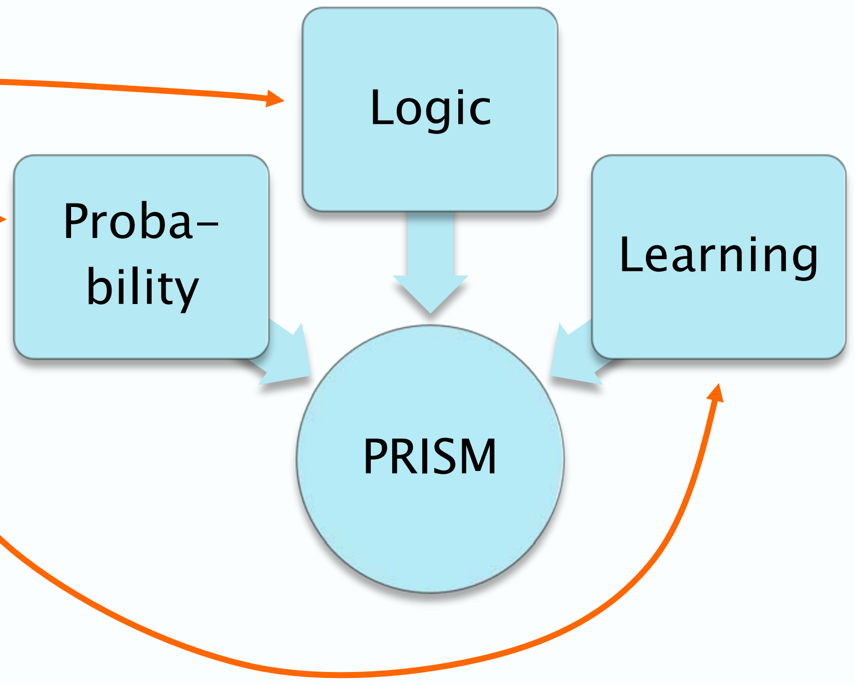
Generative Modeling by PRISM

Taisuke Sato
joint work with
Yoshitaka Kameya, Neng-Fa Zhou

Outline

- PRISM: an overview
- LP connections
  - Semantics
  - Tabling
  - Program synthesis
- ML example



Probabilistic modeling

- Major framework in machine learning
  - clustering, classification, prediction, smoothing,…
    in bioinformatics, speech/pattern recognition, text processing, robotics, Web analysis, marketing,…
- Define $p(x,y|\theta)$, $p(x|y,\theta)$ (x:hidden cause, y:observed effect, θ:parameters)
  - by graphs (Bayesian networks, Markov random fields, conditional random fields,…)
  - by rules (hidden Markov models, probabilistic context free grammars,…)
- Basic tasks:
  - probability computation (NP-hard)
  - learning parameter/structure

PLL/SRL approaches

- Graphical models for probabilistic modeling
  - Intuitive and popular but only numbers, no structured data, no variable, no relation ➔ complex modeling difficult
- More expressive formalisms (90's~)
  - PLL (probabilistic logic learning)
    - {ILP, MRDM}+probability, probabilistic abduction
  - SRL (statistical relational learning)
    - {BNs, MRFs} + relations
- Many proposals (alphabet soup)
  - Generative: $p(x,y|\theta)$, hidden x generates observation y
  - Discriminative : $p(x|y,\theta)$

Generative modeling

- Defines a generation process of an output in a sample space
  - Bayesian approach such as LDA
    - prior distribution p(θ|α) ➔ distribution p(D|θ) ➔ data D
    - Given D, predict x by

$$p(\theta \mid D) \propto \int_\theta p(\theta \mid \alpha) p(D \mid \theta) d\theta, \; p(x \mid D) = \int_\theta p(x \mid \theta) p(\theta \mid D) d\theta$$

  - Probabilistic grammars such as PCFGs
    - Rules are chosen probabilistically in the derivation
    - Prob. of sentence s :

