $G = (V, E)$ is a simple undirected graph and χ is a $\{0,1\}$-valued function partitioning the node set into white ($\chi(v) = 0$) and black nodes ($\chi(v) = 1$); $E_G(u)$ denotes the set of edges incident in a node u, and $N_G(u)$ the open neighborhood of u. An edge $e = (u, v) \in E$ can be classified into one of three categories:[10]

- **free edge:** if $\chi(u) = \chi(v) = 0$;
- **dangling edge:** if $\chi(u) \neq \chi(v)$; or
- **regular edge:** if $\chi(u) = \chi(v) = 1$.

In the graph in Figure A.16(b), the edge (f, g) is a dangling edge while the edge (g, j) is a free edge. The edge (f, g) in the graph in Figure A.16(a) is a regular edge.

[10]The classifications of edges given here are analogous to those defined in [79, 80], but not fully equivalent. In fact, in [79] and [80], graphs are uncolored, but edges might contain empty endpoints. These are analogous to white node endpoints in our terminology. Regular edges are analogous to the *normal edges* defined in [79, 80].

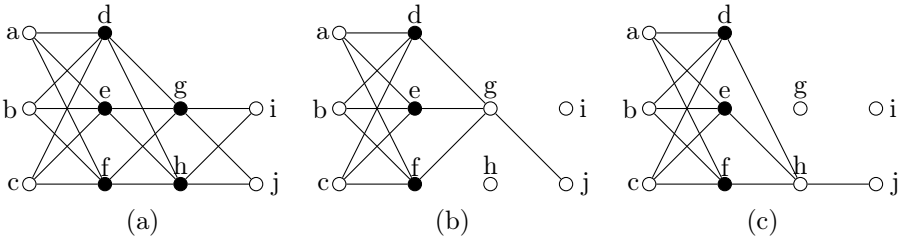


Figure A.16: (a) A graph G in $\mathcal{B}$. (b) The graph $G - E_G(h) - (i, g) - h - g$. (c) The graph $G - E_G(g) - (i, h) - g - h$.

The set $\mathsf{EC}(G)$ contains the edge covers of a bw-graph G, that is, the subsets $C \subseteq V$ such that for each black node $v \in V$, $\chi(v) = 1$, it follows that $C \cap E_G(v) \neq \emptyset$. An edge cover for the graph in Figure A.16(a) is $C = \{(a, d), (d, g), (e, g), (f, g), (h, j)\}$.

We are interested in computing the partition function $Z(G, \lambda) = \sum_{C \in \mathsf{EC}(G)} \lambda^{|C|}$ of a certain class of bw-graphs G, given a fixed real $\lambda > 0$. This problem is #P-complete in general [17], and admits an FPTAS [79, 80]. To simplify notation, we assume in the rest of this section that λ is fixed, and we write Z as function of the graph alone.

So consider the following class of graphs, collectively denoted by $\mathcal{B}$, that are clearly inspired by the contents of the previous proof. A bw-graph $G = (V, E, \chi)$ in $\mathcal{B}$ has a set of nodes that can be partitioned into four disjoint sets V_1, V_2, V_3, V_4, such that $V_1 \cup V_4$ contain only white nodes, $V_2 \cup V_3$ contain only black nodes, and for $i = 1, 2, 3$ the subgraph over nodes $V_i \cup V_{i+1}$ is bipartite complete.

We now present a dynamic programming approach to computing the partition function of graphs in $\mathcal{B}$. As we have discussed elsewhere [88], a much larger class of graphs can be tackled by the following results or by various extensions or approximation techniques.

Let e be an edge and u be a node in bw-graph G. We define the following operations and notation:

- **edge removal:** $G - e = (V, E \setminus \{e\}, \chi)$.
- **node whitening:** $G - u = (V, E, \chi')$, where $\chi'(u) = 0$ and $\chi'(v) = \chi(v)$ for $v \neq u$.

Note that these operations do not alter the node set, and that they are associative (e.g., $G-e-f = G-f-e$, $G-u-v = G-v-u$, and $G-e-u = G-u-e$). Hence, if $E = \{e_1, \ldots, e_d\}$ is a set of edges, we can write $G - E$ to denote $G - e_1 - \cdots - e_d$ applied in any order. The same is true for node whitening and for any combination of node whitening and edge removal. These operations are illustrated in the examples in Figure A.16.

The following result shows that the partition function can be decomposed in terms of two simpler problems:

Proposition 2. *Let $e = (u, v)$ be a dangling edge with u colored black. Then:*

$$Z(G) = (1 + \lambda)Z(G - e - u) - Z(G - E_G(u) - u)\,.$$

Proof. Consider the graph $G' = G - e - u$. Let A denote the set of edge covers of G that contain e and B denote the set of edge covers that do not contain e. Since A is exactly the set of edge covers of G' (after removing e) and B is exactly the set of edge covers of $G-e$, we have that $Z(G) = \sum_{C\in A} \lambda^{|C|} + \sum_{C\in B} \lambda^{|C|} = \lambda Z(G') + Z(G-e)$. Similarly, let A' be the set of edge covers of G' that contain at least one edge of $E_{G'}(u)$, and B' be the set of edge covers that contain no edge of $E_{G'}(u)$. Then $Z(G') = \sum_{C\in A'} \lambda^{|C|} + \sum_{C\in B'} \lambda^{|C|} = Z(G-e) + Z(G-E_G(u)-u)$. Isolating the term $Z(G - e)$ and substituting for it in the first identity gives us the desired result. □

Free edges can be removed by adjusting the partition function accordingly:

Proposition 3. *Let $e = (u, v)$ be a free edge of G. Then $Z(G) = (1+\lambda)Z(G-e)$.*

Proof. If C is an edge cover of $G - e$ then both C and $C \cup \{e\}$ are edge covers of G. Hence, the sum of the weights of edge covers containing e equals the sum of weights $Z(G - e)$ of edge covers not containing e up to a factor λ. □

We can use the formulas in Propositions 2 and 3 to compute the partition function of a bw-graph recursively. Each recursion computes $Z(G)$ as a function of the partition function of two graphs obtained by the removal of edges and whitening of a node. Such a naive approach however requires an exponential number of recursions (in the number of edges or nodes of the initial graph) and finishes after exponential time. We can transform such an approach into a polynomial-time algorithm by exploiting the symmetries of the graphs produced during the recursions. In particular, we take advantage of the invariance of the partition function to isomorphisms of a graph, as we discuss next.

We say that two bw-graphs $G = (V, E, \chi)$ and $G' = (V', E', \chi')$ are *isomorphic* if there is a bijection γ from V to V' (or vice-versa) such that (i) $\chi(v) = \chi'(\gamma(v))$ for all $v \in V$, and (ii) $(u, v) \in E$ if and only if $(\gamma(u), \gamma(v)) \in E'$. In other words, two bw-graphs are isomorphic if there is a color-preserving renaming of nodes that preserves the binary relation induced by E. The function γ is called an *isomorphism* from V to V'. The graphs in Figures A.16(b) and A.16(c) are isomorphic by an isomorphism that maps g to h and maps any other node to itself. If C is an edge cover of G and γ is an isomorphism between G and G', then $C' = \{(\gamma(u), \gamma(v)) : (u, v) \in C\}$ is an edge cover for G' with the same weight. Hence, $Z(G) = Z(G')$. The following result shows how to obtain isomorphic graphs with a combination of node whitenings and edge removals.

