Generative Modeling by PRISM

Taisuke Sato

Tokyo Institute of Technology, Ookayama Meguro Tokyo Japan,
`http://sato-www.cs.titech.ac.jp/`

Abstract. PRISM is a probabilistic extension of Prolog. It is a high level language for probabilistic modeling capable of learning statistical parameters from observed data. After reviewing it from various viewpoints, we examine some technical details related to logic programming, including semantics, search and program synthesis.

1 Introduction

Generative modeling is a way of probabilistic modeling that describes a generating process of an outcome in a sample space. It has however different nuances in different fields. In statistics it is oftentimes used in the context of Bayesian inference, which (hierarchically) assumes prior distributions on parameters. In ML(machine learning), it means defining joint distributions $p(x, y)$ where x is an input and y an output in contrast to discriminative modeling, which defines conditional distributions $p(y \mid x)$ for the same x and y. Or in statistical natural language processing, it usually refers to language modeling by probabilistic grammars such as HMMs (hidden Markov models) and PCFGs (probabilistic context free grammars). Here we add another nuance; by generative modeling we mean a specification of a sampling process by a probabilistic program for a given distribution.

Traditionally probabilistic models have been specified by mathematical formulas (equations) and graphs like BNs (Bayesian networks) and MRFs (Markov random fields) and programming languages were not considered as a specification language of probabilistic models. If, however, it becomes possible to use programs as probabilistic models, we will have much more flexibility in modeling because of the availability of various data structures (strings, trees, graphs) and program constructs (such as composition, if-then-else and recursion), and also a uniform mechanism (Turing machine). In addition, the expressive power of a high level programming language will reduce the coding effort to a minimum. So it seems beneficial to design a programming language and programs which denote probabilistic models. Indeed there are already a plethora of such proposals, in particular in a subfield of machine learning called PLL (probabilistic logic learning) originating in LP(logic programming)/ILP(inductive logic programming) [1–12] and SRL(statistical relational learning) originating in uncertainty reasoning by BNs [13–24].

In this talk, we examine PRISM[1][2, 5], a probabilistic extension of Prolog aiming at generative modeling by probabilistic logic programs. We will however focus on the relationship between PRISM and LP and applications to machine learning are not treated.

2 Three viewpoints

We can see PRISM from three points of view.

[**LP view**] PRISM is a probabilistic extension of Prolog[2].
Syntactically a PRISM program $DB = F \cup R$ is a Prolog program such that F is a set of probabilistic atoms called `msw` atoms (see below) and R is a set of definite clauses whose head contains no `msw` atom. We use `msw(`i`,X)` to simulate a probabilistic choice named i (ground term) which returns in `X` a value probabilistically chosen from finite outcomes associated with i. Probabilities of `msw` atoms being true are called *parameters*. Semantically PRISM's declarative semantics, the *distribution semantics*, defines a probability measure $P_{DB}(\cdot \mid \boldsymbol{\theta})$ on Herbrand models having parameters $\boldsymbol{\theta}$ associated with `msw` atoms. It subsumes the least model semantics of definite clause programs. Practically what PRISM can do but Prolog cannot do is parameter learning. PRISM programs can learn $\boldsymbol{\theta}$ from data and change their probabilistic behavior.

[**ML view**] PRISM is a high level language for probabilistic modeling.
It is an outcome of PLL/SRL research, but unlike graphical models, it uses logical rules such as definite clauses or sometimes normal clauses to define distributions. Here is a short list of machine learning facilities supported by PRISM.[3]

Sampling: For the program $DB = F \cup R$, `sample(`G`)` executes G (atom) exactly as a Prolog goal using clauses in R except `msw(`i`,X)` which returns a probabilistically chosen value (ground term) in `X`.

Search: `probf(`G`)` returns, by searching for all SLD proofs for G with respect to DB, a boolean formula $E_1 \vee \ldots \vee E_n$ such that $E_i \wedge R \vdash G$ $(1 \leq i \leq n)$. Each E_i is a conjunction of ground `msw` atoms and called an *explanation* for G. $G \Leftrightarrow E_1 \vee \ldots \vee E_n$ holds with probability one in terms of $P_{DB}(\cdot)$.