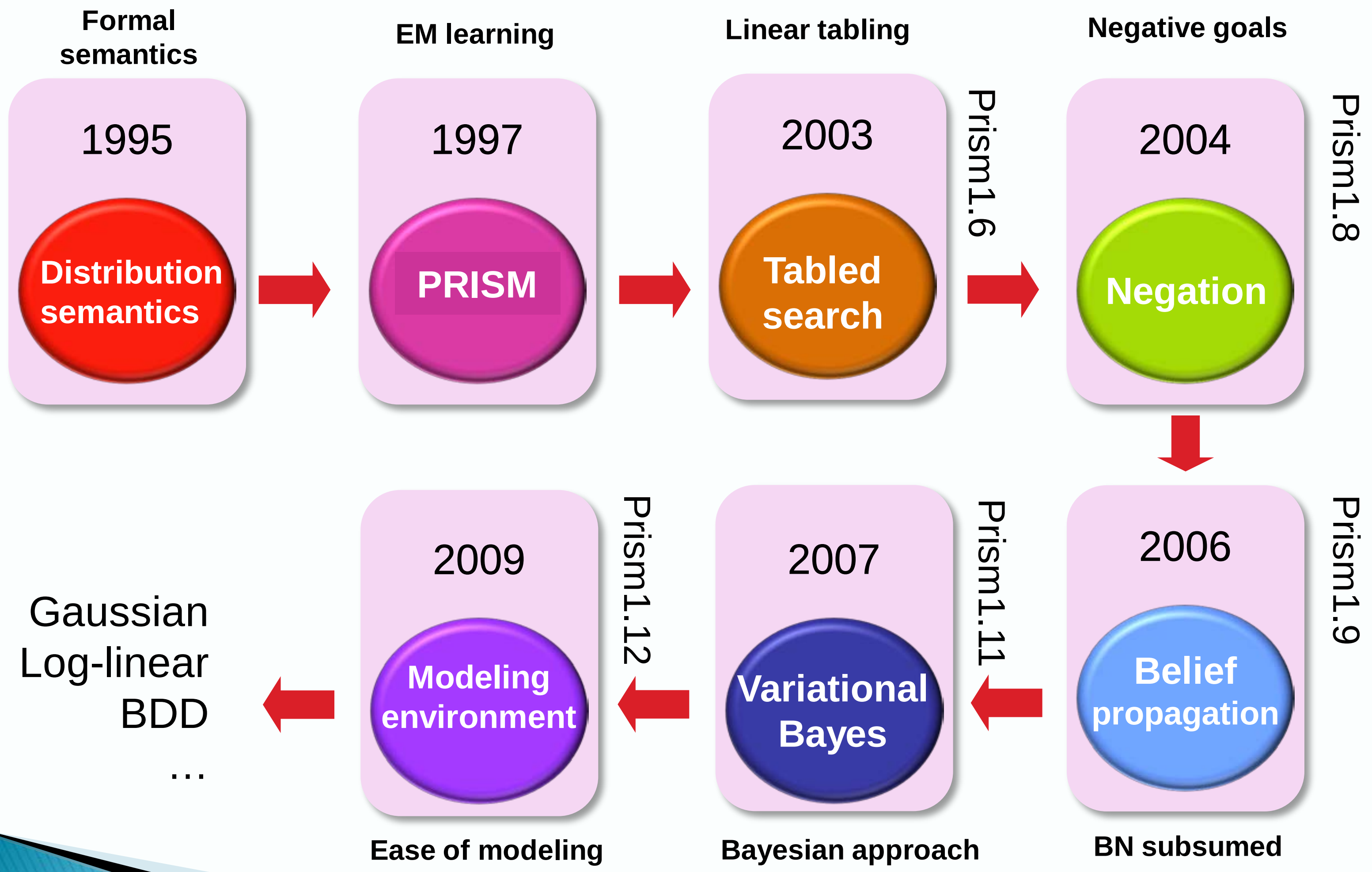
$$p(s) = \sum_{\tau \models s} \underbrace{\prod_{r \in \tau} p(r)^{\phi(r,\tau)}}_{p(\tau)}$$

- Defining distributions by (logic) programs (in PLL)
  - PHA[Poole'93], PRISM[Sato et al.'95,97], SLPs[Muggleton'96, Cussens'01], P-log[Baral et al.'04], LPAD[Vennekens et al.'04], ProbLog[De Raedt et al.'07]…

