9 Logic-based Formalisms for Statistical Relational Learning

James Cussens

This chapter provides a selective overview of logic-based approaches to statistical relational learning. Issues of representation, inference, and learning are addressed with an emphasis on representation. A distinction is drawn between "directed" representations with connections to Bayesian nets and "undirected" ones related to Markov nets. Within directed representations a further distinction is made between using conditional probabilities and using logical rules to define probability distributions. Among the formalisms discussed are: the independent choice logic, probabilistic logic programming, and stochastic logic programs. The PRISM system is used to provide concrete examples of probabilistic inference and parameter estimation. The use of "possible worlds" to provide semantics is described and its role in connecting differing formalisms is analyzed.

9.1 Introduction

This chapter provides a high-level and selective overview of formalisms which incorporate both logic and probability. Naturally, the focus is on those formalisms which fall within the ambit of statistical relational learning (SRL) or which have influenced formalisms used for SRL. Learning (in the AI sense) is the central topic of this book, but in order to understand existing and potential learning algorithms for the formalisms discussed, it is necessary to understand what is represented by each formalism: we need to know what is to be learned before examining how to do the learning. Consequently, in this chapter there is a strong focus on issues of representation.

It is worth stating some important questions concerning logic and probability which will *not* be addressed here. First, "logic" in the general nontechnical sense of a method of rational reasoning includes probabilistic reasoning quite naturally since humans are required to reason in uncertain situations. Thus two of the historically

most influential logic books include sections on probabilistic reasoning as a matter of course. The books concerned are the *Port-Royal Logic* [2] and *An Investigation of the Laws of Thought, on which are founded the Mathematical Theories of Logic and Probabilities* [4]. Here, however, we are not concerned with general issues in probabilistic reasoning, (dealing with conflicting evidence, probability kinematics, relations with other uncertainty calculi, etc). Instead both probability and logic will be treated as formalisms, i.e., in the narrow technical sense.

Secondly, there is a well-developed *logical interpretation of probability* found in the philosophical literature. The best-known advocates of this interpretation are Keynes [23] and Carnap [5]. The basic claim of this interpretation is that for *any* two propositions a and b there is an objective, logical relation of partial entailment between a and b which is measured by a unique conditional probability $P(b|a)$. See Howson and Urbach [18] for further details. This interpretation uses versions of the principle of indifference to "logically" infer such probability values. This approach is rejected in all the formalisms to be discussed: either the user defines probabilities or the probabilities are estimated from data. Whether these are the "right" probabilities is of no concern to the formalism.

9.1.1 Possible World Semantics

Although logic-based SRL formalisms reject Carnap's attempt to use logic to *determine* probabilities, his use of *possible worlds* to provide semantics for probabilistic statements is widely followed. Recall that to interpret terms and formulae of a (standard, nonprobabilistic) first-order language $\mathcal{L}$ it is necessary to consider $\mathcal{L}$-structures, also known as $\mathcal{L}$-interpretations, or more poetically, possible worlds. An $\mathcal{L}$-structure is a set (the *domain*) together with functions and relations. Each function (resp. predicate) symbol in the language has a corresponding function (resp. relation) on the domain. Standard (Tarskian) first-order semantics defines when a particular $\mathcal{L}$-formula is true in a particular $\mathcal{L}$-structure. For example, the formula $flies(tweety)$ is true in a given $\mathcal{L}$-structure iff the individual which the constant $tweety$ denotes is an element of the set which the predicate symbol $flies$ denotes.

To explain possible world semantics for probabilistic statements we will follow the account of Halpern [17]. Using Halpern's notation, the probability that $flies(tweety)$ is true is denoted by the term $w(flies(tweety))$.[1] The proposition that this probability is 0.8 is represented by the formula $w(flies(tweety)) = 0.8$.

As Halpern notes, it is not useful to ask whether a probabilistic statement such as $w(flies(tweety)) = 0.8$ is true in some particular $\mathcal{L}$-structure. In any given $\mathcal{L}$-structure, *tweety* either flies or does not. Instead we have to ask whether a *probability distribution over $\mathcal{L}$-structures* satisfies $w(flies(tweety)) = 0.8$ or not. A rigorous

1. Note on terminology: Throughout the rest of this chapter the term "the probability of F" (where F is a first-order formula) will be used as an abbreviation for "the probability that F is true."

account of how to answer this question is given by Halpern [17] but the basic idea is simple: a probability distribution μ over $\mathcal{L}$-structures—possible worlds—satisfies $w(flies(tweety)) = 0.8$ iff the set of worlds in which $flies(tweety)$ is true has probability 0.8 according to μ. Note that this means that $w(flies(tweety))$ is a *marginal* probability since it can be computed by summing over possible-world probabilities.

9.2 Representation

Probability-logic formalisms take one of two routes to defining probabilities. In the *directed* approach there is a nonempty set of formulae all of whose probabilities are explicitly stated: call these *probabilistic facts*, similarly to Sato [39]. Other probabilities are defined recursively with the probabilistic facts acting as base cases. A probability-logic model using the directed approach will be closely related to a recursive graphical model (Bayesian net). Most probability-logic formalisms fall into this category: for example, *probabilistic logic programming (PLP)* [30]; *probabilistic Horn abduction (PHA)* [36] and its later expansion the *independent choice logic (ICL)* [37]; *probabilistic knowledge bases (PKBs)* [31]; *Bayesian logic programs (BLPs)* (see chapter 10); *relational Bayesian networks (RBNs)* [20]; *stochastic logic programs (SLPs)* (see chapter 11) and the *PRISM* system [40].

The second, less common, approach is *undirected*, where no formula has its probability explicitly stated. *Relational Markov networks (RMNs)* (see chapter 6 and *Markov logic networks (MLNs)* (see chapter 12) are examples of this approach. In the undirected approach, the probability of each possible world is defined in terms of its "features," where each feature has an associated real-valued parameter. For example, in the case of MLN each feature is associated with a first-order formula: the value of the feature for a given world is simply the number of true ground instances of the formula in that world. Such approaches have much in common with undirected probabilistic models such as Markov networks. For example, to compute (perhaps conditional) probabilities of individual formulae, inference techniques from Markov networks can be used. See chapters 5 and 12, for further details.

