Languages for Probabilistic Modeling Over Structured and Relational Domains

Fabio Gagliardi Cozman

Abstract In this chapter we survey languages that specify probability distributions using graphs, predicates, quantifiers, fixed-point operators, recursion, and other logical and programming constructs. Many of these languages have roots both in probabilistic logic and in the desire to enhance Bayesian networks and Markov random fields. We examine their origins and comment on various proposals up to recent developments in probabilistic programming.

1 Introduction

Diversity is a mark of research in artificial intelligence (AI). From its inception, the field has exercised freedom in pursuing formalisms to handle monotonic, nonmonotonic, uncertain, and fuzzy inferences. Some of these formalisms have seen cycles of approval and disapproval; for instance, probability theory was taken, in 1969, to be "epistemologically inadequate" by leading figures in AI (McCarthy and Hayes 1969). At that time there was skepticism about combinations of probability and logic, even though such a combination had been under study for more than a century.

A turning point in the debate on the adequacy of probability theory to AI was Judea Pearl's development of Bayesian networks (Pearl 1988). From there many other models surfaced, based on the notion that we must be able to specify probability distributions in a modular fashion through independence relations (Sadeghi and Lauritzen 2014). In spite of their flexibility, Bayesian networks are "propositional" in the sense that random variables are not parameterized, and one cannot quantify over them. For instance, if you have 1000 individuals in a city, and for each of them you are interested in three random variables (say education, income, and age), then you must explicitly specify 3000 random variables and their independence relations.

There have been many efforts to extend graphical models so as to allow one to encode repetitive patterns, using logical variables, quantification, recursion, loops, and the like. There are specification languages based on database schema, on first-

F. G. Cozman (✉)
University of São Paulo, São Paulo, Brazil
e-mail: fgcozman@usp.br

P. Marquis et al. (eds.), *A Guided Tour of Artificial Intelligence Research*,
https://doi.org/10.1007/978-3-030-06167-8_9

order logic, on logic programming, on functional programming, even on procedural programming. Often these languages employ techniques from seminal probabilistic logics. The purpose of this chapter is to review some of these languages, starting with a brief review of probabilistic logic concepts, and then moving to relational variants of Bayesian networks and to probabilistic programming.

In Sect. 2 we present some foundational results from probabilistic logic, so as to fix useful terminology concerning syntax and semantics. In Sect. 3 we look at several formalisms that mix, using graphs, Bayesian networks and relational modeling. Section 4 is devoted to probabilistic logic programming; in Sect. 5 we go through languages inspired by various logics, and in Sect. 6 we examine Markov random fields. In Sect. 7 we consider the emerging field of probabilistic programming. Finally, in Sect. 8 we very briefly mention some inference and learning techniques.

2 Probabilistic Logics: The Laws of Thought?

Boole's book on The Laws of Thought is aptly sub-titled "on which are founded the mathematical theories of logic and probabilities" (Boole 1958). Starting from that effort, many other thinkers examined the combination of first-order logic and probabilities (Gaifman 1964; Gaifman and Snir 1982; Hoover 1978; Keisler 1985; Scott and Krauss 1966). A mix of logical and probabilistic reasoning was also central to de Finetti's concept of probability (Coletti and Scozzafava 2002; de Finetti 1964). Nilsson (1986) later rediscovered some of these ideas in the context of AI, in particular emphasizing linear programming methods that had been touched before (Bruno and Gilio 1980; Hailperin 1976).

At the beginning of the nineties, Nilsson's probabilistic logic and its extensions were rather popular amongst AI researchers (Hansen and Jaumard 1996); in particular, probabilistic first-order logic received sustained attention (Bacchus 1990; Fagin et al. 1990; Halpern 2003). That feverish work perhaps convinced some that the laws of thought had indeed been nailed down.

We now review some concepts used in probabilistic logics, as they are relevant to the remainder of this survey.

Propositional logic consists of formulas containing *propositions* $A_1, A_2, \ldots$, and the *Boolean operators* $\wedge$ (conjunction), $\vee$ (disjunction), $\neg$ (negation), $\rightarrow$ (implication) and $\leftrightarrow$ (equivalence); further details are discussed in chapter "Reasoning with Propositional Logic - From SAT Solvers to Knowledge Compilation" of this Volume. First-order logic consists of formulas containing predicates and functions, plus the Boolean operators, logical variables and existential/universal quantifiers (further details are discussed in chapter "Automated Deduction" of this Volume). Any predicate r, and any function f, is associated with a nonnegative integer, its *arity*. A predicate of arity zero is treated as a proposition. A function of arity zero is a *constant*. A *term* is either a logical variable or a constant. A predicate of arity k, followed by k terms (usually in parenthesis) is an *atom*. For instance, if baker is a predicate of arity 1, then both $\mathsf{baker}(x)$ and $\mathsf{baker}(\mathsf{John})$ are atoms.

The syntax of a minimal propositional probabilistic logic is rather simple: we can have any propositional formula φ, and moreover any *probabilistic assessment* $\mathbb{P}(\phi) = \alpha$, where ϕ is a propositional formula and α is a number in $[0, 1]$. For instance, both $A_1 \wedge \neg A_2$ and $\mathbb{P}(\neg A_3 \vee A_4 \vee \neg A_5) = 1/2$ are well-formed formulas. Conditional probabilistic assessments are often allowed; that is, $\mathbb{P}(\phi|\theta) = \alpha$ where ϕ and θ are propositional formulas.

Example 1 Suppose A_1 means "Tweety is a penguim", A_2 means "Tweety is a bird", and A_3 means "Tweety flies". Then

$$A_1 \rightarrow A_2, \qquad A_1 \rightarrow \neg A_3, \qquad \mathbb{P}(A_3|A_2) = 0.95$$

is a set of well-formed formulas. □

The usual semantics associated with this propositional syntax is as follows. Suppose we have propositions $A_1, A_2, \ldots, A_n$. There are 2^n *truth assignments* to these propositions (each proposition assigned true or false). To simplify the terminology, we refer to a truth assignment as an *interpretation*. Propositional probabilistic logic focuses on probability measures over interpretations, where the sample space Ω is the set of 2^n interpretations. Recall that such a measure $\mathbb{P}$ can be specified by associating each element of Ω with a nonnegative number in $[0, 1]$, guaranteeing that these numbers add up to one (as discussed in chapter "Representations of Uncertainty in Artificial Intelligence: Probability and Possibility" of Volume 1).

A probabilistic assessment $\mathbb{P}(\phi) = \alpha$, for some $\alpha \in [0, 1]$ is interpreted as a constraint; namely, the constraint that the probability of the set of interpretations satisfying ϕ is exactly α. A conditional probability assessment $\mathbb{P}(A_1|A_2) = \alpha$ is usually read as the constraint $\mathbb{P}(A_1 \wedge A_2) = \alpha \mathbb{P}(A_2)$.

Example 2 Consider a (rather simplified) version of Boole's challenge problem (Hailperin 1996): we have $\mathbb{P}(A_1) = \alpha_1$ and $\mathbb{P}(A_2) = \alpha_2$, and moreover $\mathbb{P}(A_3|A_1) = \beta_1$ and $\mathbb{P}(A_3|A_2) = \beta_2$; finally we know that $A_3 \rightarrow (A_1 \vee A_2)$. What are the possible values for $\mathbb{P}(A_3)$?

There are three propositions A_1, A_2, A_3; an interpretation can be encoded as a triple $(a_1 a_2 a_3)$ where $a_i = 1$ when A_i is true, and $a_i = 0$ otherwise. There are 8 interpretations that might be possible, but interpretation (001) is impossible because $A_3 \rightarrow (A_1 \vee A_2)$ holds. Each interpretation is to have a probability; we denote by p_j the probability of the jth interpretation, where we order lexicographically the triple $(a_1 a_2 a_3)$ as if it were a binary number. Thus the constraint $\mathbb{P}(A_1) = \alpha_1$ means

$$p_4 + p_5 + p_6 + p_7 = \alpha_1,$$

and the constraint $\mathbb{P}(A_3|A_2) = \beta_2$ means

$$p_3 + p_7 = \beta_2(p_2 + p_3 + p_6 + p_7).$$

By minimizing/maximizing $\mathbb{P}(A_3) = p_3 + p_5 + p_7$, subject to these constraints and $p_j \geq 0$ for all j and $\sum_j p_j = 1$, we obtain $\mathbb{P}(A_3) \in [L, U]$, where

420050_1_En_9_Chapter ☑ TYPESET ☐ DISK ☐ LE ☑ CP Disp.:6/6/2019 Pages: 37 Layout: T1-Standard

$$
\begin{aligned}
L &= \max(\alpha_1\beta_1, \alpha_2\beta_2),\\
U &= \min(\alpha_1\beta_1 + \alpha_2\beta_2, \alpha_1\beta_1 + (1-\alpha_1), \alpha_2\beta_2 + (1-\alpha_2)),
\end{aligned}
$$

provided $0 \leq L \leq U \leq 1$ (otherwise the whole problem is inconsistent). □

To use de Finetti's terminology, a set of formulas and assessments that is satisfied by at least one probability measure is said to be *coherent* (Coletti and Scozzafava 2002). Given a coherent set of assessments, reasoning should only inform us about the least commitment conclusions that are necessarily correct.

We can of course contemplate probabilistic assessments $\mathbb{P}(\phi) = \alpha$ where ϕ is a formula of first-order logic and α is a number in [0, 1].[1] The semantics of first-order formulas is given by a *domain* and an *interpretation*. A domain $\mathcal{D}$, in this technical sense, is just a set (and should not be taken as the sort of "structured domain knowledge" alluded to in the title of this chapter). An interpretation is a mapping from each predicate of arity k to a relation in $\mathcal{D}^k$, and from each function of arity k to a function from $\mathcal{D}^k$ to $\mathcal{D}$ (Enderton 1972).

Each probabilistic assessment $\mathbb{P}(\phi) = \alpha$, where ϕ is a first-order formula, is interpreted as a constraint on the probability measures over the set of interpretations for a fixed domain. If the domain is infinite, the set of interpretations is uncountable. In this overview we can bypass difficulties with uncountable sets, but still present the main points, by assuming all domains to be finite, and moreover by assuming that no functions, other than constants, are present.

Example 3 Consider predicates $\mathsf{penguim}$, bird, and flies, all of arity 1, and predicate $\mathsf{friends}$, of arity 2. The intended meaning is that $\mathsf{penguim}(\mathsf{Tweety})$ indicates that Tweety is a penguim, and likewise for bird, while $\mathsf{flies}(\mathsf{Tweety})$ indicates that Tweety flies, and $\mathsf{friends}(\mathsf{Tweety}, \mathsf{Skipper})$ indicates that Tweety and $\mathsf{Skipper}$ are friends. Suppose the domain is $\mathcal{D} = \{\mathsf{Tweety}, \mathsf{Skipper}, \mathsf{Tux}\}$. An interpretation might assign both Tweety and Tux to $\mathsf{penguim}$, only the pair $(\mathsf{Tweety}, \mathsf{Skipper})$ to $\mathsf{friends}$, and so on.

Suppose $\mathbb{P}\left(\forall x : \exists y : \mathsf{friends}(x, y)\right) = 0.01$. This assigns probability 0.01 to the set of interpretations where any element of the domain has a friend. Another possible assessment is $\forall x : \mathbb{P}(\mathsf{penguim}(x)) = 0.03$; note that here the quantifier is "outside" of the probability.

For a domain with N elements, we have 2^N possible interpretations for $\mathsf{penguim}$, and 2^{N^2} interpretations for $\mathsf{friends}$; the total number of possible interpretations for the predicates is 2^{3N+N^2}. □

We refer to the semantics just outlined as an *interpretation-based* semantics, because probabilities are assigned to sets of interpretations. There is also a *domain-based* semantics, where we assume that probability measures are defined over the domain. This may be useful to capture some common scenarios. For instance, consider: "The probability that some citizen is a computer scientist is α". We might wish

[1] We might be more even general by introducing "probabilistic quantifiers", say by writing $\P^{\geq\alpha}\phi$ to mean $\mathbb{P}(\phi) \geq \alpha$. We could then nest $\P^{\geq\alpha}\phi$ within other formulas (Halpern 2003). We avoid this generality here.

to interpret this through a probability measure over the set of citizens (the domain); the constraint is that the set of computer scientists gets probability α. Domain-based semantics, and even combinations of interpretation- and domain-based semantics, have been investigated for a while (Bacchus 1990; Fagin et al. 1990; Hoover 1978; Keisler 1985); however, interpretation-based semantics are more commonly adopted.

First-order probabilistic logic has high representational power, but very high complexity (Abadi and Halpern 1994). Another drawback is *inferential vacuity*: given a set of formulas and assessments, usually all that can be said about the probability of some other formula is that it lies in a large interval, say between 0.01 and 0.99. This happens because there are exponentially many interpretations, and a few assessments do not impose enough constraints on probability values. Finally, there is yet another problem: it has been observed that first-logic itself is not sufficient to express recursive concepts or default assumptions, and tools for knowledge representation often resort to special constructs (Baader and Nutt 2002; Baral 2003). Hence it does not seem that the machinery of first-order probabilistic logic, however elegant, can exhaust the laws of thought, after all.