PRISM for generative modeling

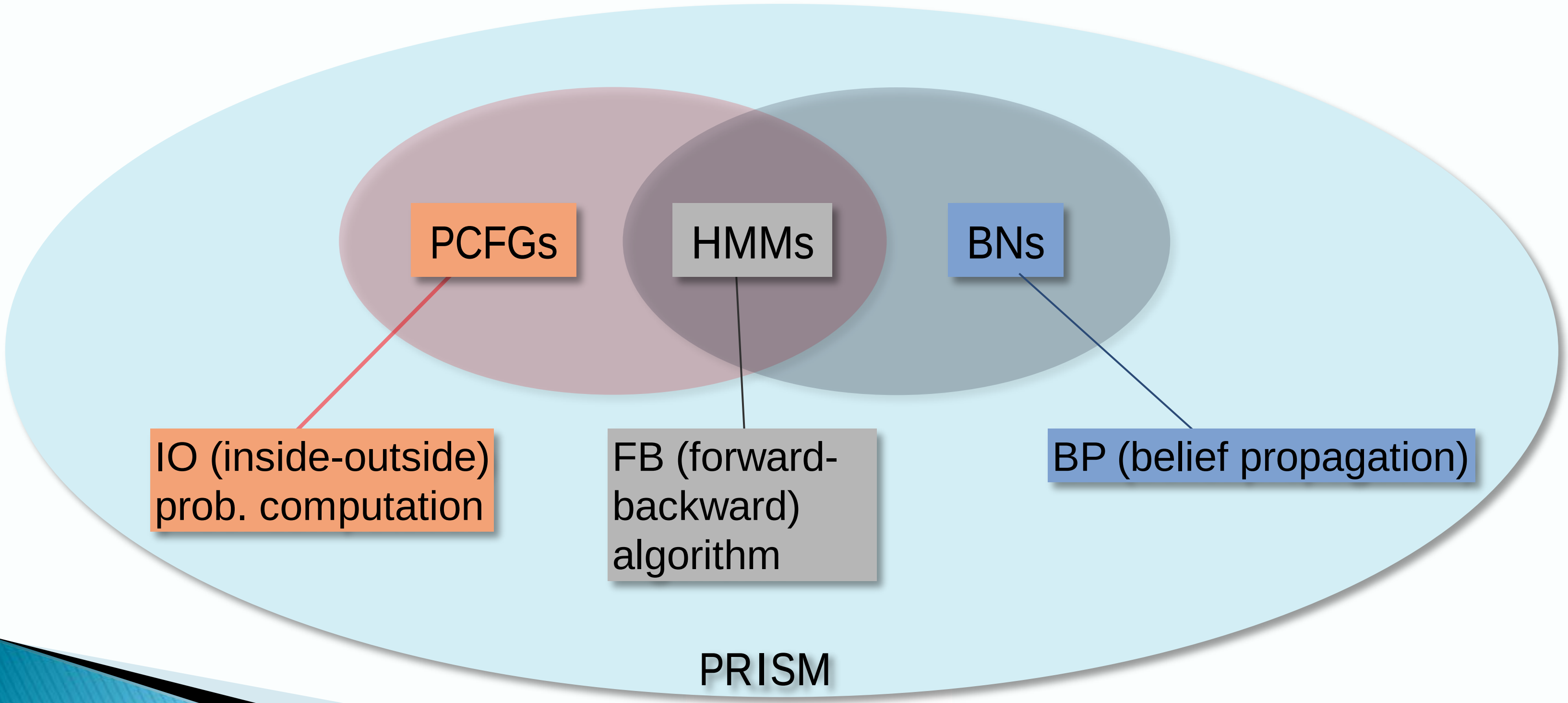
- Prolog's probabilistic extension
  - Turing machine with statistically learnable state transitions
- Syntax: Prolog + msw/2 (random choice)
  - Variables, terms, predicates, etc available for p.-modeling
- Semantics: distribution semantics
  - Program DB defines a probability measure $P_{DB}(\ )$ on least Herbrand models
- Pragmatics:(very) high level modeling language
  - Just describe probabilistic models declaratively
- Implementation:
  - B-Prolog (tabled search) + parameter learning (EM,VB-EM)
  - Single data structure : expl. graphs, dynamic programming