Proposition 4. *Consider a bw-graph G with nodes $v_1, \ldots, v_n$, where $N_G(v_1) = \cdots = N_G(v_n) \neq \emptyset$ and $\chi_G(v_1) = \cdots = \chi_G(v_n)$. For any node $w \in N_G(v_1)$, bijection $\gamma : \{v_1, \ldots, v_n\} \to \{v_1, \ldots, v_n\}$, and nonnegative integers k_1 and k_2 such that $k_1 + k_2 \leq n$ the graphs $G' = G - E_G(v_1) - \cdots - E_G(v_{k_1}) - (w, v_{k_1+1}) - \cdots - (w, v_{k_1+k_2}) - v_1 - \cdots - v_{k_1+k_2}$ and $G'' = G - E_G(\gamma(v_1)) - \cdots - E_G(\gamma(v_{k_1})) - (w, \gamma(v_{k_1+1})) - \cdots - (w, \gamma(v_{k_1+k_2})) - \gamma(v_1) - \cdots - \gamma(v_{k_1+k_2})$ are isomorphic.*

Proof. Let γ' be the bijection on the nodes of G that extends γ, that is, $\gamma'(u) = u$ for $u \notin \{v_1, \ldots, v_n\}$ and $\gamma'(u) = \gamma(v_i)$, for $i = 1, \ldots, n$. We will show that γ' is an isomorphism from G' to G''. First note that $\chi_G(u) = \chi_G(\gamma(u))$ for every node u. The only nodes that have their color (possibly) changed in G' with respect to G are the nodes $v_1, \ldots, v_{k_1+k+2}$, and these are white nodes in G'. Likewise, the only nodes that would (possibly) changed color in G'' were $\gamma(v_1), \ldots, \gamma(v_{k_1+k_2})$ and these are white in G''. Hence, $\chi_{G'}(u) = \chi_{G''}(\gamma(u))$ for every node u.

Now let us look at the edges. First note that since $N_G(v_i)$ is constant through $i = 1, \ldots, n$, G' and G'' have the same number of edges. Hence, it suffices to show that for each edge (u, v) in G' the edge $(\gamma'(u), \gamma'(v))$ is in G''. The only edges modified in obtaining G' and G'' are, respectively, those incident in $v_1, \ldots, v_{k_1+k_2}$ and in $\gamma(v_1), \ldots, \gamma(v_{k_1+k_2})$. Consider an edge (u, v) where $u, v \notin \{v_1, \ldots, v_n\}$ (hence not in $E_G(v_i)$ for any i). If $(u, v) = (\gamma'(u), \gamma'(v))$ is in G' then it is also in G''. Now consider an edge (u, v_i) in G where $u \notin \{w, v_{k_1+1}, \ldots, v_n\}$ and $k_1 < i \leq k_1 + k_2$. Then (u, v_i) is in G' and $(\gamma'(u), \gamma'(v_i))$ is in G''. Note that u could be in $N_G(v_i)$ for $k_1 + k_2 < i \leq n$. □

According to the proposition above, the graphs in Figures A.16(b) and A.16(c) are isomorphic by a bijection between g and h (and with $w = i$). Hence, the partition function of either graph is the same.

The algorithms RightRecursion and LeftRecursion described in Figures A.17 and A.18, respectively, exploit the isomorphisms described in Proposition 4 in order to achieve polynomial-time behavior when using the recursions in Propositions 2 and 3. Either algorithm requires a base white node w and integers k_1 and k_2 specifying the recursion level (with the same meaning as in Proposition 4). Unless $k_1 + k_2$ equals the number of neighbors of w in the original graph, a call to either algorithm generates two more calls to the same algorithm: one with the graph obtained by removing edge (w, v_h) and whitening v_h, and another by removing edges $E(v_h)$ and whitening v_h. Assume that $|V_2| \geq |V_3|$ (if $|V_3| > |V_2|$ we can simply manipulate node sets to obtain an isomorphic graph satisfying the assumption). The RightRecursion algorithm first checks whether the value for the current recursion level has been already computed; if yes, then it simply returns the cached value; otherwise it uses the formula in Proposition 2 (and possibly the isomorphism in Proposition 4) and generates two calls of the same algorithm on smaller graphs (i.e. with fewer edges) to compute the partition function for the current graph and stores the result in memory. The recursion continues until the recursion levels equates with the number of nodes in V_3, in which case it checks for free edges, removes them and computes the correction factor $(1 + \lambda)^k$, where k is the number of free edges, and calls the algorithm LeftRecursion to start a new recursion. At this point the graph in the input is bipartite complete and contains only nodes in V_1 and V_2. The latter algorithm behaves very similarly to the former except at the termination step. When all neighbors v_h of w have been whitened the graph no longer contains black nodes, and the corresponding partition function can be directly computed using the formulas in Proposition 3. Note that a different cache function must be used when we call LeftRecursion from RightRecursion (this can be done by in-

```
1: if Cache(w, k_1, k_2) > 0 then
2:   return Cache(w, k_1, k_2)
3: else
4:   if k_1 + k_2 < n then
5:     Let h ← k_1 + k_2 + 1
6:     Cache(w, k_1, k_2) ← (1 + λ) × RightRecursion(G − (v_h, w) −
       v_h, w, k_1, k_2 + 1) − RightRecursion(G − E_G(v_h) − v_h, w, k_1 + 1, k_2)
7:     return Cache(w, k_1, k_2)
8:   else
9:     Let k = |{(u, v) : u ∈ V_4}| be the number of free edges
10:    Remove any edges with an endpoint in V_4
11:    Set V_1 ← V_1 ∪ V_3, V_3 ← ∅
12:    if V_1 is empty then
13:      return 0
14:    end if
15:    Select an arbitrary w' ∈ V_1
16:    return (1 + λ)^k × LeftRecursion(G, w', 0, 0)
17:  end if
18: end if
```

Figure A.17: Algorithm RightRecursion: Takes a graph $G = (V_1, V_2, V_3, V_4, E)$ with $V_3 = \{v_1, \ldots, v_n\}$, $n > 0$, a node $w \in V_4$, and nonnegative integers k_1 and k_2; outputs $Z(G)$.

stantiating an object at that point and passing it as argument; we avoid stating the algorithm is this way to avoid cluttering).

Figure A.19 shows the recursion diagram of a run of RightRecursion. Each box in the figure represents a call of the algorithm with the corresponding graph as input. The left child of each box is the call RightRecursion$(G - (v_h, w) - v_h, w, k_1, k_2 + 1)$, and the right child is the call RightRecursion$(G - E_G(v_h) - v_h, w, k_1 + 1, k_2)$. The number of the graph in each box corresponds to the order in which each call was generated. Solid arcs represent non-cached calls, while dotted arcs indicate cached calls. The recursion diagram for LeftRecursion is shown in Figure A.20 (the semantics is analogous). Note that the recursion of LeftRecursion eventually reaches a graph with no black nodes, for which the partition function can be computed efficiently in closed-form.

Without the caching of computations, the algorithm would perform exponentially many recursive calls (and its corresponding diagram would be a binary tree with exponentially many nodes). The use of caching allows us to compute only one call of RightRecursion for each configuration of k_1, k_2 such that $k_1 + k_2 \leq n$, resulting in at most $\sum_{i=0}^{n}(i+1) = (n+1)(n+2)/2 = O(n^2)$ calls for RightRecursion, where $n = |V_3|$. Similarly, each call of LeftRecursion requires at most $\sum_{i=0}^{m}(i+1) = (m+1)(m+2)/2 = O(m^2)$ recursive calls for LeftRecursion, where $m = |V_2|$. Each call to RightRecursion with $k_1 + k_2 = n$ generates a call to LeftRecursion (there are $n + 1$ such configurations). Hence, the overall number

```
1: if Cache(w, k_1, k_2) is undefined then
2:    if k_1 + k_2 < m then
3:       Let h ← k_1 + k_2 + 1
4:       Cache(w, k_1, k_2) ← (1+λ)×LeftRecursion(G−(u_h, w)−u_h, k_1, k_2+
         1) − LeftRecursion(G − E_G(u_h) − u_h, k_1 + 1, k_2)
5:    else
6:       Cache(w, k_1, k_2) ← (1 + λ)^{|E|}
7:    end if
8: end if
9: return Cache(w, k_1, k_2)
```

Figure A.18: Algorithm LeftRecursion: Takes a bipartite graph $G = (V_1, V_2, E)$ with $V_2 = \{u_1, \ldots, u_m\}$, $m > 0$, a node $w \in V_1$, nonnegative integers k_1 and k_2; outputs $Z(G)$.

