Probabilistic Inductive Logic Programming

Luc De Raedt[1] and Kristian Kersting[2]

[1] Departement Computerwetenschappen, K.U. Leuven
Celestijnenlaan 200A - bus 2402, B-3001 Heverlee, Belgium
Luc.DeRaedt@cs.kuleuven.be
[2] CSAIL, Massachusetts Institute of Technologie
32 Vassar Street, Cambridge, MA 02139-4307, USA
kersting@csail.mit.edu

Abstract. Probabilistic inductive logic programming aka. statistical relational learning addresses one of the central questions of artificial intelligence: the integration of probabilistic reasoning with machine learning and first order and relational logic representations. A rich variety of different formalisms and learning techniques have been developed. A unifying characterization of the underlying learning settings, however, is missing so far.
In this chapter, we start from inductive logic programming and sketch how the inductive logic programming formalisms, settings and techniques can be extended to the statistical case. More precisely, we outline three classical settings for inductive logic programming, namely *learning from entailment*, *learning from interpretations*, and *learning from proofs or traces*, and show how they can be adapted to cover state-of-the-art statistical relational learning approaches.

1 Introduction

One of the central open questions of artificial intelligence is concerned with combining expressive knowledge representation formalisms such as relational and first-order logic with principled probabilistic and statistical approaches to inference and learning. Traditionally, relational and logical reasoning, probabilistic and statistical reasoning, and machine learning are research fields in their own rights. Nowadays, they are becoming increasingly intertwined. A major driving force is the explosive growth in the amount of heterogeneous data that is being collected in the business and scientific world. Example domains include bioinformatics, transportation systems, communication networks, social network analysis, citation analysis, and robotics. They provide *uncertain* information about varying numbers of entities and relationships among the entities, i.e., about *relational* domains. Traditional machine learning approaches are able to cope either with uncertainty or with relational representations but typically not with both

It is therefore not surprising that there has been a significant interest in integrating statistical learning with first order logic and relational representations. [14] has introduced a probabilistic variant of Comprehensive Unification Formalism (CUF). In a similar manner, Muggleton [38] and Cussens [4] have upgraded stochastic grammars towards *stochastic logic programs*; Chapter 9 presents an application of them to protein fold discovery. Sato [53] has introduced *probabilistic distributional semantics* for

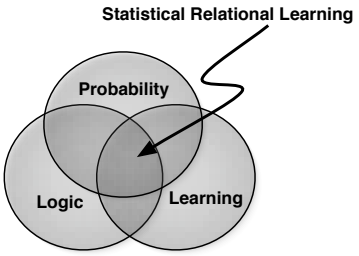

Fig. 1. Probabilistic inductive logic programming aka. statistical relational learning combines probability, logic, and learning.

logic programs; latest developments of this approach are presented in Chapter 5. Taskar *et al.* [60] have upgraded Markov networks towards *relational Markov networks*, and Domingos and Richardson [12] towards *Markov logic networks* (see also Chapter 4). Another research stream includes Poole's *independent choice Logic* [49] as reviewed in Chapter 8, Ngo and Haddawy's *probabilistic-logic programs* [45], Jäger's *relational Bayesian networks* [21], Pfeffer's [47] and Getoor's [17] *probabilistic relational models*, and Kersting and De Raedt's *Bayesian logic programs* [26, see also Chapter 7], and has investigated logical and relational extensions of Bayesian networks. Neville and Jensen [44] developed relational dependency networks. This newly emerging research field is known under the name of *statistical relational learning* or *probabilistic inductive logic programming*, cf. Figure 1, and

> deals with *machine learning* and *data mining* in *relational domains* where observations may be *missing*, *partially observed*, and/or *noisy*.

Employing relational and logical abstraction within statistical learning has two advantages. First, variables, i.e., placeholders for entities allow one to make abstraction of specific entities. Second, unification allows one to share information among entities. Thus, instead of learning regularities for each single entity independently, statistical relational learning aims at finding general regularities among groups of entities. The learned knowledge is declarative and compact, which makes it much easier for people to understand and to validated. Although, the learned knowledge must be recombined at run time using some reasoning mechanism such as backward chaining or resolution, which bears additional computational costs, statistical relational models are more flexible, context-aware, and offer — in principle — the full power of logical reasoning. Moreover, in many applications, there is a rich background theory available, which can efficiently and elegantly be represented as sets of general regularities. This is important because background knowledge often improves the quality of learning as it focuses learning on relevant patterns, i.e., restricts the search space. While learning, relational and logical abstraction allow one to reuse experience: learning about one entity improves the prediction for other entities; it might even generalize to objects, which have never been observed before. Thus, relational and logical abstraction can make statistical learning more robust and efficient. This has been proven beneficial in many fascinating

real-world applications in citation analysis, web mining, natural language processing, robotics, bio- and chemo-informatics, electronic games, and activity recognition.

Whereas most of the existing works on statistical relational learning have started from a statistical and probabilistic learning perspective and extended probabilistic formalisms with relational aspects, we will take a different perspective, in which we will start from inductive logic programming (ILP) [43], which is often also called *multi-relational data mining* (MRDM) [13]. ILP is a research field at the intersection of machine learning and logic programming. It aims at a formal framework as well as practical algorithms for inductively learning relational descriptions (in the form of logic programs) from examples and background knowledge. However, it does not explicitly deal with uncertainty such as missing or noisy information. Therefore, we will study how inductive logic programming formalisms, settings and techniques can be extended to deal with probabilities. At the same time, it is not our intention to provide a complete survey of statistical relational learning (as [9] does), but rather to focus on the principles that underlay this new and exciting subfield of artificial intelligence.

We call the resulting framework *probabilistic* ILP. It aims at a formal framework for statistical relational learning. Dealing explicitly with uncertainty makes probabilistic ILP more powerful than ILP and, in turn, than traditional attribute-value approaches. Moreover, there are several benefits of an ILP view on statistical relational learning. First of all, classical ILP learning settings — as we will argue — naturally carry over to the probabilistic case. The probabilistic ILP settings make abstraction of specific probabilistic relational and first order logical representations and inference and learning algorithms yielding — for the first time — general statistical relational learning settings. Second, many ILP concepts and techniques such as *more–general–than*, *refinement operators*, *least general generalization*, and *greatest lower bound* can be reused. Therefore, many ILP learning algorithms such as Quinlan's FOIL and De Raedt and Dehaspe's CLAUDIEN can easily be adapted. Third, the ILP perspective highlights the importance of background knowledge within statistical relational learning. The research on ILP and on artificial intelligence in general has shown that background knowledge is the key to success in many applications. Finally, an ILP approach should make statistical relational learning more intuitive to those coming from an ILP background and should cross-fertilize ideas developed in ILP and statistical learning.

We will proceed as follows. After reviewing the basic concepts of logic programming in Section 2 and of inductive logic programming in Section 3, we will extend inductive logic programming to deal explicitly with uncertainty in Section 4. Based on this foundation, we will introduce probabilistic variants of classical ILP settings for learning from entailment, from interpretations and from proofs. Each of the resulting settings will be exemplified with different probabilistic logic representations, examples and probability distributions.

2 Logic Programming Concepts

To introduce logic programs, consider Figure 2, containing two programs, *grandparent* and *nat*. Formally speaking, we have that `grandparent/2`, `parent/2`, and `nat/1` are **predicates** (with their *arity*, i.e., number of arguments listed explicitly) `jef`, `paul` and

```
parent(jef,paul).          nat(0).
parent(paul,ann).          nat(s(X)) :- nat(X).
grandparent(X,Y) :- parent(X,Z), parent(Z,Y).
```

Fig. 2. Two logic programs, *grandparent* and *nat*.

ann are **constants** and X, Y and Z are **variables** All constants and variables are also **terms** In addition, there exist structured terms such as s(X), which contains the **functor** s/1 of arity 1 and the term X. Constants are often considered as functors of arity 0. A first order **alphabet** Σ is a set of predicate symbols, constant symbols and functor symbols. *Atoms* are predicate symbols followed by the necessary number of terms, e.g., parent(jef, paul), nat(s(X)), parent(X, Z), etc. **Literals** are atoms nat(s(X)) (positive literal) and their negations not nat(s(X)) (negative literals). We are now able to define the key concept of a **definite clause**. Definite clauses are formulas of the form

$$\mathtt{A} :\!-\mathtt{B}_1, \ldots, \mathtt{B}_\mathtt{m}$$

where A and the $\mathtt{B}_\mathtt{i}$ are logical atoms and all variables are understood to be universally quantified. For instance, the clause c

$$c \equiv \mathtt{grandparent(X, Y)} :\!-\mathtt{parent(X, Z), parent(Z, Y)}$$

can be read as X is the grandparent of Y if X is a parent of Z and Z is a parent of Y. We call grandparent(X, Y) the $\mathrm{head}(c)$ of this clause, and parent(X, Z), parent(Z, Y) the $\mathrm{body}(c)$. Clauses with an empty body such as parent(jef, paul) are **facts**. A (definite) clause program (or **logic program** for short) consists of a set of clauses. In Figure 2, there are thus two logic programs, one defining grandparent/2 and one defining nat/1. The set of variables in a term, atom, conjunction or clause E, is denoted as $\mathrm{Var}(E)$, e.g., $\mathrm{Var}(c) = \{\mathtt{X}, \mathtt{Y}, \mathtt{Z}\}$. A term, atom or clause E is **ground** when there is no variable occurring in E, i.e. $\mathrm{Var}(E) = \emptyset$. A clause c is **range-restricted** when all variables in the head of the clause also appear in the body of the clause, i.e., $\mathrm{Var}(\mathrm{head}(c)) \subseteq |Vars(\mathrm{body}(c))$.

A **substitution** $\theta = \{V_1/t_1, \ldots, V_n/t_n\}$, e.g. $\{\mathtt{Y}/\mathtt{ann}\}$, is an assignment of terms t_i to variables V_i. Applying a substitution θ to a term, atom or clause e yields the instantiated term, atom, or clause $e\theta$ where all occurrences of the variables V_i are simultaneously replaced by the term t_i, e.g. $c\theta$ is

$$c' \equiv \mathtt{grandparent(X, ann)} :\!-\mathtt{parent(X, Z), parent(Z, ann)}\ .$$

A clause c_1 θ-subsumes [3] $\{\mathrm{head}(c_2)\theta\} \cup \mathrm{body}(c_2)\theta \subset \{\mathrm{head}(c_1)\} \cup \mathrm{body}(c_1)$. The **Herbrand base** of a logic program P, denoted as $\mathrm{hb}(P)$, is the set of all ground atoms constructed with the predicate, constant and function symbols in the alphabet of P.

[3] The definition of θ-subsumption also applies to conjunctions of literals, as these can also be defined as set of literals.