History of development

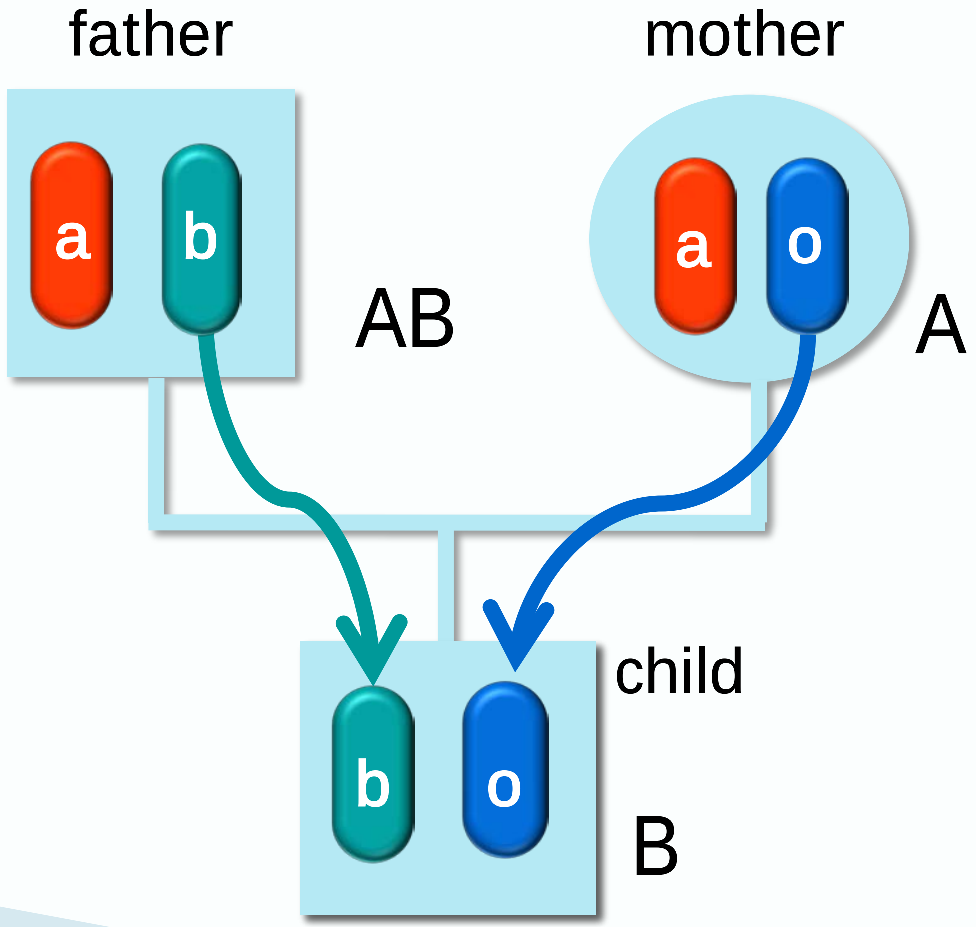


Unified treatment

- PRISM subsumes three representative generative models, PCFGs, HMMs and BNs (and their Bayesian version). They are uniformly computed/learned by a generic algorithm


PCFGs
HMMs
BNs
IO (inside-outside) prob. computation
FB (forward-backward) algorithm
BP (belief propagation)
PRISM

ABO blood type inheritance



Program DB

```
btype(X):- gtype(Gf,Gm), pg_table(X,[Gf,Gm]).
pg_table(X,Gtype):-
   ((X=a;X=b),(GT=[X,o];GT=[o,X];GT=[X,X])
   ; X=o,GT=[o,o]
   ; X=ab,(GT=[a,b];GT=[b,a])).
gtype(Gf,Gm):- msw(abo,Gf),msw(abo,Gm).
```

(probabilistic switch)

$P_{msw}(msw(abo,a)=1) = \boxed{\theta_{(abo,a)} = 0.3,}$ ... (parameter)

➔ $P_{DB}(msw(abo,a)=x_1, msw(abo,b)=x_2, msw(abo,o)=x_3,$
$btype(a)=y_1, btype(b)=y_2, btype(ab)=y_3, btype(o)=y_4)$

➔ $P_{DB}(btype(a)=1) = 0.4$ (parameter learning is inverse direction)

Three topics

- Distribution semantics
- Tabling
- Program synthesis

Representation theorem [Fenstat'67]

Let

L : countable 1st order language

α : (possibly open) formulas in L

$p(\alpha)$: probability given to α s.t.

$$\begin{cases} p(\alpha) = p(\beta), \text{ if } \vdash \alpha \Leftrightarrow \beta \\ p(\alpha) = 1, \text{ if } \vdash \alpha \end{cases}$$

Then

$$p(\alpha) = \int_S p_M(\alpha(M))\, d\lambda(M)$$

where

S : set of H-interpretations M for L'

L' : L with "special constants"

λ : σ-additive prob. measure on S

p_M : prob. measure on $\{\alpha(M) \mid \alpha \text{ in } L\}$

$\alpha(M)$: $\{a \in [I \mapsto \mathrm{Dom}_M] \mid M \models_a \alpha\}$

I: index of variables ($=$ nat. nums)

- Possible world semantics: For a closed α, $p(\alpha)$ is the sum of probabilities of possible worlds M that makes α true
  - $p_M(\alpha(M)) = 1$ if $M \models \alpha$
    $= 0$ o.w.
- When α has a free variable x, $p_M(\alpha(M))$ is the ratio of individuals in M satisfying α

Distribution semantics [Sato'95]

- DB = F U R
  - F : set of ground msw/2 atoms
    = { msw(abo,a),msw(abo,o),… }
  - R : set of definite clauses, msw/2 allowed only in the body
    = {btype(X):- gtype(Gf,Gm), pg_table(X,[Gf,Gm]) … }
  - P_F( ) : infinite product of some finite distributions on msws
- We extend P_F( ) to P_{DB}( ), probability measure over H-interpretations for DB using the least model semantics and Kolmogorov's extension theorem
  - F' ~ P_F : ground msw atoms sampled from P_F( )
  - M(R U F') : the least H-model for R U F' always exists
    ➔ (infinite) random vector taking H-interpretations
  - P_{DB}( ) : prob. measure over such H-interpretations induced by M(R U F')

Propositional example

- $DB = \{ \overbrace{a \text{ :- } b,\ a \text{ :- } c}^{R},\ \overbrace{b,\ c}^{F} \}$ $P_F(b,c)$ given

Sample (b,c) $\sim P_F(.,.)$ b	c	Sampled DB'	Herbrand model	a	$P_{DB}(a,b,c)$
0 (false)	0	a:−b, a:−c	{}	0	$= P_F(0,0)$
0	1 (true)	a:−b, a:−c c	{c,a}	1	$= P_F(0,1)$
1	0	a:−b, a:−c b	{b,a}	1	$= P_F(1,0)$
1	1	a:−b, a:−c b, c	{b,c,a}	1	$= P_F(1,1)$
	anything	else			$= 0$