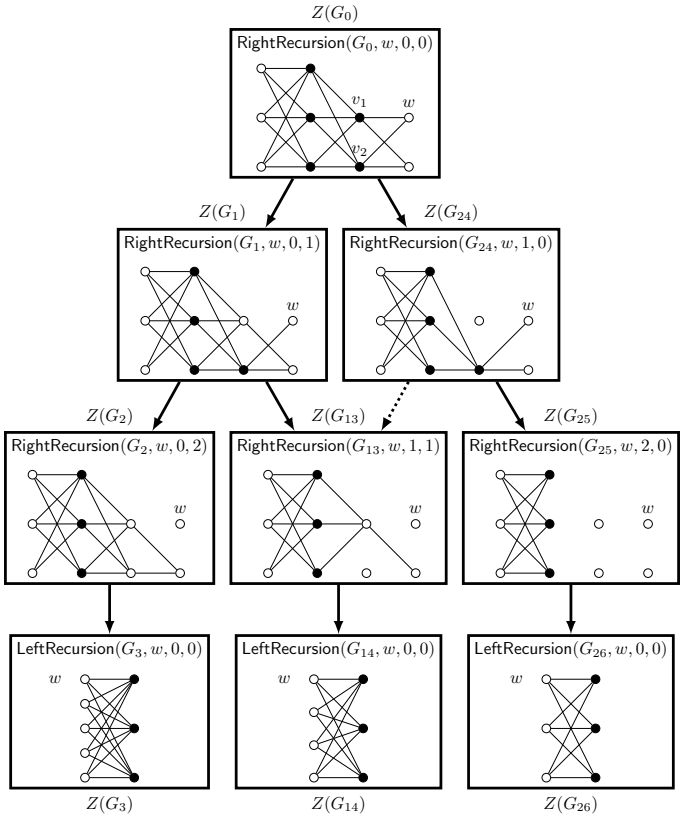


Figure A.19: Example of simulation of RightRecursion$(G_0, w, 0, 0)$.

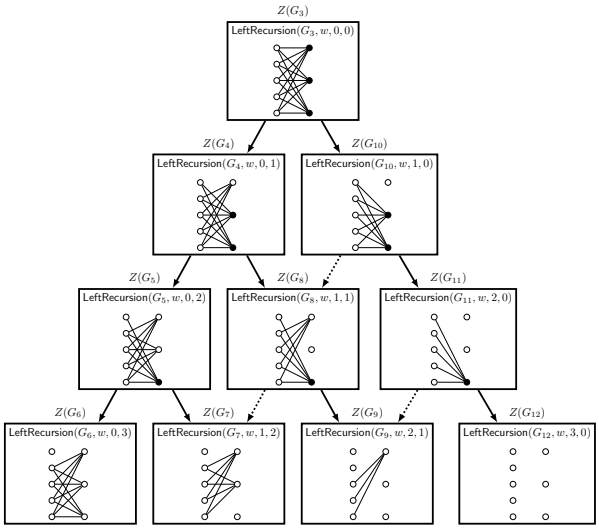


Figure A.20: Example Simulation of LeftRecursion$(G_3, w, 0, 0)$.

of recursions (i.e., call to either function) is

$$\frac{(n+1)(n+2)}{2} + (n+1)\frac{(m+1)(m+2)}{2} = O(n^2 + n \cdot m^2)\,.$$

This leads us to the following result.

Proposition 5. *Let G be a graph in $\mathcal{B}$ with $w \in V_4 \neq \emptyset$. Then the call* RightRecursion$(G, w, 0, 0)$ *outputs $Z(G)$ in time and memory at most cubic in the number of nodes of G.*

Proof. Except when $k_1 + k_2 = n$, RightRecursion calls the recursion given in Proposition 2 with the isomorphisms in Proposition 4 (any graph obtained from G by k_1 operations $-E_G(v_i)$ and k_2 operations $-(w, v_i)$ are isomorpohic). For k_1+k_2, any edge left connecting a node in V_3 and a node in V_4 must be a free edge (since all nodes in V_4 have been whitened), hence they can be removed according to Proposition 3 with the appropriate correction of the partition function. By the same result, any isolated node can be removed. When the remaining nodes in V_3 are transfered to V_1, the resulting graph is bipartite complete (with white nodes in one part and black nodes in the other). Hence, we can call LeftRecursion, which is guaranteed to compute the correct value by the same arguments.

The cubic time and space behavior is due to RightRecursion and LeftRecursion being called at most $O(n^2)$ and $O(nm^2)$, respectively, and by the fact that each call consists of local operations (edge removals and node whitenings) which take at most linear time in the number of nodes and edges of the graph. □

The algorithm RightRecursion requires the existence of a dangling edge. Now it might be that the graph contains no dangling edges; this happens when V_1

and V_4 are empty. The next result shows how to decompose the computation of the partition function into smaller graphs that either contain dangling edges (so that we can apply the previous algorithm), or are also bipartite complete.

Proposition 6. *Let G be a bipartite complete bw-graph with all nodes colored black and $e = (u, v)$ be some edge. Then $Z(G) = (1 + \lambda)Z(G - e - u - v) - Z(G - E_G(v) - v) - Z(G - E_G(u) - u) - Z(G - E_G(u) - E_G(v) - u - v)$.*

Proof. The edge covers of G can be partitioned according to whether they contain the edge e: the weight of edge covers that contain e equal the weight of edge covers that do not contain e up to the factor λ. Thus, $Z(G) = \lambda Z(G - e - u - v) + Z(G - e)$. Consider an edge cover C of $G - e - u - v$, and let $A = E_{G-e}(u) \cap C$ and $B = E_{G-e}(v) \cap C$. We have four cases: if $A \supset \emptyset$ and $B \subset \emptyset$, then C is also an edge cover for $G - e$ with the same weight in either graph; conversely, if C is an edge cover for $G - e$, then it is also an edge cover for $G - e - u - v$ (with equal weight). If $A \supset \emptyset$ and $B = \emptyset$ then C is an edge cover for $G - E_G(v) - v$ with same weight; and any edge cover for $G - E_G(v) - v$ is an edge cover for $G - e - u - v$ with equal weight. If $A = \emptyset$ and $B \supset \emptyset$ then C is an edge cover for $G - E_G(u) - u$ with equal weight; and any edge cover for $G - E_G(u) - u$ is an edge cover for $G - e - u - v$ (same weight). Finally, if $A = B = \emptyset$, then C is an edge cover for $G - E_G(u) - E_G(v) - u - v$, and any cover for this latter graph is an edge cover for $G - e - u - v$. We thus have that $Z(G - e - u - v) = Z(G - e) + Z(G - E_G(v) - v) + Z(G - E_G(u) - u) + Z(G - E_G(u) - E_G(v) - u - v)$. Substituting $Z(G - e)$ into the first equation leads to the desired result. □

In the result above, the graph $G - e - u - v$ contains dangling edges, while the graphs $G - E_G(v) - v$, $G - E_G(u) - u$ and $G - E_G(u) - E_G(v) - u - v$ are bipartite complete. Proposition 4 can be applied to show that altering the edges on which the operations are applied lead to isomorphic graphs. Therefore, a very similar algorithm to LeftRecursion, implementing the recursion in the result above in polynomial time can be easily derived.