9.2.1 Defining Random Variables Using Logic

Here we will focus on formalisms using the more common directed approach. The most basic requirement of such formalisms is to explicitly state that a given ground atomic formula has some probability of being true: a statement such as $w(flies(tweety)) = 0.8$ should be expressible. This is indeed the case for PLP, PHA/ICL, PKB, and PRISM. In all these cases, possible worlds semantics are explicitly invoked.

From a statistical point of view, asserting that $w(flies(tweety)) = 0.8$ amounts to viewing $flies(tweety)$ as a binary variable taking the values TRUE and FALSE. In many applications a restriction to binary variables would be be very inconvenient

so a number of formalisms have machinery to allow a logical representation of random variables with n values for arbitrary finite n. For each such random variable this is done by stating that a set of n atomic formulae are mutually exclusive and exhaustive. Call such sets *alternatives* and call the determination of which formula is true in an alternative an *atomic choice*, following Poole [37]. In each possible world exactly one atomic choice is true. This approach is taken in PHA/ICL, PKB, and PRISM, and a similar one is used in PKB. Indeed, in these formalisms, alternatives are used even in the binary case, thus allowing a uniform logical representation of binary and nonbinary random variables. Rather than stating that $w(flies(tweety)) = 0.8$ we can write $w(flies(tweety, yes)) = 0.8$ and $w(flies(tweety, no)) = 0.2$, and state that $\{flies(tweety, yes), flies(tweety, no)\}$ is an alternative.

The PRISM system restricts the syntactic form of alternatives quite drastically as part of its *distribution condition* [40]. Each alternative is a set of atomic formulae of the form

$$\{msw(x, i, v_1), msw(x, i, v_2), \ldots, msw(x, i, v_m)\};$$

where x and i are the *switch name* and *trial-id* respectively, and msw is short for *mult-ary random switch*.[2] Translating to the language of random variables, $msw(x, i, v_j)$ is true in a world iff in that world the variable X_i takes the value v_j. For different trial-ids i, the random variables X_i are required to be i.i.d.

This restriction seems drastic, but since switch names can be structured logical terms, *in practice* no representational power is lost. Returning to tweety we could have the alternative

$$\{msw(flies(tweety), 1, yes), msw(flies(tweety), 1, no)\};$$

where the trial-id is effectively redundant, and now $flies$ is a function symbol rather than a predicate symbol.

Since the distribution over values does not depend on the trial-id, this second argument is not present in actual PRISM code—it is implicit. Figure 9.1 gives the PRISM code for defining our tweety alternative. This code actually states that $\forall i \in \mathcal{N} : w(msw(flies(tweety), i, yes)) = 0.8 \wedge w(msw(flies(tweety), i, no)) = 0.2$, although for our example only $i = 1$ is needed.

```
values(flies(tweety),[yes,no]).
:- set_sw(flies(tweety),0.8+0.2).
```

Figure 9.1 PRISM code for tweety.

2. I have altered the notation slightly from that found in [40].

9.2.2 Using Logical Variables

Even at the rudimentary level of defining probabilities for atomic formulae some of the power of first-order methods is apparent. The basic point is that by using variables we can define probabilities for whole families of related atomic formulae. To take an example from Ngo and Haddaway [31], the PKB formula

$$P(nbrhd(X, bad)) = 0.3 \leftarrow in\_CALI(X)$$

makes up part of the definition of the distribution of random variables $nbrhd(X)$ for those X where $in\_CALI(X)$ is true. Informally, the formula says: "For those in California, there is probability 0.3 of living in a bad neighborhood." The formula $in\_CALI(X)$ is known as a *context literal*. Asserting that a context literal is true amounts to stating that it is true in all possible worlds, or equivalently, restricting the set of possible worlds under consideration to those where it is true. Thus the preceding formula can be translated into Halpern's syntax as

$$\forall x : w(nbrhd(x, bad)) = 0.3 \leftarrow w(in\_CALI(x)) = 1.$$

If, for example, we had that $w(in\_CALI(bob)) = 1$, meaning "it is certain that Bob lives in California," then it immediately follows that $w(nbrnd(bob, bad)) = 0.3$, meaning "Bob lives in a bad neighborhood with probability 0.3."

Such a mixture of probabilistic literals (i.e., $Pr(nbrnd(X, bad))$) and nonprobabilistic literals (i.e., $in\_CALI(X)$), where the latter states what is true in all worlds, is common. In PLP [30], stating what is true in all worlds is made explicit. The above formula would be written

$$nbrhd(X, bad) : [0.3, 0.3] \leftarrow in\_CALI(X) : [1, 1]$$

and would have the same informal intended meaning. As this example indicates, in PLP probability *intervals* are represented. To show the sort of formulae that can be expressed in PLP and what they mean, tables 9.1, 9.2 and 9.3 show three example PLP formulae, their informal meaning, and their representation in Halpern's notation, respectively.[3]

Table 9.1 PLP formulae

1.	$eastbound(train1) : [0.7, 0.9]$	$\leftarrow$	
2.	$bark(X) : [0.95, 1]$	$\leftarrow$	$dog(X) : [1, 1]$
3.	$not\_dog(X) : [1 - V_2, 1 - V_1]$	$\leftarrow$	$dog(X) : [V_1, V_2]$

3. Some of this material is taken from Cussens [6].

Table 9.2 Informal meanings for the PLP formulae in table 9.1

1. The probability that train1 is eastbound is between 0.7 and 0.9.
2. All dogs have a probability at least 0.95 of barking.
3. If the probability that something is a dog lies between V_1 and V_2, then the probability that it is not a dog lies between $1 - V_2$ and $1 - V_1$.

Table 9.3 The PLP formulae in table 9.1 in Halpern's notation

1.	$w(eastbound(train1)) \in [0.7, 0.9]$		
2.	$\forall x : w(bark(x)) \in [0.95, 1]$	$\leftarrow$	$w(dog(x)) = 1$
3.	$\forall x, v_1, v_2 : not\_dog(x) \in [1 - v_2, 1 - v_1]$	$\leftarrow$	$w(dog(x)) \in [v_1, v_2]$

9.2.3 Logical Implication versus Conditional Probabilities

So far we have been looking mainly at formulae which directly define probabilities for atomic formulae. Directly stating all probabilities of interest is too restrictive and so formalisms provide mechanisms for defining probabilities which must be inferred rather than just "looked up."