Probability computation: `prob(`G`)` computes $P_{DB}(G)$, the probability of G by $P_{DB}(\cdot)$ whereas `chindsight(`G`,`G'`)` computes the conditional probability $P_{DB}(G' \mid G)$ of a subgoal G' that occurs in a proof of G.

Viterbi inference: `viterbif(`G`)` returns the most probable explanation for G together with its probability.

Parameter learning: PRISM provides MLE(maximum likelihood estimation), MAP(maximum a posteriori) estimation for parameter leaning and

[1] `http://sato-www.cs.titech.ac.jp/prism/index.html`
[2] Currently PRISM is built on top of B-Prolog (`http://www.probp.com/`).
[3] See the PRISM manual for the complete list of available predicates.

VB (variational Bayes) for hyper parameter learning (priors are Dirichlet distributions). These are available through `learn/1` built-in predicate.

Model selection: To help structure learning, PRISM has special built-in predicates to compute criteria for model selection. They include BIC (Bayes information criterion), CS (Cheeseman-Stutz) score and VFE (variational free energy).

The primary benefit of PRISM modeling from the ML point of view is the ease of probabilistic modeling. We have only to write a program by a high level language and use it. There is no need for a laborious chain of deriving a learning algorithm, designing a data structure and implementing and debugging them. The result is a significant saving of time and energy. This is especially true when we attempt to develop a new model while going through cycles of probabilistic modeling. Think of developing some variants of HMMs for example. Once we write a basic HMM program, it is relatively easy to modify it. If the modified model goes wrong, just rewrite the program. We are free of implementing similar algorithms for similar HMMs all over again from scratch.

[**AI view**] PRISM is a system for statistical abduction.

PRISM performs search, computation and learning, all necessary elements of AI, in a unified manner under the distribution semantics. They are seamlessly integrated as *statistical abduction* [25]. In logical abduction, we abduce an explanation E for an observed goal G by search, using background knowledge $\mathbf{B}$, such that $E \wedge \mathbf{B} \vdash G$ and $E \wedge \mathbf{B}$ is consistent. Usually E is restricted to a conjunction of special atoms called abducibles. In statistical abduction, we further assume a distribution on abducibles and learn their probabilities from data. By doing so we can select E having the highest probability as the best explanation for G. In PRISM's case, the set R of definite clauses in a program $DB = F \cup R$ corresponds to $\mathbf{B}$ and `msw`s in F play the role of abducibles.

Here is a small PRISM program.

```
values_x(p1,[rock,paper,scissors],fix@[0.4,0.4,0.2]).
values_x(p2,[rock,paper,scissors],[0.1,0.3,0.6]).

rps(R1,R2):-
    msw(p1,X),msw(p2,Y),
    ( X=Y -> R1=draw,R2=draw
    ; ((X=rock,Y=paper);(X=paper,Y=scissors);(X=sissors,Y=rock))
          -> R1=lose,R2=win
    ; R1=win,R2=lose ).
```

Fig. 1. Rock-paper-scissors program

This program simulates the rock-paper-scissors game. The first `values_x/3` clause introduces a probabilistic choice `msw(p1,X)` with a player `p1` and a gesture `X` being one of {`rock`, `paper`, `scissors`}, with corresponding parameters (probabilities) 0.4, 0.4 and 0.2 for each. "`fix@`" means parameters associated with `p1` do not change by learning. The second clause is understood similarly but the parameters, 0.1, 0.3 and 0.6, are temporarily set and changeable by learning. The last clause plays the rock-paper-scissors game. It first calls `msw(p1,X)` to probabilistically choose a gesture `X` for `p1` and similarly `Y` for `p2` by `msw(p2,Y)`. It then determines `win`, `lose` or `draw` by comparing `X` and `Y`.

```
?- prism(rock_paper_scissors).
...
?- get_samples(1000,rps(R1,R2),Gs),learn(Gs).
...
#em-iterations: 0.......(79) (Converged: -1091.799641688)
Statistics on learning:
       Graph size: 18
       Number of switches: 2
       Number of switch instances: 6
       Number of iterations: 79
       Final log likelihood: -1091.799641688
       Total learning time: 0.004 seconds
       Explanation search time: 0.000 seconds

?- show_sw.
Switch p1: rock (p: 0.4000) paper (p: 0.4000) scissors (p: 0.2000)
Switch p2: rock (p: 0.0641) paper (p: 0.3466) scissors (p: 0.5892)

?- viterbif(rps(win,lose)).
rps(win,lose) <= msw(p1,rock) & msw(p2,scissors)
```