Proposition 7. *Let G be a graph in $\mathcal{B}$ (possibly with $V_1 = V_4 = \emptyset$. Then $Z(G)$ can be computed in time and memory at most polynomial in the number of nodes of G.*

Appendix A.5. Plates (Section 7)

Theorem 19. INF[PLATE] *and* QINF[PLATE] *are* PP*-complete with respect to many-one reductions, and* DINF[PLATE] *requires constant computational effort. These results hold* even if the domain size is given in binary notation.

Proof. Consider first INF[PLATES]. To prove membership, take a plate model with relations $X_1, \ldots, X_n$. Suppose we ground this specification on a domain of size N. To compute $\mathbb{P}(\mathbf{Q}|\mathbf{E})$, the only relevant groundings are the ones that are ancestors of each of the ground atoms in $\mathbf{Q} \cup \mathbf{E}$. Our strategy will be to bound the number of such relevant groundings. To do that, take a grounding $X_i(a_1, \ldots, a_{k_i})$ in $\mathbf{Q} \cup \mathbf{E}$, and suppose that X_i is not a root node. Each parent

X_j of X_i may appear once in the definition axiom related to X_i. And each parent of these parents will again have a limited number of parent groundings; in the end there are at most $(n-1)$ relevant groundings that are ancestors of $X_i(a_1, \ldots, a_{k_i})$. We can take the union of all groundings that are ancestors of groundings of $\mathbf{Q} \cup \mathbf{E}$, and the number of such groundings is still polynomial in the size of the input. Thus in polynomial time we can build a polynomially-large Bayesian network that is a fragment of the grounded Bayesian network. Then we can run a Bayesian network inference in this smaller network, an effort within PP; note that each random variable may actually have more than two values and membership to PP is still obtained (that is, if we were to consider "relations" with values other than true and false, the grounding into a Bayesian network would still yield an inference problem within PP). Note also that domain size is actually not important so it can be specified either in unary or binary notation. To prove hardness, note that INF[Prop$(\wedge, \neg)$] is PP-hard, and a propositional specification can be reproduced within PLATES.

Now consider QINF[PLATES]. First, to prove membership, note that even INF[PLATES] is in PP. To prove hardness, reproduce the proof of Theorem 14 by encoding a #3SAT$(>)$ problem, specified by sentence ϕ and integer k, with the definition axioms:

$$\begin{aligned}
\mathsf{clause0}(\mathcal{X}, y, z) &\equiv \neg\mathsf{left}(\mathcal{X}) \vee \neg\mathsf{middle}(y) \vee \neg\mathsf{right}(z),\\
\mathsf{clause1}(\mathcal{X}, y, z) &\equiv \neg\mathsf{left}(\mathcal{X}) \vee \neg\mathsf{middle}(y) \vee \mathsf{right}(z),\\
\mathsf{clause2}(\mathcal{X}, y, z) &\equiv \neg\mathsf{left}(\mathcal{X}) \vee \mathsf{middle}(y) \vee \neg\mathsf{right}(z),\\
&\vdots \quad \vdots \quad \vdots\\
\mathsf{clause7}(\mathcal{X}, y, z) &\equiv \mathsf{left}(\mathcal{X}) \vee \mathsf{middle}(y) \vee \mathsf{right}(z),\\
\mathsf{equal}(\mathcal{X}, y, z) &\equiv \mathsf{left}(\mathcal{X}) \leftrightarrow \mathsf{middle}(y) \leftrightarrow \mathsf{right}(z),
\end{aligned}$$

and $\mathbb{P}(\mathsf{left}(\mathcal{X}) = 1) = \mathbb{P}(\mathsf{middle}(\mathcal{X}) = 1) = \mathbb{P}(\mathsf{right}(\mathcal{X}) = 1) = 1/2$. The resulting plate model is depicted in Figure A.21. The query is again just a set of assignments $\mathbf{Q}$ ($\mathbf{E}$ is empty) containing an assignment per clause. If a clause is $\neg A_2 \vee A_3 \vee \neg A_1$, then take the corresponding assignment $\{\mathsf{clause2}(a_2, a_3, a_1) = 1\}$, and so on. Moreover, add the assignments $\{\mathsf{equal}(a_i, a_i, a_i) = 1\}$ for each $i \in \{1, \ldots, n\}$, to guarantee that left, middle and right have identical truth assignments for all elements of the domain. The #3SAT$(>)$ is solved by deciding whether $\mathbb{P}(\mathbf{Q}) > k/2^n$ with domain of size n; hence the desired hardness is proved.

And DINF[PLATES] requires constant effort: in fact, domain size is not relevant to a fixed inference, as can be seen from the proof of inferential complexity above. □

Appendix A.6. Valiant's counting hierarchy (Section 8)

Theorem 20. *Consider the class of functions that gets as input a relational Bayesian network specification based on* FFFO*, a domain size N (in binary or unary notation), and a set of assignments $\mathbf{Q}$, and returns $\mathbb{P}(\mathbf{Q})$. This class of functions is* #EXP*-equivalent.*

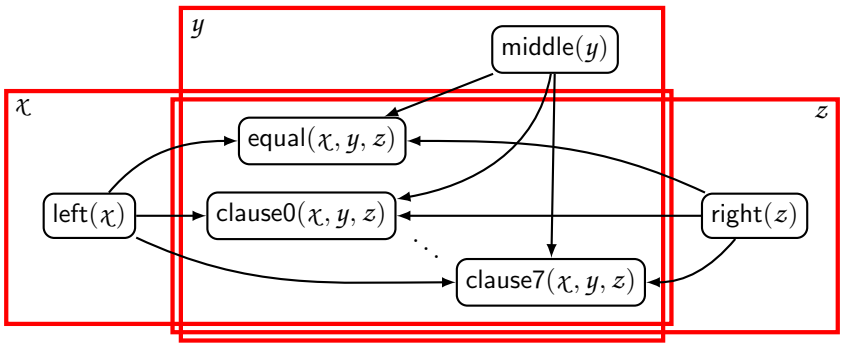


Figure A.21: A plate model that decides a #3SAT(>) problem.

Proof. Build a relational Bayesian network specification as in the proof of Theorem 4. Note that the $p = \mathbb{P}\left(\mathbf{E} \wedge \bigwedge_{i=1}^{6} Z_i\right)$ is the probability that a tiling of the torus is built satisfying all horizontal and vertical restrictions and the initial condition, and moreover containing the accepting state q_a.