For directed approaches, there are two basic ways in which this is done: using conditional probabilities and using logical rules. PKB [31], for example, focuses on the former approach allowing formulae such as

$$P(bglry(X, yes)|nbd(X, bad)) = 0.6 \leftarrow in\_CALI(X), \tag{9.1}$$

which corresponds to this statement in Halpern's notation:

$$\forall x : [w(bglry(x, yes) \wedge nbd(x, bad)) = 0.6 \times w(nbd(x, bad)) \leftarrow w(in\_CALI(x)) = 1],$$

which can be written in disjunctive form:

$$\forall x : [w(bglry(x, yes) \wedge nbd(x, bad)) = 0.6 \times w(nbd(x, bad)) \vee w(in\_CALI(x)) \neq 1]. \tag{9.2}$$

(All these formulae informally mean "For those living in California, the probability of being burgled if they live in a bad neighborhood is 0.6.") This approach implicitly defines a (possibly huge) Bayesian network (the *ground network*) where each node corresponds to a ground atomic formula. The conditional probability tables (CPTs) in the ground network are often defined with the help of combining rules such as noisy-or. The full ground network is never actually constructed. Instead only just enough of it is constructed to answer any given probabilistic query. This is a technique known as *knowledge-based model construction (KBMC)*.

The alternative approach is to use logical rules: statements of what is true in all worlds. But this raises a problem. From

$$w(flies(tweety)) = 0.8 \tag{9.3}$$

and the statement that $\forall x : happy(x) \leftarrow flies(x)$ is true in all worlds:

$$w(\forall x : happy(x) \leftarrow flies(x)) = 1, \tag{9.4}$$

we get not $w(happy(tweety)) = 0.8$ but merely $w(happy(tweety)) \geq 0.8$ (since *tweety* may be happy "for other reasons"). No specific probability is determined for $happy(tweety)$. There are two attitudes to this problem. The first is just to live with having mere bounds on probabilities. This is the approach taken by Boole [4] and Ng and Subrahmanian [30] and, using a propositional approach, Nilsson [32]. The second method is to invoke a version of the closed-world assumption (CWA), so that (9.4) is interpreted to mean

$$w(\forall x : happy(x) \leftrightarrow flies(x)) = 1$$

from which, together with (9.3), $w(happy(tweety)) = 0.8$ *does* follow. The CWA approach is taken in PHA/ICL [36, 37] and PRISM [40]. In both these formalisms there is a strict separation between the probabilistic facts—like $flies(tweety)$—whose probabilities are explicitly given, and formulae like $happy(tweety)$ whose probabilities have to be inferred from the probabilistic facts, the rules, and the CWA. This separation is achieved by syntactic restrictions on the rules so that no probabilistic fact can be inferred using the rules. The rules basically extend the distribution defined over the facts to the other atomic formulae. As Sato puts it:

> When a joint distribution P_F is given to a set F of facts in a logic program $DB = F \cup R$ where R is a set of rules, we can further extend it to a joint distribution P_{DB} over the set of least models of DB [39].

Further details on this point, and on the relation between PHA/ICL, PRISM, and SLPs are given by Cussens [9]. Two further points are worth mentioning. First, it is possible to combine conditional probability and rule-based methods, as shown by (9.1). In such cases it is important to distinguish between formulae in the antecedent of a rule and those in the conditional part of a conditional probability. For example, if in formula (9.1) we move the antecedent literal into the conditional part of the probability we get

$$P(bglry(X, yes)|nbd(X, bad), in\_CALI(X)) = 0.6, \tag{9.5}$$

which in Halpern's notation is

$$\begin{aligned} \forall x : [\quad & w(bglry(x, yes) \wedge nbd(x, bad) \wedge in\_CALI(x)) = \\ & 0.6 \times w(nbd(x, bad) \wedge in\_CALI(x)) \quad]. \end{aligned} \tag{9.6}$$

This results in a strictly stronger formula: (9.2) only impacts on those known to be Californians, whereas (9.6) states a conditional probability that applies to all individuals. Formally, (9.6) $\models$ (9.2) but (9.2) $\not\models$ (9.6). To see this abbreviate (9.6)

to $\forall x : [p(x)]$ and (9.2) to $\forall x : [q(x) \vee w(in\_CALI(x)) \neq 1]$, then

$$
\begin{array}{lllll}
(9.6) & \equiv & \forall x : [& p(x) &] \\
& \Leftrightarrow & \forall x : [& p(x) \wedge (w(in\_CALI(x)) = 1 \quad \vee \quad w(in\_CALI(x)) \neq 1) &] \\
& \Leftrightarrow & \forall x : [& (p(x) \wedge w(in\_CALI(x)) = 1) \quad \vee \quad (p(x) \wedge w(in\_CALI(x)) \neq 1) &] \\
& \Leftrightarrow & \forall x : [& q(x) \quad \vee \quad (p(x) \wedge w(in\_CALI(x)) \neq 1) &] \\
& \Rightarrow & \forall x : [& q(x) \quad \vee \quad w(in\_CALI(x)) \neq 1 &] \\
& \equiv & (9.2) & &
\end{array}
\tag{9.7}
$$

Secondly, although the connection to Bayesian networks is more direct when using conditional probabilities, it is also straightforward to encode Bayesian networks using the rule-based approach [36], since both cases share an underlying directedness.

9.2.4 Defining Joint Distributions

So far we have considered probabilities on the truth values of atomic formulae only. It is necessary to go further and have a mechanism for defining probabilities on (at least) conjunctions of atomic formulae. This defines the *joint* distribution over the truth values of atomic formulae. If each possible world has a conjunction that it alone satisfies, this will give us a complete distribution over possible worlds.

One approach is to assume independence in all cases where this is possible, an approach going back to Boole:

> The events whose probabilities are given are to be regarded as independent of any connexion but such as is either expressed, or necessarily implied, in the data ... ([4], pp. 256-7.)

Where alternatives (in the Poole sense) are used, it is clear that the atomic formulae in any given alternative are highly *dependent*. Equally, those formulae on either side of an implication must be dependent. However, we are at liberty to assume that formulae from different alternatives are independent and this is what is done in PHA/ICL and PRISM; indeed this is why the independent choice logic is so called. Note that this is only possible because these formalisms disallow "inferred probabilities," like $happy(tweety)$, from appearing in alternatives.