Unique features

- Unconditionally definable
  - Arbitrary definite program allowed (even a:- a)
  - No syntactic restriction (such as acyclic, range-restricted)
- Infinite domain
  - Countably many constant/function/predicate symbols
  - Infinite Herbrand universe allowed
- Infinite joint distribution (prob. measure)
  - Not a distribution on infinite ground atoms
  - Countably many i.i.d. ground atoms available
    ➔ recursion, PCFG possible
- Parameterized with LP semantics
  - Currently the least model semantics used
  - The greatest model semantics, three valued semantics,…

Three topics

- Distribution semantics
- Tabling
- Program synthesis

From semantics to probability computation

- $P_{DB}(\mathrm{iff}(DB))=1$ holds in our semantics
- We rewrite goal G by SLD to an equivalent random boolean formula $G \Leftrightarrow E_1 \vee \cdots \vee E_N$, $E_i = msw_1 \& \cdots \& msw_k$
- Assume the exclusiveness of E_is, then $P_{DB}(G) = P_{DB}(E_1)+\cdots+P_{DB}(E_N)$ and $P_{DB}(E_i) = P_{DB}(m_1) \cdots P_{DB}(m_k)$

$$\begin{aligned}\mathrm{iff}(DB) \models_H\ & \mathtt{btype(a)} \Leftrightarrow \\ & (\mathtt{msw(abo,a)} \wedge \mathtt{msw(abo,a)}) \\ & \vee(\mathtt{msw(abo,a)} \wedge \mathtt{msw(abo,o)}) \\ & \vee(\mathtt{msw(abo,o)} \wedge \mathtt{msw(abo,a)})\end{aligned}$$

⇓

$$\begin{aligned}P_{DB}(\mathtt{btype(a)}) =\ & \\ & P_{DB}(\mathtt{msw(abo,a)})P_{DB}(\mathtt{msw(abo,a)}) \\ & +P_{DB}(\mathtt{msw(abo,a)})P_{DB}(\mathtt{msw(abo,o)}) \\ & +P_{DB}(\mathtt{msw(abo,o)})P_{DB}(\mathtt{msw(abo,a)})\end{aligned}$$

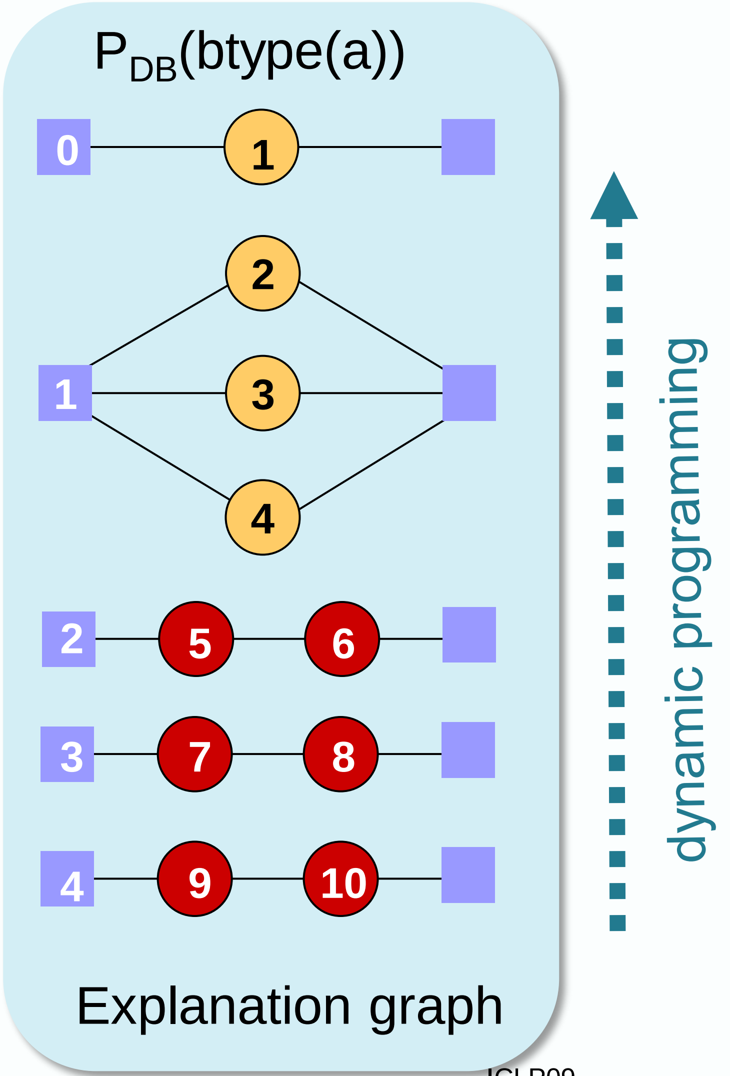
⇓

- Simple but exponential in #explanations ➔ tabling

Tabled search

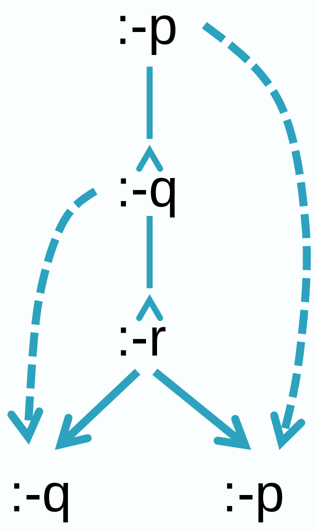
All solution search for ?- btype(a) with tabling btype/1, gtype/2 yields AND/OR boolean formulas