Example 1. The Herbrand bases of the *nat* and *grandparent* logic programs are

$$\begin{aligned}
\mathrm{hb}(nat) &= \{\mathtt{nat(0)}, \mathtt{nat(s(0))}, \mathtt{nat(s(s(0)))}, ...\} \\
\text{and } \mathrm{hb}(grandparent) &= \{\mathtt{parent(ann, ann)}, \mathtt{parent(jef, jef)}, \\
&\quad \mathtt{parent(paul, paul)}, \mathtt{parent(ann, jef)}, \mathtt{parent(jef, ann)}, ..., \\
&\quad \mathtt{grandparent(ann, ann)}, \mathtt{grandparent(jef, jef)}, ...\}.
\end{aligned}$$

A **Herbrand interpretation** for a logic program P is a subset of $\mathrm{hb}(P)$. A Herbrand interpretation I is a **model** if and only if for all substitutions θ such that $\mathrm{body}(c)\theta \subseteq I$ holds, it also holds that $\mathrm{head}(c)\theta \in I$. The interpretation I is a model of a logic program P if I is a model of all clauses in P. A clause c (logic program P) **entails** another clause c' (logic program P'), denoted as $c \models c'$ ($P \models P'$), if and only if, each model of c (P) is also a model of c' (P'). Clearly, if clause c (program P) θ-subsumes clause c' (program P') then c (P) entails c' (P'), but the reverse is not true.

The **least Herbrand model** $\mathrm{LH}(P)$, which constitutes the semantics of the logic program P, consists of all facts $f \in \mathrm{hb}(P)$ such that P logically entails f, i.e. $P \models f$. All ground atoms in the least Herbrand model are provable. **Proofs** are typically constructed using the **SLD-resolution** procedure: given a **goal** :-$\mathtt{G_1}, \mathtt{G_2} \ldots, \mathtt{G_n}$ and a clause $\mathtt{G}$:-$\mathtt{L_1}, \ldots, \mathtt{L_m}$ such that $\mathtt{G_1}\theta = \mathtt{G}\theta$, applying SLD resolution yields the new goal :-$\mathtt{L_1}\theta, \ldots, \mathtt{L_m}\theta, \mathtt{G_2}\theta \ldots, \mathtt{G_n}\theta$. A *successful* refutation, i.e., a proof of a goal is then a sequence of resolution steps yielding the empty goal, i.e. :- . *Failed* proofs do not end in the empty goal.

Example 2. The atom $\mathtt{grandparent(jeff, ann)}$ is true because of

:-$\mathtt{grandparent(jeff, ann)}$
:-$\mathtt{parent(jeff, Z)}, \mathtt{parent(Z, ann)}$
:-$\mathtt{parent(paul, ann)}$
:-

Resolution is employed by many theorem provers (such as Prolog). Indeed, when given the goal $\mathtt{grandparent(jeff, ann)}$, Prolog would compute the above successful resolution refutation and answer that the goal is true.

For a detailed introduction to logic programming, we refer to [33], for a more gentle introduction, we refer to [15], and for a detailed discussion of Prolog, see [58].

3 Inductive Logic Programming (ILP) and its Settings

Inductive logic programming is concerned with finding a hypothesis H (a logic program, i.e. a definite clause program) from a set of positive and negative examples *Pos* and *Neg*.

Example 3 (Adapted from Example 1.1 in [32]). Consider learning a definition for the $\mathtt{daughter}/2$ predicate, i.e., a set of clauses with head predicates over $\mathtt{daughter}/2$,

given the following facts as learning examples

Pos	daughter(dorothy, ann).
	daughter(dorothy, brian).
Neg	daughter(rex, ann).
	daughter(rex, brian).

Additionally, we have some general knowledge called background knowledge B, which describes the family relationships and sex of each person:

```
mother(ann, dorothy). female(dorothy).        female(ann).
mother(ann, rex).     father(brian, dorothy). father(brian, rex).
```

From this information, we could induce H

```
daughter(C, P) :- female(C), mother(P, C).
daughter(C, P) :- female(C), father(P, C).
```

which perfectly explains the examples in terms of the background knowledge, i.e., *Pos* are entailed by H together with B, but *Neg* are not entailed.

More formally, ILP is concerned with the following learning problem.

Definition 1 (ILP Learning Problem). Given *a set of positive and negative examples* Pos *and* Neg *over some language $\mathcal{L}_E$, a background theory B, in the form of a set of definite clauses, a hypothesis language $\mathcal{L}_H$, which specifies the clauses that are allowed in hypotheses, and a covers relation $covers(e, H, B) \in \{0, 1\}$, which basically returns the classification of an example e with respect to H and B,* **find** *a hypothesis H in $\mathcal{H}$ that covers (with respect to the background theory B) all positive examples in* Pos *(completeness) and none of the negative examples in* Neg *(consistency).*

The language $\mathcal{L}_E$ chosen for representing the examples together with the *covers* relation determines the inductive logic programming setting. Various settings have been considered in the literature [7]. In the following, we will formalize learning from *entailment* [48] and from *interpretations* [20, 8]. We further introduce a novel, intermediate setting, which we call learning from *proofs*. It is inspired on the seminal work by [55].

3.1 Learning from Entailment

Learning from entailment is by far the most popular ILP setting and it is addressed by a wide variety of well-known ILP systems such as FOIL [50], PROGOL [37], and ALEPH [56].

Definition 2 (Covers Relation for Learning from Entailment). *When learning from entailment, the examples are definite clauses and a hypothesis H covers an example e with respect to the background theory B if and only if $B \cup H \models e$, i.e., each model of $B \cup H$ is also a model of e.*

In many well-known systems, such as FOIL, one requires that the examples are ground facts, a special form of clauses. To illustrate the above setting, consider the following example inspired on the well-known mutagenicity application [57].

Example 4. Consider the following facts in the background theory B, which describe part of molecule 225.

$\texttt{molecule}(225).$
$\texttt{logmutag}(225, 0.64).$
$\texttt{lumo}(225, -1.785).$
$\texttt{logp}(225, 1.01).$
$\texttt{nitro}(225, [\texttt{f1\_4}, \texttt{f1\_8}, \texttt{f1\_10}, \texttt{f1\_9}]).$
$\texttt{atom}(225, \texttt{f1\_1}, \texttt{c}, 21, 0.187).$
$\texttt{atom}(225, \texttt{f1\_2}, \texttt{c}, 21, -0.143).$
$\texttt{atom}(225, \texttt{f1\_3}, \texttt{c}, 21, -0.143).$
$\texttt{atom}(225, \texttt{f1\_4}, \texttt{c}, 21, -0.013).$
$\texttt{atom}(225, \texttt{f1\_5}, \texttt{o}, 52, -0.043).$
$\texttt{bond}(225, \texttt{f1\_1}, \texttt{f1\_2}, 7).$
$\texttt{bond}(225, \texttt{f1\_2}, \texttt{f1\_3}, 7).$
$\texttt{bond}(225, \texttt{f1\_3}, \texttt{f1\_4}, 7).$
$\texttt{bond}(225, \texttt{f1\_4}, \texttt{f1\_5}, 7).$
$\texttt{bond}(225, \texttt{f1\_5}, \texttt{f1\_1}, 7).$
$\texttt{bond}(225, \texttt{f1\_8}, \texttt{f1\_9}, 2).$
$\texttt{bond}(225, \texttt{f1\_8}, \texttt{f1\_10}, 2).$
$\texttt{bond}(225, \texttt{f1\_1}, \texttt{f1\_11}, 1).$
$\texttt{bond}(225, \texttt{f1\_11}, \texttt{f1\_12}, 2).$
$\texttt{bond}(225, \texttt{f1\_11}, \texttt{f1\_13}, 1).$
...
$\texttt{ring\_size\_5}(225, [\texttt{f1\_5}, \texttt{f1\_1}, \texttt{f1\_2}, \texttt{f1\_3}, \texttt{f1\_4}]).$
$\texttt{hetero\_aromatic\_5\_ring}(225, [\texttt{f1\_5}, \texttt{f1\_1}, \texttt{f1\_2}, \texttt{f1\_3}, \texttt{f1\_4}]).$
...

Consider now the positive example $\texttt{mutagenic}(225)$. It is covered by H

$$\texttt{mutagenic}(\texttt{M}) \ :- \ \texttt{nitro}(\texttt{M}, \texttt{R1}), \texttt{logp}(\texttt{M}, \texttt{C}), \texttt{C} > 1.$$

together with the background knowledge B, because $H \cup B$ entails the example. To see this, we unify $\texttt{mutagenic}(225)$ with the clause's head. This yields

$$\texttt{mutagenic}(225) \ :- \ \texttt{nitro}(225, \texttt{R1}), \texttt{logp}(225, \texttt{C}), \texttt{C} > 1.$$

Now, $\texttt{nitro}(225, \texttt{R1})$ unifies with the fifth ground atom (left-hand side column) in B, and $\texttt{logp}(225, \texttt{C})$ with the fourth one. Because $1.01 > 1$, we found a proof of $\texttt{mutagenic}(225)$.

3.2 Learning from Interpretations

The learning from interpretations setting [8] upgrades boolean concept-learning in computational learning theory [61].

Definition 3 (Covers Relational for Learning from Interpretations). *When learning from interpretations, the examples are Herbrand interpretations and a hypothesis H covers an example e with respect to the background theory B if and only if e is a model of $B \cup H$.*

Recall that Herbrand interpretations are sets of true ground facts and they completely describe a possible situation.

Example 5. Consider the interpretation I, which is the union of B

$$\begin{aligned} B = \{&\texttt{father}(\texttt{henry},\texttt{bill}), \texttt{father}(\texttt{alan},\texttt{betsy}), \texttt{father}(\texttt{alan},\texttt{benny}),\\ &\texttt{father}(\texttt{brian},\texttt{bonnie}), \texttt{father}(\texttt{bill},\texttt{carl}), \texttt{father}(\texttt{benny},\texttt{cecily}),\\ &\texttt{father}(\texttt{carl},\texttt{dennis}), \texttt{mother}(\texttt{ann},\texttt{bill}), \texttt{mother}(\texttt{ann},\texttt{betsy}),\\ &\texttt{mother}(\texttt{ann},\texttt{bonnie}), \texttt{mother}(\texttt{alice},\texttt{benny}), \texttt{mother}(\texttt{betsy},\texttt{carl}),\\ &\texttt{mother}(\texttt{bonnie},\texttt{cecily}), \texttt{mother}(\texttt{cecily},\texttt{dennis}), \texttt{founder}(\texttt{henry}),\\ &\texttt{founder}(\texttt{alan}), \texttt{founder}(\texttt{ann}), \texttt{founder}(\texttt{brian}), \texttt{founder}(\texttt{alice})\} \end{aligned}$$

and

$$C = \{\texttt{carrier}(\texttt{alan}), \texttt{carrier}(\texttt{ann}), \texttt{carrier}(\texttt{betsy})\}\ .$$

The interpretation I is covered by the clause c

$$\texttt{carrier}(\texttt{X}) \ :- \ \texttt{mother}(\texttt{M},\texttt{X}), \texttt{carrier}(\texttt{M}), \texttt{father}(\texttt{F},\texttt{X}), \texttt{carrier}(\texttt{F}).$$

because I is a model of c, i.e., for all substitutions θ such that $\mathrm{body}(c)\theta \subseteq I$, it holds that $\mathrm{head}(c)\theta \in I$.

The key difference between learning from interpretations and learning from entailment is that interpretations carry much more — even complete — information. Indeed, when learning from entailment, an example can consist of a *single* fact, whereas when learning from interpretations, all facts that hold in the example are known. Therefore, learning from interpretations is typically easier and computationally more tractable than learning from entailment, cf. [7].