When a formalism (implicitly) defines a Bayesian network whose nodes are ground atomic formulae, then the probability of any conjunction is just the probability of the relevant joint instantiation of the Bayesian net in the normal way. A quite different way of combining Bayesian networks with logic is provided by *relational Bayesian networks* [20]. Each node in an RBN corresponds to a *relation*[4] instead of to an atomic formula. The possible values for a relation r are the possi-

4. This includes monadic relations such as $flies$.

ble interpretations of r. An interpretation for a relation r is a set of "true" ground atomic formulae with r as the predicate symbol: something that varies across possible worlds. For example, in one possible world the relation *mother* might have the interpretation

$$\{mother(gill, rob), mother(gill, jane), mother(dot, james)\},$$

whereas in another it might be

$$\{mother(gill, rob), mother(gill, jane), mother(alison, gill)\}.$$

Any joint instantiation of an RBN fixes an interpretation for all relations in the RBN and thus corresponds to some possible world. So an RBN defines a distribution over possible worlds. RBNs are, in fact, a special case of a more general class of models called *random relational structure models*. See Jaeger [19] for the full story.

9.2.5 Avoiding Possible World Semantics

Not all probability-logic formalisms are framed in terms of possible worlds. The hallmark of *Bayesian logic programs* [22] is a one-to-one mapping between ground atomic formulae and random variables *where there is no restriction on what these random variables might be.* In particular a random variable need not represent the probability with which the ground atomic formula *is true*; indeed it need not be binary. One advantage of this design decision is that continuous random variables can be represented. There is, however, a logical aspect to BLPs which has associated semantics. In BLPs, first-order clauses are used, together with combining rules, to define the *structure* of a BLP in much the same way that parent-child edges define the structure of a Bayesian network. Essentially, a ground instance of an atomic formula in the head of a clause corresponds to a child node, whereas those in the body are its parents. Using logical formulae to define the structure of a large (possibly infinite) Bayesian network in this way means that logical methods can be used to reason about the structure of the network. More on BLPs can be found in chapter 9 in this book.

The example of BLPs shows that it can be fruitful to use first-order logic as a convenient way of representing and manipulating data (and models) with complex structure, without too much concern about what the resulting probability distributions "mean." Much, but not all, of the work on SLPs [28, 7] takes this view, The easiest way to understand SLPs is by relating them to stochastic context-free grammars (SCFGs) as Muggleton [28] did in the original paper. In an SCFG each grammar rule has an associated probability. This provides a mechanism for probabilistically generating strings from the grammar: when there is a choice of grammar rules for expanding a nonterminal, one is chosen according to the probabilities. Any derivation in the grammar thus has a probability which is simply the product of the probabilities of all rules used in the derivation. The probability of any string is given by the sum of the probabilities of all derivations which

generate that string. SLPs "lift" this basic idea to logic programs: probabilities are attached to first-order clauses, thus defining probabilities for proofs. SLPs are more complex than SCFGs since they are not generally context-free: not all sequences of clauses constitute a proof; some end in failure. There are different ways of dealing with this—one option is to use backtracking—which define different probability distributions [8]. More on SLPs can be found in chapter 10 of this book.

A "semantics-independent" approach appears pragmatic and flexible: is there not the problem that a formalism with possible-worlds semantics cannot model probability distributions over spaces other than possible worlds? In fact, the distinction between the two approaches is not so fundamental since, with a little imagination, any probability distribution can be viewed as one over some set of possible worlds. Conversely, having possible-worlds semantics certainly does not stop a formalism being applicable to real-world problems.

Moreover, imposing a possible-world semantics on a formalism can provide a useful "bridge" to related formalisms. For example, Cussens [9] provides a possible-worlds semantics to SLPs by translating SLPs into PRISM programs, the latter already having possible-world semantics. This amounts to mapping each proof to a possible world. A characterization of the sort of possible-world distributions thus defined is given by Sato and Kameya [40]. The connection between PHA/ICL and PRISM can then be used to connect SLPs with PHA/ICL.

9.3 Inference

Having defined a probability distribution in a logic-based formalism there remains the problem of computing probabilities to answer specific queries, such as "What's the probability that Tweety flies?" This problem is generally known as "inference" and the term is particularly apposite for a logic-based formalism, since for such formalisms it is possible to exploit nonprobabilistic logical inference to perform complex probabilistic computations. Here we will only consider inference for those formalisms (such as PHA/ICL and PRISM) which use logical implication to define probability distributions, since in such cases normal first-order inference can be used particularly directly to compute probabilities.

Consider, first, standard logical inference—using the first-order logical theory H in (9.8) by way of example. H defines possible output sequences (via the *hmm/2* predicate) for a hidden Markov model (HMM)whose parameters are yet to be defined. The HMM has two states (*s0* and *s1*) and two symbols in its output alphabet (a and b). Both states can emit both symbols and all four possible state transitions are possible. The only formula of any interest is the second which uses the *cons* function symbol to encode a nonempty sequence.

$$\begin{aligned} &\forall s : hmm(s, null) \\ \forall s, x, y, t : &\, hmm(s, cons(x, y)) \leftarrow emit(s, x), next(s, t), hmm(t, y) \\ &\quad emit(s0, a) \wedge emit(s0, b) \wedge emit(s1, a) \wedge emit(s1, b) \\ &\quad next(s0, s0) \wedge next(s0, s1) \wedge next(s1, s0) \wedge next(s1, s1) \end{aligned} \tag{9.8}$$

Because the first-order language $\mathcal{L}$ used in (9.8) contains the function symbol *cons*, the language includes an infinite number of terms, and so there are an infinite number of groundings of the second universally quantified formula. However, to prove, for example, that the ground formula $hmm(s0, cons(a, cons(b, null)))$ follows from H it is not necessary to ground the formulae in H. In practice a theorem prover like Prolog [25] uses *unification* to partially instantiate formulae "on the fly" to establish a proof.

Recall that for PHA/ICL and PRISM there is a strict separation between the probabilistic facts for which probabilities are explicitly stated and all other formulae. To compute the probability of a formula F which is not a probabilistic fact, it suffices to find those conjunctions of facts which, together with the logical rules, entail F. Thanks to the ever-convenient CWA; $P(F)$ is then exactly the sum of the probabilities of these conjunctions. The key point is that finding the required conjunctions is a *purely logical* operation—*abduction*—and so algorithms and implementations for first-order logical inference can be used for it. The key importance of abduction is reflected in the name *probabilistic Horn abduction* [36].