If we can recover the number of tilings of the torus from this probability, we obtain the number of accepting computations of the exponential-time Turing machine we started with. Assume we have p. There are 2^{2n} elements in our domain; if the plate model is grounded, there are $2^{2n}(2n+c)$ grounded root random variables, hence there are $2^{2^{2n}(2n+c)}$ interpretations. Hence $p \times 2^{2^{2n}(2n+c)}$ is the number of truth assignments that build the board satisfying all horizontal and vertical constraints and the initial conditions. However, this number is *not* equal to the number of tilings of the board. To see this, consider the grounded Bayesian network where each a in the domain is associated with a "slice" containing groundings $X_i(a)$, $Y_i(a)$, $C_j(a)$ and so on. If a particular configuration of these indicator variables corresponds to a tiling, then we can produce the same tiling by permuting all elements of the domain with respect to the slices of the network. Intuitively, we can fix a tiling and imagine that we are labelling each point of the torus with an element of the domain; clearly every permutation of these labels produces the same tiling (this intuition is appropriate because each a corresponds to a different point in the torus). So, in order to produce the number of tilings of the torus, we must compute $p \times 2^{2^{2n}(2n+c)}/(2^{2n}!)$, where we divide the number of satisfying truth assignments by the number of repeated tilings. □

Theorem 21. *Consider the class of functions that gets as input a relational Bayesian network specification based on* FFFO *with relations with bounded arity, a domain size N in unary notation, and a set of assignments $\mathbf{Q}$, and returns $\mathbb{P}(\mathbf{Q})$. This class of functions is* ♮PSPACE*-equivalent.*

Proof. First we describe a counting Turing machine that produces a count proportional to $\mathbb{P}(\mathbf{Q})$ using a polynomial number of nondeterministic guesses. This nondeterministic machine guesses a truth assignment for each one of the

polynomially-many grounded root nodes (and writes the guess in the working tape). Note that each grounded root node X is associated with an assessment $\mathbb{P}(X = 1) = c/d$, where c and d are integers. The machine must replicate its computation paths to handle such rational assessments exactly as in the proof of Theorem 7. The machine then verifies, in each computation path, whether the guessed truth assignment satisfies $\mathbf{Q}$; if it does, then accept; if not, then reject. Denote by R the number of grounded root nodes and by $\#A$ the number of accepting paths of this machine; then $\mathbb{P}(\mathbf{Q}) = \#A/2^R$.

Now we show that $\mathbf{Q}$ is $\natural$PSPACE-hard with respect to weighted reductions. Define $\varphi(\mathcal{X}_1, \ldots, \mathcal{X}_m)$ to be a quantified Boolean formula with free logvars $\mathcal{X}_1, \ldots, \mathcal{X}_m$:

$$\forall \mathcal{Y}_1 : Q_2 \mathcal{Y}_2 : \ldots Q_M \mathcal{X}_M : \phi(\mathcal{X}_1, \ldots, \mathcal{X}_m),$$

where each logvar can only be true or false, each Q_j is a quantifier (either $\forall$ or $\exists$). And define $\#\varphi$ to be the number of instances of $\mathcal{X}_1, \ldots, \mathcal{X}_m$ such that $\varphi(\mathcal{X}_1, \ldots, \mathcal{X}_m)$ is true. Denote by $\natural$QBF the function that gets a formula $\varphi(\mathcal{X}_1, \ldots, \mathcal{X}_m)$ and returns $\#\varphi$; Ladner shows that $\natural$QBF is $\natural$PSPACE-complete [75, Theorem 5(2)]. So, adapt the hardness proof of Theorem 10: introduce the definition axiom

$$Y \equiv\equiv \forall \mathcal{Y}_1 : \ldots Q_m \mathcal{Y}_m : \phi'(X_1, \ldots, X_m),$$

where ϕ' has the same structure of ϕ but logvars are replaced as follows. First, each $\mathcal{X}_j$ is replaced by a relation X_j of arity zero (that is, a proposition). Second, each logvar $\mathcal{Y}_j$ is replaced by the atom $X(\mathcal{Y}_j)$ where X is a fresh unary relation. These relations are associated with assessments $\mathbb{P}(X_j = 1) = 1/2$ and $\mathbb{P}(X(\mathcal{X}) = 1) = 1/2$. This completes the relational Bayesian network specification. Now for domain $\{0, 1\}$, first compute $\mathbb{P}(\mathbf{Q})$ for $\mathbf{Q} = \{Y = 1, X(0) = 0, X(1) = 1\}$ and then compute $2^m(\mathbb{P}(\mathbf{Q})/(1/4))$. The latter number is the desired value of $\natural$QBF; note that $\mathbb{P}(\mathbf{Q})/(1/4) = \mathbb{P}(Y = 1 | X(0) = 0, X(1) = 1)$. □

Theorem 22. *Consider the class of functions that gets as input a relational Bayesian network specification based on* FFFO^k *for* $k \geq 2$*, a domain size* N *in unary notation, and a set of assignments* $\mathbf{Q}$*, and returns* $\mathbb{P}(\mathbf{Q})$*. This class of functions is* #P*-equivalent.*

Proof. Hardness is trivial: even Prop$(\wedge, \neg)$ is #P-equivalent, as Prop$(\wedge, \neg)$ suffices to specify any propositional Bayesian network, and equivalence is then obtained [113]. To prove membership, use the Turing machine described in the proof of membership in Theorem 11 without assignments $\mathbf{E}$ (that is, the machine only processes $\mathbf{Q}$), without the final computations in Park's construction. This machine produces the number $\#A$ of computation paths that satisfy $\mathbf{Q}$; then return $\#A/2^R$, where R is the number of grounded root nodes. □

Theorem 23. *Consider the class of functions that get as input a plate model based on* FFFO*, a domain size* N *(either in binary or unary notation), and a set of assignments* $\mathbf{Q}$*, and returns* $\mathbb{P}(\mathbf{Q})$*. This class of functions is* #P*-equivalent.*

Proof. Hardness is trivial: a propositional Bayesian network can be encoded with a plate model. To prove membership, build the same fragment of the grounded Bayesian network as described in the proof of Theorem 19: inference with the plate model is then reduced to inference with this polynomially large Bayesian network. □

References

[1] A. Artale, D. Calvanese, R. Kontchakov, M. Zakharyashev, The *DL-Lite* family and relations, Journal of Artificial Intelligence Research 36 (2009) 1–69.

[2] F. Baader, Terminological cycles in a description logic with existential restrictions, in: IJCAI, 2003, pp. 325–330.

[3] F. Baader, W. Nutt, Basic description logics, in: Description Logic Handbook, Cambridge University Press, 2002, pp. 47–100.

[4] F. Bacchus, Representing and Reasoning with Probabilistic Knowledge: A Logical Approach, MIT Press, Cambridge, 1990.

[5] F. Bacchus, Using first-order probability logic for the construction of Bayesian networks, in: Conference on Uncertainty in Artificial Intelligence, 1993, pp. 219–226.

[6] D. D. Bailey, V. Dalmau, P. G. Kolaitis, Phase transitions of PP-complete satisfiability problems, Discrete Applied Mathematics 155 (2007) 1627–1639.

[7] M. Bauland, E. Bohler, N. Creignou, S. Reith, H. Schnoor, H. Vollmer, The complexity of problems for quantified constraints, Theory of Computing Systems 47 (2010) 454–490.

[8] P. Beame, G. Van den Broeck, E. Gribkoff, D. Suciu, Symmetric weighted first-order model counting, in: ACM Symposium on Principles of Database Systems (PODS), 2015.

[9] O. Benjelloun, A. D. Sarma, A. Halevy, M. Theobald, J. Widom, Databases with uncertainty and lineage, The International Journal on Very Large Data Bases 17 (2) (2008) 243–264.

[10] D. M. Blei, A. Y. Ng, M. I. Jordan, Latent Dirichlet allocation, Journal of Machine Learning Research 3 (2003) 993–1022.

[11] R. Bubley, M. Dyer, Graph orientations with no sink and an approximation for a hard case of #SAT, in: ACM-SIAM Symposium on Discrete Algorithms, 1997, pp. 248–257.

[12] D. Buchman, D. Poole, Negative probabilities in probabilistic logic programs, International Journal of Approximate Reasoning 83 (2017) 43–59.

[13] D. Buchman, D. Poole, Why rules are complex: real-valued probabilistic logic programs are not fully expressive, in: Conference on Uncertainty in Artificial Intelligence, 2017.