3 Bayesian Networks and Their Diagrammatic Relational Extensions

Bayesian networks offer a pleasant way to visualize independence relations, and an efficient way to encode a probability distribution; as such, they have been widely applied within AI and in many other fields (Darwiche 2009; Koller and Friedman 2009; Pourret et al. 2008). To fix terminology, here is a definition (see also chapter "Belief Graphical Models for Uncertainty Representation and Reasoning" of this Volume). A Bayesian network consists of a pair: there is a directed acyclic graph $\mathbb{G}$, where each node is a random variable X_i, and a probability distribution $\mathbb{P}$ over the random variables, so that $\mathbb{P}$ satisfies the Markov condition with respect to $\mathbb{G}$: each X_i is independent of its nondescendants (in $\mathbb{G}$) given its parents (in $\mathbb{G}$) (Pearl 1988). Even though we can have discrete and continuous random variables in a Bayesian network, in this survey we simplify matters by focusing on *binary* random variables. When we have a finite set of binary random variables, the Markov condition implies a factorization of the joint probability distribution; for any configuration $\{X_1 = x_1, \ldots, X_n = x_n\}$,

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

where $\mathrm{pa}(X_i)$ denotes the parents of X_i, π_i is the projection of $\{X_1 = x_1, \ldots, X_n = x_n\}$ on $\mathrm{pa}(X_i)$. Often each $\mathbb{P}\left(X_i = x_i | \mathrm{pa}(X_i) = \pi_i\right)$ is described by a *local conditional probability table*.

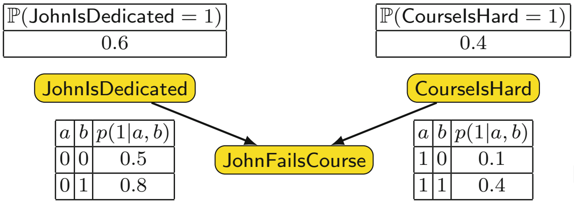


Fig. 1 The Bayesian network for the "propositional" version of the University World, where $p(1|a, b)$ denotes $\mathbb{P}\left(\mathsf{JohnFailsCourse} = 1|\mathsf{JohnIsDedicated} = a, \mathsf{CourseIsHard} = b\right)$

Consider, as an example, a simplified version of the ubiquitous "University World" (Getoor et al. 2007). We have random variables JohnIsDedicated, CourseIsHard, and JohnFailsCourse, each one with values 0 and 1. Figure 1 depicts a Bayesian network, where JohnIsDedicated and CourseIsHard are independent, and where both directly affect JohnFailsCourse.

Bayesian networks do encode structured domain knowledge through its independent relations. However, domain knowledge may come with much more structured patterns. For example, a University World usually has many students and many courses, and a very repetitive structure. We might say: for *any* pair $(\mathcal{X}, \mathcal{Y})$, where $\mathcal{X}$ is a student and $\mathcal{Y}$ is a course, the probability that $\mathcal{X}$ fails given that she is dedicated and the course is easy is 0.1. Figure 2 depicts a Bayesian network with various students and courses, where probabilities are obtained by repeating the conditional probability tables in Fig. 1.

The question then arises as to how we should specify such "very structured" Bayesian networks. It makes sense to import some tools from first-order logic. For instance, we clearly have predicates, such as the predicate fails, that can be *grounded* by replacing logical variables by elements of appropriate domains (thus we obtain the grounded predicate fails(Tina, Physics), and so on). However, here the "grounded predicates" that appear in a graph are not just propositions, but rather random variables. A symbol such as fails must be understood with a dual purpose: it can be viewed as a predicate, or as a function that yields a random variable for each pair of elements of the domain. And a symbol such as fails(Tina, Physics) also has a dual purpose: it can be viewed as a grounded predicate, or as a random variable that yields 1 when the grounded predicate is true, and 0 otherwise.

We adopt the following terminology (Poole 2003). A *parvariable* (for *parameterized random variable*) is a function that yields a random variable (its *grounding*) when its parameters are substituted for elements of the domain. The number of parameters of a parvariable is its *arity*. To specify a parameterized Bayesian network, we use parvariables instead of random variables.

Of course, when we have parvariables we must adapt our conditional probability tables accordingly, as they depend on the values of parvariables. For example, for the Bayesian network in Fig. 2 we might have

$$\mathbb{P}\left(\mathsf{isDedicated}(\mathcal{X}) = 1\right) = 0.6,$$

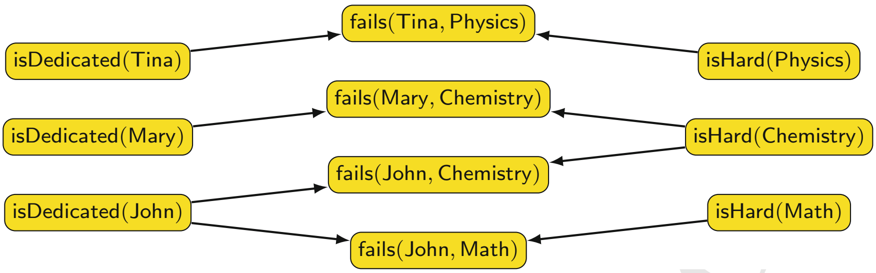


Fig. 2 The Bayesian network for the University World with three students and three courses

meaning $\mathbb{P}\left(\mathsf{isDedicated}(a)\right) = 0.6$ for each student a. Also, we might write

$$\mathbb{P}\left(\mathsf{fails}(x, y) = 1 | \mathsf{isDedicated}(x) = 0, \mathsf{isHard}(y) = 0\right) = 0.5,$$

$$\mathbb{P}\left(\mathsf{fails}(x, y) = 1 | \mathsf{isDedicated}(x) = 0, \mathsf{isHard}(y) = 1\right) = 0.8,$$

and so on, assessments that are imposed on every pair (x, y).

Each "parameterized table" specifying a conditional probability table for each substitution of logical variables is called a *parfactor*. Thus for the Bayesian network in Fig. 2 we need only three parfactors.

Possibly the most popular diagrammatic scheme to specify parvariables and parfactors is offered by *plate models*. A plate consists of a set of parvariables that share a domain (that is, the parvariables are all indexed by elements of a domain). A plate model for the University World is presented in Fig. 3 (left); plates are usually depicted as rectangles enclosing the related parvariables. The main constraint imposed on plates is that the domains used to index the parents of a parvariable must also be used to index the parvariable. Thus in Fig. 3 (left) we see that fails appears in the intersection of two plates, each one of them associated with a domain: fails is indexed by both domains.

Plate models appeared within the BUGS project (Gilks et al. 1993; Lunn et al. 2009) around 1992. At that time other template languages were discussed under the general banner of "knowledge-based model construction" (Goldman and Charniak 1990; Horsch and Poole 1990; Wellman et al. 1992). Plates were promptly adopted in machine learning (Buntine 1994) and in many statistical applications (Lunn et al. 2012).

The BUGS package was innovative not only in proposing plates but also in introducing a textual language in which to specify large Bayesian networks through plates. Figure 3 (right) shows a plate model rendered by WinBUGS (a version of BUGS); this plate model can be specified textually by using a loop to describe the plate, and by introducing a statement for each parvariable in the plate model:

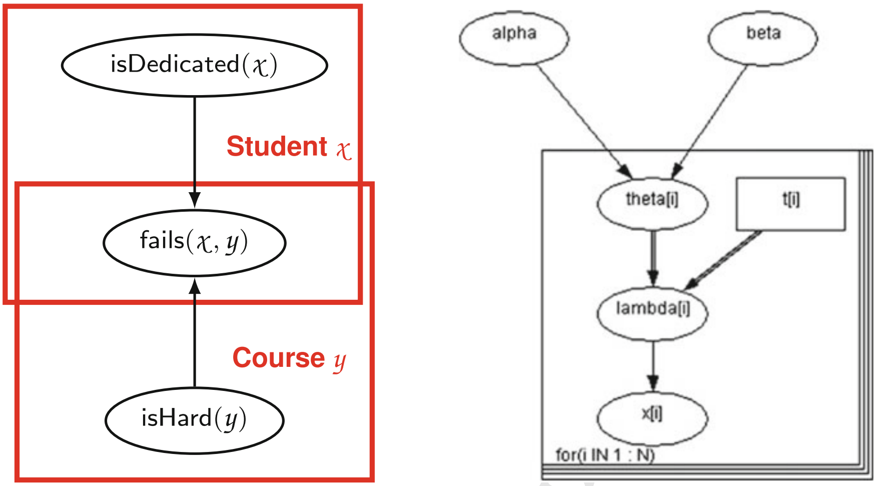


Fig. 3 Left: Plate model for the University World. Right: A plate model rendered in WinBUGS (described in the WinBUGS manual)

```
model {
      for (i in 1 : N) {
         theta[i] ~ dgamma(alpha, beta)
         lambda[i] <- theta[i] * t[i]
         x[i] ~ dpois(lambda[i])
      }
      alpha ~ dexp(1)
      beta ~ dgamma(0.1, 1.0)
}
```

The symbol $\sim$ indicates that the left hand side is distributed according to the distribution in the right hand side (Gamma, Poisson, Exponential distributions, respectively specified by `dgamma`, `dpois`, `dexp`), while the symbol $<\!-$ indicates a deterministic expression. Rectangular nodes in the plate model, such as the one containing `t[i]`, denote constants that are specified elsewhere in the program. BUGS is particularly powerful in that it can go beyond finite modeling, allowing discrete and continuous distributions (Lunn et al. 2012).

A possible extension of plate models is to allow a parvariable to have children outside of its plate. Figure 4 presents a popular example of such an "enhanced" plate; many existing topic models and factorization schemes are similarly drawn with "enhanced" plates.

When a parvariable does not belong to the plate of its parent parvariables, one must worry about *aggregation*. To understand this, consider an example. Suppose that in the University World we have a parvariable highFailureRate, of arity zero, whose value depends on fails(χ, y) for all pairs (χ, y). This parvariable should be drawn outside

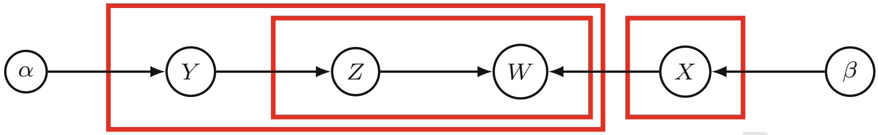


Fig. 4 The usual "enhanced" plate model for *smoothed Latent Dirichlet Allocation (sLDA)* (Blei et al. 2003). Logical variables and domains are omitted. Here X is the parent of W, but X and W belong to non-intersecting plates

of all plates in Fig. 3 (left). Consider specifying the parfactor for highFailureRate. All the groundings of fails(x, y), for all pairs (x, y), affect highFailureRate. Thus to specify probabilities for the latter we must somehow *aggregate* values of the former. This is akin to quantification in first-order logic. Most languages that allow probability values to depend on several objects at once resort to so-called *combination functions* (some of them will be discussed later).

Plates were not the only tools proposed in the nineties to specify repetitive Bayesian networks; *network fragments* and *object-oriented networks* were also contemplated (Glesner and Koller 1995; Koller and Pfeffer 1997, 1998; Mahoney and Laskey 1996), These ideas evolved to *Probabilistic Relational Models (PRMs)*, a clever mix of Bayesian networks and entity-relationship diagrams that spurred many efforts on knowledge representation and on *Statistical Relational Learning* (Friedman et al. 1999; Getoor et al. 2007; Koller and Pfeffer 1998).

In short, a PRM consists of a set of classes; each class contains a set of objects (similarly to a domain), and is associated with a set of parvariables. Usually a class is drawn as a box containing parvariables. Figure 5 (top) depicts a possible PRM for the University World; each class contains a parvariable, and there are *association* edges between classes, indicating for instance that each **Registration** is paired with a **Student** and with a **Course**. Associations appear as dashed edges; in PRMs these associations are often called *slot-chains* (Getoor et al. 2007).

Each parvariable X in a PRM is then associated with a parfactor specifying the probability for each instance of X given instances of parents of X. For instance, for the University World PRM we would need a parfactor to specify probability of fails(z) as dependent on the value of appropriate instances of isDedicated(x) and isHard(y). Here "appropriate instance" means "instance related by appropriate association". It may happen that a parvariable depends on many instances of another parvariable; for example, we may have a parvariable highFailureRate that depends on all groundings of fails. In PRMs a parfactor may depend on an *aggregate* of instances (an aggregate is just a combination function).

A *relational skeleton* for a PRM is an explicit specification of objects in each class, plus the explicit specification of pairs of objects that are associated. The semantics of a PRM with respect to a skeleton is the Bayesian network that is produced by instantiating the parvariables and parfactors.

Suppose we build a graph where each node is a parvariable, and the parents of a parvariable r are the parvariables that appear in the parfactor for r. If this graph is acyclic, then every skeleton will induce an acyclic (thus valid) Bayesian network.