To make this concrete, consider the PRISM program H' in figure 9.2. This is just a parameterized version of H, where, for simplicity, there is an implicit assumption that the initial state of the HMM is given. H' defines a distribution over a countably infinite set of possible worlds (each of which encodes a particular realization of the HMM). The facts are the (infinitely many) $msw/3$ atoms $msw(out(s0), 1, a)$, $msw(out(s0), 2, a)$, $\ldots msw(tr(s1), 1, s0)$, $msw(tr(s1), 2, s0), \ldots$ where the second argument has been suppressed as a programming convenience.

To compute the probability that, say, $hmm(s0, [a, b, a])$ is true, it is necessary to find conjunctions of facts the truth of which entail the truth of $hmm(s0, [a, b, a])$. Fortunately, we can use the PRISM built-in `probf/1` to explicitly show these, as displayed in figure 9.3. Figure 9.3 differs from the actual PRISM output in that the implicit second argument has been made explicit.

The first three lines of figure 9.3 state that $hmm(s0, [a, b, a])$ is true iff either

$$hmm(s0, [b, a]) \wedge msw(out(s0), 1, a) \wedge msw(tr(s0), 1, s0)$$

or

$$hmm(s1, [b, a]) \wedge msw(out(s0), 1, a) \wedge msw(tr(s0), 1, s1)$$

```
values(tr(S),[s0,s1,stop]).
:- set_sw(tr(s0),0.3+0.4+0.3).
:- set_sw(tr(s1),0.4+0.1+0.5).
values(out(S),[a,b]).
:- set_sw(out(s0),0.3+0.7).
:- set_sw(out(s1),0.6+0.4).

hmm(S,[X|Y]) :-
   msw(out(S),X),
   msw(tr(S),T),
   (
     T == stop, Y = []
   ;
     T \= stop, hmm(T,Y)
   ).
```

Figure 9.2 H': a PRISM encoding of a parameterized version of the hidden Markov model in (9.8).

```
| ?- probf(hmm(s0,[a,b,a])).

hmm(s0,[a,b,a])
   <=> hmm(s0,[b,a]) & msw(out(s0),1,a) & msw(tr(s0),1,s0)
     v hmm(s1,[b,a]) & msw(out(s0),1,a) & msw(tr(s0),1,s1)
hmm(s0,[b,a])
   <=> hmm(s0,[a]) & msw(out(s0),2,b) & msw(tr(s0),2,s0)
     v hmm(s1,[a]) & msw(out(s0),2,b) & msw(tr(s0),2,s1)
hmm(s1,[b,a])
   <=> hmm(s0,[a]) & msw(out(s1),3,b) & msw(tr(s1),3,s0)
     v hmm(s1,[a]) & msw(out(s1),3,b) & msw(tr(s1),3,s1)
hmm(s0,[a])
   <=> msw(out(s0),a) & msw(tr(s0),4,stop)
hmm(s1,[a])
   <=> msw(out(s1),a) & msw(tr(s1),4,stop)

yes
| ?- prob(hmm(s0,[a,b,a]),P).

P = 0.012429?
yes
```

Figure 9.3 Using abduction to compute a probability. The PRISM output has been altered so that the second argument on `msw/3` is explicit.

is true. Note that these formulae are guaranteed to be mutually exclusive since $msw(tr(s0), 1, s0)$ and $msw(tr(s0), 1, s1)$ are defined to be alternatives. The next three lines state when $hmm(s0, [b, a])$ is true, and so on. It is not difficult to see that there are exactly eight mutually exclusive conjunctions of $msw/3$ facts which (together with the rules) entail $hmm(s0, [a, b, a])$. For example, one of these conjunctions is $msw(out(s0), 1, a)$, $msw(tr(s0), 1, s0)$, $msw(out(s0), 2, b)$, $msw(tr(s0), 2, s0)$, $msw(out(s0), 3, a)$, $msw(tr(s0), 4, stop)$. Since the $msw/3$ probabilistic facts are defined as independent, the probability of each conjunction is simply a product of the probabilities of the conjuncts. The probability of $hmm(s0, [a, b, a])$ is just the sum of these eight products, which, as figure 9.3 shows, happens to be 0.012429. Naturally, the sum is computed by dynamic programming similar to the variable elimination algorithm used in Bayesian networks. Sophisticated logic programming "tabling" technology can be exploited to do this elegantly and efficiently.

It should be stressed that this example of probabilistic inference was able to exploit the restrictions on PRISM programs that "all the probabilistic ground atoms in the body of each clause are probabilistically independent and the clauses defining a probabilistic predicate are probabilistically exclusive" [42], as well as the CWA. In other cases inference is much harder. For example, inference in PLP requires linear programming to deal with the inequalities involved and the linear program $LP(P)$ needed for a PLP program P "contains exponentially many linear programming variables w.r.t. the size of the Herbrand base of P" [29]. (The *Herbrand base* is the set of all ground atomic formulae expressible in the language used to define P.)

Naturally, one option for hard inference problems is to resort to approximate methods. For example, Angelopoulos and Cussens [1], used an SLP to represent a prior probability distribution over classification trees similarly to the way that the PRISM program above defined a distribution over HMM outputs. Since they adopt a Bayesian approach, learning reduces to probabilistic inference and so the key problem is to compute posterior probabilities: probabilities conditional on the observed data. Using exact inference to compute such probabilities (for example, the posterior class distribution for a test example) seems a hopeless task, so instead, the Metropolis-Hastings algorithm is used to sample from the posterior and thus to produce approximations to the desired probabilities.

9.4 Learning

Having considered how probability-logic formalisms represent probability distributions and how inference can be used to compute probabilities of interest, we can now turn to the issue of learning a model from data. As always, we consider the observed data as a sample generated by some unknown "true" model whose identity we wish to "learn" (or rather estimate). In some cases only the parameters of the model are unknown. In the general case both the structure and parameters of

the true model are unknown. These two cases are considered in section 9.4.1 and section 9.4.2, respectively.

The paper by De Raedt and Kersting [10] provides an excellent overview of learning in probability-logic formalisms. The current section is complementary to De Raedt and Kersting's broad survey since, (1) for the sake of concreteness it examines parameter estimation in some detail for a particular formalism (PRISM) and (2) discusses the use of probabilities in pre-SRL inductive logic programming (ILP). An examination of pre-SRL ILP is useful since it seems likely that some of the techniques found there may be useful for more recent formalisms.