3.3 Learning from Proofs

Because learning from entailment (with ground facts as examples) and interpretations occupy extreme positions with respect to the information the examples carry, it is interesting to investigate intermediate positions. Shapiro's [55] Model Inference System (MIS) fits nicely within the learning from entailment setting where examples are facts. However, to deal with missing information, Shapiro employs a clever strategy: MIS queries the users for missing information by asking them for the truth-value of facts. The answers to these queries allow MIS to reconstruct the trace or the proof of the positive examples. Inspired by Shapiro, we define the learning from proofs setting.

Definition 4 (Covers Relation for Learning from Proofs). *When learning from proofs, the examples are ground proof-trees and an example e is covered by a hypothesis H with respect to the background theory B if and only if e is a proof-tree for $H \cup B$.*

At this point, there exist various possible forms of proof-trees. Here, we will — for reasons that will become clear later — assume that the proof-tree is given in the form of a ground and-tree where the nodes contain ground atoms. More formally:

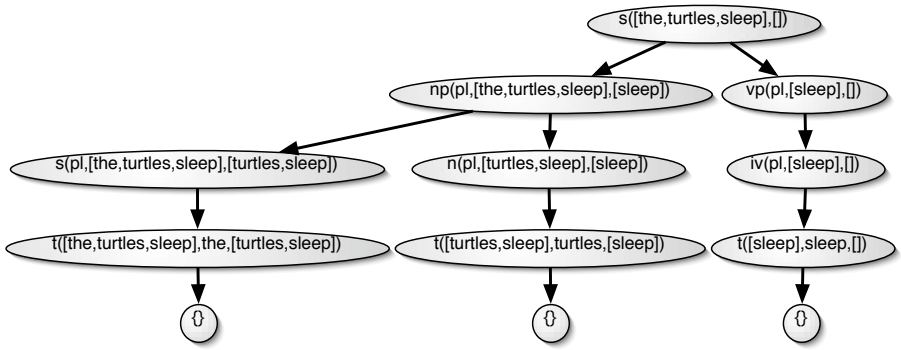

Fig. 3. A proof tree, which is covered by the definite clause grammar in Example 6. Symbols are abbreviated.

Definition 5 (Proof Tree). *A tree t is a proof-tree for a logic program T if and only if t is a rooted tree where for every node $n \in t$ with* children(n) *satisfies the property that there exists a substitution θ and a clause $c \in T$ such that $n = \mathrm{head}(c)\theta$ and* children$(n) = \mathrm{body}(c)\theta$.

Example 6. Consider the following *definite clause grammar*.

$$\begin{aligned}
\mathtt{sentence}(\mathtt{A},\mathtt{B}) &: -\mathtt{noun_phrase}(\mathtt{C},\mathtt{A},\mathtt{D}), \mathtt{verb_phrase}(\mathtt{C},\mathtt{D},\mathtt{B}).\\
\mathtt{noun_phrase}(\mathtt{A},\mathtt{B},\mathtt{C}) &: -\mathtt{article}(\mathtt{A},\mathtt{B},\mathtt{D}), \mathtt{noun}(\mathtt{A},\mathtt{D},\mathtt{C}).\\
\mathtt{verb_phrase}(\mathtt{A},\mathtt{B},\mathtt{C}) &: -\mathtt{intransitive_verb}(\mathtt{A},\mathtt{B},\mathtt{C}).\\
\mathtt{article}(\mathtt{singular},\mathtt{A},\mathtt{B}) &: -\mathtt{terminal}(\mathtt{A},\mathtt{a},\mathtt{B}).\\
\mathtt{article}(\mathtt{singular},\mathtt{A},\mathtt{B}) &: -\mathtt{terminal}(\mathtt{A},\mathtt{the},\mathtt{B}).\\
\mathtt{article}(\mathtt{plural},\mathtt{A},\mathtt{B}) &: -\mathtt{terminal}(\mathtt{A},\mathtt{the},\mathtt{B}).\\
\mathtt{noun}(\mathtt{singular},\mathtt{A},\mathtt{B}) &: -\mathtt{terminal}(\mathtt{A},\mathtt{turtle},\mathtt{B}).\\
\mathtt{noun}(\mathtt{plural},\mathtt{A},\mathtt{B}) &: -\mathtt{terminal}(\mathtt{A},\mathtt{turtles},\mathtt{B}).\\
\mathtt{intransitive_verb}(\mathtt{singular},\mathtt{A},\mathtt{B}) &: -\mathtt{terminal}(\mathtt{A},\mathtt{sleeps},\mathtt{B}).\\
\mathtt{intransitive_verb}(\mathtt{plural},\mathtt{A},\mathtt{B}) &: -\mathtt{terminal}(\mathtt{A},\mathtt{sleep},\mathtt{B}).\\
\mathtt{terminal}([\mathtt{A}|\mathtt{B}],\mathtt{A},\mathtt{B}) &.
\end{aligned}$$

It covers the proof tree shown in Figure 3.

Proof-trees contain — as interpretations — a lot of information. Indeed, they contain instances of the clauses that were used in the proofs. Therefore, it may be hard for the user to provide this type of examples. Even though this is generally true, there exist specific situations for which this is feasible. Indeed, consider tree banks such as the UPenn Wall Street Journal corpus [35], which contain parse trees. These trees directly correspond to the proof-trees we talk about.

4 Probabilistic ILP Settings

Indeed, ILP has been developed for coping with relational data. It does not, however, handle uncertainty in a principled way. In the reminder of this paper, we will show how to extend the ILP settings to the probabilistic case. To do so, we shall quickly review some of the key concepts and notation that will be used [4].

Let $\mathtt{X}$ be a random variable with a finite domain $\mathrm{domain}(\mathtt{X}) = \{x_1, x_2, \ldots, x_n\}$ of possible states. We use the notation $\mathbf{P}(\mathtt{X})$ to denote the probability distribution over $\mathrm{domain}(\mathtt{X})$, and $P(\mathtt{X} = x_i)$ or $P(x_i)$ to denote the probability that the random variable $\mathtt{X}$ takes the value $x_i \in \mathrm{domain}\,\mathtt{X}$. For instance, consider the random variable (or proposition) $\mathtt{earthquake}$ with domain $\{\mathit{true}, \mathit{false}\}$. Then, $\mathbf{P}(\mathtt{earthquake})$ denotes the probability distribution, and $P(\mathtt{earthquake} = \mathit{false})$ (or, in shorthand notation, $P(\neg\mathtt{earthquake})$) denotes the probability that $\mathtt{earthquake}$ is *false*. The distribution $\mathbf{P}(\mathtt{X}_1, \cdots, \mathtt{X}_n)$ over a set of random variables $\{\mathtt{X}_1, \cdots, \mathtt{X}_n\}$, $n > 1$, is called **joint probability distributions**. For instance, we may be interested in the joint probability that $\mathtt{earthquake} = \mathit{true}$ and $\mathtt{burglary} = \mathit{true}$ at the same time. Generalizing the above notation, we will be using the notation $\mathbf{P}(\mathtt{earthquake}, \mathtt{burglary})$ and $P(\mathtt{earthquake} = \mathit{true}, \mathtt{burglary} = \mathit{true})$ respectively.

Some useful definitions and properties of probability theory can now be listed. The **conditional probability** is defined as $\mathbf{P}(\mathtt{X}|\mathtt{Y}) = \mathbf{P}(\mathtt{X}, \mathtt{Y})/\mathbf{P}(\mathtt{Y})$ if $\mathbf{P}(\mathtt{Y}) > 0$. Note that the use of $\mathbf{P}$ in equalities is a short hand notation that denotes that the equality is valid *for all* possible states of the involved random variables. The **chain rule** says that $\mathbf{P}(\mathtt{X}_1, \cdots, \mathtt{X}_n) = \mathbf{P}(\mathtt{X}_1) \prod_{i=2}^{n} \mathbf{P}(\mathtt{X}_i|\mathtt{X}_{i-1}, \cdots, \mathtt{X}_1)$. The **law of Bayes** states that $\mathbf{P}(\mathtt{X}|\mathtt{Y}) = \mathbf{P}(\mathtt{Y}|\mathtt{X}) \cdot \mathbf{P}(\mathtt{X})/\mathbf{P}(\mathtt{Y})$. The distribution of $\mathtt{X}$ is related to the joint distribution $\mathbf{P}(\mathtt{X}, \mathtt{Y})$ by $\mathbf{P}(\mathtt{X}) = \sum_{y \in \mathrm{domain}(\mathtt{Y})} \mathbf{P}(\mathtt{X}, y)$, which is called **marginalization**. Finally, two random variables $\mathtt{X}$ and $\mathtt{Y}$ are **conditionally independent** given a third random variable $\mathtt{Z}$ if and only if $\mathbf{P}(\mathtt{X}, \mathtt{Y}|\mathtt{Z}) = \mathbf{P}(\mathtt{X}|\mathtt{Z}) \cdot \mathbf{P}(\mathtt{Y}|\mathtt{Z})$. In case that the property holds without needing to condition on variables $\mathtt{Z}$, we say that $\mathtt{X}$ and $\mathtt{Y}$ are **independent**.

When working with probabilistic ILP representations, there are essentially two changes:

1. clauses are annotated with probabilistic information such as conditional probabilities, and
2. the covers relation becomes probabilistic.

A probabilistic covers relation softens the hard covers relation employed in traditional ILP and is defined as the probability of an example given the hypothesis and the background theory.

Definition 6 (Probabilistic Covers Relation). *A probabilistic covers relation takes as arguments an example e, a hypothesis H and possibly the background theory B, and returns the probability value $\mathbf{P}(e \mid H, B)$ between 0 and 1 of the example e given H and B, i.e.,* $\mathrm{covers}(e, H, B) = \mathbf{P}(e \mid H, B)$.

[4] The reader not familiar with the basics of probability theory is encouraged to consult [52] for an excellent overview from an artificial intelligence perspective.

Using the probabilistic covers relation of Definition 6, our first attempt at a definition of the probabilistic ILP learning problem is as follows.

Preliminary Definition 1 (Probabilistic ILP Learning Problem)
Given *a probabilistic-logical language $\mathcal{L}_H$ and a set E of examples over some language $\mathcal{L}_E$,* **find** *the hypothesis H^* in $\mathcal{L}_H$ that maximizes* $\mathbf{P}(E \mid H^*, B)$.

Under the usual **i.i.d.** assumption, i.e., examples are sampled independently from identical distributions, this results in the maximization of

$$\mathbf{P}(E \mid H^*, B) = \prod_{e \in E} \mathbf{P}(e \mid H^*, B) = \prod_{e \in E} \text{covers}(e, H^*, B) \ .$$

Similar to the ILP learning problem, the language $\mathcal{L}_E$ selected for representing the examples together with the probabilistic covers relation determines different learning setting. In ILP, this lead to learning from interpretations, from proofs, and from entailment. It should therefore be no surprise that this very same distinction also applies to probabilistic knowledge representation formalisms. Indeed, Bayesian networks [46] essentially define a probability distribution over *interpretations* or *possible worlds*, and stochastic grammars [34] define a distribution over *proofs*, derivations or traces.

Guided by Definition 1, we will now introduce three probabilistic ILP settings, which extend the purely logical ones sketched before. Afterwards, we will refine Definition 1 in Definition 7.