420050_1_En_9_Chapter ☑ TYPESET ☐ DISK ☐ LE ☑ CP Disp.:6/6/2019 Pages: 37 Layout: T1-Standard

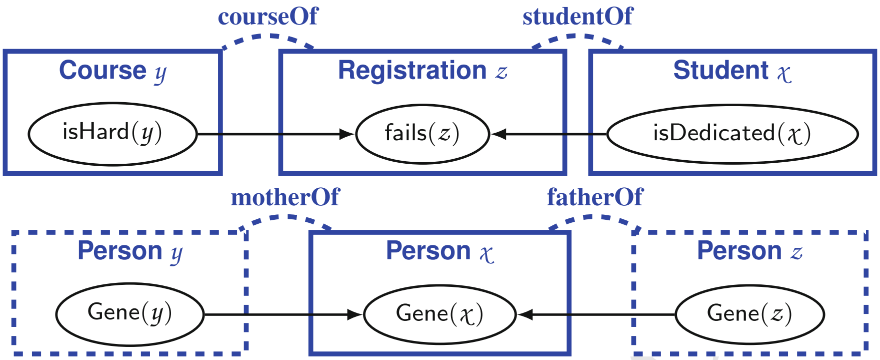


Fig. 5 Top: PRM for the University World. Bottom: PRM representing genetic relationships

However, if that graph has cycles, we face a *global semantics* problem (Jaeger 2002): can we guarantee that an acyclic Bayesian network emerges for any skeleton we expect to have? Suppose for example that a particular gene in a person may depend on that gene in the person's father and mother. Some specification languages for PRMs allow classes to appear more than once in a diagram, as depicted in Fig. 5 (bottom). The dependences may look cyclic, but we know that "fatherOf" and "motherOf" are acyclic relations, so no cycles can appear when we consider specific skeletons. Algorithms that decide existence of global semantics have been provided for special cases (Getoor et al. 2007; Milch et al. 2005b), but the answer to this question in full generality seems quite hard, and possibly undecidable (De Bona and Cozman 2017; Jaeger 2002).

As a digression, note that we can think of associations themselves as random variables that are fully observed in the relational skeleton. For instance, we might have a parvariable isCourseOf that indicates whether a course y is actually in a particular registration z. By grounding such a specification, one can produce a ground Bayesian network as depicted in Fig. 2, where only a few selected pairs student/course are associated with a grade. A parvariable that corresponds to an association, and that is fully observed in the relational skeleton, is called a *guard parvariable* (Koller and Friedman 2009).

There is much in favor of PRMs: they are as modular and visually elegant as Bayesian networks, and as featured as entity-relationship diagrams. They allow one to represent uncertainty about the structure, by associating probabilities with associations; they even allow for uncertainty about existence of particular instances (Getoor et al. 2007). Moreover, inference and learning algorithms for Bayesian networks can be easily adapted to PRMs (Sect. 8).

However, a drawback of PRMs is that they have no unified formal syntax; indeed the term "PRM" has a rather loose meaning and it is difficult to draw boundaries on what is, and what is not, a PRM. There have been some attempts to formalize PRMs. One of them is the DAPER language, where entity-relationship diagrams are extended to cope with probabilities (Heckerman et al. 2007). Figure 6 shows DAPER diagrams for the University World, for the genetic problem, and for a Hidden Markov

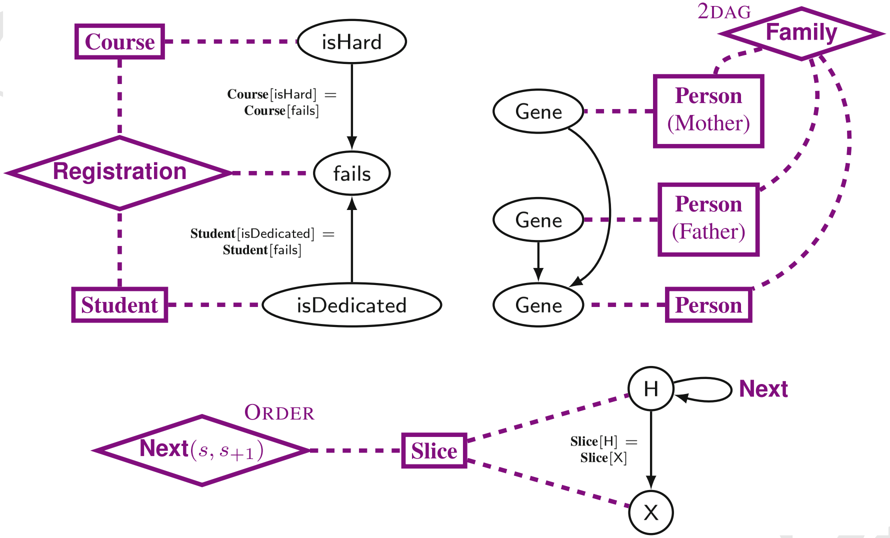


Fig. 6 Top left: A DAPER diagram for the University World. Top right: A DAPER diagram for the genetic problem. Bottom: A simplified DAPER diagram for a Hidden Markov Model (H denotes the state, while X denotes the observation; at each time slice there is a state transition that depends on the previous state)

420050_1_En_9_Chapter ✓ TYPESET ☐ DISK ☐ LE ✓ CP Disp.:6/6/2019 Pages: 37 Layout: T1-Standard

Model, a rather simple Dynamic Bayesian network (Heckerman et al. 2007). Note that temporal modeling is in essence a relational problem, where time steps index random variables; we later consider other formalisms with a focus on temporal evolution. A DAPER diagram can be annotated with a variety of constraints, such as the 2DAG in Fig. 6 (meaning that each child of a node has at most two parents and cannot be its own ancestor!). As edges can be annotated with constraints in first-order logic, there are few guarantees one can offer with respect to the complexity of manipulating a DAPER diagram.

There are also textual languages that aim at encoding PRMs in a formal manner; for instance, the Probabilistic Relational Language (PRL) (Getoor and Grant 2006) and also the CLP($\mathcal{BN}$) language (Costa et al. 2003) resort to logic programming to specify parvariables and parfactors.

Another comprehensive framework that combines logical constructs and Bayesian networks is conveyed by *Multi-Entity Bayesian Networks (MEBNs)* (da Costa and Laskey 2005; Laskey 2008). An MEBN consists of a set of network fragments (called *MFrags*), each containing parvariables (referred to as *template random variables*) and constraints over logical variables, all with associated parfactors. The language is very expressive as it allows for parfactors that are specified programmatically, and constraints that are based on first-order formulas. But MEBNs go beyond first-order logic in several aspects: instead of just true and false, in MEBNs some variables may assume the value absurd; moreover, there are ways to specify default probabilities. The modeling framework has been implemented and applied to practical problems (Carvalho et al. 2010). Figure 7 shows an MEBN for the University World: each MFrag corresponds to a parvariable in the original plate model. Each MFrag also contains *context parvariables*, depicted in Fig. 7 in rectangles. These correspond to guard parvariables that induce the structure of any instantiated Bayesian network. Finally, each MFrag contains *input parvariables*, whose parfactors are specified in other MFrags; input variables appear in Fig. 7 as dark ovals.

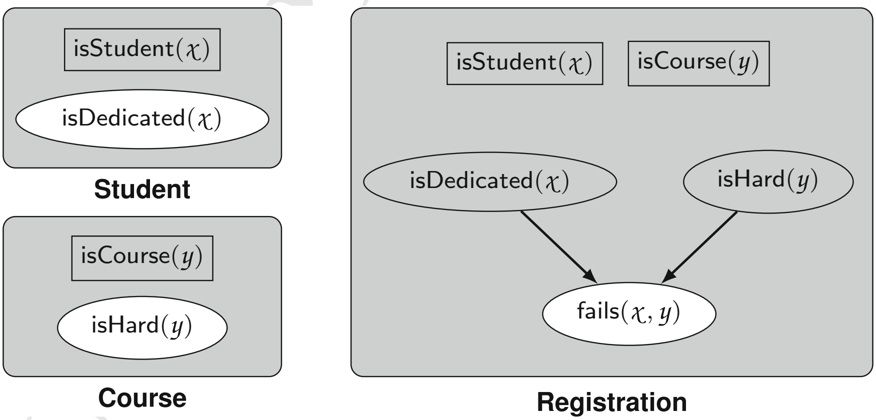


Fig. 7 An MEBN, with three network fragments, for the University World

The formalisms reviewed in this section rely on graphs to encode classes, associations, parvariables and their dependences; they do not offer much syntactic guidance as to how one should encode the parfactors. Most of the languages we discuss in the remainder of this chapter can specify both the dependences amongst parvariables and the parfactors; to do so, they use textual descriptions.

4 Probabilistic Logic Programming

Logic programming is an old and core AI technology (Baral 2003), as discussed in chapter "Logic Programming" of this Volume. It is not surprising that probabilistic logic programming has been pursued for some time (Lukasiewicz 1998; Lakshmanan and Sadri 1994; Ng and Subrahmanian 1992; Ngo and Haddawy 1997).

Poole was an early advocate of the idea that logic programming can be used to extend Bayesian networks with relational modeling (Poole 1993b). In Poole's Probabilistic Horn Abduction (PHA) language one can write:

```
symptom(S) <- carries(D), causes(D,S).
disjoint([causes(D,S): 0.7, nc(D,S): 0.3]).
```

Here we have a rule, indicating how a symptom shows up, and a statement indicating that either $\mathsf{causes}(D, S)$ is true with probability 0.7, or $\mathsf{nc}(D, S)$ is true with probability 0.3. The PHA language later evolved into Independent Choice Logic (ICL), where decision-making and multi-agent scenarios can be modeled (Poole 1997, 2008).

Another seminal proposal for a mixture of logic programming and probabilities was Sato's *distribution semantics*, embodied in the popular package PRISM (Sato 1995; Sato and Kameya 2001). A similar syntax and semantics appeared in Fuhr's work on Probabilistic Datalog (Fuhr 1995). The distribution semantics has been adopted by many languages (Riguzzi et al. 2014; De Raedt and Kimmig 2015); to understand it, a few definitions are needed.

A logic program is a set of declarative rules that just describe a problem; finding a solution is the task of an inference engine (Dantsin et al. 2001; Eiter et al. 2009). A *normal logic program* consists of rules written as (Dantsin et al. 2001)

$$A_0 \coloneq A_1, \ldots, A_m, \mathbf{not}\, A_{m+1}, \ldots, \mathbf{not}\, A_n.$$

where the A_i are atoms and **not** indicates negation as failure. The *head* of this rule is A_0; the remainder of the rule is its *body*. A rule without a body, written simply as $A_0.$, is a *fact*. A *subgoal* in the body is either an atom A (a *positive* subgoal) or **not** A (a *negative* subgoal). A program without negation is *definite*, and a program with only grounded atoms is *propositional*.

The *Herbrand base* of a program is the set of all grounded atoms built from constants and predicates in the program. As before, we allow for constants, but we do not consider functions, so as to stay with finite Herbrand bases. The grounding of a program is the propositional program obtained by applying every possible grounding to each rule, using only the constants in the program (i.e., using only grounded atoms in the Herbrand base).

The *dependency graph* of a program is a directed graph where each predicate is a node, and where there is an edge from a node B to a node A if there is a rule where A appears in the head and B appears in the body; if B appears right after **not** in any such rule, the edge is *negative*; otherwise, it is *positive*. The *grounded* dependency graph is the dependency graph of the propositional program obtained by grounding.

A program is *acyclic* when its grounded dependency graph is acyclic. The semantics of an acyclic program is rather simple, and given by the program's *Clark completion* (Clark 1978): roughly, take the grounding of the program, and for each head A, make it true if some rule with head A has all its subgoals recursively assigned to true. At the end of this process, we have an interpretation for the predicates.

Take a probabilistic logic program to consist of *probabilistic facts* in addition to rules and facts. A probabilistic fact is written as α::A**.**, where α is a number in [0, 1] and A**.** is a fact (here we use the syntax of the popular package Problog (Fierens et al. 2014).

Sato's distribution semantics is, in essence, a distribution over logic programs: for each probabilistic fact α::A**.**, with probability α fact A**.** is added to the program, and with probability $1 - \alpha$ the fact is not added to the program (all probabilistic facts are selected independently). Note that each selection of probabilistic facts generates a logic program.

Example 4 To understand the distribution semantics, consider a probabilistic logic program in Problog's syntax. The first line establishes some deterministic facts, followed by probabilistic ones, and then we have a few rules (each rule uses an auxiliary predicate associated with a probabilistic fact):

```
isStudent(john).   isStudent(mary).   isCourse(math).
0.6::isDedicated(X).
0.4::isHard(Y).
0.5::a1(X,Y).    0.1::a2(X,Y).
0.8::a3(X,Y).    0.4::a4(X,Y).
fails(X,Y) :- isStudent(X), isDedicated(X),
                  isCourse(Y), isHard(Y), a1(X,Y).
fails(X,Y) :- isStudent(X), isDedicated(X),
                  isCourse(Y), not isHard(Y), a2(X,Y).
fails(X,Y) :- isStudent(X), not isDedicated(X),
                  isCourse(Y), isHard(Y), a3(X,Y).
fails(X,Y) :- isStudent(X), not isDedicated(X),
                  isCourse(Y), not isHard(Y), a4(X,Y).
```

The Herbrand base is then the set of groundings obtained by replacing logical variables by the constants john, mary, and math. Altogether, these facts, probabilistic facts, and rules are similar to the plate mode in Fig. 3 (left). Note that predicates isStudent and isCourse correspond to guard parvariables; they can also be seen as the context parvariables depicted in Fig. 7. □

One convenient feature of probabilistic logic programs (with Sato's distribution semantics) is that they inherit the "default" assumptions used in logic programming. For instance, if we do not say that isCourse(mary) is true, then it is automatically false. This simplifies the specification of "guard parvariables": for instance, in the example above the parvariables isStudent and isCourse aptly specify the structure of the graph, and are not associated with probabilities.