Before focusing on these two areas it is worth mentioning two key points about learning in probability-logic formalisms which are provided by De Raedt and Kersting [10].

- Much of the machinery for learning Bayesian networks can be used to learn directed probability-logic models such as BLPs. When first-order clauses are used for the structure of a directed model, then specialization and generalization of clauses corresponds to using a macro-operator for adding and deleting arcs in a Bayesian network. Parameter estimation for logical directed models corresponds to parameter estimation with tied parameters in a normal Bayesian network. This is because one first-order clause typically represents a collection of network fragments in the underlying Bayesian network via its ground instances.
- The probabilistic models associated with a PHA/ICL, PRISM, or SLP model depend on parameters associated with many predicates in the underlying logic program. This means that structure learning for such models is, in general, at least as hard as multiple-predicate learning / theory revision in ILP: which is known to be a hard problem. However, there exists work on learning SLPs in a restricted setting [27], and also work on applying grammatical inference techniques to learn SLPs [3].

As for other types of statistical inference, the key to learning in probability-logic formalisms is the likelihood function: the probability of the observed data as a function of the model. If the structure is fixed, then the likelihood is just a function of the model parameters. If a probability-logic formalism defines a distribution over possible worlds, then ideally we would like the data to be a collection of independent observations of possible worlds, each viewed as a sample drawn from the unknown "true" model. The probability of each world can then be computed using inference (section 9.3) and the likelihood of the data is just a product of these probabilities. In the case of alternative-based formalisms like PHA/ICL, PRISM, and SLPs, each data point would then be associated with a unique conjunction of atomic choices. Maximum likelihood estimation of the multinomial distribution over each alternative is then possible by simple counting in the normal way. A Bayesian approach using Dirichlet priors is equally simple.

However, in many cases each observation is a ground atomic formula, which is true in many worlds. This means the data-generating process is best viewed in terms of missing data: the "true" model generates a world, but we do not get to

```
hmm_out(X) :- hmm(s0,1,X).

hmm(S,N,[X|Y]) :-
  msw(out(S),X),
  msw(tr(S),T),
  (
    T == stop, Y = []
  ;
    T \= stop, NN is N+1, hmm(T,NN,Y)
  ).
```

Figure 9.4 PRISM program such that only one ground instance of *hmm_out*/1 is true in each possible world

see this world, only some ground atomic formula that is true in it. The rest of the information required to determine the sampled world is missing. Unsurprisingly, the expectation maximization (EM) algorithm is generally used in such situations.

An alternative approach is presented by Kok and Domingos [24]. Here the data is contained in a (multitable) relational database. Each row in each table defines a ground atomic formula in the usual way. The entire database is equivalent to a conjunction of all these ground atomic formulae (so it is equivalent to a Datalog Prolog program). Using the CWA, this defines a unique world: all formulae which are not consequences of the conjunction are deemed false. (This unique world is the *minimal Herbrand model* of the associated Prolog program.) So, on the one hand we have only a single observation, but on the other it is an entire world that is observed. See Kok and Domingos [24] and chapter 11 this book for further details.

9.4.1 Parameter Estimation

To analyze parameter estimation, a modeling convention introduced by Sato and Kameya [40] will be adopted. There will be a *target predicate* such that in each possible world exactly one ground instance of this target predicate is true. This turns out not to be much of a restriction and simplifies the analysis greatly. For example, consider using PRISM to learn the parameters of the HMM of figure 9.2. The predicate $hmm/2$ is not a suitable target predicate, since any world in which all the following formulae are true—$msw(out(s0),1,a)$, $msw(tr(s0),1,s0)$, $msw(out(s0),2,b)$, $msw(tr(s0),2,s0)$, $msw(out(s0),3,a)$, $msw(tr(s0),4,stop)$—will also have $hmm(s0,[a,b,a])$, $hmm(s0,[b,a])$, and $hmm(s0,[a])$ true. This is illustrated by figure 9.3. Also, figure 9.2 as it stands is a conditional model—conditional on the initial state—since no distribution over the initial state has been defined. Changing the program to the one given in figure 9.4 and declaring that `hmm_out/1` is the target predicate— `target(hmm_out,1)` — is enough to fix both these problems. Now exactly one *hmm_out*/1 formula will be true in each world.

However, each *hmm_out*/1 formula will be true in many worlds, so that for maximum likelihood estimation of model parameters the EM algorithm is used. Fig-

```
| ?- learn([hmm_out([a,b,a,a]),hmm_out([b,b]),hmm_out([a,a,b])]).
..
Finished learning
        Number of iterations: 13.
        Final likelihood:-9.440724
        Total learning time: 0.01 seconds.
        All solution search time: 0.01 seconds.
        Total table space used: 6304 out of 240000000 bytes
Type show_sw to show the probability distributions.
yes
| ?- show_sw
Switch tr(s1): unfixed: s0 (0.235899) s1 (0.000014) stop (0.764086)
Switch out(s1): unfixed: a (0.254708) b (0.745291)
Switch tr(s0): unfixed: s0 (0.226177) s1 (0.773818) stop (0.000003)
Switch out(s0): unfixed: a (0.788360) b (0.211639)
```

Figure 9.5 EM learning with PRISM. (I have edited the output to reduce the precision of parameters.)

ure 9.5 shows a run of PRISM using the EM algorithm to estimate the parameters of the HMM using a data set of three examples. The algorithm was initialized with the values shown in figure 9.2. Abduction is used *once* to produce the data structure shown in figure 9.3. This can then be used in each iteration of the EM algorithm to compute the expected values required in the E-part of the EM algorithm.

So far a very simple and hopefully familiar example—HMM parameter estimation—has been used to explain parameter estimation in PRISM. Of course, there is no pressing reason to use SRL for this problem. The whole point of SRL is to address problems outside the remit of more standard approaches. So now consider a simple elaboration of the HMM learning problem which highlights some of the flexibility of a logic-based approach. Suppose that the HMM is *constrained* so that not all outputs from the HMM are permitted. This amounts to altering the definition of `hmm_out/1` to

```
hmm_out(X) :- hmm(s0,1,X), constraint(X),          (9.9)
```

where the predicate *constraint*/1 is *any predicate* which can be defined using (clausal) first-order logic. Now there will be worlds in which no ground instance of the target predicate is true: these worlds will not be associated with a possible data point. To take a very simple example, if *constraint*/1 were defined thus:

```
constraint(X) :- X = [Y,Y|Z],
```

then the possible world illustrated by figure 9.3 would no longer entail $hmm\_out([a, b, a])$, since the first two elements differ. The distribution over ground instances of *hmm_out*/1 is now a conditional one: conditional on the logically defined constraint being satisfied. This turns out to be an exponential-family distribution

where the partition function Z is the probability that the constraint is true in a world sampled from the original, unconditional distribution.