4.1 Probabilistic Learning from Interpretations

In order to integrate probabilities in the learning from interpretations setting, we need to find a way to assign probabilities to interpretations covered by an annotated logic program. In the past decade, this issue has received a lot of attention and various different approaches have been developed. The most popular propositional frameworks are Bayesian network and Markov networks. Later on, these propositional frameworks have been extended to the relational case such probabilistic-logic programs [45], probabilistic relational models [47], relational Bayesian networks [21], and Bayesian logic programs [24, 25].

In this book, the two most popular propositional formalisms, namely Bayesian networks and Markov networks, are considered, as well as their relational versions. The present chapter focuses on Bayesian networks, and their extension towards Bayesian logic programs [27, more details in Chapter 7], whereas Chapter 6 by Santos Costas *et al.* discusses an integration of Bayesian networks and logic programs called CLP($\mathcal{BN}$) and Chapter 9 by Domingos *et al.* focuses on Markov networks and their extension to Markov Logic [12].

Bayesian Networks The most popular formalism for defining probabilities on possible worlds is that of Bayesian networks. As an example of a Bayesian network, consider Judea Pearl's famous alarm network graphically illustrated in Figure 4. Formally speaking, a **Bayesian network** is an augmented, directed acyclic graph, where each node corresponds to a random variable $\mathtt{X}_i$ and each edge indicates a *direct influence* among

$\mathbf{P}(\texttt{burglary})$
$(0.001, 0.999)$

$\mathbf{P}(\texttt{earthquake})$
$(0.002, 0.998)$

burglary	earthquake	$\mathbf{P}(\texttt{alarm})$
true	*true*	$(0.95, 0.05)$
true	*false*	$(0.94, 0.06)$
false	*true*	$(0.29, 0.71)$
false	*false*	$(0.001, 0.999)$

alarm	$\mathbf{P}(\texttt{johncalls})$
true	$(0.90, 0.10)$
false	$(0.05, 0.95)$

alarm	$\mathbf{P}(\texttt{marycalls})$
true	$(0.70, 0.30)$
false	$(0.01, 0.99)$

Table 1. The conditional probability distributions associated with the nodes in the alarm network, cf. Figure 4.

the random variables. It represents the joint probability distribution $\mathbf{P}(\mathtt{X}_1, \ldots, \mathtt{X}_n)$. The influence is quantified with a conditional probability distribution $cpd(\mathtt{X}_i)$ associated to each node X_i. It is defined in terms of the parents of the node X, which we denote by $\mathbf{Pa}(\mathtt{X}_i)$, and specifies $\mathrm{cpd}(\mathtt{X}_i) = \mathbf{P}(\mathtt{X}_i \mid \mathbf{Pa}(\mathtt{X}_i))$.

Example 7. Consider the Bayesian network in Figure 4. It contains the random variables `alarm`, `earthquake`, `marycalls`, `johncalls` and `alarm`. The CPDs associated to each of the nodes are specified in Table 1. They include the CPDs $\mathbf{P}(\texttt{alarm} \mid \texttt{earthquake}, \texttt{burglary})$, and $\mathbf{P}(\texttt{earthquake})$, etc.

The Bayesian network thus has two components: a qualitative one, i.e. the directed acyclic graph, and a quantitative one, i.e. the conditional probability distributions. Together they specify the joint probability distribution

As we will – for simplicity – assume that the random variables are all boolean, i.e., they can have the domain $\{true, false\}$, this actually amounts to specifying a probability distribution on the set of all possible interpretations. Indeed, in our alarm example, the Bayesian network defines a probability distribution over truth-assignments to $\{\mathsf{alarm}, \mathsf{earthquake}, \mathsf{marycalls}, \mathsf{johncalls}, \mathsf{burglary}\}$.

The qualitative component specifies a set of conditional independence assumptions. More formally, it stipulates the following conditional independency assumption:

Assumption 1 *Each node X_i in the graph is conditionally independent of any subset $\mathbf{A}$ of nodes that are not descendants of X_i given a joint state of $\mathbf{Pa}(X_i)$, i.e. $\mathbf{P}(X_i \mid \mathbf{A}, \mathbf{Pa}(X_i)) = \mathbf{P}(X_i \mid \mathbf{Pa}(X_i))$.*

For example, alarm is conditionally independent of $\mathsf{marycalls})$ given a joint state of its parents $\{\mathsf{earthquake}, \mathsf{burglary}\}$. Because of the conditional independence assumption, we can write down the joint probability density as

$$\mathbf{P}(X_1, \ldots, X_n) = \prod_{i=1}^{n} \mathbf{P}(X_i \mid \mathbf{Pa}(X_i)) \tag{1}$$

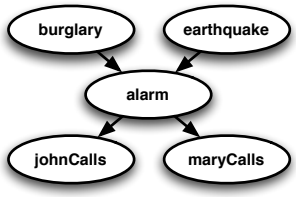

Fig. 4. The Bayesian alarm network. Nodes denote random variables and edges denote direct influences among the random variables.

by applying the independency assumption and the chain rule to the joint probability distribution.

Bayesian Logic Programs The idea underlying Bayesian logic programs is to view ground atoms as random variables that are defined by the underlying definite clause programs. Furthermore, two types of predicates are distinguished: deterministic and probabilistic ones. The former are called *logical*, the latter *Bayesian*. Likewise we will also speak of Bayesian and logical atoms. A Bayesian logic program now consists of a set of of Bayesian (definite) clauses, which are expressions of the form $\mathsf{A} \mid \mathsf{A}_1, \ldots, \mathsf{A}_n$ where A is a Bayesian atom, $\mathsf{A}_1, \ldots, \mathsf{A}_n$, $n \geq 0$, are Bayesian and logical atoms and all variables are (implicitly) universally quantified. To quantify probabilistic dependencies, each Bayesian clause c is annotated with its conditional probability distribution $\mathrm{cpd}(c) = \mathbf{P}(\mathsf{A} \mid \mathsf{A}_1, \ldots, \mathsf{A}_n)$, which quantifies as a macro the probabilistic dependency among ground instances of the clause.

Let us illustrate Bayesian logic programs on an example inspired on Jensen's stud farm example [22], which describes the processes underlying a life threatening hereditary disease.

Example 8. Consider the following Bayesian clauses:

$$\mathsf{carrier}(\mathsf{X}) \mid \mathsf{founder}(\mathsf{X}). \tag{2}$$

$$\mathsf{carrier}(\mathsf{X}) \mid \mathsf{mother}(\mathsf{M},\mathsf{X}),\ \mathsf{carrier}(\mathsf{M}),\ \mathsf{father}(\mathsf{F},\mathsf{X}),\ \mathsf{carrier}(\mathsf{F}). \tag{3}$$

$$\mathsf{suffers}(\mathsf{X}) \mid \mathsf{carrier}(\mathsf{X}). \tag{4}$$

They specify the probabilistic dependencies governing the inheritance process. For instance, clause (3) says that the probability for a horse being a carrier of the disease depends on its parents being carriers. In this example, the mother, father, and founder are logical, whereas the other ones, such as carrier and suffers, are Bayesian. The logical predicates are then defined by a classical definite clause program which constitute the background theory for this example. It is listed as interpretation B in Example 5. Furthermore, the conditional probability distributions for the Bayesian clauses are

$P(\mathsf{carrier}(\mathsf{X}) = true)$
0.6

$\mathsf{carrier}(\mathsf{X})$	$P(\mathsf{suffers}(\mathsf{X}) = true)$
$true$	0.7
$false$	0.01

carrier(M)	carrier(F)	$P(\mathsf{carrier(X)} = true)$
$true$	$true$	0.6
$true$	$false$	0.5
$false$	$true$	0.5
$false$	$false$	0.0

Observe that logical atoms, such as $\mathsf{mother(M, X)}$, do not affect the distribution of Bayesian atoms, such as $\mathsf{carrier(X)}$, and are therefore not considered in the conditional probability distribution. They only provide variable bindings, e.g., between $\mathsf{carrier(X)}$ and $\mathsf{carrier(M)}$.

By now, we are able to define the covers relation for Bayesian logic programs. A Bayesian logic program together with the background theory induces a Bayesian network. The random variables A of the Bayesian network are the Bayesian ground atoms in the least Herbrand model I of the annotated logic program. A Bayesian ground atom, say $\mathsf{carrier(alan)}$, influences another Bayesian ground atom, say $\mathsf{carrier(betsy)}$, if and only if there exists a Bayesian clause c such that

1. $\mathsf{carrier(alan)} \in body(c)\theta \subseteq I$, and
2. $\mathsf{carrier(betsy)} \equiv head(c)\theta \in I$.

Each node A has $\mathrm{cpd}(c\theta)$ as associated conditional probability distribution. If there are multiple ground instances in I with the same head, a *combining rule* $\mathrm{combine}\{\cdot\}$ is used to quantified the combined effect. A combining rule is a function that maps finite sets of conditional probability distributions onto one (*combined*) conditional probability distribution. Examples of combining rules are *noisy-or*, and *noisy-and*, see e.g. [22].

Note that we assume that the induced network is acyclic and has a finite branching factor. The probability distribution induced is now

$$\mathbf{P}(I|H) = \prod_{\text{Bayesian atom } \mathsf{A} \in I} \mathrm{combine}\{\mathrm{cpd}(c\theta) \mid body(c)\theta \subseteq I, head(c)\theta \equiv \mathsf{A}\}. \quad (5)$$

Let us illustrate this fro the stud farm example.

Example 9. Using the above definition, the probability of the interpretation

$$\begin{array}{lll} \{\mathsf{carrier(henry)} = false, & \mathsf{suffers(henry)} = false, & \mathsf{carrier(ann)} = true, \\ \mathsf{suffers(ann)} = false, & \mathsf{carrier(brian)} = false, & \mathsf{suffers(brian)} = false, \\ \mathsf{carrier(alan)} = false, & \mathsf{suffers(alan)} = false, & \mathsf{carrier(alice)} = false, \\ \mathsf{suffers(alice)} = false, & \ldots\} & \end{array}$$

can be computed using a standard Bayesian network inference engine because the facts together with the program induce the Bayesian network shown in Figure 5. Thus (5) defines a probabilistic coverage relation. In addition, various types of inference would be possible. One might, for instance, ask for the probability $P(\mathsf{suffers(henry)}|\mathsf{carrier(henry)} = true)$, which can again be computed using a standard Bayesian network inference engine.

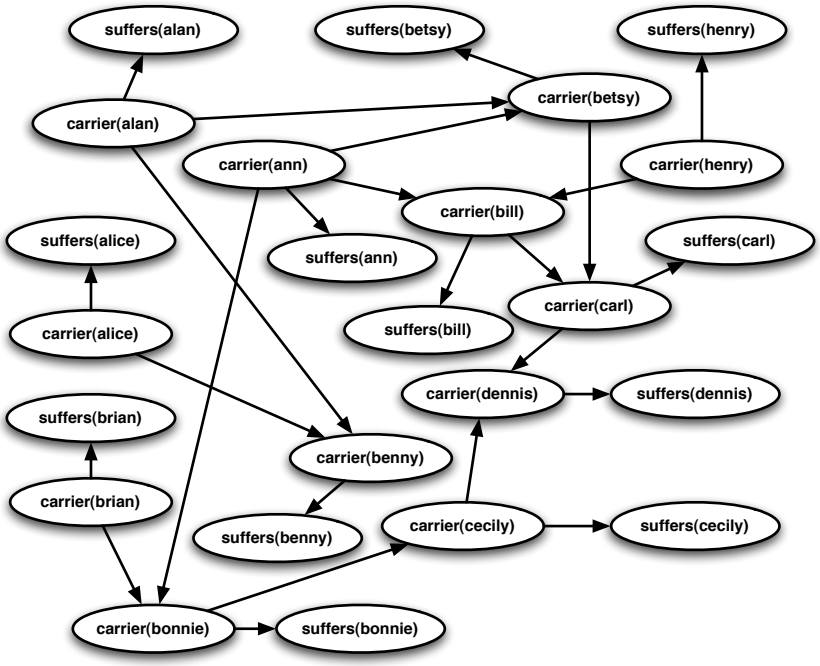

Fig. 5. The structure of the Bayesian network induced by the Stud farm Bayesian logic program. For the ease of comprehensibility, we have omitted the *logical* Bayesian atoms over `founder`/1, `father`/2, and `mother`/2.