[14] H. Buhrman, L. Fortnow, T. Thierauf, Nonrelativizing separations, in: Proceedings of IEEE Complexity, 1998, pp. 8–12.

[15] A. Bulatov, M. Dyer, L. A. Goldberg, M. Jalsenius, M. Jerrum, D. Richerby, The complexity of weighted and unweighted #CSP, Journal of Computer and System Sciences 78 (2012) 681–688.

[16] W. L. Buntine, Operations for learning with graphical models, Journal of Artificial Intelligence Research 2 (1994) 159–225.

[17] J.-Y. Cai, P. Lu, M. Xia, Holographic reduction, interpolation and hardness, Computational Complexity 21 (4) (2012) 573–604.

[18] D. Calvanese, G. D. Giacomo, D. Lembo, M. Lenzerini, R. Rosati, DL-Lite: Tractable description logics for ontologies, in: AAAI, 2005, pp. 602–607.

[19] D. Calvanese, G. D. Giacomo, D. Lembo, M. Lenzerini, R. Rosati, Data complexity of query answering in description logics, in: Knowledge Representation, 2006, pp. 260–270.

[20] R. N. Carvalho, K. B. Laskey, , P. C. Costa, PR-OWL 2.0 — bridging the gap to OWL semantics, in: URSW 2008-2010/UniDL 2010, LNAI 7123, 2013, pp. 1–18.

[21] I. I. Ceylan, R. Peñaloza, The Bayesian description logic $\mathcal{BEL}$, in: International Joint Conference on Automated Reasoning, 2014, pp. 480–494.

[22] M. Chavira, A. Darwiche, On probabilistic inference by weighted model counting, Artificial Intelligence 172 (6-7) (2008) 772–799.

[23] P. C. G. Costa, K. B. Laskey, PR-OWL: A framework for probabilistic ontologies, in: Conference on Formal Ontology in Information Systems, 2006.

[24] V. S. Costa, D. Page, M. Qazi, J. Cussens, CLP($\mathcal{BN}$): Constraint logic programming for probabilistic knowledge, in: U. Kjaerulff, C. Meek (eds.), Conference on Uncertainty in Artificial Intelligence, Morgan-Kaufmann, 2003, pp. 517–524.

[25] F. G. Cozman, D. D. Mauá, Bayesian networks specified using propositional and relational constructs: Combined, data, and domain complexity, in: AAAI Conference on Artificial Intelligence, 2015.

[26] F. G. Cozman, D. D. Mauá, On the semantics and complexity of probabilistic logic programs, Journal of Artificial Intelligence Research 60 (2017) 221–262.

[27] F. G. Cozman, R. B. Polastro, Complexity analysis and variational inference for interpretation-based probabilistic description logics, in: Proceedings of the Conference on Uncertainty in Artificial Intelligence, AUAI Press, 2009, pp. 117–125.

[28] N. Dalvi, D. Suciu, Efficient query evaluation on probabilistic databases, VLDB Journal, 16 (2007) 523–544.

[29] N. Dalvi, D. Suciu, The dichotomy of probabilistic inference for unions of conjunctive queries, Journal of the ACM 59 (6) (2012) 30:1–30:87.

[30] C. d'Amato, N. Fanizzi, T. Lukasiewicz, Tractable reasoning with Bayesian description logics, in: International Conference on Scalable Uncertainty Management, 2008, pp. 146–159.

[31] A. Darviche, G. Provan, Query DAGSs: A practical paradigm for implementing belief-network inference, in: E. Horvitz, F. Jensen (eds.), Conference on Uncertainty in Artificial Intelligence, Morgan Kaufmann, 1996, pp. 203–210.

[32] A. Darwiche, A differential approach to inference in Bayesian networks, Journal of the ACM 50 (3) (2003) 280–305.

[33] A. Darwiche, Modeling and Reasoning with Bayesian Networks, Cambridge University Press, 2009.

[34] A. Darwiche, P. Marquis, A knowledge compilation map, Journal of Artificial Intelligence Research 17 (2002) 229–264.

[35] G. De Bona, F. G. Cozman, On the coherence of probabilistic relational formalisms, Entropy 20 (2018) 229/1–229/24.

[36] Z. Ding, Y. Peng, R. Pan, BayesOWL: Uncertainty modeling in semantic web ontologies, in: Soft Computing in Ontologies and Semantic Web, vol. 204 of Studies in Fuzziness and Soft Computing, Springer, Berlin/Heidelberg, 2006, pp. 3–29.

[37] P. Domingos, W. A. Webb, A tractable first-order probabilistic logic, in: AAAI, 2012, pp. 1902–1909.

[38] A. Durand, M. Hermann, P. G. Kolaitis, Subtractive reductions and complete problems for counting complexity classes, Theoretical Computer Science 340 (3) (2005) 496–513.

[39] D. Fierens, H. Blockeel, M. Bruynooghe, J. Ramon, Logical Bayesian networks and their relation to other probabilistic logical models, in: Int. Conference on Inductive Logic Pogramming, 2005, pp. 121–135.

[40] D. Fierens, H. Blockeel, J. Ramon, M. Bruynooghe, Logical Bayesian networks, in: Workshop on Multi-Relational Data Mining, 2004, pp. 19–30.

[41] D. Fierens, G. Van den Broeck, J. Renkens, D. Shrerionov, B. Gutmann, G. Janssens, L. de Raedt, Inference and learning in probabilistic logic programs using weighted Boolean formulas, Theory and Practice of Logic Programming 15 (3) (2014) 358–401.

[42] J. Flum, M. Grohe, The parameterized complexity of counting problems, SIAM Journal of Computing 33 (4) (2004) 892–922.

[43] N. Friedman, L. Getoor, D. Koller, A. Pfeffer, Learning probabilistic relational models, in: International Joint Conference on Artificial Intelligence, 1999, pp. 1300–1309.

[44] L. Getoor, N. Friedman, D. Koller, A. Pfeffer, B. Taskar, Probabilistic relational models, in: Introduction to Statistical Relational Learning, 2007.

[45] L. Getoor, J. Grant, PRL: A probabilistic relational language, Machine Learning 62 (2006) 7–31.

[46] L. Getoor, B. Taskar, Introduction to Statistical Relational Learning, MIT Press, 2007.

[47] W. Gilks, A. Thomas, D. Spiegelhalter, A language and program for complex Bayesian modelling, The Statistician 43 (1993) 169–178.

[48] J. Gill, Computational complexity of probabilistic Turing machines, SIAM Journal on Computing 6 (4) (1977) 675–695.

[49] S. Glesner, D. Koller, Constructing flexible dynamic belief networks from first-order probabilistic knowledge bases, in: Symbolic and Quantitative Approaches to Reasoning with Uncertainty, 1995, pp. 217–226.

[50] R. P. Goldman, E. Charniak, Dynamic construction of belief networks, in: Conference of Uncertainty in Artificial Intelligence, 1990, pp. 90–97.

[51] J. Goldsmith, M. Hagen, M. Mundhenk, Complexity of DNF minimization and isomorphism testing for monotone formulas, Information and Computation 206 (6) (2008) 760–775.

[52] E. Grädel, Finite model theory and descriptive complexity, in: Finite Model Theory and its Applications, Springer, 2007, pp. 125–229.

[53] E. Gribkoff, G. Van den Broeck, D. Suciu, Understanding the complexity of lifted inference and asymmetric weighted model counting, in: Conference on Uncertainty in Artificial Intelligence, 2014.

[54] A. Grove, J. Halpern, D. Koller, Asymptotic conditional probabilities: the unary case, SIAM Journal on Computing 25 (1) (1996) 1–51.