Fig. 2. Learning session

Fig. 2 is a sample learning session (predicates used there are all built-ins). We first load the program in Fig. 1 on a file `rock_paper_scissors.psm` by `prism/1`. We then generate learning data `Gs` = [`rps(win,lose)`, `rps(draw,draw)`, ...] by `get_samples/3` which sampled `rps(R1,R2)` 1,000 times[4]. `learn(Gs)` internally invokes a built-in EM algorithm to estimate parameters. The learning is completed after 79 iterations. The estimated values are shown by `show_sw/0`. Using the learned parameters, we compute by `viterbif/1` the most probable gestures that cause `rps(win,lose)`, i.e. `p1` wins and `p2` loses. They are `rock` for `p1` and `scissors` for `p2` with log-probability -2.1972.

[4] `p1` wins 343 times, `p2` wins 375 times, draw 282 times.

3 Inside PRISM: three topics

As mentioned before, PRISM can be seen from three points of view. In this section we pick up the LP view and look into some details of three topics which illustrate how PRISM is connected to LP. They are semantics [2, 5], tabling [26] and program synthesis [27].

3.1 Probabilistic semantics

The distribution semantics of PRISM is a probabilistic generalization of the least model semantics in LP. It defines a probability measure on the set of Herbrand models. Let $DB = F \cup R$ be a PRISM program. Also let $\mathtt{msw}_1, \mathtt{msw}_2, \ldots$ be an enumeration of the `msw` atoms in F. We identify an infinite 0-1 vector $\omega_F = (x_1, x_2, \ldots)$ where $x_i \in \{0, 1\}$ with a Herbrand model that assigns $\mathtt{msw}_1 = x_1$, $\mathtt{msw}_2 = x_2, \ldots$ where 1 means true and 0 false. Let $P_F(\cdot)$ be an arbitrary *base measure* on such ω_Fs such that for a choice named i with possible outcomes $\{v_1, \ldots, v_k\}$, $P_F(\cdot)$ makes $\{\mathtt{msw(}i\mathtt{,}v_1\mathtt{)}, \ldots, \mathtt{msw(}i\mathtt{,}v_k\mathtt{)}\}$ exhaustive and mutually exclusive. That is $P_F(\mathtt{msw(}i\mathtt{,}v_1\mathtt{)} \vee \cdots \vee \mathtt{msw(}i\mathtt{,}v_k\mathtt{)}) = 1$ and $P_F(\mathtt{msw(}i\mathtt{,}v_h\mathtt{)} \wedge \mathtt{msw(}i\mathtt{,}v_{h'}\mathtt{)}) = 0$ $(h \neq h')$. It is straightforward to construct such $P_F(\cdot)$.

We now extend the $P_F(\cdot)$, using the mechanism of the least Herbrand model construction, to a probability measure $P_{DB}(\cdot)$ for the whole DB. Let $\omega_{F'}$ be a sample from $P_F(\cdot)$ and F' the set of `msw` atoms made true by $\omega_{F'}$. Construct the least Herbrand model $\mathbf{M}(F' \cup R)$ of the definite program $F' \cup R$. It uniquely determines the truth value of every ground atom and by construction every ground atom is a measurable function of $\omega_{F'}$ with respect to $P_F(\cdot)$. It follows from this fact and Kolmogorov's extension theorem that we can extend $P_F(\cdot)$ to the probability measure $P_{DB}(\cdot)$ on the set of possible Herbrand models for DB. $P_{DB}(\cdot)$ is the denotation of DB in the distribution semantics [5]. If $P_F(\cdot)$ puts all probability mass on a single interpretation F', P_{DB} puts all probability mass on the least model $\mathbf{M}(F' \cup R)$ also. Hence we can say the distribution semantics is a probabilistic generalization of the least model semantics. Hereafter for intuitiveness, we identify $P_{DB}(\cdot)$ with an infinite joint distribution $P_{DB}(A_1 = x_1, A_2 = x_2, \ldots)$ on the probabilistic ground atoms $A_1, A_2, \ldots$ in the Herbrand base of DB where $x_i \in \{0, 1\}$, when appropriate.