It is still possible to use the EM algorithm to search for maximum likelihood estimates of the parameters of such a distribution; it is just that the generative characterization of this conditional distribution is more complicated. We assume, as always, that worlds are sampled from the true underlying distribution. If no ground instance of the target predicate is true in a sampled world, then that world is rejected and no data point is generated; otherwise the unique ground instance of the target predicate which is true in that world is added to the data. Viewing the observed data as being generated in this fashion we have an extra sort of missing data: the worlds which were entirely rejected. So the data is now *truncated* data. Fortunately, Dempster et al. [13] show that the EM algorithm is applicable even when the data has been truncated like this. The method was applied to SLPs by Cussens [7] under the name *failure-adjusted maximization (FAM)* and is used in the most recent version of the PRISM system [41].

9.4.2 Structure Learning

The structure of a probability-logic model is by definition some sort of first-order theory, frequently a set of first-order clauses, i.e., a logic program. ILP is the branch of machine learning concerned with inducing logic programs from data, so it is no surprise that ILP techniques are often used when learning the structure of probability-logic models from data.

Since its inception ILP has had no option but to induce de facto probabilistic models for the simple reason that deterministic, purely logical, rules rarely fit the data—however, the probabilistic aspect of induced rules has not always been properly formalized. In its simplest form, data for ILP is a set of true facts (positive examples) and a set of untrue facts (negative examples). The ideal is to find a set of clauses which, when added to an existing logic program (the background knowledge), entail all of the positives and none of the negatives. There is generally a bias for simple theories so that, for example, just returning the positive examples as the induced theory is decidedly suboptimal. In most real applications, this logical ideal is unreachable. So, instead of (hopelessly) searching for rules which fit the data exactly, many ILP systems search for *accurate* rules: ones which entail many more positives than negatives. Of course, training set accuracy can be an unreliable guide to true accuracy, so, for example, Bayesian estimates of true accuracy can be used Džeroski [14]. Bayesian estimation is available in the ALEPH system by using `mestimate` as the clause evaluation function, and setting the `m` parameter to define the underlying prior distribution. Each induced rule can now be returned with an associated parameter: its expected accuracy, or informally, its *probability*.

Using ILP to distinguish positives from negatives is most readily applicable to binary classification. In many other cases (particularly those where probability is explicitly represented) it is more appropriate to induce a logic program that *instantiates variables* to perform classification, regression [21], or more complex

tasks akin to program synthesis. For example, figure 9.6 shows a clause induced using the ALEPH ILP system (from example input that comes with that system to demonstrate ILP learning of classification trees). The clause probabilistically classifies days according to whether they are suitable for playing or not by the simple expedient of "putting probability in the background." No negative examples

```
class(A,B) :-
   not (outlook(A,rain),windy(A,true)), outlook(A,sunny),
   humidity(A,C), lteq(C,70),
   random(B,[0.75-play,0.25-dont_play]).
```

Figure 9.6 Probabilistic classification rule induced by ALEPH.

are used to induce such a rule: the key is to declare that the class `B` is an output to be computed from the day `A` which is an input. In the ALEPH and Progol systems this is done with the declaration in figure 9.7.

```
:- modeh(1,class(+day,-class)).
```

Figure 9.7 ALEPH declaration that the `class/2` variable takes an input (indicated by the "+") of type `day` and generates an output (indicated by the "-") of type `class`.

It is possible to use an ILP algorithm to search for rules and then build some probabilistic model from these rules afterward. One option is to use a *combining rule* to compute probabilities for test examples entailed by more than one induced rule. (See chapter 9 for further details on combining rules.) Pompe and Kononenko [34] use a naive Bayes model to combine first-order classification rules with a later approach splitting induced first-order rules to better approximate the naive Bayes assumption [35]. This work is an example of the often used technique of viewing first-order rules (or parts of rules) as features for a nonlogical probabilistic model. If induced rules are going to be used eventually as the structural component of a probabilistic model, then naturally it is better that the algorithm searching for rules is designed to find rules suitable for this purpose.

A more thoroughly probabilistic approach is to use ILP techniques as subroutines in an algorithm that directly learns a probabilistic model from data. This is the approach taken by Dehaspe [11] with his MACCENT algorithm. The goal of MACCENT is to learn a conditional distribution giving a distribution over classes (C) for any given example (I). MACCENT uses the ILP framework of *learning from interpretations* where each example I is a Prolog program. There are thus connections to the approach of Kok and Domingos [24] mentioned earlier. The

distribution is always a conditional exponential-family distribution of the form

$$p_\lambda(C|I) = \frac{1}{Z_\lambda(I)} \exp\left(\sum_{m=1}^{m=M} \lambda_m f_{j_m,k_m}(I,C)\right),$$

where each feature $f_{j,k}$ is a *Boolean clausal indicator function* defined using a class C_j and a Prolog query Q_k as follows:

$$f_{j,k}(I,C) = \begin{cases} 1 & \text{if } C = C_j \text{ and Prolog query } Q_k \text{ succeeds in instance } I \\ 0 & \text{otherwise} \end{cases}.$$

Both parameters λ_m and the features f_{j_m,k_m} are learned using an adaptation of the algorithm of Della Pietra et al. [12]. MACCENT searches the lattice of clausal indicator functions for good features using standard ILP search where these functions are ordered by logical generality (or more precisely θ-subsumption [33]).

9.5 Conclusion

In this chapter we have looked at the big three issues of representation, inference, and learning for probability-logic models, with a focus on representation. What is exciting about the current interest in SRL is that techniques for all three of these (often originating from different communities) are coming together to produce powerful techniques for learning from structured and relational data. (It is worth noting that there are initiatives with similar goals originating from the statistical community [16], although there logical approaches are not currently used.) The number of applications which involve such data are many: almost any real-world problem for which standard ILP is a reasonable choice—and many more besides—is also a target for SRL. To take just three recent examples, Frasconi et al. [15] applied a "declarative kernel" approach to (1) predicting mutagenicity, (2) information extraction, and (3) prediction of mRNA signal structure; Lodhi and Muggleton [26] applied failure-adjusted maximization to learn SLPs to model metabolic pathways and Riedel and Klein [38] learnt MLN based on discourse representation structures of a sentence to extract gene-protein interactions from annotated Medline abstracts.