[55] P. Haddawy, Generating Bayesian networks from probability logic knowledge, in: Conference on Uncertainty in Artificial Intelligence, 1994, pp. 262–269.

[56] Joseph Y. Halpern, Reasoning about Uncertainty, MIT Press, 2003.

[57] D. Heckerman, An empirical comparison of three inference methods, in: R. D. Shachter, L. N. Kanal, J. F. Lemmer (eds.), Uncertainty in Artificial Intelligence 4, Elsevier Science Publishers, North-Holland, 1990, pp. 283–303.

[58] D. Heckerman, C. Meek, D. Koller, Probabilistic entity-relationship models, PRMs, and plate models, in: L. Getoor, B. Taskar (eds.), Introduction to Statistical Relational Learning, MIT Press, 2007, pp. 201–238.

[59] M. C. Horsch, D. Poole, A dynamic approach to probabilistic inference using Bayesian networks, in: Conference of Uncertainty in Artificial Intelligence, 1990, pp. 155–161.

[60] M. Jaeger, Relational Bayesian networks, in: D. Geiger, P. P. Shenoy (eds.), Conference on Uncertainty in Artificial Intelligence, Morgan Kaufmann, San Francisco, California, 1997, pp. 266–273.

[61] M. Jaeger, Complex probabilistic modeling with recursive relational Bayesian networks, Annals of Mathematics and Artificial Intelligence 32 (2001) 179–220.

[62] M. Jaeger, Probabilistic role models and the guarded fragment, in: Information Processing and Management of Uncertainty in Knowledge-Based Systems (IPMU), 2004, pp. 235–242.

[63] M. Jaeger, Lower complexity bounds for lifted inference, Theory and Practice of Logic Programming 15 (2) (2014) 246–264.

[64] M. Jaeger, G. Van Den Broeck, Liftability of probabilistic inference: Upper and lower bounds, in: 2nd Statistical Relational AI (StaRAI-12) Workshop, 2012.

[65] S. M. Kazemi, A. Kimmig, G. Van den Broeck, D. Poole, New liftable classes for first-order probabilistic inference, in: NIPS, 2016.

[66] K. Kersting, Lifted probabilistic inference, in: L. D. Raedt, C. Bessiere, D. Dubois, P. Doherty, P. Frasconi, F. Heintz, P. Lucas (eds.), European Conference on Artificial Intelligence, IOS Press, 2012.

[67] K. Kersting, L. D. Raedt, S. Kramer, Interpreting Bayesian logic programs, in: AAAI-2000 Workshop on Learning Statistical Models from Relational Data, 2000.

[68] C. Koch, MayBMS: A system for managing large uncertain and probabilistic databases, in: C. C. Aggarwal (ed.), Managing and Mining Uncertain Data, Springer-Verlag, 2009, pp. 149–184.

[69] D. Koller, N. Friedman, Probabilistic Graphical Models: Principles and Techniques, MIT Press, 2009.

[70] D. Koller, A. Y. Levy, A. Pfeffer, P-CLASSIC: A tractable probabilistic description logic, in: AAAI, 1997, pp. 390–397.

[71] D. Koller, A. Pfeffer, Object-oriented Bayesian networks, in: Conference on Uncertainty in Artificial Intelligence, 1997, pp. 302–313.

[72] D. Koller, A. Pfeffer, Probabilistic frame-based systems, in: National Conference on Artificial Intelligence (AAAI), 1998, pp. 580–587.

[73] J. Kwisthout, The computational complexity of probabilistic inference, Tech. Rep. ICIS–R11003, Radboud University Nijmegen, The Netherlands (2011).

[74] J. H. P. Kwisthout, H. L. Bodlaender, L., V. der Gaag, The necessity of bounded treewidth for efficient inference in Bayesian networks, in: European Conference on Artificial Intelligence, 2010, pp. 237–242.

[75] R. E. Ladner, Polynomial space counting problems, SIAM Journal of Computing 18 (6) (1989) 1087–1097.

[76] K. B. Laskey, MEBN: A language for first-order Bayesian knowledge bases, Artificial Intelligence 172 (2-3) (2008) 140–178.

[77] H. R. Lewis, Complexity results for classes of quantificational formulas, Journal of Computer and System Sciences 21 (1980) 317–353.

[78] L. Libkin, Elements of Finite Model Theory, Springer, 2004.

[79] C. Lin, J. Liu, P. Lu, A simple FPTAS for counting edge covers, in: ACM-SIAM Symposium on Discrete Algorithms, 2014, pp. 341–348.

[80] J. Liu, P. Lu, C. Zhang, FPTAS for counting weighted edge covers, in: European Symposium on Algorithms, 2014, pp. 654–665.

[81] T. Lukasiewicz, U. Straccia, Managing uncertainty and vagueness in description logics for the semantic web, Journal of Web Semantics 6 (2008) 291–308.

[82] D. Lunn, C. Jackson, N. Best, A. Thomas, D. Spiegelhalter, The BUGS Book: A Practical Introduction to Bayesian Analysis, CRC Press/Chapman and Hall, 2012.

[83] D. Lunn, D. Spiegelhalter, A. Thomas, N. Best, The BUGS project: Evolution, critique and future directions, Statistics in Medicine 28 (2009) 3049–3067.

[84] S. Mahoney, K. B. Laskey, Network engineering for complex belief networks, in: Conference on Uncertainty in Artificial Intelligence, 1996.

[85] V. Mansinghka, A. Radul, CoreVenture: a highlevel, reflective machine language for probabilistic programming, in: NIPS Workshop on Probabilistic Programming, 2014.

[86] D. D. Mauá, F. G. Cozman, The effect of combination functions on the complexity of relational Bayesian networks, International Journal of Approximate Reasoning 85 (2017) 178–195.

[87] D. D. Mauá, C. P. de Campos, F. G. Cozman, The complexity of MAP inference in Bayesian networks specified through logical languages, in: International Joint Conference on Artificial Intelligence, 2015, pp. 889–895.

[88] D. D. Mauá, F. G. Cozman, A tractable class of model counting problems, in: Encontro Nacional de Inteligência Artificial, 2015.

[89] B. Milch, B. Marthi, D. Sontag, S. Russell, D. L. Ong, A. Kolobov, BLOG: Probabilistic models with unknown objects, in: IJCAI, 2005.

[90] B. Milch, L. S. Zettlemoyer, K. Kersting, M. Haimes, L. P. Kaelbling, Lifted probabilistic inference with counting formulas, in: AAAI, 2008, pp. 1062–1068.

[91] T. Mitchell, W. Cohen, E. Hruschka, P. Talukdar, J. Betteridge, A. Carlson, B. Dalvi, M. Gardner, B. Kisiel, J. Krishnamurthy, N. Lao, K. Mazaitis, T. Mohamed, N. Nakashole, E. Platanios, A. Ritter, M. Samadi, B. Settles, R. Wang, D. Wijaya, A. Gupta, X. Chen, A. Saparov, M. Greaves, J. Welling, Never-ending learning, in: Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI-15), 2015.

[92] R. E. Neapolitan, Learning Bayesian Networks, Prentice Hall, 2003.

[93] L. Ngo, P. Haddawy, Answering queries from context-sensitive probabilistic knowledge bases, Theoretical Computer Science 171 (1–2) (1997) 147–177.

[94] M. Niepert, G. Van den Broeck, Tractability through exchangeability: a new perspective on efficient probabilistic inference, in: AAAI Conference on Artificial Intelligence, 2014, pp. 2467–2475.

[95] M. Ogiwara, A complexity theory for feasible closure properties, Journal of Computer and System Sciences 46 (1993) 295–325.