We remark that our semantics (probability measure on possible models, or worlds) is not new. Fenstad proved a representation theorem forty years ago [28]. It states that if we assign probabilities $P(\varphi)$ to closed formulas φ in a countable language $\mathcal{L}$ without equality, respecting Kolmogorov's axioms for probability while satisfying $P(\varphi) = 1$ if $\vdash \varphi$ and $P(\varphi) = P(\phi)$ if $\vdash \varphi \Leftrightarrow \phi$, $P(\varphi)$ is given as an integration

$$P(\varphi) = \int_{\Omega} \varphi(\omega)\, \mu(d\omega)$$

where $\mu(\cdot)$ is a probability measure on a certain set Ω of models related to $\mathcal{L}$ and $\varphi(\omega) = 1$ if a model $\omega \in \Omega$ satisfies φ, else 0[5]. What is semantically new here is that we construct such $\mu(\cdot) = P_{DB}(\cdot)$ concretely from a logic program DB so that $P_{DB}(G)$ is computable for a goal G. We point out some unique features of the distribution semantics.

- $P_{DB}(\cdot)$ is an infinite joint distribution on countably many random atoms. *It is definable, unconditionally, for any DB*. Other PLL formalisms often place restrictions on DB such as acyclicity [1, 8] and range-restrictedness [6, 9, 3] for their distributions to be definable. SRL formalisms attempting to define infinite distributions also have restrictions on their programs [29, 21, 30].
- Probabilistic grammars such as HMMs and PCFGs that define finite stochastic processes but whose length is unbounded are formally captured by PRISM programs with the distribution semantics. Thanks to the rigor of the distribution semantics, it is even possible to write a PRISM program defining prefix probabilities for a given PCFG, though their computation requires an infinite sum and cannot be handled by the current PRISM system[6] [31].
- The distribution semantics is parameterized with a non-probabilistic semantics $\mathbf{M}$ used to extend the base measure $P_F(\cdot)$. That is, if we choose as $\mathbf{M}$ the greatest model semantics instead of the least model semantics, we will have another distribution, which is always definable but not necessarily computable, giving non-zero probability to infinite recursion. $\mathbf{M}$ may be stable model semantics [32, 11] or well-founded semantics [33, 8]. In such cases, we will have distributions for normal probabilistic logic programs.

3.2 Tabling and dynamic programming

In PRISM the probability $P_{DB}(G)$ of an atom G is computed by first reducing G logically to a disjunction $E_1 \vee \ldots \vee E_n$ of explanations and then computing $P_{DB}(G)$ by $P_{DB}(G) = \sum_{i=1}^{n} P_{DB}(E_i)$, $P_{DB}(E_i) = \prod_{k=1}^{h_i} \theta_{i,k}$ where $E_i = \mathtt{msw}_{i,1} \wedge \cdots \wedge \mathtt{msw}_{i,h_i}$ and $\theta_{i,k}$ is the parameter of $\mathtt{msw}_{i,k}$ $(1 \leq k \leq h_i)$. A computational barrier here is that usually there are exponentially many explanations. In the case of parsing where G represents a sentence and E_i a parse tree, it is not rare to have millions of parse trees. One standard way to avoid such intractable computation is applying DP (dynamic programming) to $E_1 \vee \ldots \vee E_n$ that factors out common probability computations. But the real problem is not DP but how to realize it *without* constructing $E_1 \vee \ldots \vee E_n$.

Our solution to this problem is tabling, or memoizing, which is a general technique to record what has been computed and reuse it later, thereby saving repeated computation. In addition, tabling has the side-effect of stopping infinite recursion. This makes it possible to write a DCG grammar containing left recursive rules such as `NP` $\rightarrow$ `NP S`. LP has a long history of tabling [34–38, 26]

[5] The actual Fenstad's theorem is more complicated than stated here. We show the case of closed formulas for simplicity.