4.2 Probabilistic Proofs

To define probabilities on proofs, ICL [49, Chapter 8], PRISMs [53, 54, Chapter 5], and stochastic logic programs [14, 38, 5, an application can be found in Chapter 9] attach probabilities to facts (respectively clauses) and treat them as stochastic choices within resolution. Relational Markov models [2] and logical hidden Markov models [28], which we will briefly review in Chapter 2, can be viewed as a simple fragment of them, where heads and bodies of clauses are singletons only, so-called iterative clauses. We will illustrate probabilistic learning from proofs using stochastic logic programs. For a discussion of the close relationship among stochastic logic programs, ICL, and PRISM, we refer to [6].

Stochastic logic programs are inspired on stochastic context free grammars [1, 34]. The analogy between context free grammars and logic programs is that

- grammar rules correspond to definite clauses,
- sentences (or strings) to atoms, and
- productions to derivations.

Furthermore, in stochastic context-free grammars, the rules are annotated with probability labels in such a way that the sum of the probabilities associated to the rules defining a non-terminal is 1.0 .

Eisele and Muggleton have exploited this analogy to define stochastic logic programs. These are essentially definite clause programs, where each clause c has an associated probability label p_c such that the sum of the probabilities associated to the rules defining any predicate is 1.0 (though [4] considered less restricted versions as well).

This framework allows ones to assign probabilities to proofs for a given predicate q given a stochastic logic program $H \cup B$ in the following manner. Let $D_{\mathtt{q}}$ denote the set of all possible ground proofs for atoms over the predicate $\mathtt{q}$. For simplicity reasons, it will be assumed that there is a finite number of such proofs and that all proofs are finite (but again see [4] for the more general case). Now associate to each proof $t_q \in D_q$ the probability

$$v_t = \prod_c p_c^{n_{c,t}}$$

where the product ranges over all clauses c and $n_{c,t}$ denotes the number of times clause c has been used in the proof t_q. For stochastic context free grammars, the values v_t correspond to the probabilities of the production. However, the difference between context free grammars and logic programs is that in grammars two rules of the form $\mathtt{n} \rightarrow \mathtt{q}, \mathtt{n}_1, ..., \mathtt{n}_m$ and $\mathtt{q} \rightarrow \mathtt{q}_1, ..., \mathtt{q}_k$ always 'resolve' to give $\mathtt{n} \rightarrow \mathtt{q}_1, ..., \mathtt{q}_k, \mathtt{n}_1, ..., \mathtt{n}_m$ whereas resolution may fail due to unification. Therefore, the probability of a proof tree t in D_q, i.e., a successful derivation is

$$P(t \mid H, B) = \frac{v_t}{\sum_{s \in D_q} v_s} \ . \tag{6}$$

The probability of a ground atom $\mathtt{a}$ is then defined as the sum of all the probabilities of all the proofs for that ground atom.

$$P(\mathtt{a} \mid H, B) = \sum_{\substack{s \in D_q \\ s \text{ is a proof for } a}} v_s \ . \tag{7}$$

Example 10. Consider a stochastic variant of the definite clause grammar in Example 6 with uniform probability values for each predicate. The value v_u of the proof (tree) u in Example 6 is $v_u = \frac{1}{3} \cdot \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{12}$. The only other ground proofs s_1, s_2 of atoms over the predicate `sentence` are those of

$$\mathtt{sentence}([\mathtt{a}, \mathtt{turtle}, \mathtt{sleeps}], [])$$
$$\text{and } \mathtt{sentence}([\mathtt{the}, \mathtt{turtle}, \mathtt{sleeps}], []) \ .$$

Both get the value $v_{s_1} = v_{s_2} = \frac{1}{12}$. Because there is only one proof for each of the sentences,

$$P(\mathtt{sentence}([\mathtt{the}, \mathtt{turtles}, \mathtt{sleep}], [])) = v_u = \frac{1}{3} \ .$$

For stochastic logic programs, there are at least two natural learning settings.

Motivated by Equation (6), we can learn them from proofs. This makes structure learning for stochastic logic programs relatively easy, because proofs carry a lot information about the structure of the underlying stochastic logic program. Furthermore, the

learning setting can be considered as an extension of the work on learning stochastic grammars from proof-banks. It should therefore also be applicable to learning unification based grammars. We will present a probabilistic ILP approach within the learning from proofs setting in Section 5.4.

On the other hand, we can use Equation (7) as covers relation and, hence, employ the learning from entailment setting. Here, the examples are ground atoms entailed by the target stochastic logic program. Learning stochastic logic programs from atoms only is much harder than learning them from proofs because atoms carry much less information than proofs. Nevertheless, this setting has been studied by [5] and by [54], who solves the parameter estimation problem for stochastic logic programs respectively PRISM programs, and by [39, 41], who presents an approach to structure learning of stochastic logic programs: adding one clause at a time to an existing stochastic logic program. In the following section, we will introduce the probabilistic learning from entailment. Instead of considering stochastic logic programs, however, we will study a Naïve Bayes framework, which has a much lower computational complexity.

4.3 Probabilistic Learning from Entailment

In order to integrate probabilities in the entailment setting, we need to find a way to assign probabilities to clauses that are entailed by an annotated logic program. Since most ILP systems working under entailment employ ground facts for a single predicate as examples, and the authors are unaware of any existing probabilistic ILP formalisms that implement a probabilistic covers relation for definite clauses as examples in general, we will restrict our attention to assign probabilities to facts for a single predicate. It remains an open question as how to formulate more general frameworks for working with entailment.

More formally, let us annotate a logic program H consisting of a set of clauses of the form $p \leftarrow b_i$, where p is an atom of the form $p(V_1, ..., V_n)$ with the V_i different variables, and the b_i are different bodies of clauses. Furthermore, we associate to each clause in H the probability values $\mathbf{P}(b_i \mid p)$; they constitute the conditional probability distribution that for a random substitution θ for which $p\theta$ is ground and true (resp. false), the query $b_i\theta$ succeeds (resp. fails) in the knowledge base B. [5] Furthermore, we assume the prior probability of p is given as $\mathbf{P}(p)$, it denotes the probability that for a random substitution θ, $p\theta$ is true (resp. false). This can then be used to define the covers relation $\mathbf{P}(p\theta \mid H, B)$ as follows (we delete the B as it is fixed):

$$\mathbf{P}(p\theta \mid H) = \mathbf{P}(p\theta \mid b_1\theta, ..., b_k\theta) = \frac{\mathbf{P}(b_1\theta, ..., b_k\theta \mid p\theta) \times \mathbf{P}(p\theta)}{\mathbf{P}(b_1\theta, ..., b_k\theta)} \tag{8}$$

For instance, applying the naïve Bayes assumption yields

$$\mathbf{P}(p\theta \mid H) = \frac{\prod_i \mathbf{P}(b_i\theta \mid p\theta) \times \mathbf{P}(p\theta)}{\mathbf{P}(b_1\theta, ..., b_k\theta)} \tag{9}$$

Finally, since $P(p\theta \mid H) + P(\neg p\theta \mid H) = 1$, we can compute $P(p\theta \mid H)$ without $P(b_1\theta, ..., b_k\theta)$ through normalization.

[5] The query q succeeds in B if there is a substitution σ such that $B \models q\sigma$.

Example 11. Consider again the mutagenicity domain and the following annotated logic program:

$$(0.01, 0.21) : \texttt{mutagenetic(M)} \leftarrow \texttt{atom(M, \_, \_, 8, \_)}$$
$$(0.38, 0.99) : \texttt{mutagenetic(M)} \leftarrow \texttt{bond(M, \_, A, 1), atom(M, A, c, 22, \_), bond(M, A, \_, 2)}$$

We denote the first clause by $\texttt{b}_1$ and the second one by $\texttt{b}_2$. The vectors on the left-hand side of the clauses specify $P(b_i\theta = true \mid p\theta = true)$ and $P(b_i\theta = true \mid p\theta = false)$ respectively. The covers relation (assuming the Naïve Bayes assumption) assigns probability 0.97 to example 225 because both features fail for $\theta = \{\texttt{M} \leftarrow \texttt{225}\}$. Hence,

$$\begin{aligned} P(\ \texttt{mutagenetic(225)} = true, & \texttt{b}_1\theta = false,\ \texttt{b}_2\theta = false) \\ &= P(\ \texttt{b}_1\theta = false \mid \texttt{mutagenetic(225)} = true\ ) \\ &\quad \cdot P(\ \texttt{b}_2\theta = false \mid \texttt{mutagenetic(225)} = true\ ) \\ &\quad \cdot P(\ \texttt{mutagenetic(225)} = true\ ) \\ &= 0.99 \cdot 0.62 \cdot 0.31 \approx 0.19 \end{aligned}$$

and $P(\ \texttt{mutagenetic(225)} = false, \texttt{b}_1\theta = false,\ \texttt{b}_2\theta = false) = 0.79 \cdot 0.01 \cdot 0.68 \approx 0.005$. This yields

$$P(\ \texttt{muta(225)} = true \mid \texttt{b}_1\theta = false,\ \texttt{b}_2\theta = false\}) = \frac{0.19}{0.19 + 0.005} \approx 0.97\ .$$

5 Probabilistic ILP: A Definition and Example Algorithms

Guided by Definition 1, we have introduced several probabilistic ILP settings for statistical relational learning. The main idea was to lift traditional ILP settings by associating probabilistic information with clauses and interpretations and by replacing ILP's deterministic covers relation by a probabilistic one. In the discussion, we made one trivial but important observation:

Observation 1 *Derivations might fail.*

The probability of a failure is zero and, consequently, failures are never observable. Only succeeding derivations are observable, i.e., the probabilities of such derivations are greater zero. As an extreme case, recall the negative examples *Neg* employed in the ILP learning problem definition 1. They are supposed to be not covered, i.e., $P(\mathit{Neg}|H, B) = 0$.

Example 12. Reconsider Example 3. `Rex` is a male person; he cannot be the `daughter` of `ann`. Thus, $\texttt{daughter(rex, ann)}$ was listed as a negative example.

Negative examples conflict with the usual view on learning examples in statistical learning. In statistical learning, we seek to find that hypothesis H^*, which is most likely given the learning examples:

$$H^* = \arg\max_H P(H|E) = \arg\max_H \frac{P(E|H) \cdot P(F)}{P(E)} \quad \text{with} \quad P(E) > 0\ .$$

Thus, examples E are observable, i.e., $P(E) > 0$. Therefore, we refine the preliminary probabilistic ILP learning problem definition 1. In contrast to the purely logical case of ILP, we do not speak of *positive* and *negative* examples anymore but of *observed* and *unobserved* ones.