Poole's PHA focused on acyclic probabilistic logic programs, while Sato's original proposal focused on definite, but not necessarily acyclic, probabilistic logic programs. These syntactic restrictions have been removed in a variety of ways. One natural extension is to allow *stratified negation*; that is, to allow cycles in the dependency graph as long as there is no negative edge in any cycle (Dantsin et al. 2001). Here is a well-formed probabilistic logic program in Problog's syntax:

```
smokes(X) :- not relaxed(X).
smokes(X) :- influences(Y,X), smokes(Y).
0.3::influences(john,mary).
0.3::influences(mary,john).
0.2::relaxed(mary).
```

For any sample of the three probabilistic facts, the resulting logic program has a cycle. But this is not a problem, for cyclic logic programs still have a semantics (Dantsin et al. 2001); in fact, any probabilistic logic program such that the rules form a stratified logic program induces a *unique* probability measure over interpretations. The Problog package deals with probabilistic stratified logic programs (Fierens et al. 2014; De Raedt and Kimmig 2015).

Probabilistic logic programs that contain cycles and arbitrary negation have been also studied (Hadjichristodoulou and Warren 2012; Lukasiewicz 2005; Riguzzi 2015; Sato et al. 2005). Several semantics have been proposed for such programs. We mention here two semantics: one is based on the *stable model* semantics, where a logic program admits more than a single stable model as its meaning (Gelfond and Lifschitz 1988), and the *well-founded semantics*, where a logic program admits a single well-founded model, but that model may contain true, false and undefined assignments (Van Gelder et al. 1991). The stable model semantics induces not a single probability measure over interpretations, but a set of probability measures, while the well-founded semantics induces a single probability measure over assignments of three-valued logic (Cozman and Mauá 2017c). Another possibility, adopted by the P-log language, is to use the stable model semantics, but to assume a uniform probability distribution over the set of stable models of any sampled logic program (Baral et al. 2009). The LP$^{\text{MLN}}$ language also distributes probability over sets of interpretations

420050_1_En_9_Chapter ☑ TYPESET ☐ DISK ☐ LE ☑ CP Disp.:6/6/2019 Pages: 37 Layout: T1-Standard

(Lee and Wang 2015), resorting to a syntax that resembles Markov logic (Sect. 6). Yet another approach is represented by a probabilistic version of the Datalog$^{\pm}$ language (Ceylan et al. 2016), where the stable semantics is adopted but program repairs are automatically invoked when the program is inconsistent.

There is no consensus yet on how to handle probabilistic logic programs with disjunctive heads and other common constructs (Cozman and Mauá 2017c). Logic Programs with Annotated Disjunctions (LPADs) offer syntax to handle disjunction probabilistically (Vennekens et al. 2004), as also done in CP-logic (Vennekens et al. 2009). A LPAD may contain a rule such as

```
(heads(Coin):0.6); (tails(Coin):0.4) :- toss(Coin).
```

to model a biased coin. An alternative is to adapt the stable model semantics of disjunctive programs to handle probabilities (Cozman and Mauá 2017a).

Even more general combinations of answer set programming, first-order sentences, and probabilities have been investigated (Nickles and Mileo 2015); another extension that has received attention is the specification of continuous random variables (Nitti et al. 2016).

All of these extensions of Sato's semantics still aim at defining a distribution over interpretations of grounded atoms. A few years later than Sato's distribution semantics, a different approach to probabilistic logic programming was advocated, with the combination of Inductive Logic Programming and probabilities (Muggleton 1996). The resulting mixture has been pursued in many guises, usually under the banner of *Probabilistic Inductive Logic Programming* (De Raedt et al. 2010; De Raedt 2008). Assumptions are often distinct from Sato's, for instance by placing probabilities over the set of proofs of a logic program (sometimes referred to as *proof theoretic semantics*) (Cussens 1999; De Raedt and Kersting 2004).

Two additional formalisms that specify Bayesian networks using logic programming principles, but not all syntactic conventions of logic programming, are Logical Bayesian Networks (LBNs) (Fierens et al. 2005, 2004) and Bayesian Logic Programs (BLPs) (De Raedt and Kersting 2004; Kersting et al. 2000). Both distinguish between guard parvariables and parvariables associated with probabilities. To illustrate, consider the following Bayesian Logic Program fragment (De Raedt and Kersting 2004):

```
carrier(x) | founder(x)
carrier(x) | mother(m,x), carrier(m),
                       father(f,x), carrier(f)
```

where mother and father are *logical predicates* (that is, guard parvariables), while founder and carrier are associated with probabilities that must be given separately. Logical Bayesian Networks adopt additional elements of logic programming in the manipulation of logical predicates (Fierens et al. 2005).

5 Probabilistic Logic, Again; and Probabilistic Description Logics

A natural idea would be the mix graphs as used in Bayesian networks with general assessments as allowed in probabilistic logics. This is exactly the scheme adopted in Andersen and Hooker's Bayesian Logic and its variants (Andersen and Hooker 1994; de Campos et al. 2009; Cozman et al. 2008). The resulting probabilistic logics can often be viewed as specification languages for *credal networks*; that is, for graph-based models that encode *sets* of probability distributions rather than a single distribution (Cozman 2000).

Another natural idea is to specify Bayesian networks using extended first-order formulas (Bacchus 1993). For instance, one might write

$$\begin{gathered}\forall \chi, y : \mathsf{fails}(\chi, y) \equiv (\mathsf{isDedicated}(\chi) \wedge \mathsf{a1}(\chi, y)) \vee (\mathsf{isHard}(y) \wedge \mathsf{a2}(\chi, y)), \\ \forall \chi, y : \mathbb{P}\left(\mathsf{a1}(\chi, y) = 1\right) = 0.2, \qquad \forall \chi, y : \mathbb{P}\left(\mathsf{a2}(\chi, y) = 1\right) = 0.6,\end{gathered} \tag{1}$$

where the symbol $\equiv$ is used to emphasize that we have a definition, and both $\mathsf{a1}$ and $\mathsf{a2}$ are auxiliary predicates (similar to the auxiliary probabilistic facts used in Example 4). It is worth noting that, for fixed χ and y, Expression (1) specifies a Noisy-OR gate for fails (Pearl 1988).

Expression (1) suggests a general strategy, where a Bayesian network is specified by associating each random variable X_i with a definition $X_i \equiv f(\mathrm{pa}(X_i), Y_1, \ldots, Y_m)$, for a deterministic function f of the parents of X_i and some auxiliary random variables $Y_1, \ldots, Y_m$ that are assigned probabilities $\mathbb{P}\left(Y_j = 1\right) = \alpha_j$. Poole refers to the latter random variables as *independent choices*, as he argues in favor of this specification strategy (Poole 2010). This sort of strategy is used in structural models to handle continuous random variables (Pearl 2009).

Another popular strategy in combining logic and probabilities is to mix Bayesian networks and *description logics*. Recall that description logics are (usually) decidable fragments of first-order logic that suffice for many knowledge representation tasks (Baader et al. 2017), as discussed in chapter "Reasoning with Ontologies" of Volume 1. A description logic (usually) contains a vocabulary containing *individuals*, *concepts*, and *roles*. New concepts can be defined by *intersection*, *union*, or *complement* of other concepts. And a concept can be defined using constructs $\exists r.C$ or $\forall r.C$, where r is a role and C is a concept. An inclusion axiom is written $C \sqsubseteq D$, and a definition is written $C \equiv D$, where C and D are concepts. A semantics for a description logic can (usually) be given by translation of individuals into constants, concepts into unary predicates, roles into binary predicates, and by translation of various operators into logical operators (Borgida 1996). That is, we have a domain $\mathcal{D}$ that again is a set, and for each concept C we have a set $\mathbb{I}(C)$ of elements of the domain, and for each role r we have a set $\mathbb{I}(r)$ of pairs of elements of the domain. Inclusion $C \sqsubseteq D$ means that $\mathbb{I}(C) \subseteq \mathbb{I}(D)$, and $C \equiv D$ means that $\mathbb{I}(C) = \mathbb{I}(D)$. Moreover, the intersection $C \sqcap D$, for concepts C and D, is translated into $C(\chi) \wedge D(\chi)$; similarly union is translated into disjunction and complement is translated into negation.

A more involved translation is needed for $\exists r.C$; this concept should be interpreted as "the set of all χ such that there is y satisfying $r(\chi, y)$ and $C(y)$". For instance, the expression

$$\mathsf{Student} \sqcap \exists \mathsf{hasChild}.\mathsf{Female}$$

defines the set of students who have at least a daughter. In symbols, $\exists r.C$ is interpreted as $\{x \in \mathcal{D} | \exists y \in \mathcal{D} : (x, y) \in \mathbb{I}(r) \wedge y \in \mathbb{I}(C)\}$. Similarly, $\forall r.C$ is interpreted as $\{x \in \mathcal{D} | \forall y \in \mathcal{D} : (x, y) \in \mathbb{I}(r) \rightarrow y \in \mathbb{I}(C)\}$.

An early partnership between Bayesian networks and description logics is the P-CLASSIC language: there the description logic CLASSIC is enlarged with probabilities in a domain-based semantics (Koller et al. 1997a). One must draw a graph where nodes refer either to concepts or to parvariables that indicate how many individuals are related by roles. Inference in P-CLASSIC produces, for instance, "the probability that a randomly selected individual is a student and has a daughter"; in symbols: $\mathbb{P}(\mathsf{Student} \sqcap \exists \mathsf{hasChild}.\mathsf{Female})$. Other combinations of Bayesian networks and description logics have produced tools that resemble P-CLASSIC (Ding et al. 2006; Staker 2002; Yelland 1999).

An interpretation-based semantics is adopted by PR-OWL (Carvalho et al. 2013; Costa and Laskey 2006), a language that combines constructs from description logics with Multi-Entity Bayesian Networks (Sect. 3). Another popular strategy has been to use versions of Poole's independent choices together with description logics (Lukasiewicz et al. 2011; Riguzzi et al. 2015); for instance, in DISPONTE one can write

$$0.9{::}\mathsf{Bird} \sqsubseteq \mathsf{Flies},$$

thus mixing Problog's syntax with an inclusion axiom (Riguzzi et al. 2015). Some languages even employ Bayesian networks to specify probabilistic choices (Ceylan and Peñaloza 2014; d'Amato et al. 2008). For instance, in the Bayesian DL-Lite$_{\mathcal{R}}$ language one can write

$$\mathsf{Bird} \sqsubseteq \mathsf{Flies} : \mathsf{condition} = \mathsf{true}$$

to mean that the inclusion axiom is present with probability $\mathbb{P}(\mathsf{condition} = \mathsf{true})$ that is in turn given by a Bayesian network (d'Amato et al. 2008).

There are many other probabilistic description logics that go beyond Bayesian networks, often including sophisticated logical operators and interval-valued assessments (Klinov and Parsia 2011; Lukasiewicz 2008). These languages have found application in the semantic web, in information retrieval, and in knowledge representation by ontologies. The reader can benefit from substantial surveys in the literature (Lukasiewicz and Straccia 2008; Predoiu and Stuckenschmidt 2009).

6 Markov Random Fields: Undirected Graphs

We have so far considered languages where probabilities are specified directly; that is, if the probability that John is dedicated is 0.9, one just writes $\mathbb{P}(\mathsf{JohnIsDedicated} = 1) = 0.9$. An entirely different strategy is employed in Markov random fields (Koller and Friedman 2009). There one uses an undirected graph, where each node is a random variable, and where each clique C_i of the graph is associated with a positive function $f_i(C_i)$ (recall that a clique is a complete subgraph); see also chapter "Belief Graphical Models for Uncertainty Representation and Reasoning" of this Volume. For instance, the Markov random field defined in Fig. 8 has nine cliques containing two random variables, and two cliques containing three random variables. With rather general assumptions, the joint distribution then factorizes as

$$\mathbb{P}(X_1 = x_1, \ldots, X_n = x_n) = (1/\nu) \prod_i f_i(C_i = c_i),$$

where the product goes over the set of cliques, and for each clique C_i the symbol c_i is the projection of $\{X_1 = x_1, \ldots, X_n = x_n\}$ on the random variables in C_i. The normalization constant ν is called the *partition function*, and it is equal to $\sum_{x_1,\ldots,x_n} \prod_i f_i(C_i = c_i)$ (see also chapter "Belief Graphical Models for Uncertainty Representation and Reasoning" of this Volume). There are technical complications if we want to find a Markov condition for such graphs in the presence of zero probabilities; we do not dwell on such details here.

The graph in Fig. 8 induces a very structured Markov random field, as we can expect the function for clique $\mathsf{relaxed}(\mathsf{Ann}) - \mathsf{smokes}(\mathsf{Ann})$ to be equal to the function for clique $\mathsf{relaxed}(\mathsf{Bob}) - \mathsf{smokes}(\mathsf{Bob})$, and so on. It is thus natural to think about languages that would allow us to specify repetitive Markov random fields. We mention two approaches: relational Markov networks and Markov logic networks.