[6] Prefix probabilities can be computed by matrix operations [31].

[7] and what we have found through the development of PRISM is that tabling is well-suited, or vital to probability computation in machine learning.

By introducing tabling for all explanations search for a goal G, we can obtain a boolean formula, equivalent to $G \Leftrightarrow E_1 \vee \ldots \vee E_n$, as a descendingly ordered list of equivalences $G \Leftrightarrow W_0, A_1 \Leftrightarrow W_1, \ldots, A_M \Leftrightarrow W_M$ such that A_i $(1 \leq i \leq M)$, a tabled goal appearing in a proof of G, represents a subexpression occurring multiple times in the explanations $E_1, \ldots, E_n$ and W_i is a conjunction of `msw` atoms and tabled goals in the lower layers. We consider this list as a graph whose node are atoms and call it an *explanation graph* for G. In the explanation graph, G is a root node and subgraphs (tabled goals) at one layer are shared by subgraphs at higher layers. Hence probability computation (sum-product computation) applied to it naturally becomes DP. Thus we can realize DP by tabling while avoiding the construction of $E_1 \vee \ldots \vee E_n$. We encode the DP process as the g-IO (generalized IO) algorithm working on explanation graphs. It is a generic routine in PRISM to compute probabilities [5].

The effect of tabling is decisive. The g-IO algorithm simulates known standard probability computation/learning algorithms with the same time complexity; $O(N^2L)$ for the Baum-Welch algorithm used in HMMs [39] where N is the number of states, L input length, $O(N^3L^3)$ for the Inside-Outside algorithm used in PCFGs in Chomsky normal form [40] where N is the number of symbols and L sentence length and $O(N)$ for Pearl's $\pi\lambda$ message passing [41] used in the probability computation of singly connected BNs where is N the number of nodes in a BN [5].

Also, recently, it is discovered that the celebrated BP (belief propagation) algorithm used for the probability computation of multiply connected BNs is nothing but the g-IO algorithm applied to logically described junction trees [42]. In other words, to use BP, we have only to write a program describing a junction tree[8]. We may say PRISM subsumes both probabilistic grammars and BNs not only at the semantic level but also at the probability computation/learning level.

Tabling is useful in Bayesian inference as well. In [43] we introduced the VB (variational Bayes) approach to PRISM and implemented the VB-EM algorithm that learns hyper parameters of Dirichlet priors associated with `msw` atoms, in a dynamic programming manner using explanation graphs and the slightly extended g-IO algorithm. Hyper parameter learning is done with the same time complexity as usual parameter learning because both types of learning use the same explanation graphs and isomorphic learning algorithms. We test the left-corner parsing model and the profile-HMM model. Although there is no report on their hyper parameter learning to our knowledge, all we need to do is to write a declarative PRISM program for each model, and the rest of the task

[7] Our tabling is linear-tabling [26] which does not use a suspend-resume mechanism for tabled execution of logic programs but iteratively computes answers until they saturate.

[8] The distribution of the PRISM system includes an example of logical junction tree. Querying the tree with `chindsight_agg/2` is equivalent to running BP.

- hyper parameter learning followed by Viterbi inference based on the learned hyper parameters - is carried out automatically by the PRISM system.

3.3 Log-linear models and logic program synthesis

The last topic is non-generative modeling. Generative modeling, typically PCFGs to us, assumes no failure in the process of generating an outcome. However logic programs may fail as we all know. The problem caused by failure to logic-based probabilistic modeling such as SLPs (stochastic logic programs) [3, 44] and PRISM [45] is loss of probability mass. If the execution eventually fails after a probabilistic choice is made, the probability mass put on the choice is lost. As a result the total sum of probabilities for possible generation processes will be less than unity, implying that our probability is not mathematically correct.