$$
\begin{aligned}
{}^{1}\mathtt{btype(a)} &\Leftrightarrow \mathtt{gtype(a,a)}^{2} \\
&\quad \vee\ \mathtt{gtype(a,o)}^{3} \\
&\quad \vee\ \mathtt{gtype(o,a)}^{4} \\
{}^{2}\mathtt{gtype(a,a)} &\Leftrightarrow \\
&\mathtt{msw(abo,a)}^{5} \wedge \mathtt{msw(abo,a)}^{6} \\
{}^{3}\mathtt{gtype(a,o)} &\Leftrightarrow \\
&\mathtt{msw(abo,a)}^{7} \wedge \mathtt{msw(abo,o)}^{8} \\
{}^{4}\mathtt{gtype(o,a)} &\Leftrightarrow \\
&\mathtt{msw(abo,o)}^{9} \wedge \mathtt{msw(abo,a)}^{10}
\end{aligned}
$$



Linear tabling

- PRISM uses linear tabling (Zhou et al.'08)
  - single thread (not suspend/resume scheme)
  - iteratively computes all answers by backtracking for each top-most-looping subgoal
- Looping subgoals
  - … :-A,B ➔…➔:-A',C and A, A' are variants, they are looping subgoals
  - If A has no ancestor in any loop containing A, it is the top-most goal



SLD tree

Computational complexity

- Thanks to tabling, PRISM's prob. computation is as efficient as the existing model-specific algorithms

Model family	EM algorithm	Time complexity
Hidden Markov models	Baum-Welch algorithm	$O(N^2L)$ N: number of states L: max. length of sequences
Probabilistic context-free grammars	Inside-outside algorithm	$O(N^3L^3)$ N: number of non-terms L: max. length of sentences
Singly-connected Bayesian networks	EM based on π-λ computation	$O(N)$ N: number of nodes

BP (belief propagation) is an instance of PRISM's general probability computation scheme(Sato'07)

PCFG program

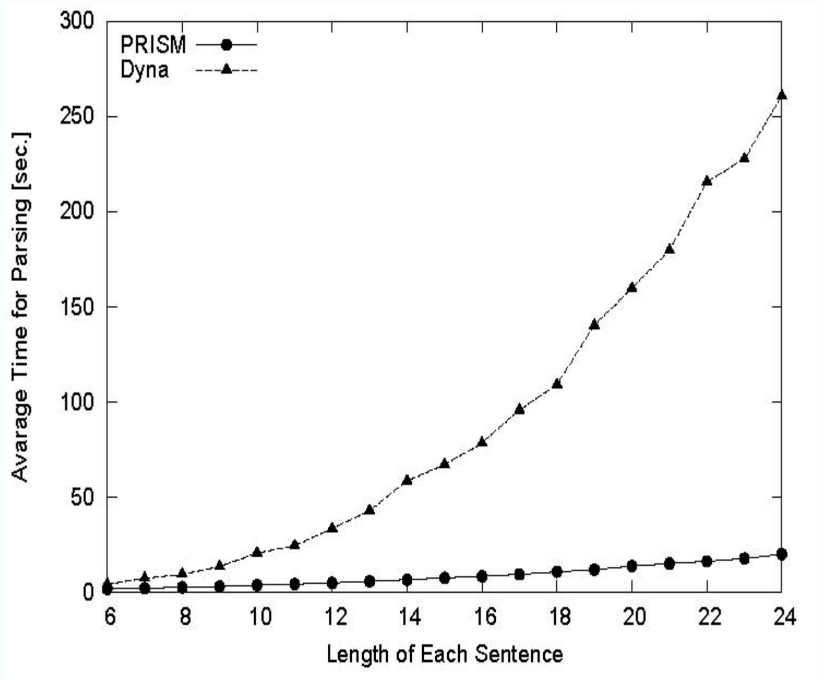
```
  S → NP VP (1.0)
NP → NP PP (0.2) |
      cars (0.1) |
      stars (0.2) |
      telescopes (0.3) |
      astronomers (0.2)
PP → P NP (1.0)
  V → see (0.5) |
      saw (0.5)
  P → in (0.3) |
      at (0.4) |
      with (0.3)
```