A *relational Markov network* is the exact counterpart of PRMs in the context of undirected graphs (Taskar et al. 2007). A relational Markov network consists, first, of a set of clique templates; each template contains parvariables and conditions on parvariables that determine which instantiations are connected by

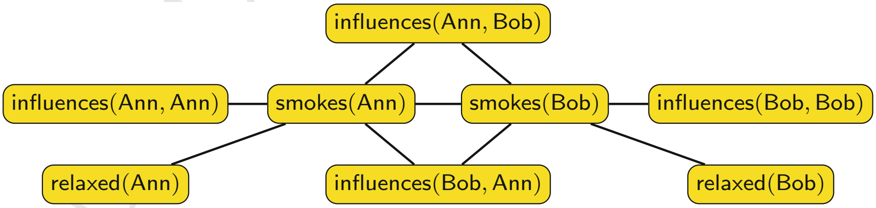


Fig. 8 The Markov random field about smoking

edges (some implementations specify such conditions using database queries written in SQL). Given a domain (or a database), a template is instantiated into a set of random variables and edges between those random variables. The second part of a relational Markov network is a set of functions that map parvariables to positive numbers; we refer to these functions again as *parfactors* (note that these parfactors are not restricted to values in $[0, 1]$). As an example, we could build the Markov random field in Fig. 8 by specifying parvariables $\mathsf{relaxed}$, smokes, and $\mathsf{influences}$, and parfactors $f(\mathsf{relaxed}(\mathcal{X}), \mathsf{smokes}(\mathcal{X}))$ and $g(\mathsf{smokes}(\mathcal{X}), \mathsf{influences}(\mathcal{X}, \mathcal{Y}), \mathsf{smokes}(\mathcal{Y}))$. For instance, we might specify the first parfactor as follows:

$\mathsf{relaxed}(\mathcal{X})$	$\mathsf{smokes}(\mathcal{X})$	$f(\mathsf{relaxed}(\mathcal{X}), \mathsf{smokes}(\mathcal{X}))$
0	0	2
0	1	1
1	0	1
1	1	3

Given an instantiation of the parvariables, the parfactors can be instantiated as well, and the result is a Markov random field such as the one in Fig. 8.

Because relational Markov networks can represent uncertainty about links, they have been successful in tasks such as collective information extraction (Bunescu and Mooney 2004) and activity recognition (Liao et al. 2006).

Markov logic networks adopt a different, albeit related, strategy (Domingos and Lowd 2009; Richardson and Domingos 2006). Instead of parvariables and parfactors, we really start with predicates and first-order sentences; the only departure from dry first-order logic is that we associate a *weight* with each sentence.

In our "smoking" example, we might contemplate two weighted sentences:

$$\begin{array}{ll} \text{Weight} = 2: & \forall \mathcal{X} : \mathsf{relaxed}(\mathcal{X}) \rightarrow \mathsf{smokes}(\mathcal{X}), \\ \text{Weight} = 4: & \forall \mathcal{X}, \mathcal{Y} : \mathsf{influences}(\mathcal{X}, \mathcal{Y}) \wedge \mathsf{smokes}(\mathcal{X}) \rightarrow \mathsf{smokes}(\mathcal{Y}). \end{array} \tag{2}$$

As before, predicates have a dual role as parvariables, and a grounding of a predicate works as a random variable.

The semantics of Markov logic networks can be explained operationally, by a procedure that takes a Markov logic network and a set $\mathcal{D}$, the domain, and produces a Markov random field. The procedure is rather simple: first, ground all sentences with respect to all groundings, repeating weights as appropriate. For instance, in the "smoking" example with domain $\{\mathsf{Ann}, \mathsf{Bob}\}$, we obtain

$$\begin{aligned}
&\text{Weight} = 2: && \mathsf{relaxed}(\mathsf{Ann}) \to \mathsf{smokes}(\mathsf{Ann}),\\
&\text{Weight} = 2: && \mathsf{relaxed}(\mathsf{Bob}) \to \mathsf{smokes}(\mathsf{Bob}),\\
&\text{Weight} = 4: && \mathsf{influences}(\mathsf{Ann}, \mathsf{Ann}) \wedge \mathsf{smokes}(\mathsf{Ann}) \to \mathsf{smokes}(\mathsf{Ann}),\\
&\text{Weight} = 4: && \mathsf{influences}(\mathsf{Ann}, \mathsf{Bob}) \wedge \mathsf{smokes}(\mathsf{Ann}) \to \mathsf{smokes}(\mathsf{Bob}),\\
&\text{Weight} = 4: && \mathsf{influences}(\mathsf{Bob}, \mathsf{Ann}) \wedge \mathsf{smokes}(\mathsf{Bob}) \to \mathsf{smokes}(\mathsf{Ann}),\\
&\text{Weight} = 4: && \mathsf{influences}(\mathsf{Bob}, \mathsf{Bob}) \wedge \mathsf{smokes}(\mathsf{Bob}) \to \mathsf{smokes}(\mathsf{Bob}).
\end{aligned}$$

Now read each one of these weighted propositional sentences as a function:

$$\text{“Weight} = \alpha : \phi\text{” yields } \begin{cases} e^{\alpha} & \text{when } \phi \text{ is } \mathsf{true},\\ 1 & \text{otherwise.} \end{cases}$$

This somewhat mysterious convention yields a single joint probability distribution for any Markov logic network with a given domain. If we denote by $X_1, \ldots, X_n$ the random variables that correspond to all groundings of all predicates, the convention is that:

$$\mathbb{P}\left(X_1 = x_1, \ldots, X_n = x_n\right) = (1/Z) \exp\left(\sum_j w_j n_j\right),$$

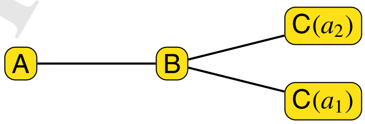
where j runs over the weighted sentences, w_j is the weight of the jth weighted sentence, and n_j is the number of groundings of the jth sentence that are true for the configuration $\{X_1 = x_1, \ldots, X_n = x_n\}$ (and Z is the normalization constant). In our “smoking” example, the Markov random field induced by Expression (2) has the undirected graph depicted in Fig. 8.

Example 5 A short exercise is instructive. Consider the Markov logic network

$$\text{Weight} = \ln(2): \ \mathsf{A} \wedge \mathsf{B}, \qquad \text{Weight} = \ln(10): \ \neg\mathsf{A} \wedge \neg\mathsf{B},$$

$$\text{Weight} = \ln(10): \ \mathsf{B} \wedge \mathsf{C}(x).$$

For a domain $\{a_1, \ldots, a_N\}$, we get random variables A, B, $\mathsf{C}(a_1), \ldots \mathsf{C}(a_N)$. For instance, for $N = 2$ we get the following Markov random field:



The partition function is, using a bit of combinatorics,

$$\begin{aligned}
Z &= \left(10 \sum_{i=0}^{N} \binom{N}{i}\right) + \left(\sum_{i=0}^{N} 10^i \binom{N}{i}\right) + \left(\sum_{i=0}^{N} \binom{N}{i}\right) + \left(2 \sum_{i=0}^{N} 10^i \binom{N}{i}\right)\\
&= 11 \times 2^N + 3 \times 11^N,
\end{aligned}$$

420050_1_En_9_Chapter ☑ TYPESET ☐ DISK ☐ LE ☑ CP Disp.:6/6/2019 Pages: 37 Layout: T1-Standard

and then $\mathbb{P}(\mathsf{A} = 1) = (2^N + 2 \times 11^N)/Z$ and $\mathbb{P}(\mathsf{B} = 1) = 3 \times 11^N/Z$. □

Markov logic networks offer an attractive combination of logic and probability; it is really a clever idea that has been vigorously applied in practical problems (Domingos and Lowd 2009; De Raedt et al. 2016). A clear advantage of such "undirected" languages is that they have no problem with cycles, as for instance PRMs do. One can only wonder if we have finally nailed down the laws of thought.

However, Markov random fields are much less transparent than PRMs when it comes to the meaning of the functions attached to cliques. These functions are not probabilities; rather, their normalized product yields the joint distribution. Interpreting the numbers is already a challenge in the propositional setting, but the relational setting introduces significant complications because the relative effect of functions depends on the number of groundings — this in turn depends on the size of the domain and on the arity of predicates. Consider Example 5: for $N = 1$ we have $\mathbb{P}(\mathsf{A} = 1) = 24/55 < \mathbb{P}(\mathsf{A} = 0)$, but as N grows, $\mathbb{P}(\mathsf{A} = 1)$ grows as well, and we have $\lim_{N\to\infty} \mathbb{P}(\mathsf{A} = 1) = 2/3$. In general, it is very hard to know, at design time, what the weights mean when instantiated (Poole et al. 2012; Jain et al. 2007).

An additional difficulty with Markov logic networks is that a material implication $\mathsf{A} \rightarrow \mathsf{B}$ is not really related to a conditional probability of B given A: $\mathsf{A} \rightarrow \mathsf{B}$ may be true with very high probability just because A is almost always false. Thus attaching a weight to a material implication says little about conditional probabilities.

And to close these remarks on Markov logic networks, consider the following short and startling example.[2]

Example 6 Suppose we have a Markov logic network consisting of three weighted (propositional) sentences:

$$\text{Weight} = 2: \ \mathsf{A}, \qquad \text{Weight} = 1: \ \mathsf{A} \rightarrow \mathsf{B}, \qquad \text{Weight} = 0: \ \mathsf{C} \rightarrow \mathsf{B}.$$

Then the probability that A is true is 0.83481, but the probability that the sentence $\mathsf{C} \rightarrow \mathsf{B}$ is true is 0.864645. That is, the latter sentence has higher probability than the former, thus reversing the order of their weights! □

There are other graph-based probabilistic models besides Bayesian networks and Markov random fields (Sadeghi and Lauritzen 2014), but it does not seem that they have been lifted to relational settings, with the notable exception of *Relational Dependency Networks* (Neville and Jensen 2007). A dependency network consists of a graph, possibly with cycles, where each node is a random variable and edges are bidirectional, and each node is associated with a conditional probability table specifying its probabilities given its neighbors in the graph (Heckerman et al. 2000). The joint distribution is not obtained in closed-form, but rather by a limiting procedure; it is thus a bit hard to relate local parameters and global behavior. Relational Dependency Networks employ the same toolset of PRMs to extend dependency networks: now the nodes of a graph are parvariables organized in classes, and a dependency

[2]This example is due to my colleague Marcelo Finger (personal communication).

420050_1_En_9_Chapter ☑ TYPESET ☐ DISK ☐ LE ☑ CP Disp.:6/6/2019 Pages: 37 Layout: T1-Standard

network is obtained by grounding the parvariables and associated parfactors (Neville and Jensen 2007). The ability to encode cyclic patterns has been useful in applications, for instance in natural language processing (Toutanova et al. 2003).

7 Probabilistic Programming

The emergence of PRMs and related formalisms during the nineties prompted some researchers to explore programming languages with probabilistic constructs. Some early formalisms, like RBNs, IBAL, and BLOG, illustrate important design choices.

Jaeger's *Relational Bayesian Networks (RBNs)* associate probabilities with predicates so that, given a finite domain, one obtains a Bayesian network that specifies a probability distribution over interpretations of the predicates. The design philosophy behind this language is to have a few powerful constructs that are quite expressive but that still allow for theoretical analysis. We have a vocabulary of predicates; a predicate name stands also for a parvariable with the same arity. Each predicate is associated with a *probability formula* (in essence, a parfactor specification). Here is an example:

```
burglary(x) = 0.005;
alarm(x) = 0.95 burglary(x) + 0.01 (1-burglary(x));
cityAlarm = NoisyOr{0.8 alarm(x)|x; x = x};
```

The probability formula for burglary is the simplest one: a number. That probability formula can be understood as: $\forall \mathcal{X} : \mathbb{P}(\mathsf{burglary}(\mathcal{X}) = 1) = 0.005$. The probability formula for alarm basically consists of arithmetic operations with parvariables; only a few operations are allowed in RBNs. One could read this probability formula as follows: for each $\mathcal{X}$, if $\mathsf{burglary}(\mathcal{X})$ is true, then return 0.95; otherwise, return 0.01. The probability formula for cityAlarm is more involved: it defines a *combination expression* that is a Noisy-OR gate of all instances of alarm, each one of them associated with a probability 0.8.

The syntax of combination expressions is rather intimidating. A combination expression is written $\mathsf{comb}(F_1, \ldots, F_k | \mathcal{Y}_1, \ldots, \mathcal{Y}_m : \phi)$, where each F_i is a probability formula, each $\mathcal{Y}_j$ is a logical variable that appears in the formulas F_i, ϕ is a formula containing only logical variables, Boolean operators and equality; finally, comb is a function that takes a multiset of numbers (a set with possibly repeated numbers) and returns a number in [0, 1]. The semantics of combination expressions can be explained operationally as follows. First, collect all logical variables that appear in any $F_1, \ldots, F_k$, and not in $\{\mathcal{Y}_1, \ldots, \mathcal{Y}_m\}$. These are the *free* logical variables of the combination expression. Basically, a combination expression is a function that yields a number in [0, 1] for each instance of the free logical variables. To compute this number for a fixed instance of the free logical variables, go over all instances of the logical variables $\mathcal{Y}_1, \ldots, \mathcal{Y}_m$ that satisfy ϕ. For each one of these instances, compute the value of $F_1, \ldots, F_k$. This produces a multiset of numbers in [0, 1]. Finally, apply

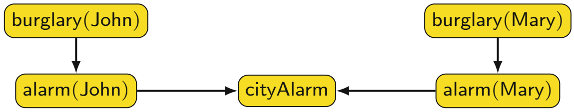


Fig. 9 The Bayesian network for the “alarm” RBN with domain {John, Mary}

comb to this multiset to obtain the value of the combination function for the fixed instance of the free variables. Clearly this is not a simple scheme, and it reveals the difficulty of producing a specification language with a small set of constructs that still can capture a large number of practical scenarios.

In any case, any RBN defines, for a given finite domain, a Bayesian network. For instance, we obtain the Bayesian network in Fig. 9 when we take the RBN specified previously and domain {John, Mary}. The conditional probability tables for this Bayesian network can be read from the RBN; the only non-trivial specification is the Noisy-OR gate for cityAlarm.