Suppose there is a PRISM program DB about `q(X)` which defines a distribution $P_{DB}(\cdot)$. Let $\mathtt{t}_1, \ldots, \mathtt{t}_N$ be all answer substitutions for the query `?-q(X)`. If failure computation occurs during the search for all answers for `?-q(X)` and $Z = P_{DB}(\exists \mathtt{X}\,\mathtt{q(X)}) = \sum_{i=1}^{N} P_{DB}(\mathtt{q(t}_i\mathtt{)}) < 1$ happens, we consider a normalized distribution $Z^{-1}P_{DB}(\mathtt{q(X)})$ over $\{\mathtt{q(t}_1\mathtt{)}, \ldots, \mathtt{q(t}_N\mathtt{)}\}$ to recover probability.

However $Z^{-1}P_{DB}(\mathtt{q(X)})$ is a log-linear model[9]and parameter learning of log-linear models is known to be much harder than BNs and PCFGs due to the computation of Z, a normalizing constant. Cussens proposed the FAM (failure-adjusted maximization) algorithm for parameter learning of SLPs whose computation may fail and hence defines log-linear models [44]. It is an EM algorithm but requires the computation of “failure probability” $1 - Z$ (Z is the probability of success computation).

We incorporated the FAM algorithm into PRISM by applying a logic program synthesis technique to PRISM programs to derive special programs called *failure programs* to compute failure probabilities $1 - Z$. Given a program DB for the target goal `q(X)` which has failed computation paths, we consider another goal $\mathtt{failure} \Leftrightarrow \forall \mathtt{X}(\mathtt{q(X)} \Rightarrow \mathtt{false})$ and synthesize a failure program for this `failure` predicate so that `?-failure` faithfully traces every failed computation path for `?-q(X)` in the original program DB. Under a certain condition[10], it can be proved $P_{DB}(\mathtt{failure}) = 1 - P_{DB}(\mathtt{q(X)}) = 1 - Z$ [46]. The point here is not that we can compute $1 - Z$ exactly but that we are now able to compute it using DP by applying tabled search to the synthesized failure program. In [46], an example of HMMs with constraints which may fail is presented. The synthesized failure program runs by tabled execution in time linear in the length of input for the original HMM program.

[9] Log-linear models take the form $\log p(x) = \sum_i w_i f_i(x)$ where $f_i(x)$ is a real-valued function called feature and w_i is a real number called weight. In the case of SLPs, $f_i(x)$ is the number of occurrences of an i-th clause in a refutation x.

[10] Roughly every computation path for `q(X)` must terminate with finite failure or success.

The program synthesis for failure programs is done by FOC (first-order compiler) [27]. It is an unfold/fold program transformation system for logic programs with universally quantified implicational goals $\forall y(p(x,y) \Rightarrow q(y,z))$[11] in the clause body. FOC transforms the original PRISM program while considering the probabilistic semantics of `msw` atoms into a PRISM program with disequality constraints.

```
failure :- not(success).  |   failure:-closure_success0(f0).
success :- agree(_).      |   closure_success0(A):-closure_agree0(A).
                          |
agree(A):-                |   closure_agree0(_):-
  msw(coin(a),A),         |     msw(coin(a),A),
  msw(coin(b),B),         |     msw(coin(b),B),
  A=B.                    |      \+A=B.
```

Fig. 3. Agreement program (left) and the synthesized failure program (right)

The program in Fig. 3 models probabilistic singular/plural agreement between nouns and verbs in some hypothetical language. `coin(a)` determines the singularity/plurality of a noun with probability 0.4/0.6 respectively and so does `coin(b)` for a verb. If they do not agree, the sentence generation fails. `failure` predicate on the left hand side is defined as the negation of $\exists$`A agree(A)` (success of `agree(_)`). FOC compiles it into the failure program on the right hand side by removing negation while introducing new predicates `closure_success/1` and `closure_agree0/1` (see [27] for details). As you can see, the compiled program correctly computes failure probability.

4 Concluding remarks

We have examined PRISM, an extension of Prolog with `msw/2` predicate for probabilistic choice, the distribution semantics, tabled search and generic routines for probability computation and parameter learning. We have been developing PRISM for more than a decade, to achieve generality and efficiency for probabilistic modeling, but there remains a long way to go. The future work includes an implementation of Gaussian distributions, also that of log-linear models, and removing some modeling condition (the exclusiveness condition [5]) by the introduction of BDDs.