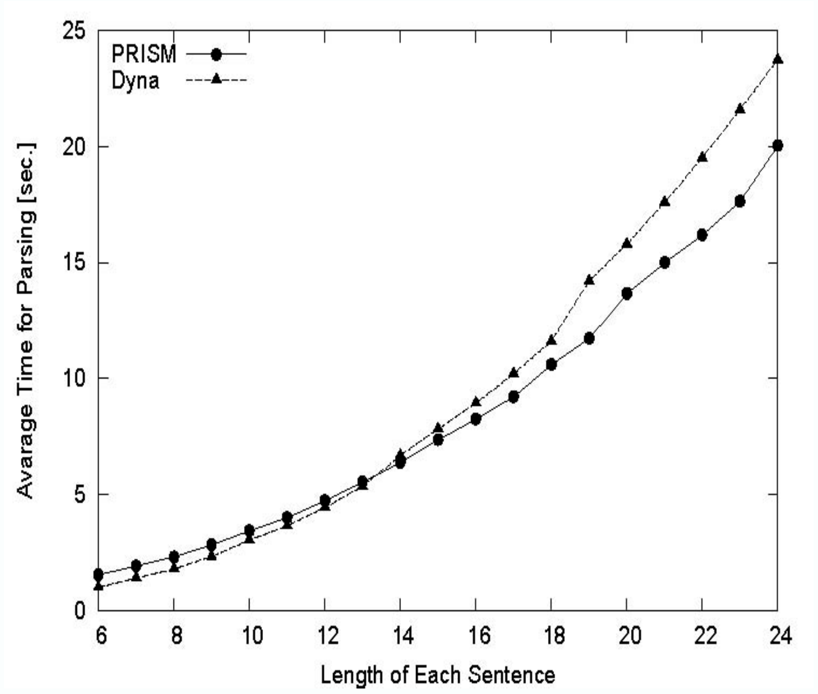
```
s(X,[]) :- np(X,Y), vp(Y,[]).
np(X,Z) :- msw(np,RHS),
   ( RHS=[np,pp], np(X,Y), pp(Y,Z)
   ; RHS=[ears], X=[ears|Z] ; … ).
pp(X,Z]) :- p(X,Y), np(Y,Z).
vp(X,Z) :- msw(np,RHS),
    ( RHS=[vp,pp], vp(X,Y), pp(Y,Z)
    ; RHS=[v,np], v(X,Y), np(Y,Z) )
v(X,Y) :- msw(v,RHS), ( RHS=[see], X=[see|Y] ;
      RHS=[saw], X=[saw|Y] ).
p(X,Y) :- msw(p,RHS),
    ( RHS=[in], X=[in|Y] ; RHS=[at], X=[at|Y]
    ; RHS=[with] & X=[with|Y] ).

values_x(np, [[np,pp],[ears],…], [0.1,0.2,…]).
values_x(v, [[see],[saw]], [0.5,0.5]).
values_x(p,[ [in],[at],[with]], [0.3,0.4,0.3]).
```

- compact
- readable

Parsing performance
-- PRISM1.10 vs. Dyna0.3.9





Parsing by 20,000 CFG rules extracted from 49,000 (POS) sentences from WSJ portion of Penn tree bank with uniform prob.
Randomly selected 20 sentences are used for the average probability computation (on the left) and Viterbi parsing (on the right)

Three topics

- Distribution semantics
- Tabling
- Program synthesis

Constraint probabilistic modeling

- Agreement of number (A=singular, plural)

```
agree(A):-
    msw(subj,A),
    msw(verb,B),
    A=B.
```

A, B randomly chosen
agree(A) succeeds only when A=B, o.w. fails

- Observable distribution is a conditional one

$$P(\text{agree}(A) \mid \exists X\ \text{agree}(X))$$
$$= P(\text{msw}(\text{subj},A))P(\text{msw}(\text{verb},A)) / P(\exists X\ \text{agree}(X))$$
$$P(\exists X\ \text{agree}(X)) = \Sigma_{A=sg,pl} P(\text{msw}(\text{subj},A))P(\text{msw}(\text{verb},A))$$

- Parameters are learnable by FAM(Cussens '01) but it requires a failure program

Failure program synthesis

- A failure program for agree/1: “failure 🡐 not(∃X agree(X))” expresses how ?- agree(X) probabilistically fails

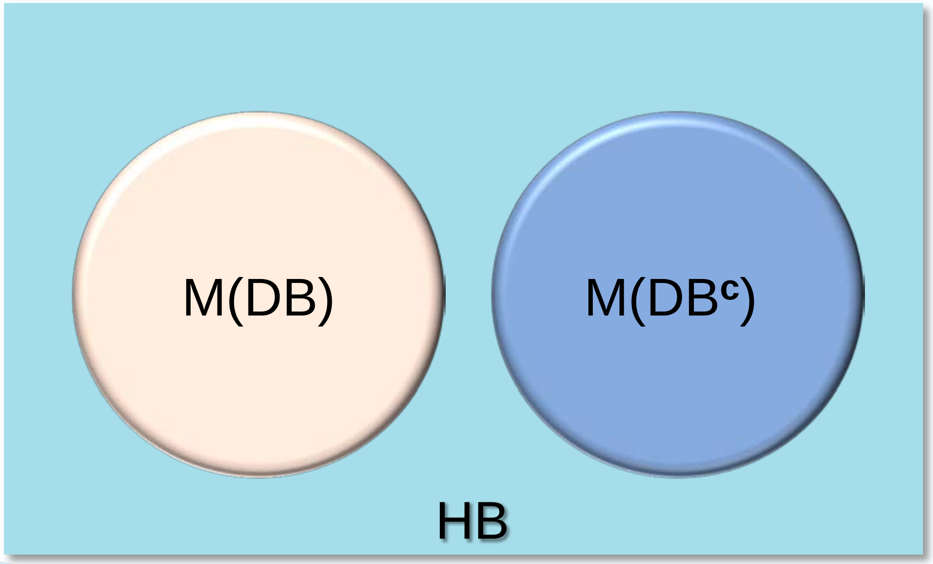
```
agree(A):-
    msw(subj,A),
    msw(verb,B),
    A=B.
```

```
failure:-
    msw(subj,A),
    msw(verb,B),
    ¥+A=B.
```