Definition 7 (Probabilistic ILP Problem). Given *a set $E = E_p \cup E_i$ of observed and unobserved examples E_p and E_i (with $E_p \cap E_i = \emptyset$) over some example language $\mathcal{L}_E$, a probabilistic covers relation* $\text{covers}(e, H, B) = P(e \mid H, B)$*, a probabilistic logical language $\mathcal{L}_H$ for hypotheses, and a background theory B,* **find** *a hypothesis H^* in $\mathcal{L}_H$ such that $H^* = \arg\max_H \text{score}(E, H, B)$ and the following constraints hold: $\forall\ e_p \in E_p\ :\ \text{covers}(e_p, H^*, B) > 0$ and $\forall\ e_i \in E_i\ :\ \text{covers}(e_i, H^*, B) = 0$. The score is some objective function,usually involving the probabilistic covers relation of the observed examples such as the observed likelihood $\prod_{e_p \in E_p} \text{covers}(e_p, H^*, B)$ or some penalized variant thereof.*

The probabilistic ILP learning problem of Definition 7 unifies ILP and statistical learning in the following sense: using a deterministic covers relation (,which is either 1 or 0) yields the classical ILP learning problem, see Definition 1, whereas sticking to propositional logic and learning from *observed* examples, i.e., $P(E) > 0$, only yields traditional statistical learning.

To come up with algorithms solving probabilistic ILP learning problems, say for density estimation, one typically distinguishes two subtasks because $H = (L, \lambda)$ is essentially a logic program L annotated with probabilistic parameters λ:

1. *Parameter estimation* where it is assumed that the underlying logic program L is fixed, and the learning task consists of estimating the parameters λ that maximize the likelihood.
2. *Structure learning* where both L and λ have to be learned from the data.

Below, we will sketch basic parameter estimation and structure learning techniques, and illustrate them for each setting. In the remainder of the thesis, we will then discuss selected probabilistic ILP approaches for learning from interpretations and probabilistic learning from traces in detail. A more complete survey of learning probabilistic logic representations can be found in [9] and in the related work sections of this thesis.

5.1 Parameter Estimation

The problem of parameter estimation is thus concerned with estimating the values of the parameters λ of a fixed probabilistic program $H = (L, \lambda)$ that best explains the examples E. So, λ is a set of parameters and can be represented as a vector. As already indicated above, to measure the extent to which a model fits the data, one usually employs the likelihood of the data, i.e. $P(E \mid L, \lambda)$, though other scores or variants could be used as well.

When all examples are fully observable, maximum likelihood reduces to frequency counting. In the presence of missing data, however, the maximum likelihood estimate typically cannot be written in closed form. It is a numerical optimization problem, and all known algorithms involve nonlinear optimization The most commonly adapted

technique for probabilistic logic learning is the Expectation-Maximization (EM) algorithm [11, 36]. EM is based on the observation that learning would be easy (i.e., correspond to frequency counting), if the values of all the random variables would be known. Therefore, it estimates these values, maximizes the likelihood based on the estimates, and then iterates. More specifically, EM assumes that the parameters have been initialized (e.g., at random) and then iteratively performs the following two steps until convergence:

(E-Step) On the basis of the observed data and the present parameters of the model, it computes a distribution over all possible completions of each partially observed data case.

(M-Step) Treating each completion as a fully observed data case weighted by its probability, it computes the improved parameter values using (weighted) frequency counting.

The frequencies over the completions are called the *expected counts*. Examples for parameter estimation of probabilistic relational models in general can be found in Chapters 2 and 10 for sequential relational models , in Chapter 4 for Markov logic, in Chapter 5 for PRISM, in Chapter 6 for CLP($\mathcal{BN}$), in Chapter 7 for Bayesian logic programs, and in Chapters 9 and 11 for stochastic logic programs and variants.

5.2 Structure Learning

The problem is now to learn both the structure L and the parameters λ of the probabilistic program $H = (L, \lambda)$ from data. Often, further information is given as well. As in ILP, the additional knowledge can take various different forms, including a *language bias* that imposes restrictions on the syntax of L, and an *initial hypothesis* (L, λ) from which the learning process can start.

Nearly all (score-based) approaches to structure learning perform a heuristic search through the space of possible hypotheses. Typically, hill-climbing or beam-search is applied until the hypothesis satisfies the logical constraints and the $\mathrm{score}(H, E)$ is no longer improving. The steps in the search-space are typically made using refinement operators, which make small, syntactic modification to the (underlying) logic program.

At this points, it is interesting to observe that the logical constraints often require that the observed examples are covered in the logical sense. For instance, when learning stochastic logic programs from entailment, the observed example clauses must be entailed by the logic program, and when learning Markov logic networks, the observed interpretations must be models of the underlying logic program. Thus, for a probabilistic program $H = (L_H, \lambda_H)$ and a background theory $B = (L_B, \lambda_B)$ it holds that $\forall e_p \in E_p : P(e|H, B) > 0$ if and only if $\mathrm{covers}(e, L_H, L_B) = 1$, where L_H (respectively L_B) is the underlying logic program (logical background theory) and $\mathrm{covers}(e, L_H, L_B)$ is the purely logical *covers* relation, which is either 0 or 1.

Let us now sketch for each probabilistic ILP setting one learning approach.

5.3 Learning from Probabilistic Interpretations

The large majority of statistical relational learning techniques proposed so far fall into the learning from interpretations setting including parameter estimation of probabilis-

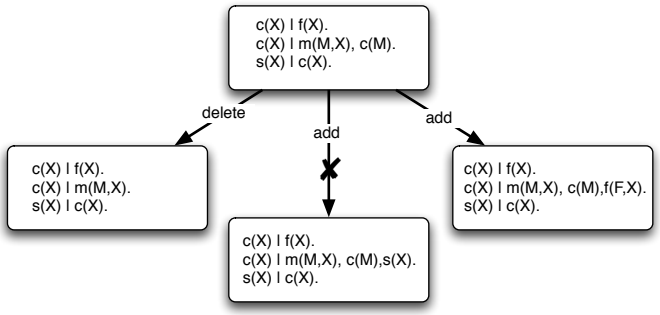

Fig. 6. The use of refinement operators during structural search within the framework of Bayesian logic programs. We can add an atom or delete an atom from the body of a clause. Candidates crossed out are illegal because they are cyclic. Other refinement operators are reasonable such as adding or deleting logically valid clauses.

tic logic programs [30], learning of probabilistic relational models [18], parameter estimation of relational Markov models [60], learning of object-oriented Bayesian networks [3], learning relational dependency networks [44], and learning logic programs with annotated disjunctions [62, 51]. This book provides details on learning sequential relational models in Chapter 2 and 10, on learning Markov logic programs in Chapter 4, and on learning CLP($\mathcal{BN}$) in Chapter 6.

As an example, which will be discussed in detail in Chapter 7, consider learning Bayesian logic programs. SCOOBY [26] is a greedy hill-climbing approach for learning Bayesian logic programs. SCOOBY takes the initial Bayesian logic program $H = (L, \lambda)$ as starting point and computes the parameters maximizing $score(L, \lambda, E)$. Then, refinement operators generalizing respectively specializing H are used to to compute all legal neighbors of H in the hypothesis space, see Figure 6. Each neighbor is scored. Let $H' = (L', \lambda')$ be the legal neighbor scoring best. If $score(L, \lambda, E) < score(L', \lambda', E)$ then SCOOBY takes H' as new hypothesis. The process is continued until no improvements in score are obtained.

SCOOBY is akin to theory revision approaches in inductive logic programming, which also form the basis for learning biochemical reaction models in Chapter 11. In case that only propositional clauses are considered, SCOOBY coincides with greedy hill-climbing approaches for learning Bayesian networks [19].

5.4 Learning from Probabilistic Proofs

Given a training set E containing ground proofs as examples, one possible approach to learning from observed proofs only combines ideas from the early ILP system GOLEM [42] that employs Plotkin's [48] least general generalization (LGG) with bottom-up generalization of grammars and hidden Markov models [59]. The resulting algorithm employs the likelihood of the proofs $score(L, \lambda, E)$ as the scoring function.

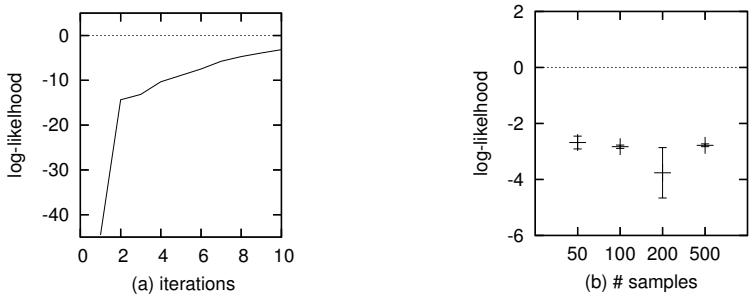

Fig. 7. Experimental results on learning stochastic logic programs from proofs. **(a)** A typical learning curve. **(b)** Final log-likelihood averaged over 4 runs. The error bars show the standard deviations.

It starts by taking as L_0 the set of ground clauses that have been used in the proofs in the training set and scores it to obtain λ_0. After initialization, the algorithm will then repeatedly select a pair of clauses in L_i, and replace the pair by their LGG to yield a candidate L'. The candidate that scores best is then taken as $H_{i+1} = (L_{i+1}, \lambda_{i+1})$, and the process iterates until the score no longer improves. One interesting issue is that strong logical constraints can be imposed on the LGG. These logical constraints directly follow from the fact that the example proofs should still be valid proofs for the logical component L of all hypotheses considered. Therefore, it makes sense to apply the LGG only to clauses that define the same predicate, that contain the same predicates, and whose (reduced) LGG also has the same length as the original clauses.