References

[1] N. Angelopoulos and J. Cussens. Exploiting informative priors for Bayesian classification and regression trees. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 2005.

[2] A. Arnauld and P. Nicole. *Port-Royal Logic*. Translated by Bobbs-Merrill, Indianapolis, IN, 1964.

[3] M. Bernard and A. Habrard. Learning stochastic logic programs. In *Proceedings of the Work-in-Progress Track at the International Conference on Inductive Logic Programming*, 2001.

[4] G. Boole. *An Investigation of the Laws of Thought, on which are founded the Mathematical Theories of Logic and Probabilities*. Reprint, Dover Publications, New York, NY, 1958.

[5] R. Carnap. *Logical Foundations of Probability*. University of Chicago Press, Chicago, 1950.

[6] J. Cussens. Bayesian inductive logic programming with explicit probabilistic bias. Technical Report PRG-TR-24-96, Oxford University Computing Laboratory, Oxford, UK, 1996.

[7] J. Cussens. Parameter estimation in stochastic logic programs. *Machine Learning*, 44(3):245–271, 2001.

[8] J. Cussens. Stochastic logic programs: Sampling, inference and applications. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, 2000.

[9] J. Cussens. Integrating by separating: Combining probability and logic with ICL, PRISM and SLPs. APRIL project report, January 2005.

[10] L. De Raedt and K. Kersting. Probabilistic logic learning. *SIGKDD Explorations*, 5(1):31–48, 2003.

[11] L. Dehaspe. Maximum entropy modeling with clausal constraints. In *Inductive Logic Programming*, 1997.

[12] S. Della Pietra, V. Della Pietra, and J. Lafferty. Inducing features of random fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19 (4):380–393, 1997.

[13] 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, Series B*, 39(1):1–38, 1977.

[14] S. Džeroski. Handling Noise in Inductive Logic Programming. Master's thesis, Faculty of Electrical Engineering and Computer Science, University of Ljubljana, Ljubljana, Slovenia, 1991.

[15] P. Frasconi, A. Passerini, S. Muggleton, and H. Lodhi. Declarative kernels. Inductive Logic Programming, 2005.

[16] P. Green, N. Hjort, and S. Richardson, editors. *Highly Structured Stochastic Systems*. OUP, 2003.

[17] J. Halpern. An analysis of first-order logics of probability. *Artificial Intelligence*, 46:311–350, 1990.

[18] C. Howson and P. Urbach. *Scientific Reasoning: The Bayesian Approach*. Open Court, La Salle, Illinois, 1989.

[19] M. Jaeger. Relational Bayesian networks: A survey. *Electronic Transactions in Artificial Intelligence*, 6, 2002.

[20] Manfred Jaeger. Relational Bayesian networks. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, 1997.

[21] A. Karalič and I. Bratko. First order regression. *Machine Learning*, 26(2-3): 147–176, 1997. ISSN 0885-6125.

[22] K. Kersting and L. De Raedt. Bayesian logic programs. Technical Report 151, University of Freiburg, Freiburg, Germany, April 2001.

[23] J. Keynes. *A Treatise on Probability*. Macmillan, London, 1921.

[24] S. Kok and P. Domingos. Learning the structure of Markov logic networks. In *Proceedings of the International Conference on Machine Learning*, 2005.

[25] J. Lloyd. *Foundations of Logic Programming*. Springer, Berlin, second edition, 1987.

[26] H. Lodhi and S. Muggleton. Modelling metabolic pathways using stochastic logic programs-based ensemble methods. In *Proceedings of the International Conference on Computational Methods in System Biology*, 2004.

[27] S. Muggleton. Learning the structure and parameters of stochastic logic programs. In *Inductive Logic Programming*, 2002.

[28] S. Muggleton. Stochastic logic programs. In L. De Raedt, editor, *Advances in Inductive Logic Programming*, volume 32 of *Frontiers in Artificial Intelligence and Applications*, pages 254–264. IOS Press, Amsterdam, 1996.

[29] R. Ng and V.S. Subrahmanian. A semantical framework for supporting subjective and conditional probabilities in deductive databases. *Journal of Automated Reasoning*, 10(2):191–235, 1993.

[30] R. Ng and V.S. Subrahmanian. Probabilistic logic programming. *Information and Computation*, 101(2):150–201, 1992.

[31] L. Ngo and P. Haddaway. Answering queries from context-sensitive probabilistic knowledge bases. *Theoretical Computer Science*, 171:147–171, 1997.

[32] N. Nilsson. Probabilistic logic. *Artificial Intelligence*, 28:71–87, 1986.

[33] Gordon D. Plotkin. A note on inductive generalization. *Machine Intelligence*, 5:153–163, 1970.

[34] U. Pompe and I. Kononenko. Naive Bayesian classifier within ILP-R. In *Inductive Logic Programming*, 1995.

[35] U. Pompe and I. Kononenko. Probabilistic first-order classification. In *Inductive Logic Programming*, 1997.

[36] D. Poole. Probabilistic Horn abduction and Bayesian networks. *Artificial Intelligence*, 64(1):81–129, 1993.

[37] D. Poole. The independent choice logic for modelling multiple agents under uncertainty. *Artificial Intelligence*, 94(1–2):5–56, 1997.

[38] S. Riedel and E. Klein. Genic interaction extraction with semantic and syntactic chains. In *Proceedings of the Learning Language in Logic Workshop*, 2005.

[39] T. Sato. A statistical learning method for logic programs with distribution semantics. In *Inductive Logic Programming*, 1995.

[40] T. Sato and Y. Kameya. Parameter learning of logic programs for symbolic-statistical modeling. *Journal of Artificial Intelligence Research*, 15:391–454, 2001.

[41] T. Sato, Y. Kameya, and N. Zhou. Generative modeling with failure in PRISM. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 2005.

[42] N. Zhou, T. Sato, and Y. Kameya. *A Reference Guide to PRISM Version 1.7*, March 2004.