References

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

[11] Negation $\neg p(x,y) = (p(x,y) \Rightarrow$ `false`$)$ is a special case.

2. Sato, T.: A statistical learning method for logic programs with distribution semantics. In: Proceedings of the 12th International Conference on Logic Programming (ICLP'95). (1995) 715–729
3. Muggleton, S.: Stochastic logic programs. In De Raedt, L., ed.: Advances in Inductive Logic Programming. IOS Press (1996) 254–264
4. Poole, D.: The independent choice logic for modeling multiple agents under uncertainty. Artificial Intelligence **94**(1-2) (1997) 7–56
5. Sato, T., Kameya, Y.: Parameter learning of logic programs for symbolic-statistical modeling. Journal of Artificial Intelligence Research **15** (2001) 391–454
6. Kersting, K., De Raedt, L.: Basic principles of learning bayesian logic programs. Technical Report Technical Report No. 174, Institute for Computer Science, University of Freiburg (2002)
7. Blockeel, H.: Prolog for Bayesian networks: a meta-interpreter approach. In: Proceedings of the 2nd International Workshop on Multi-Relational Data Mining (MRDM'03). (2003) 1–13
8. Vennekens, J., Verbaeten, S., Bruynooghe.M.: Logic programs with annotated disjunctions. In: Proceedings of the 20th International Conference on Logic Programming (ICLP'04). Lecture Notes in Computer Science 3132 (2004) 431–445
9. Fierens, D., Blockeel, H., Bruynooghe, M., Ramon, J.: Logical Bayesian networks and their relation to other probabilistic logical models. In: Proceedings of the 15th International Conference on Inductive Logic Programming (ILP'05), volume 3625 of Lecture Notes in Computer Science. (2005) 121–135
10. De Raedt, L., Kimmig, A., Toivonen, H.: ProbLog: A probabilistic Prolog and its application in link discoverry. In: Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI'07). (2007) 2468–2473
11. Baral, C., Gelfond, M., Rushton, N.: Probabilistic reasoning with answer sets. Theory and Practice of Logic Programming (TPLP) **9**(1) (2009) 57–144
12. De Raedt, L., Kersting, K.: Probabilistic inductive logic programming. In De Raedt, L., Frasconi, P., Kersting, K., Muggleton, S., eds.: Probabilistic Inductive Logic Programming - Theory and Applications. Lecture Notes in Computer Science. Springer (2008) 1–27
13. Breese, J.S.: Construction of belief and decision networks. Computational Intelligence **8**(4) (1992) 624–647
14. Koller, D., Pfeffer, A.: Learning probabilities for noisy first-order rules. In: Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI'97). (1997) 1316–1321
15. Friedman, N., Getoor, L., Koller, D., Pfeffer, A.: Learning probabilistic relational models. In: Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI'99). (1999) 1300–1309
16. Pfeffer, A.: IBAL: A probabilistic rational programming language. In: Proceedings of the 17th International Conference on Artificial Intelligence (IJCAI'01). (2001) 733–740
17. Jaeger, J.: Complex probabilistic modeling with recursive relational Bayesian networks. Annals of Mathematics and Artificial Intelligence **32**(1-4) (2001) 179–220
18. Getoor, L., Friedman, N., Koller, D., Taskar, B.: Learning Probabilistic Models of Relational Structure. Journal of Machine Learning Research **3** (2002) 679–707
19. Costa, V., Page, D., Qazi, M., Cussens, J.: CLP(BN): Constraint logic programming for probabilistic knowledge. In: Proceedings of the 19th Conference on Uncertainty in Artificial Intelligence (UAI'03). (2003) 517–524