Preliminary results with a prototype implementation are promising. In one experiment, we generated from the target stochastic logic program

$$
\begin{aligned}
&1 : \mathsf{s}(\mathsf{A},\mathsf{B}) \leftarrow \mathsf{np}(\mathsf{Number},\mathsf{A},\mathsf{C}),\mathsf{vp}(\mathsf{Number},\mathsf{C},\mathsf{B}).\\
&1/2 : \mathsf{np}(\mathsf{Number},\mathsf{A},\mathsf{B}) \leftarrow \mathsf{det}(\mathsf{A},\mathsf{C}),\mathsf{n}(\mathsf{Number},\mathsf{C},\mathsf{B}).\\
&1/2 : \mathsf{np}(\mathsf{Number},\mathsf{A},\mathsf{B}) \leftarrow \mathsf{pronom}(\mathsf{Number},\mathsf{A},\mathsf{B}).\\
&1/2 : \mathsf{vp}(\mathsf{Number},\mathsf{A},\mathsf{B}) \leftarrow \mathsf{v}(\mathsf{Number},\mathsf{A},\mathsf{B}).\\
&1/2 : \mathsf{vp}(\mathsf{Number},\mathsf{A},\mathsf{B}) \leftarrow \mathsf{v}(\mathsf{Number},\mathsf{A},\mathsf{C}),\mathsf{np}(\mathsf{D},\mathsf{C},\mathsf{B}).\\
&1 : \mathsf{det}(\mathsf{A},\mathsf{B}) \leftarrow \mathsf{term}(\mathsf{A},\mathsf{the},\mathsf{B}).\\
&1/4 : \mathsf{n}(\mathsf{s},\mathsf{A},\mathsf{B}) \leftarrow \mathsf{term}(\mathsf{A},\mathsf{man},\mathsf{B}).\\
&1/4 : \mathsf{n}(\mathsf{s},\mathsf{A},\mathsf{B}) \leftarrow \mathsf{term}(\mathsf{A},\mathsf{apple},\mathsf{B}).\\
&1/4 : \mathsf{n}(\mathsf{pl},\mathsf{A},\mathsf{B}) \leftarrow \mathsf{term}(\mathsf{A},\mathsf{men},\mathsf{B}).\\
&1/4 : \mathsf{n}(\mathsf{pl},\mathsf{A},\mathsf{B}) \leftarrow \mathsf{term}(\mathsf{A},\mathsf{apples},\mathsf{B}).\\
&1/4 : \mathsf{v}(\mathsf{s},\mathsf{A},\mathsf{B}) \leftarrow \mathsf{term}(\mathsf{A},\mathsf{eats},\mathsf{B}).\\
&1/4 : \mathsf{v}(\mathsf{s},\mathsf{A},\mathsf{B}) \leftarrow \mathsf{term}(\mathsf{A},\mathsf{sings},\mathsf{B}).\\
&1/4 : \mathsf{v}(\mathsf{pl},\mathsf{A},\mathsf{B}) \leftarrow \mathsf{term}(\mathsf{A},\mathsf{eat},\mathsf{B}).\\
&1/4 : \mathsf{v}(\mathsf{pl},\mathsf{A},\mathsf{B}) \leftarrow \mathsf{term}(\mathsf{A},\mathsf{sing},\mathsf{B}).\\
&1 : \mathsf{pronom}(\mathsf{pl},\mathsf{A},\mathsf{B}) \leftarrow \mathsf{term}(\mathsf{A},\mathsf{you},\mathsf{B}).\\
&1 : \mathsf{term}([\mathsf{A}|\mathsf{B}],\mathsf{A},\mathsf{B}) \leftarrow
\end{aligned}
$$

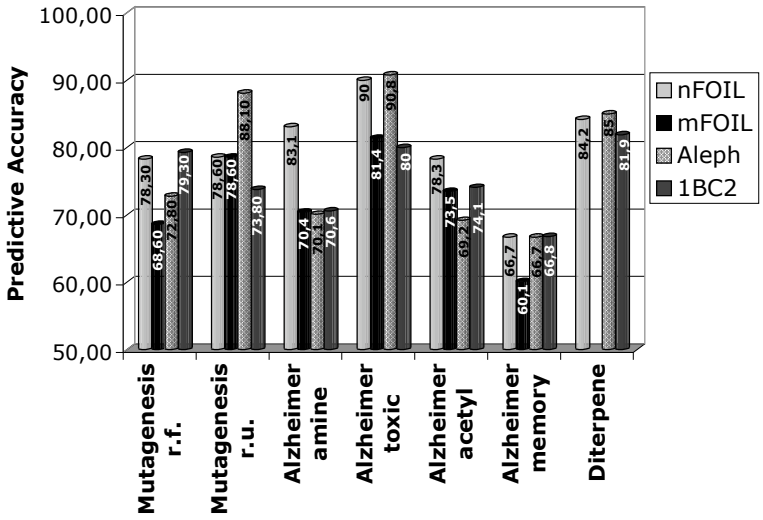

Fig. 8. Cross-validated accuracy results of NFOIL on ILP benchmark data sets. For **Mutagenesis r.u.**, leave-one-out cross-validated accuracies are reported because of the small size of the data set. For all other domains, 10-fold cross-validated results are given. **mFOIL** [32] and **Aleph** [56] are standard ILP algorithms. **1BC2** [16] is a first order logical variant of Naïve Bayes. For **1BC2**, we do not test significance because the results on **Mutagenesis** are taken from [16]. **Diterpene** is a multiclass problem but **mFOIL** has been developed for two-class problems only. Therefore, we do not report results for **mFOIL** on Diterpene.

(independent) training sets of 50, 100, 200, and 500 proofs. For each training set, 4 different random initial sets of parameters were tried. We ran the learning algorithm on each data set starting from each of the initial sets of parameters. The algorithm stopped when a limit of 200 iterations was exceeded or a change in log-likelihood between two successive iterations was smaller than 0.0001.

Figure 7 **(a)** shows a typical learning curve, and Figure 7 **(b)** summarizes the overall results. In all runs, the original structure was induced from the proof-trees. Moreover, already 50 proof-trees suffice to rediscover the structure of the original stochastic logic program. Further experiments with 20 and 10 samples respectively show that even 20 samples suffice to learn the given structure. Sampling 10 proofs, the original structure is rediscovered in one of five experiments. This supports that the *learning from proof trees* setting carries a lot information. Furthermore, our methods scales well. Runs on two independently sampled sets of 1000 training proofs yield similar results: -4.77 and -3.17, and the original structure was learned in both cases. More details can be found in [10].

Other statistical relational learning frameworks that have been developed within the learning from proofs setting are relational Markov models [2] and logical hidden Markov models [28, 29, see also Chapters 2 and 10].

5.5 Probabilistic Learning from Entailment

This setting has been investigated for learning stochastic logic programs [39, 40, 5, 41] and for parameter estimation of PRISM programs [54, 23] from observed examples only, cf. Chapters 5 and 11. Here, we will illustrate a promising, alternative approach with less computational complexity, which adapts FOIL [50] with the conditional likelihood as described in Equation (9) as the scoring function $score(L, \lambda, E)$. This idea has been followed with NFOIL, see [31] for more details.

Given a training set E containing positive and negative examples (i.e. true and false ground facts), this algorithm stays in the learning from observed examples only to induce a probabilistic logical model to distinguish between the positive and negative examples. It computes Horn clause features $b_1, b_2, \ldots$ in an outer loop. It terminates when no further improvements in the score are obtained, i.e, when $score(\{b_1, \ldots, b_i\}, \lambda_i, E) < score(\{b_1, \ldots, b_{i+1}\}, \lambda_{i+1}, E)$, where λ denotes the maximum likelihood parameters. A major difference with FOIL is, however, that the covered positive examples are *not* removed. The inner loop is concerned with inducing the next feature b_{i+1} top-down, i.e., from general to specific. To this aim it starts with a clause with an empty body, e.g., $\texttt{muta(M)} \leftarrow$. This clause is then specialized by repeatedly adding atoms to the body, e.g., $\texttt{muta(M)} \leftarrow \texttt{bond(M,A,1)}$, $\texttt{muta(M)} \leftarrow \texttt{bond(M,A,1)}, \texttt{atom(M,A,c,22,\_)}$, etc. For each refinement b'_{i+1} we then compute the maximum-likelihood parameters λ'_{i+1} and $score(\{b_1, \ldots, b'_{i+1}\}, \lambda'_{i+1}, E)$. The refinement that scores best, say b''_{i+1}, is then considered for further refinement and the refinement process terminates when $score(\{b_1, \ldots, b_{i+1}\}, \lambda_{i+1}, E) < score(\{b_1, \ldots, b''_{i+1}\}, \lambda''_{i+1}, E)$. As Figure 8 shows, NFOIL performs well compared to other ILP systems on traditional ILP benchmark data sets. MFOIL and ALEPH, two standard ILP systems, were never significantly better than NFOIL (paired sampled t-test, $p = 0.05$). NFOIL achieved significantly higher predictive accuracies than MFOIL on Alzheimer amine, toxic, and acetyl. Compared to ALEPH, NFOIL achieved significantly higher accuracies on Alzheimer amine and acetyl (paired sampled t-test, $p = 0.05$). For more details, we refer to [31].

6 Conclusions

This chapter has defined the formal framework of *probabilistic ILP* for statistical relational learning and presented three learning setting settings: *probabilistic learning from entailment, from interpretations*, and *from proofs*. They differ in their representation of examples and the corresponding covers relation. The probabilistic ILP settings and learning approaches are by no means the only possible settings for probabilistic ILP. Nevertheless, two of the settings have – to the best of our knowledge – not been introduced before. Furthermore, we have sketched how the settings combine and generalize ILP and statistical learning. Finally, we have shown how state-of-the-art SRL frameworks fit into these learning settings

At present, it is still an open question as to what the relation among these different settings is. It is, however, apparent that they provide different levels of information about the target probabilistic program, cf. Figure 9. Learning from entailment provides the least information, whereas learning from proofs or traces the most. Learning from

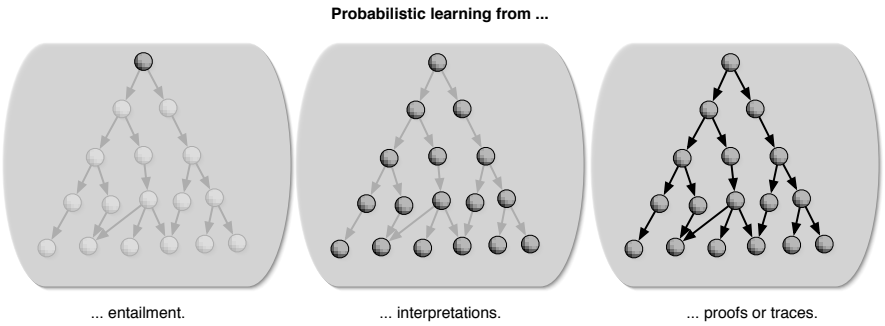

Fig. 9. The level of information on the target probabilistic program provided by probabilistic ILP settings: shaded parts denote unobserved information. Learning from entailment provides the least information. Only roots of proof tree are observed. In contrast, learning from proofs or traces provides the most information. All ground clauses and atoms used in proofs are observed. Learning from interpretations provides an intermediate level of information. All ground atoms but not the clauses are observed.

interpretations occupies an intermediate position. This is interesting because learning is expected to be even more difficult as the less information is observed. Furthermore, the presented learning settings are by no means the only possible settings. Examples might be weighted and proofs might be partially observed.

Acknowledgements

This work was supported by the European Union, contract number FP6-508861, Applications of Probabilistic Inductive Logic Programming II.

References

1. S. P. Abney. Stochastic Attribute-Value Grammars. *Computational Linguistics*, 23(4):597–618, 1997.
2. C. R. Anderson, P. Domingos, and D. S. Weld. Relational Markov Models and their Application to Adaptive Web Navigation. In D. Hand, D. Keim, and R. Ng, editors, *Proceedings of the Eighth International Conference on Knowledge Discovery and Data Mining (KDD-02)*, pages 143–152, Edmonton, Canada, July 2002. ACM Press.
3. O. Bangsø, H. Langseth, and T. D. Nielsen. Structural learning in object oriented domains. In I. Russell and J. Kolen, editors, *Proceedings of the Fourteenth International Florida Artificial Intelligence Research Society Conference (FLAIRS-01)*, pages 340–344, Key West, Florida, USA, 2001. AAAI Press.
4. J. Cussens. Loglinear models for first-order probabilistic reasoning. In K. Blackmond Laskey and H. Prade, editors, *Proceedings of the Fifteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI-99)*, pages 126–133, Stockholm, Sweden, 1999. Morgan Kaufmann.
5. J. Cussens. Parameter estimation in stochastic logic programs. *Machine Learning Journal*, 44(3):245–271, 2001.