In the IBAL language (Pfeffer 2001), elements of functional programming are mixed with probabilistic assessments. A basic construct is a *stochastic choice*, syntactically expressed as $\mathsf{dist}[p_1 : e_1, \ldots, p_n : e_n]$, where each p_i is a number and each e_i is an expression: with probability p_i, the result of the statement is the evaluation of e_i. Useful syntactic sugar is provided by the $\mathsf{flip}(p)$ construct, yielding 1 with probability p and 0 with probability $1 - p$ (the flip construct appears already in stochastic programs that inspired IBAL (Koller et al. 1997b).

For example, this IBAL statement

```
alarm = (quake & flip(0.1)) | (burglary & flip(0.7));
```

assigns a Noisy-OR gate to alarm, by taking a disjunction (|) of conjunctions (&). And the following IBAL code is related to the University World:

```
student() = { isDedicated = flip 0.6; };
course() = { isHard = flip 0.4; };
registration(s,c) = {
   fails = if s.isDedicated
     then (if c.isHard then flip 0.4 else flip 0.1)
     else (if c.isHard then flip 0.8 else flip 0.5);};
```

Note how PRM classes are encapsulated in functions, and parvariables appear as “local variables”. In IBAL one can also specify a domain and observations, and ask about probability values. Moreover, IBAL is not just a template language, as it offers many programming constructs that allow for arbitrary (Turing-complete) computations (Pfeffer 2001).

Another “generative” language is Milch et al.’s Bayesian Logic, referred to as BLOG (Milch et al. 2005a). A distinctive feature of BLOG is that one can place a distribution over the size of the domain; consider for instance (Wu et al. 2016):

```
Type House;
#House ~ Poisson(10);
random Boolean Burglary(House h)~BooleanDistrib(0.1);
random Boolean Earthquake ~ BooleanDistrib(0.002);
random Boolean Alarm(House h) ~
  case [Burglary(h), Earthquake] in {
    [false, false] -> BooleanDistrib(0.01),
    [false, true]  -> BooleanDistrib(0.40),
    [true, false]  -> BooleanDistrib(0.80),
    [true, true]   -> BooleanDistrib(0.90)
  };
```

The second line associates a Poisson distribution with the number of houses. A language where domain size is not necessarily fixed is sometimes called an *open-universe* language (Russell 2015). Such a language must deal with a number of challenges; for instance, the need to consider infinitely many parents for a random variable (Milch et al. 2005a). The flexibility of BLOG has met the needs of varied practical applications (Russell 2015).

It is only natural to think that even more powerful specification languages can be built by adding probabilistic constructs to existing programming languages. An early language that adopted this strategy is CES, where probabilistic constructs are added to C (Thrun 2000); that effort later led to the PTP language, whose syntax augments CAML (Park et al. 2008). A similar strategy appeared in the community interested in planning: existing languages, sometimes based on logic programming, have been coupled with probabilities — two important examples are Probabilistic PDDL (Yones and Littman 2004) and RDDL (Sanner 2011). The latter languages have been used extensively in demonstrations and in competitions, and have been influential in using actions with deterministic and with uncertain effects to obtain decision making with temporal effects.

A rather influential language that adds probabilistic constructs to a functional programming language (in this case, Scheme) is Church (Goodman et al. 2008). Even though the goal of Church was to study cognition, the language is heavily featured; for instance, we can use the "flip" construct, plus conjunction and disjunction, to define conditional probability tables as follows:

```
(define flu (flip 0.1))
(define cold (flip 0.2))
(define fever (or (and cold (flip 0.3))
                  (and flu (flip 0.5))))
```

and we can even use recursion to define a genometric distribution:

```
(define (geometric p)
          (if (flip p)  0  (+ 1 (geometric p))))
```

A descendant of Church is WebPPL; here the probabilistic constructs are added to JavaScript. For instance, a conditional probability table is written as follows:

```
var flu = flip(0.1);
var cold = flip(0.2);
var fever = ((cold && flip(0.3))||(flu && flip(0.5)));
```

Many other probabilistic programming languages have been proposed by adding probabilistic constructs to programming languages from procedural to functional persuasions (Gordon et al. 2014b; Kordjamshidi et al. 2015; Narayanan et al. 2016; Mansinghka and Radul 2014; McCallum et al. 2009; Paige and Wood 2014; Pfeffer 2016; Wood et al. 2014). These languages offer at least "flip"-like commands, and some offer declarative constructs that mimic plate models. For instance, in Haikaru (Narayanan et al. 2016) one can specify a Latent Dirichlet Allocation model (Blei et al. 2003) in a few lines of code, for instance specifying a plate as follows:

```
phi <~ plate _ of K:  dirichlet(word_prior)
```

In some cases probabilistic languages have been proposed in connection with probabilistic databases (Gordon et al. 2014a; Saad and Mansinghka 2016), a related technology that we mention again in Sect. 8.

Another strategy in probabilistic programming is to add a powerful library to an existing language. For instance, the Figaro toolkit offers a mature and complete probabilistic modeler and reasoner on top of the programming language Scala, together with solid methodological guidelines (Pfeffer 2016). Another powerful toolkit is available in the Infer.NET project, a framework that meshes with several languages (Minka et al. 2014). There are other toolkits that provide substantial probabilistic programming features (Bessiere et al. 2013; Salvatier et al. 2016), and even recent efforts to mix probabilistic programming with techniques from deep learning (Tran et al. 2017).

A popular probabilistic programming language with a unique and powerful feature set is Stan (Carpenter et al. 2017), whose syntax looks deceptively similar to the declarative BUGS language. In fact Stan offers imperative and modular programming, loops and even recursion. For instance, in Stan a loop can assign values to a variable repeatedly:

```
for (n in 1:N) {
    t = inv_logit(a+x[n]*b);
    y[n] ~ bernoulli(t);
    total = total + x[n];
}
```

and one can define a recursive function such as

```
int random_fib(int n) {
```

```
        if (n<2) return n + 0.02 * bernoulli(0.1);
        else return random_fib(n-1) + random_fib(n-2);
    }
```

There is now substantial activity in probabilistic programming, and the final word on the subject is not yet written.[3] Some languages include continuous distributions, disintegrations, and even symbolic manipulation of probabilities. Most languages go well beyond relational versions of Bayesian networks and Markov random fields, for instance by including recursion.

The main goal of probabilistic programming within artificial intelligence has been to "enable probabilistic modeling and machine learning to be accessible to the working programmer, who has sufficient domain expertise, but perhaps not enough expertise in probability theory or machine learning" (Gordon et al. 2014b). Probabilistic programming is not only relevant to artificial intelligence, but also attracts users interested in randomized algorithms and cryptography, and even quantum computing (Barthe et al. 2015); in fact, several issues now investigated in probabilistic programming stay closer to programming language design than to knowledge representation.

8 Inference and Learning: Some Brief Words

This survey focuses on the syntax and semantics of various languages that aim at probabilistic modeling; the goal is to serve the reader a taste of what these languages can do. Now, once a model is specified, it may be necessary to compute the probability of various events; such a computation is called an *inference*. While the first relational extensions of Bayesian networks worried mostly about inference (Bacchus 1993; Gilks et al. 1993; Poole 1993a; Wellman et al. 1992), a turning point was the development of machine learning methods for PRMs around 1999 (Friedman et al. 1999). Most languages have been, since then, accompanied by appropriate learning methods (De Raedt 2008; Getoor and Taskar 2007; Milch and Russell 2007; De Raedt et al. 2016).

A discussion of inference and learning algorithms for all languages described previously would certainly require another (even longer) survey. To keep matters at a manageable size, only a few central results are mentioned in the remainder of this section, mostly on inference algorithms.

Consider then the challenge of computing probabilities for a PRM, or a probabilistic logic program, or some probabilistic program. One strategy is to take the relational specification and translate it to a Bayesian network (or Markov random field), and then run inference in the latter (Koller et al. 1997a; Wellman et al. 1992). It may be that a particular inference does not require the whole Bayesian network;

[3] Lists of languages can be found at http://probabilistic-programming.org/wiki/Home and https://en.wikipedia.org/wiki/Probabilistic_programming_language.

only a sub-network may be *requisite* for the inference of interest. And even if the grounded Bayesian network is infinite, it may be possible to approximate inference with a suitable grounded sub-network (Laskey 2008; Pfeffer and Koller 2000).

In some cases, one may compute a desired probability without even generating this requisite grounded sub-network. Consider an example: suppose we have parvariables $X(\chi)$ and Y, associated with assessments:

$$\forall \chi : \mathbb{P}\left(X(\chi) = 1\right) = 0.4, \qquad Y = \begin{cases} 1 & \text{if } \exists \chi : \{X(\chi) = 1\}, \\ 0 & \text{otherwise.} \end{cases}$$

If we have a domain with N elements, then $\mathbb{P}\left(Y = 1\right) = 1 - (0.6)^N$; there is no need to generate the $N + 1$ requisite random variables. Example 5 shows that a more sophisticated combinatorial argument may be used to compute an inference without generating the requisite Markov random field.

Informally, *lifted inference* refers to a computation of probabilities that does not generate the requisite sub-network. Research on lifted inference started with a seminal paper by Poole (2003), who proposed a few combinatorial operators to handle some important cases. The number of lifted operators and algorithms has grown enormously, and existing surveys convey extensive references (Kersting 2012; De Raedt et al. 2016; Van den Broeck and Suciu 2017). Research on lifted inference has emphasized an abstract framework where the "input language" is just a set of parfactors (de Salvo Braz et al. 2007), similar to the ones adopted in relational Markov networks. Several techniques of lifted inference have also been applied to probabilistic logic programming (Riguzzi et al. 2017). One can also consider algorithms for approximate inference that look at parvariables rather than grounded random variables, thus working in a lifted fashion.

It is likely that most applied work must resort to approximate inference, both based on variational methods or on sampling algorithms (Koller and Friedman 2009). In fact, plate models were originally built to specify models in BUGS, a package focused on Gibbs sampling (Gilks et al. 1993). For many languages, approximate inference is all that one can hope, and the survey papers already mentioned contain a substantial number of relevant references. The use of sampling methods is particularly important in probabilistic programming, where exact inference is very difficulty. Most probabilistic programming languages in essence do inference by repeatedly running code and storing statistics.

During the development of lifted inference, an important connection has surfaced between probabilistic logic programming and probabilistic databases. The latter consist of databases where data can be annotated with probabilities; querying such a database raises the same computational problems as running inference with parfactors (Suciu et al. 2011). The literature on probabilistic databases has produced deep results on the classes of queries that can be actually be solved in polynomial time.

A natural question then is whether one can, for a fixed a language, answer any inference in polynomial time. If a language guarantees polynomial time inference, when parvariables, parfactors and query are fixed, and the input consists of the domain, then the language is *domain-liftable* (Van den Broeck 2011). While pioneering results by Jaeger show that even some rather simple languages fail to be domain-liftable (Jaeger 2000, 2014), the work on lifted inference has demonstrated that several important languages are domain-liftable (Taghipour et al. 2013; Van den Broeck et al. 2014).

Domain-liftability has a narrow focus: it concentrates on polynomial complexity when the input is just the domain. One can study the complexity of inference in a broader framework. First, we may have, as input, the parvariables and parfactors and domains and query: we then have the *inferential complexity* of the language. Another possibility is to have, as input, the domains and query (the parvariables and parfactors are fixed): we then have the *query complexity* of the language. Finally we may have, as input, just the query (the parvariables, parfactors and domains are fixed): we then have the *domain complexity* of the language. There is now a substantial set of results on these notions of complexity, both for languages based on first-order logic (Cozman and Mauá 2015; Mauá and Cozman 2016) and for probabilistic logic programming (Cozman and Mauá 2017b, c). Besides complexity questions, the theory of probabilistic programming has produced serious analysis of programming patterns and semantic foundations (Gordon et al. 2014b).

Lifted inference is not the only genuinely relational aspect of inference for the languages we have surveyed. Another important problem is deciding whether a probabilistic program has global semantics; that is, deciding whether all possible groundings of a probabilistic program do define a probability distribution (Jaeger 2002). Yet another problem is *referential uncertainty*, where one is uncertain about an association between individuals in a domain (Getoor et al. 2007). Inference in relational probabilistic languages covers a large range of techniques.

An even more bewildering variety of techniques has been developed in connection with machine learning for languages discussed in this survey. In fact, most of those languages have been proposed already with corresponding learning algorithms. This is certainly true for Probabilistic Inductive Logic Programming (De Raedt 2008; De Raedt et al. 2010), and indeed for most work on probabilistic logic programming (Riguzzi et al. 2014; Sato 1995). Machine learning has also been a central concern of probabilistic programming (Goodman et al. 2008; Gordon et al. 2014b), and a basic feature of PRMs from their inception (Friedman et al. 1999; Getoor and Taskar 2007).

Broadly speaking, there are two distinct problems that learning algorithms try to solve: one is *parameter learning*, where a set of syntactically correct statements is given, and the exact probability values must be estimated from data; the other problem is *structure learning*, where both the statements and the probability values must be extracted from data. Methods based on Inductive Logic Programming typically assume that a set of positive and negative examples is available, and the goal is to guarantee that positive examples are derived by the learned model, while negative examples are not (De Raedt 2008). Statistical methods have different assumptions, usually focusing on maximization of some score that depends on the data and the model. For parameter learning, the most popular score is the likelihood function; for structure learning, it is necessary to penalize the complexity of the statements, and a multitude of scores is available, most of them imported from the literature on Bayesian networks (Koller and Friedman 2009). The main difficulty in structure learning is the size of the space of possible sets of statements; generally some heuristic search is employed (Getoor and Taskar 2007; De Raedt et al. 2016). The vast literature on data mining and machine learning offers many possible techniques to apply, as the reader can appreciate by reading the papers cited in this survey.