20. Milch, B., Marthi, B., Russell, S., Sontag, D., Ong, D., Kolobov, A.: BLOG: Probabilistic models with unknown objects. In: Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI'05). (2005) 1352–1359
21. Laskey, K.: MEBN: A logic for open-world probabilistic reasoning. C4I Center Technical Report C4I06-01, George Mason University Department of Systems Engineering and Operations Research (2006)
22. Richardson, M., Domingos, P.: Markov logic networks. Machine Learning **62** (2006) 107–136
23. Getoor, L., Grant, J.: PRL: A probabilistic relational language. Journal of Machine Learning **62**(1-2) (2006) 7–31
24. Getoor, L., Taskar, B., eds.: Introduction to Statistical Relational Learning. MIT Press, Cambridge, MA (2007)
25. Sato, T., Kameya, Y.: Statistical abduction with tabulation. In Kakas, A., Sadri, F., eds.: Computational Logic: Logic Programming and Beyond. LNAI 2408, Springer (2002) 567–587
26. Zhou, N.F., Sato, T., Shen, Y.D.: Linear tabling strategies and optimization. Theory and Practice of Logic Programming **8**(1) (2008) 81–109
27. Sato, T.: First Order Compiler: A deterministic logic program synthesis algorithm. Journal of Symbolic Computation **8** (1989) 605–627
28. Fenstad, J.E.: Representation of probabilities defined on first order languages. In Crossley, J.N., ed.: Sets, Models and Recursion Theory. North-Holland (1967) 156–172
29. Milch, B., Marthi, B., Sontag, D., Russell, S., Ong, D., Kolobov, A.: Approximate Inference for Infinite Contingent Bayesian Networks. In: Proceedings of the 10th International Workshop on Artificial Intelligence and Statistics (AISTATS'05). (2005) 1352–1359
30. Domingos, P., Singla, P.: Markov logic in infinite domains. In De Raedt, L., Dietterich, T., Getoor, L., Kersting, K., Muggleton, S., eds.: Probabilistic, Logical and Relational Learning - A Further Synthesis. Number 07161 in Dagstuhl Seminar Proceedings (2008)
31. Stolcke, A.: An efficient probabilistic context-free parsing algorithm that computes prefix probabilities. Computational Linguistics **21**(2) (1995) 165–201
32. Gelfond, M., Lifshcitz, V.: The stable model semantics for logic programming. (1988) 1070–1080
33. Van Gelder, A., Ross, K., Schlipf, J.: The well-founded semantics for general logic programs. The journal of ACM (JACM) **38(3)** (1991) 620–650
34. Tamaki, H., Sato, T.: OLD resolution with tabulation. In: Proceedings of the 3rd International Conference on Logic Programming (ICLP'86). Volume 225 of Lecture Notes in Computer Science., Springer (1986) 84–98
35. Sagonas, K., Swift, T., Warren, D.: XSB as an efficient deductive database engine. In: Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data. (1994) 442–453
36. Ramakrishnan, I., Rao, P., Sagonas, K., Swift, T., Warren, D.: Efficient tabling mechanisms for logic programs. In: Proceedings of the 12th International Conference on Logic Programming (ICLP'95), The MIT Press (1995) 687–711
37. Guo, H.F., Gupta, G.: A simple scheme for implementing tabled logic programming systems based on dynamic reordering of alternatives. In: Proceedings of the 17th International Conference on Logic Programming, London, UK, Springer-Verlag (2001) 181–196

38. Sagonas, K., Stuckey, J.: Just enough tabling. In: Proceedings of the 6th ACM SIGPLAN international conference on Principles and practice of declarative programming (PPDP '04), New York, NY, USA, ACM 78–89
39. Rabiner, L.R., Juang, B.: Foundations of Speech Recognition. Prentice-Hall (1993)
40. Baker, J.K.: Trainable grammars for speech recognition. In: Proceedings of Spring Conference of the Acoustical Society of America. (1979) 547–550
41. Pearl, J.: Probabilistic Reasoning in Intelligent Systems. Morgan Kaufmann (1988)
42. Sato, T.: Inside-Outside probability computation for belief propagation. In: Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI'07). (2007) 2605–2610
43. Sato, T., Kameya, Y., Kurihara, K.: Variational bayes via propositionalized probability computation in prism. Annals of Mathematics and Artificial Intelligence, to appear.
44. Cussens, J.: Parameter estimation in stochastic logic programs. Machine Learning **44**(3) (Sept. 2001) 245–271
45. Sato, T., Kameya, Y.: PRISM: a language for symbolic-statistical modeling. In: Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI'97). (1997) 1330–1335
46. Sato, T., Kameya, Y., Zhou, N.F.: Generative modeling with failure in PRISM. In: Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI'05). (2005) 847–852