6. J. Cussens. Integrating by separating: Combining probability and logic with ICL, PRISM and SLPs. Technical report, APrIL Projetc, January 2005.
7. L. De Raedt. Logical settings for concept-learning. *Artificial Intelligence Journal*, 95(1):197–201, 1997.
8. L. De Raedt and S. Džeroski. First-Order jk-Clausal Theories are PAC-Learnable. *Artificial Intelligence Journal*, 70(1-2):375–392, 1994.
9. L. De Raedt and K. Kersting. Probabilistic Logic Learning. *ACM-SIGKDD Explorations: Special issue on Multi-Relational Data Mining*, 5(1):31–48, 2003.
10. L. De Raedt, K. Kersting, and S. Torge. Towards learning stochastic logic programs from proof-banks. In M. Veloso and S. Kambhampati, editors, *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI-05)*, pages 752–757, Pittsburgh, Pennsylvania, USA, July 9–13 2005. AAAI.
11. A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*, B 39:1–39, 1977.
12. P. Domingos and M. Richardson. Markov Logic: A Unifying Framework for Statistical Relational Learning. In T. G. Dietterich, L. Getoor, and K. Murphy, editors, *Proceedings of the ICML-2004 Workshop on Statistical Relational Learning and its Connections to Other Fields (SRL-04)*, pages 49–54, Banff, Alberta, Canada, July 8 2004.
13. S. Džeroski and N. Lavrač, editors. *Relational data mining*. Springer-Verlag, Berlin, 2001.
14. A. Eisele. Towards Probabilistic Extensions of Contraint-based Grammars. In J. Dörne, editor, *Computational Aspects of Constraint-Based Linguistics Decription-II*. DYNA-2 deliverable R1.2.B, 1994.
15. P. Flach. *Simply Logical: Intelligent Reasoning by Example*. John Wiley, 1994.
16. P. A. Flach and N. Lachiche. Naive Bayesian classification of structured data. *Machine Learning Journal*, 57(3):233–269, 2004.
17. L. Getoor. *Learning Statistical Models from Relational Data*. PhD thesis, Stanford University, USA, June 2001.
18. L. Getoor, N.Friedman, D. Koller, and B Taskar. Learning Probabilistic Models of Link Structure. *Journal of Machine Leaning Research (JMLR)*, 3:679 – 707, 2002.
19. D. Heckerman. A Tutorial on Learning with Bayesian Networks. Technical Report MSR-TR-95-06, Microsoft Research,, 1995.
20. N. Helft. Induction as nonmonotonic inference. In R. J. Brachman and H. J. Levesque, editors, *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning(KR-89)*, pages 149–156, Toronto, Canada, May 15-18 1989. Kaufmann.
21. M. Jäger. Relational Bayesian Networks. In K. B. Laskey and H. Prade, editors, *Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence (UAI-97)*, pages 266–273, Stockholm, Sweden, July 30–August 1 1997. Morgan Kaufmann.
22. F. V. Jensen. *Bayesian networks and decision graphs*. Springer-Verlag New, 2001.
23. Y. Kameya, T. Sato, and N.-G. Zhou. Yet more efficient EM learning for parameterized logic programs by inter-goal sharing. In R. Lopez de Mantaras and L. Saitta, editors, *Proceedings of the 16th European Conference on Artificial Intelligence (ECAI-04)*, pages 490–494, Valencia, Spain, August 22-27 2004. IOS Press.
24. K. Kersting. Bayes'sche-logische Programme. Master's thesis, Institute for Computer Science, University of Freiburg, 2000.
25. K. Kersting and L. De Raedt. Bayesian logic programs. Technical Report 151, Institute for Computer Science, University of Freiburg, Freiburg, Germany, April 2001.
26. K. Kersting and L. De Raedt. Towards Combining Inductive Logic Programming with Bayesian Networks. In Céline Rouveirol and M. Sebag, editors, *Proceedings of the 11th International Conference on Inductive Logic Programming (ILP-01)*, volume 2157 of *LNAI*, pages 118–131, Strasbourg, France, September 2001. Springer.

27. K. Kersting and L. De Raedt. Bayesian Logic Programming: Theory and Tool. In L. Getoor and B. Taskar, editors, *An Introduction to Statistical Relational Learning*, pages 291–321. MIT Press, 2007.
28. K. Kersting, L. De Raedt, and T. Raiko. Logial Hidden Markov Models. *Journal of Artificial Intelligence Research (JAIR)*, 25:425–456, 2006.
29. K. Kersting and T. Raiko. 'Say EM' for Selecting Probabilistic Models for Logical Sequences. In F. Bacchus and T. Jaakkola, editors, *Proceedings of the Twenty-First Conference on Uncertainty in Artificial Intelligence (UAI-05)*, pages 300–307, Edinburgh, Scotland, July 26-29 2005.
30. D. Koller and A. Pfeffer. Learning probabilities for noisy first-order rules. In M. Pollack, editor, *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI-97)*, pages 1316–1321, Nagoya, Japan, 1997. Morgan Kaufmann.
31. N. Landwehr, K. Kersting, and L. De Raedt. nFOIL: Integrating Naïve Bayes and Foil. In M. Veloso and S. Kambhampati, editors, *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI-05)*, pages 795–800, Pittsburgh, Pennsylvania, USA, July 9–13 2005. AAAI.
32. N. Lavrač and S. Džeroski. *Inductive Logic Programming*. Ellis Horwood, 1994.
33. J. W. Lloyd. *Foundations of Logic Programming*. Springer, Berlin, 2. edition, 1989.
34. C. H. Manning and H. Schütze. *Foundations of Statistical Natural Language Processing*. The MIT Press, 1999.
35. M. Marcus, G. Kim, M. A. Marcinkiewicz, R. MacIntyre, A. Bies, M. Ferguson, K. Katz, and B. Schasberger. The Penn treebank: Annotating predicate argument structure. In C. J. Weinstein, editor, *In ARPA Human Language Technology Workshop*, pages 114–119, Plainsboro, NJ, USA, March 8–11 1994.
36. G. McLachlan and T. Krishnan. *The EM Algorithm and Extensions*. Wiley, New York, 1997.
37. S. H. Muggleton. Inverse Entailment and Progol. *New Generation Computing Journal*, pages 245–286, 1995.
38. S. H. Muggleton. Stochastic logic programs. In L. De Raedt, editor, *Advances in Inductive Logic Programming*, pages 254–264. IOS Press, 1996.
39. S. H. Muggleton. Learning Stochastic Logic Programs. *Electronic Transactions in Artificial Intelligence*, 4(041), 2000.
40. S. H. Muggleton. Learning stochastic logic programs. In L. Getoor and D. Jensen, editors, *Working Notes of the AAAI-2000 Workshop on Learning Statistical Models from Relational Data (SRL-00)*, pages 36–41, Austin, Texas, July 31 2000. AAAI Press.
41. S. H. Muggleton. Learning Structure and Parameters of Stochastic Logic Programs. In S. Matwin and C. Sammut, editors, *Proceedgins of the Twelfth International Conference on Inductive Logic Prgramming (ILP-02)*, volume 2583 of *LNCS*, Sydney, Australia, July 9–11 2002. Springer.
42. S. H. Muggleton and C. Feng. Efficient Induction of Logic Programs. In S. H. Muggleton, editor, *Inductive Logic Programming*. Acadamic Press, 1992.
43. S. H Muggleton and L. De Raedt. Inductive Logic Programming: Theory and Methods. *Journal of Logic Programming*, 19(20):629–679, 1994.
44. J. Neville and D. Jensen. Dependency Networks for Relational Data. In R. Rastogi, K. Morik, M. Bramer, and X. Wu, editors, *Proceedings of The Fourth IEEE International Conference on Data Mining (ICDM-04)*, pages 170–177, Brighton, UK, November 1–4 2004. IEEE Computer Society Press.
45. L. Ngo and P. Haddawy. Answering Queries from Context-Sensitive Probabilistic Knowledge Bases. *Theoretical Computer Science*, 171:147–177, 1997.
46. J. Pearl. *Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 2. edition, 1991.

47. A. J. Pfeffer. *Probabilistic Reasoning for Complex Systems*. PhD thesis, Computer Science Department, Stanford University, December 2000.
48. G. D. Plotkin. A note on inductive generalization. In *Machine Intelligence 5*, pages 153–163. Edinburgh University Press, 1970.
49. D. Poole. Probabilistic Horn abduction and Bayesian networks. *Artificial Intelligence Journal*, 64:81–129, 1993.
50. J. R. Quinlan and R. M. Cameron-Jones. Induction of logic programs:FOIL and related systems. *New Generation Computing*, pages 287–312, 1995.
51. F. Riguzzi. Learning logic programs with annotated disjunctions. In A. Srinivasan, R. King, and R. Camacho, editors, *Proceedings of the 14th International Conference on Inductive Logic Programming (ILP-04)*, volume 3194 of *LNCS*, pages 270–287, Porto, Portugal, September 6-8 2004. Springer.
52. S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Inc., 2. edition, 2002.
53. T. Sato. A Statistical Learning Method for Logic Programs with Distribution Semantics. In L. Sterling, editor, *Proceedings of the Twelfth International Conference on Logic Programming (ICLP-95)*, pages 715 – 729, Tokyo, Japan, 1995. MIT Press.
54. T. Sato and Y. Kameya. Parameter Learning of Logic Programs for Symbolic-Statistical Modeling. *Journal of Artificial Intelligence Research (JAIR)*, 15:391–454, 2001.
55. E. Shapiro. *Algorithmic Program Debugging*. MIT Press, 1983.
56. A. Srinivasan. *The Aleph Manual*, 1999. Available at `http://www.comlab.ox.ac.uk/oucl/řesearch/areas/machlearn/Aleph/`.
57. A. Srinivasan, S. H. Muggleton, R. D. King, and M. J. E. Sternberg. Theories for Mutagenicity: A Study of First-Order and Feature -based Induction. *Artificial Intelligence Journal*, 85:277–299, 1996.
58. L. Sterling and E. Shapiro. *The Art of Prolog: Advanced Programming Techniques*. The MIT Press, 1986.
59. A. Stolcke and S. Omohundro. Hidden Markov model induction by Bayesian model merging. In S. J. Hanson, J. D. Cowan, and C. L. Giles, editors, *Advances in Neural Information Processing Systems*, volume 5, pages 11–18. Morgan Kaufman, 1993. (Proceedings of NIPS-92, Denver, Colorado, USA, November 30 - December 3, 1992).
60. B. Taskar, P. Abbeel, and D. Koller. Discriminative Probabilistic Models for Relational Data. In A. Darwiche and N. Friedman, editors, *Proceedings of the Eighteenth Conference on Uncertainty in Artificial Intelligence (UAI-02)*, pages 485–492, Edmonton, Alberta, Canada, August 1-4 2002.
61. L. G. Valiant. A theory of the Learnable. *Communications of the ACM*, 27(11):1134–1142, November 1984.
62. J. Vennekens, S. Verbaeten, and M. Bruynooghe. Logic Programs with Annotated Disjunctions. In B. Demoen and V. Lifschitz, editors, *Proceedings of 20th International Conference on Logic Programming (ICLP-04)*, pages 431–445, Saint-Malo, France, September 6-10 2004.