9 Conclusion

This chapter has visited many different languages that aim at probabilistic modeling in knowledge representation and machine learning. As noted previously, the focus of this survey has been to provide a gentle discussion of syntactic and semantic features of existing languages, without too much technical detail; a careful review of inference has not been attempted, and the broad topic of learning algorithms has been barely scratched. Moreover, we have focused on specification languages with an "artificial intelligence motivation", and avoided languages that address specific tasks such as risk analysis or information retrieval. A hopefully satisfactory way to continue the study of languages for probabilistic modeling in artificial intelligence is to consult the list of references for this chapter.

It seems fair to say that there is no single language that can serve all purposes; a skilled practitioner cannot hope to use a single language in every application. One can find languages that are narrow and efficient, and languages that are flexible and that resist exact inference. One can find declarative and imperative languages, with both pros and cons of these programming paradigms. And certainly the future will bring an even more diverse zoo of languages. No unified set of Laws of Thought emerges from the current literature.

Acknowledgements The author was partially supported by CNPq (grant 308433/2014-9). This work was partially supported by FAPESP (grant 2016/18841-0).

References

Abadi M, Halpern JY (1994) Decidability and expressiveness for first-order logics of probability. Inf Comput 112(1):1–36

Andersen KA, Hooker JN (1994) Bayesian logic. Decis Support Syst 11:191–210

Baader F, Nutt W (2002) Basic description logics. Description logic handbook. Cambridge University Press, Cambridge, pp 47–100

Baader F, Horrocks I, Lutz C, Sattler U (2017) An introduction to description logic. Cambridge University Press, Cambridge

Bacchus F (1990) Representing and reasoning with probabilistic knowledge: a logical approach. MIT Press, Cambridge

Bacchus F (1993) Using first-order probability logic for the construction of Bayesian networks. In: Conference on uncertainty in artificial intelligence, pp 219–226

Baral C (2003) Knowledge representation, reasoning, and declarative problem solving. Cambridge University Press, Cambridge

Baral C, Gelfond M, Rushton N (2009) Probabilistic reasoning with answer sets. Theory Pract Log Program 9(1):57–144

Barthe G, Gordon AD, Katoen JP, McIver A (2015) Challenges and trends in probabilistic programming. In: Dagstuhl reports (Seminar 15181), vol 5. Dagstuhl Publishing, pp 123–141

Bessiere P, Mazer E, Ahuactzin JM, Mekhnacha K (2013) Bayesian programming. CRC Press, Boca Raton

Blei DM, Ng AY, Jordan MI (2003) Latent Dirichlet allocation. J Mach Learn Res 3:993–1022

Boole G (1958) The laws of thought. Dover edition, New York

Borgida A (1996) On the relative expressiveness of description logics and predicate logics. Artif Intell 82(1–2):353–367

Bruno G, Gilio A (1980) Applicazione del metodo del simplesso al teorema fondamentale per le probabilità nella concezione soggettivistica. Statistica 40:337–344

Bunescu R, Mooney RJ (2004) Collective information extraction with relational Markov networks. In: Annual meeting of the association for computational linguistics

Buntine WL (1994) Operations for learning with graphical models. J Artif Intell Res 2:159–225

Carpenter B, Gelman A, Hoffman MD, Lee D, Goodrich B, Betancourt M, Brubaker M, Guo J, Li P, Riddell A (2017) Stan: a probabilistic programming language. J Stat Softw 76(1):1–32. https://doi.org/10.18637/jss.v076.i01

Carvalho RN, Laskey KB, Costa PCG, Ladeira M, Santos LL, Matsumoto S (2010) UnBayes: modeling uncertainty for plausible reasoning in the semantic web. In: Wu G (ed) Semantic web. InTech, pp 1–28

Carvalho RN, Laskey KB, Costa PC (2013) PR-OWL 2.0 — bridging the gap to OWL semantics. In: URSW 2008-2010/UniDL 2010, LNAI 7123. Springer, pp 1–18

Ceylan İİ, Peñaloza R (2014) The Bayesian description logic $\mathcal{BEL}$. In: International joint conference on automated reasoning, pp 480–494

Ceylan İİ, Lukasiewicz T, Peñaloza R (2016) Complexity results for probabilistic Datalog$^{\pm}$. In: European conference on artificial intelligence, pp 1414–1422

Clark KL (1978) Negation as failure. Logic and data bases. Springer, Berlin, pp 293–322

Coletti G, Scozzafava R (2002) Probabilistic logic in a coherent setting. In: Trends in logic, vol 15. Kluwer, Dordrecht

Costa PCG, Laskey KB (2006) PR-OWL: a framework for probabilistic ontologies. In: Conference on formal ontology in information systems

Costa VS, Page D, Qazi M, Cussens J (2003) CLP($\mathcal{BN}$): constraint logic programming for probabilistic knowledge. In: Kjaerulff U, Meek C (eds) Conference on uncertainty in artificial intelligence. Morgan-Kaufmann, San Francisco, pp 517–524

Cozman FG (2000) Credal networks. Artif Intell 120:199–233

Cozman FG, Mauá DD (2015) The complexity of plate probabilistic models. In: Scalable uncertainty management. LNCS, vol 9310. Springer, Cham, pp 36–49

Cozman FG, Mauá DD (2017a) The complexity of inferences and explanations in probabilistic logic programming. Symbolic and quantitative approaches to reasoning with uncertainty. Lecture notes in computer science, vol 10369. Springer, Cham, pp 449–458

Cozman FG, Mauá DD (2017b) On the complexity of propositional and relational credal networks. Int J Approx Reason 83:298–319

Cozman FG, Mauá DD (2017c) On the semantics and complexity of probabilistic logic programs. J Artif Intell Res 60:221–262

Cozman FG, de Campos CP, da Rocha JCF (2008) Probabilistic logic with independence. Int J Approx Reason 49:3–17

Cussens J (1999) Parameter estimation in stochastic logic programs. Mach Learn 44(3):245–271

da Costa PCG, Laskey KB (2005) Of Klingons and starships: Bayesian logic for the 23rd century. In: Conference on uncertainty in artificial intelligence d'Amato C, Fanizzi N, Lukasiewicz T (2008) Tractable reasoning with Bayesian description logics. In: Greco S, Lukasiewicz T (eds) International conference on scalable uncertainty management. Lecture notes in computer science, vol 5291. Springer, pp 146–159

Dantsin E, Eiter T, Voronkov A (2001) Complexity and expressive power of logic programming. ACM Comput Surv 33(3):374–425

Darwiche A (2009) Modeling and reasoning with Bayesian networks. Cambridge University Press, Cambridge

De Bona G, Cozman FG (2017) Encoding the consistency of relational Bayesian networks. In: Encontro Nacional de Inteligência Artificial e Computacional, Uberlândia, Brasil

de Campos CP, Cozman FG, Luna JEO (2009) Assembling a consistent set of sentences in relational probabilistic logic with stochastic independence. J Appl Log 7:137–154

de Finetti B (1964) Foresight: its logical laws, its subjective sources. In: Kyburg HE Jr, Smokler HE (eds) Studies in subjective probability. Wiley, New York

De Raedt L (2008) Logical and relational learning. Springer, Berlin

De Raedt L, Kersting K (2004) Probabilistic inductive logic programming. In: International conference on algorithmic learning theory, pp 19–36

De Raedt LD, Kimmig A (2015) Probabilistic (logic) programming concepts. Mach Learn 100:5–47

De Raedt L, Frasconi P, Kersting K, Muggleton S (2010) Probabilistic inductive logic programming. Springer, Berlin

De Raedt LD, Kersting K, Natarajan S, Poole D (2016) Statistical relational artificial intelligence: logic, probability, and computation. Morgan & Claypool Publishers, San Rafael

de Salvo Braz R, Amir E, Roth D (2007) Lifted first-order probabilistic inference. In: Getoor L, Taskar B (eds) An introduction to statistical relational learning. MIT Press, Cambridge, pp 433–451

Ding Z, Peng Y, Pan R (2006) BayesOWL: uncertainty modeling in semantic web ontologies. In: Soft computing in ontologies and semantic web. Studies in fuzziness and soft computing, vol 204. Springer, Berlin, pp 3–29

Domingos P, Lowd D (2009) Markov logic: an interface layer for artificial intelligence. Morgan and Claypool, San Rafael

Eiter T, Ianni G, Krennwalner T (2009) Answer set programming: a primer. Reasoning web. Springer, Berlin, pp 40–110

Enderton HB (1972) A mathematical introduction to logic. Academic, Orlando

Fagin R, Halpern JY, Megiddo N (1990) A logic for reasoning about probabilities. Inf Comput 87:78–128

Fierens D, Blockeel H, Ramon J, Bruynooghe M (2004) Logical Bayesian networks. In: Workshop on multi-relational data mining, pp 19–30

Fierens D, Blockeel H, Bruynooghe M, Ramon J (2005) Logical Bayesian networks and their relation to other probabilistic logical models. In: Conference on inductive logic programming, pp 121–135

Fierens D, Van den Broeck G, Renkens J, Shrerionov D, Gutmann B, Janssens G, De Raedt L (2014) Inference and learning in probabilistic logic programs using weighted Boolean formulas. Theory Pract Log Program 15(3):358–401

Friedman N, Getoor L, Koller D, Pfeffer A (1999) Learning probabilistic relational models. In: International joint conference on artificial intelligence, pp 1300–1309

Fuhr N (1995) Probabilistic datalog - a logic for powerful retrieval methods. Conference on research and development in information retrieval, Seattle, Washington, pp 282–290

Gaifman H (1964) Concerning measures on first-order calculi. Isr J Math 2:1–18

Gaifman H, Snir M (1982) Probabilities over rich languages, testing and randomness. J Symb Log 47(3):495–548

Gelfond M, Lifschitz V (1988) The stable model semantics for logic programming. Proceedings of international logic programming conference and symposium 88:1070–1080

Getoor L, Grant J (2006) PRL: a probabilistic relational language. Mach Learn 62:7–31

Getoor L, Taskar B (2007) Introduction to statistical relational learning. MIT Press, Cambridge

420050_1_En_9_Chapter ☑ TYPESET ☐ DISK ☐ LE ☑ CP Disp.:6/6/2019 Pages: 37 Layout: T1-Standard

Getoor L, Friedman N, Koller D, Pfeffer A, Taskar B (2007) Probabilistic relational models. In: Introduction to statistical relational learning, MIT Press, Cambridge

Gilks WR, Thomas A, Spiegelhalter D (1993) A language and program for complex Bayesian modelling. The Statistician 43:169–178

Glesner S, Koller D (1995) Constructing flexible dynamic belief networks from first-order probabilistic knowledge bases. In: Symbolic and quantitative approaches to reasoning with uncertainty, pp 217–226

Goldman RP, Charniak E (1990) Dynamic construction of belief networks. In: Conference of uncertainty in artificial intelligence, pp 90–97

Goodman ND, Mansinghka VK, Roy D, Bonawitz K, Tenenbaum JB (2008) Church: a language for generative models. In: Conference in uncertainty in artificial intelligence, pp 220–229

Gordon AD, Grapple T, Rolland N, Russo C, Bergstrom J, Guiver J (2014a) Tabular: a schema-driven probabilistic programming language. ACM SIGPLAN Not 49(1):321–334

Gordon AD, Henzinger TA, Nori AV, Rajmani SK (2014b) Probabilistic programming. In: Proceedings of the conference on future of software engineering. ACM, New York, pp 167–181. https://doi.org/10.1145/2593882.2593900

Hadjichristodoulou S, Warren DS (2012) Probabilistic logic programming with well-founded negation. In: International symposium on multiple-valued logic, pp 232–237

Hailperin T (1976) Boole's logic and probability: a critical exposition from the standpoint of contemporary algebra, logic, and probability theory. North-Holland, Amsterdam

Hailperin T (1996) Sentential probability logic. Lehigh University Press, Bethlehem

Halpern JY (2003) Reasoning about uncertainty. MIT Press, Cambridge

Hansen P, Jaumard B (1996) Probabilistic satisfiability. Technical report G-96-31, Les Cahiers du GERAD, École Polytechique de Montréal

Heckerman D, Chickering DM, Meek C, Rounthwaite R, Kadie C (2000) Dependency networks for inference, collaborative filtering, and data visualization. J Mach Learn Res 1:49–75

Heckerman D, Meek C, Koller D (2007) Probabilistic entity-relationship models, PRMs, and plate models. In: Taskar B, Getoor L (eds) Introduction to statistical relational learning. MIT Press, Cambridge, pp 201–238

Hoover DN (1978) Probability logic. Ann Math Log 14:287–313

Horsch MC, Poole D (1990) A dynamic approach to probabilistic inference using Bayesian networks. In: Conference of uncertainty in artificial intelligence, pp 155–161

Jaeger M (2000) On the complexity of inference about probabilistic relational models. Artif Intell 117(2):297–308

Jaeger M (2002) Relational Bayesian networks: a survey. Linkop Electron Artic Comput Inf Sci 6

Jaeger M (2014) Lower complexity bounds for lifted inference. Theory Pract Log Program 15(2):246–264