[96] C. H. Papadimitriou, A note on succinct representations of graphs, Information and Control 71 (1986) 181–185.

[97] C. H. Papadimitriou, Computational Complexity, Addison-Wesley Publishing, 1994.

[98] J. Pearl, Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference, Morgan Kaufmann, San Mateo, California, 1988.

[99] J. Pearl, Causality: Models, Reasoning, and Inference (2nd edition), Cambridge University Press, Cambridge, United Kingdom, 2009.

[100] A. Pfeffer, IBAL: a probabilistic rational programming language, in: International Joint Conference on Artificial Intelligence, 2001, pp. 733–740.

[101] D. Poole, Probabilistic Horn abduction and Bayesian networks, Artificial Intelligence 64 (1993) 81–129.

[102] D. Poole, The independent choice logic for modelling multiple agents under uncertainty, Artificial Intelligence 94 (1/2) (1997) 7–56.

[103] D. Poole, First-order probabilistic inference, in: International Joint Conference on Artificial Intelligence (IJCAI), 2003, pp. 985–991.

[104] D. Poole, The Independent Choice Logic and beyond, in: L. D. Raedt, P. Frasconi, K. Kersting, S. Muggleton (eds.), Probabilistic Inductive Logic Programming, vol. 4911 of Lecture Notes in Computer Science, Springer, 2008, pp. 222–243.

[105] D. Poole, Probabilistic programming languages: Independent choices and deterministic systems, in: R. Dechter, H. Geffner, J. Y. Halpern (eds.), Heuristics, Probability and Causality — A Tribute to Judea Pearl, College Publications, 2010, pp. 253–269.

[106] H. Poon, P. Domingos, Sum-product networks: a new deep architecture, in: Conference on Uncertainty in Artificial Intelligence, 2011.

[107] O. Pourret, P. Naim, B. Marcot, Bayesian Networks — A Practical Guide to Applications, Wiley, 2008.

[108] J. S. Provan, M. O. Ball, The complexity of counting cuts and of computing the probability that a graph is connected, SIAM Journal on Computing 12 (4) (1983) 777–788.

[109] L. D. Raedt, Logical and Relational Learning, Springer, 2008.

[110] L. D. Raedt, K. Kersting, Probabilistic inductive logic programming, in: International Conference on Algorithmic Learning Theory, 2004, pp. 19–36.

[111] R. Ramachandran, G. Qi, K. Wang, J. Wang, J. Thornton, Probabilistic reasoning in DL-Lite, in: Pacific Rim International Conference on Trends in Artificial Intelligence, 2012, pp. 480–491.

[112] F. Riguzzi, The distribution semantics is well-defined for all normal programs, in: F. Riguzzi, J. Vennekens (eds.), International Workshop on Probabilistic Logic Programming, vol. 1413 of CEUR Workshop Proceedings, 2015, pp. 69–84.

[113] D. Roth, On the hardness of approximate reasoning, Artificial Intelligence 82 (1-2) (1996) 273–302.

[114] S. Sanner, D. McAllester, Affine algebraic decision diagrams (AADDs) and their application to structured probabilistic inference, in: International Joint Conference on Artificial Intelligence, 2005, pp. 1384–1390.

[115] T. Sato, A statistical learning method for logic programs with distribution semantics, in: Int. Conference on Logic Programming, 1995, pp. 715–729.

[116] T. Sato, Y. Kameya, Parameter learning of logic programs for symbolic-statistical modeling, Journal of Artificial Intelligence Research 15 (2001) 391–454.

[117] J. Simon, On some central problems in computational complexity, Tech. Rep. TR75-224, Department of Computer Science, Cornell University (1975).

[118] S. Singh, C. Mayfield, S. Mittal, S. Prabhakar, S. Hambrusch, R. Shah, Orion 2.0: Native support for uncertain data, in: SIGMOD, 2008, pp. 1239–1241.

[119] T. Sommestad, M. Ekstedt, P. Johnson, A probabilistic relational model for security risk analysis, Computers and Security 29 (2010) 659–679.

[120] D. Sontag, D. Roy, Complexity of inference in latent Dirichlet allocation, in: Advances in Neural Information Processing Systems, vol. 24, 2011, pp. 1008–1016.

[121] D. Suciu, D. Oiteanu, C. Ré, C. Koch, Probabilistic Databases, Morgan & Claypool Publishers, 2011.

[122] N. Taghipour, D. Fierens, G. Van den Broeck, J. Davis, H. Blockeel, Completeness results for lifted variable elimination, in: Conference on Artificial Intelligence and Statistics (AISTATS), Scottsdale, USA, 2013, pp. 572–580.

[123] M. Tenorth, M. Beetz, KnowRob: A knowledge processing infrastructure for cognition-enabled robots, The International Journal of Robotics Research 32 (5) (2013) 566–590.

[124] S. Tobies, The complexity of reasoning with cardinality restrictions and nominals in expressive description logics, Journal of Artificial Intelligence Research 12 (2000) 199–217.

[125] S. Toda, PP is as hard as the polynomial-time hierarchy, SIAM Journal of Computing 20 (5) (1991) 865–877.

[126] S. Toda, O. Watanabe, Polynomial-time 1-Turing reductions from #PH to #P, Theoretical Computer Science 100 (1992) 205–221.

[127] J. Tóran, Complexity classes defined by counting quantifiers, Journal of the ACM 38 (3) (1991) 753–774.

[128] S. P. Vadhan, The complexity of counting in sparse, regular and planar graphs, SIAM Journal of Computing 31 (2) (2001) 398–427.

[129] L. G. Valiant, The complexity of computing the permanent, Theoretical Computer Science 8 (1979) 189–201.

[130] L. G. Valiant, The complexity of enumeration and reliability problems, SIAM Journal of Computing 8 (3) (1979) 410–421.

[131] L. G. Valiant, Accidental algorithms, in: Annual IEEE Symposium on Foundations of Computer Science, 2006, pp. 509–517.

[132] G. Van den Broeck, On the completeness of first-order knowledge compilation for lifted probabilistic inference, in: Neural Processing Information Systems, 2011, pp. 1386–1394.

[133] G. Van den Broeck, J. Davis, Conditioning in first-order knowledge compilation and lifted probabilistic inference, in: AAAI Conference on Artificial Intelligence, 2012, pp. 1961–1967.

[134] G. Van den Broeck, M. Wannes, A. Darwiche, Skolemization for weighted first-order model counting, in: International Conference on Principles of Knowledge Representation and Reasoning, 2014, pp. 111–120.

[135] M. Y. Vardi, The complexity of relational query languages, in: Annual ACM Symposium on Theory of Computing, 1982, pp. 137–146.

[136] K. W. Wagner, The complexity of combinatorial problems with succinct input representation, Acta Informatica 23 (1986) 325–356.

[137] D. Z. Wang, E. Michelaks, M. Garofalakis, J. M. Hellerstein, BayesStore: Managing large, uncertain data repositories with probabilistic graphical models, in: VLDB Endowment, vol. 1, 2008, pp. 340–351.

[138] M. P. Wellman, J. S. Breese, R. P. Goldman, From knowledge bases to decision models, Knowledge Engineering Review 7 (1) (1992) 35–53.

[139] J. Widom, Trio: A system for data, uncertainty, and lineage, in: C. C. Aggarwal (ed.), Managing and Mining Uncertain Data, Springer-Verlag, 2009, pp. 113–148.

[140] M. Xia, W. Zhao, #3-regular bipartite planar vertex cover is #P-complete, in: Theory and Applications of Models of Computation, vol. 3959, 2006, pp. 356–364.