Jain D, Kirchlechner B, Beetz M (2007) Extending Markov logic to model probability distributions in relational domains. In: KI 2007: advances in artificial intelligence. Lecture Notes in Computer Science, vol 4667. Springer, Berlin

Keisler HJ (1985) Probabilistic quantifiers. In: Barwise J, Feferman S (eds) Model-theoretic logic. Springer, New York, pp 509–556

Kersting K (2012) Lifted probabilistic inference. In: De Raedt L, Bessiere C, Dubois D, Doherty P, Frasconi P, Heintz F, Lucas P (eds) European conference on artificial intelligence. IOS Press, Amsterdam

Kersting K, De Raedt L, Kramer S (2000) Interpreting Bayesian logic programs. In: AAAI-2000 workshop on learning statistical models from relational data

Klinov P, Parsia B (2011) A hybrid method for probabilistic satisfiability. In: Bjorner N, Sofronie-Stokkermans V (eds) International conference on automated deduction. Springer, Berlin, pp 354–368

Koller D, Friedman N (2009) Probabilistic graphical models: principles and techniques. MIT Press, Cambridge

Koller D, Pfeffer A (1997) Object-oriented Bayesian networks. In: Conference on uncertainty in artificial intelligence, pp 302–313

Koller D, Pfeffer A (1998) Probabilistic frame-based systems. In: National conference on artificial intelligence (AAAI), pp 580–587

Koller D, Levy AY, Pfeffer A (1997a) P-CLASSIC: a tractable probablistic description logic. In: AAAI, pp 390–397

Koller D, McAllester D, Pfeffer A (1997b) Effective Bayesian inference for stochastic programs. In: AAAI, pp 740–747

Kordjamshidi P, Roth D, Wu H (2015) Saul: towards declarative learning based programming. In: International joint conference on artificial intelligence (IJCAI), pp 1844–1851

Lakshmanan LVS, Sadri F (1994) Probabilistic deductive databases. In: Symposium on logic programming, pp 254–268

Laskey KB (2008) MEBN: a language for first-order Bayesian knowledge bases. Artif Intell 172(2–3):140–178

Lee J, Wang Y (2015) A probabilistic extension of the stable model semantics. In: AAAI spring symposium on logical formalizations of commonsense reasoning, pp 96–102

Liao L, Fox D, Kautz H (2006) Location-based activity recognition. In: Advances in neural information processing systems, pp 787–794

Lukasiewicz T (1998) Probabilistic logic programming. In: European conference on artificial intelligence, pp 388–392

Lukasiewicz T (2005) Probabilistic description logic programs. In: Proceedings of the 8th European conference on symbolic and quantitative approaches to reasoning with uncertainty (ECSQARU 2005). pp 737–749, Springer, Barcelona

Lukasiewicz T (2008) Expressive probabilistic description logics. Artif Intell 172(6–7):852–883

Lukasiewicz T, Straccia U (2008) Managing uncertainty and vagueness in description logics for the semantic web. J Web Semant 6:291–308

Lukasiewicz T, Predoiu L, Stuckenschmidt H (2011) Tightly integrated probabilistic description logic programs for representing ontology mappings. Ann Math Artif Intell 63(3/4):385–425

Lunn D, Spiegelhalter D, Thomas A, Best N (2009) The BUGS project: evolution, critique and future directions. Stat Med 28:3049–3067

Lunn D, Jackson C, Best N, Thomas A, Spiegelhalter D (2012) The BUGS book: a practical introduction to Bayesian analysis. CRC Press/Chapman and Hall, Boca Raton

Mahoney S, Laskey KB (1996) Network engineering for complex belief networks. In: Conference on uncertainty in artificial intelligence

Mansinghka V, Radul A (2014) CoreVenture: a highlevel, reflective machine language for probabilistic programming. In: NIPS workshop on probabilistic programming

Mauá DD, Cozman FG (2016) The effect of combination functions on the complexity of relational Bayesian networks. In: Conference on probabilistic graphical models — JMLR workshop and conference proceedings, vol 52, pp 333–344

McCallum A, Schultz K, Singh S (2009) Factorie: probabilistic programming via imperatively defined factor graphs. In: Advances in neural information processing systems (NIPS), pp 1249–1257

McCarthy J, Hayes PJ (1969) Some philosophical problems from the standpoint of artificial intelligence. In: Meltzer B, Michie D (eds) Machine intelligence, vol 4. Edinburgh University Press, pp 463–502

Milch B, Russell S (2007) First-order probabilistic languages: into the unknown. In: International conference on inductive logic programming

Milch B, Marthi B, Russell S, Sontag D, Ong DL, Kolobov A (2005a) BLOG: probabilistic models with unknown objects. In: IJCAI

Milch B, Marthi B, Sontag D, Russell S, Ong DL, Kolobov A (2005b) Approximate inference for infinite contingent Bayesian networks. In: Artificial intelligence and statistics

Minka T, Winn JM, Guiver JP, Webster S, Zaykov Y, Yangel B, Spengler A, Bronskill J (2014) Infer.NET 2.6. Technical report, Microsoft Research Cambridge

Muggleton S (1996) Stochastic logic programs. Advances in inductive logic programming. IOS Press, Amsterdam, pp 254–264
Narayanan P, Carette J, Romano W, Shan C, Zinkov R (2016) Probabilistic inference by program transformation in Hakaru (system description). In: Functional and logic programming, pp 62–79
Neville J, Jensen D (2007) Relational dependency networks. J Mach Learn Res 8:653–692
Ng R, Subrahmanian VS (1992) Probabilistic logic programming. Inf Comput 101(2):150–201
Ngo L, Haddawy P (1997) Answering queries from context-sensitive probabilistic knowledge bases. Theor Comput Sci 171(1–2):147–177
Nickles M, Mileo A (2015) A system for probabilistic inductive answer set programming. In: International Conference on scalable uncertainty management, vol 9310. Lecture notes in computer science, pp 99–105
Nilsson NJ (1986) Probabilistic logic. Artif Intell 28:71–87
Nitti D, Laet TD, Raedt LD (2016) Probabilistic logic programming with hybrid domains. Mach Learn 103(3):407–449
Paige B, Wood F (2014) A compilation target for probabilistic programming languages. International conference on machine learning, JMLR 32:1935–1943
Park S, Pfenning F, Thrun S (2008) A probabilistic language based on sampling functions. ACM Trans Program Lang Syst 31(1):4:1–4:45
Pearl J (1988) Probabilistic reasoning in intelligent systems: networks of plausible inference. Morgan Kaufmann, San Mateo
Pearl J (2009) Causality: models, reasoning, and inference, 2nd edn. Cambridge University Press, Cambridge
Pfeffer A (2001) IBAL: a probabilistic rational programming language. In: International joint conference on artificial intelligence, pp 733–740
Pfeffer A (2016) Practical probabilistic programming. Manning Publications, Shelter Island
Pfeffer A, Koller D (2000) Semantics and inference for recursive probability models. In: AAAI, pp 538–544
Poole D (1993a) Average-case analysis of a search algorithm for estimating prior and posterior probabilities in Bayesian networks with extreme probabilities. In: 13th international joint conference on artificial intelligence, pp 606–612
Poole D (1993b) Probabilistic Horn abduction and Bayesian networks. Artif Intell 64:81–129
Poole D (1997) The independent choice logic for modelling multiple agents under uncertainty. Artif Intell 94(1/2):7–56
Poole D (2003) First-order probabilistic inference. In: International joint conference on artificial intelligence (IJCAI), pp 985–991
Poole D (2008) The independent choice logic and beyond. In: De Raedt L, Frasconi P, Kersting K, Muggleton S (eds) Probabilistic inductive logic programming. Lecture Notes in Computer Science, vol 4911. Springer, Berlin, pp 222–243
Poole D (2010) Probabilistic programming languages: independent choices and deterministic systems. In: Dechter R, Geffner H, Halpern JY (eds) Heuristics, probability and causality — a tribute to Judea pearl. College Publications, pp 253–269
Poole D, Buchman D, Natarajan S, Kersting K (2012) Aggregation and population growth: the relational logistic regression and Markov logic cases. In: International Workshop on Statistical Relational AI
Pourret O, Naim P, Marcot B (2008) Bayesian networks – a practical guide to applications. Wiley, New York
Predoiu L, Stuckenschmidt H (2009) Probabilistic models for the semantic web. The semantic web for knowledge and data management: technologies and practices. IGI Global, Hershey, pp 74–105
Richardson M, Domingos P (2006) Markov logic networks. Mach Learn 62(1–2):107–136
Riguzzi F (2015) The distribution semantics is well-defined for all normal programs. In: Riguzzi F, Vennekens J (eds) International workshop on probabilistic logic programming, CEUR workshop proceedings, vol 1413, pp 69–84

420050_1_En_9_Chapter ☑ TYPESET ☐ DISK ☐ LE ☑ CP Disp.:6/6/2019 Pages: 37 Layout: T1-Standard

Riguzzi F, Bellodi E, Zese R (2014) A history of probabilistic inductive logic programming. Front Robot AI 1:1–5
Riguzzi F, Bellodi E, Lamma E, Zese R (2015) Probabilistic description logics under the distribution semantics. Semant Web 6(5):477–501
Riguzzi F, Bellodi E, Zese R, Cota G, Lamma E (2017) A survey of lifted inference approaches for probabilistic logic programming under the distribution semantics. Int J Approx Reason 80:313–333
Russell S (2015) Unifying logic and probability. Commun ACM 58(7):88–97
Saad F, Mansinghka VK (2016) A probabilistic programming approach to probabilistic data analysis. In: Advances in neural information processing systems (NIPS)
Sadeghi K, Lauritzen S (2014) Markov properties for mixed graphs. Bernoulli 20(2):676–696
Salvatier J, Wiecki TV, Fonnesbeck C (2016) Probabilistic programming in Python using PyMC3. PeerJ
Sanner S (2011) Relational dynamic influence diagram language (RDDL): language description. Technical report, NICTA and Australian National University
Sato T (1995) A statistical learning method for logic programs with distribution semantics. In: Conference on logic programming, pp 715–729
Sato T, Kameya Y (2001) Parameter learning of logic programs for symbolic-statistical modeling. J Artif Intell Res 15:391–454
Sato T, Kameya Y, Zhou NF (2005) Generative modeling with failure in PRISM. In: International joint conference on artificial intelligence, pp 847–852
Scott D, Krauss P (1966) Assigning probabilities to logical formulas. In: Suppes Hintikka (ed) Aspects of inductive logic. North-Holland, Amsterdam, pp 219–264
Staker R (2002) Reasoning in expressive description logics using belief networks. International conference on information and knowledge engineering. Las Vegas, USA, pp 489–495
Suciu D, Oiteanu D, Ré C, Koch C (2011) Probabilistic databases. Morgan & Claypool Publishers, San Rafael
Taghipour N, Fierens D, Van den Broeck G, Davis J, Blockeel H (2013) Completeness results for lifted variable elimination. International conference on artificial intelligence and statistics (AISTATS). Scottsdale, USA, pp 572–580
Taskar B, Abbeel P, Wong MF, Koller D (2007) Relational Markov networks. In: Getoor L, Taskar B (eds) Introduction to statistical relational learning. MIT Press, Cambridge, pp 175–199
Thrun S (2000) Towards programming tools for robots that integrate probabilistic computation and learning. In: IEEE international conference on robotics and automation (ICRA)
Toutanova K, Klein D, Manning CD, Singer Y (2003) Feature-rich part-of-speech tagging with a cyclic dependency network. Conference of the North American chapter of the association for computational linguistics on human language technology 1:173–180
Tran D, Hoffman MD, Saurous RA, Brevdo E, Murphy K, Blei DM (2017) Deep probabilistic programming. In: International conference on learning representations
Van den Broeck G (2011) On the completeness of first-order knowledge compilation for lifted probabilistic inference. In: Neural processing information systems, pp 1386–1394
Van den Broeck G, Suciu D (2017) Query processing on probabilistic data: a survey. Found Trends Databases 7:197–341
Van den Broeck G, Wannes M, Darwiche A (2014) Skolemization for weighted first-order model counting. In: International conference on principles of knowledge representation and reasoning, pp 111–120
Van Gelder A, Ross KA, Schlipf JS (1991) The well-founded semantics for general logic programs. J Assoc Comput Mach 38(3):620–650
Vennekens J, Verbaeten S, Bruynooghe M (2004) Logic programs with annotated disjunctions. In: Logic programming - ICLP. LNCS, vol 3132. Springer, Berlin, pp 431–445
Vennekens J, Denecker M, Bruynoogue M (2009) CP-logic: a language of causal probabilistic events and its relation to logic programming. Theory Pract Log Program 9(3):245–308

Wellman MP, Breese JS, Goldman RP (1992) From knowledge bases to decision models. Knowl Eng Rev 7(1):35–53
Wood F, van de Meent JW, Mansinghka V (2014) A new approach to probabilistic programming inference. In: International conference on artificial intelligence and statistics, pp 1024–1032
Wu Y, Li L, Russell S, Bodik R (2016) Swift: compiled inference for probabilistic programming languages. In: International joint conference on artificial intelligence (IJCAI)
Yelland PM (1999) Market analysis using a combination of Bayesian networks and description logics. Technical report SMLI TR-99-78, Sun Microsystems Laboratories
Yones HLS, Littman ML (2004) PPDDL 1.0: an extension to PDDL for expressing planning domains with probabilistic effects. Technical report CMU-CS-04-167, Carnegie Mellon University, Pittsburgh, PA