- PRISM uses FOC(first-order compiler) to automatically synthesize failure programs (negation elimination)

FOC(first-order compiler)

- FOC automatically eliminates negation from the source program using continuation (Sato ’89)
- Compiled program DB^c positively computes the finite failure set of DB



If DB^c is terminating, failure = negation and $M(DB^c)$= HB-M(DB)

Compilation example

Source program DB_{even}

```
even(0).
even(s(X)) :- not(even(X)).
```

$$even(X) \Leftrightarrow X = 0 \vee \exists Y\,(X = s(Y) \wedge \neg even(Y))$$

$$\begin{cases} even_c(X, C) \Leftrightarrow (even(X) \Rightarrow cont(C)) \\ cont(f0) \Leftrightarrow fail \end{cases}$$

$$even(s(X)) \Leftarrow even_c(X, f0)$$

$$even_c(X, C) \Leftrightarrow (X = 0 \Rightarrow cont(C)) \wedge \forall Y (X = s(Y) \Rightarrow (\neg even(Y) \Rightarrow cont(C)))$$

$$even_c(X, C) \Leftrightarrow (X = 0 \Rightarrow cont(C)) \wedge \forall Y (X = s(Y) \Rightarrow even(Y) \vee cont(C))$$

Compiled program DB^c_{even}

even(0).
even(s(A)):- $even_c$(A,f0).
$even_c$(s(A),_):- even(A).

ML example by PRISM

- Automated construction/evaluation of probabilistic classifiers
  - Modeling is part of machine learning, post- processing is as troublesome as modeling
  - PRISM 1.12 provides facilities to ease model-evaluation that make your code much shorter
- votes' dataset from UCI ML repository
- Classifier: Naive Bayes
  - Many missing values in the dataset → We use (VB)EM
  - From known vote records, we classify unknown votes as republican or democrat using 16 yes/no features
  - We perform 5-fold cross-validation

Probabilistic inferences in PRISM

- **Basic:**

  Sampling
  For a given goal G, return answer substitution s with the probability $P_q(Gs)$

  Probability computation
  For a given goal G, compute $P_q(G)$

  Viterbi computation
  For a given goal G, find the most probable explanation $E^* = \text{argmax}_{E\hat{\imath}\, y(G)} P_q(E)$
  where $y(G)$ are possible explanations for G

  Hindsight computation
  For a given goal G, compute $P_q(G')$ or $P_q(G'/G)$ where G' is a subgoal of G

  EM learning
  Given a bag $\{G_1, G_2, ..., G_T\}$ of goals, estimate the parameters q
  that maximizes the likelihood $\text{P}_t P_q(G_t)$

- **Advanced:**
  - Handling failures in the generation process (version 1.8)
  - Model selection (version 1.10)
  - Variational Bayesian learning (version 1.11)
  - Top-N Viterbi computation (version 1.11)
  - Data-parallel EM learning (version 1.11)
  - Deterministic annealing EM algorithm (version 1.11)

Vote data-learning

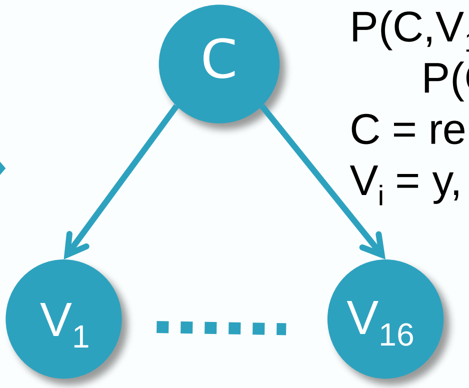
predict

16 features (vote record)

```
republican, n,y,n,y,y,y,n,n,n,y,?,y,y,y,n,y
republican, n,y,n,y,y,y,n,n,n,n,n,y,y,y,n,?
democrat,   ?,y,y,?,y,y,n,n,n,n,y,n,y,y,n,n
democrat,   n,y,y,n,?,y,n,n,n,n,y,n,y,n,n,y
democrat    y,y,y,n,y,y,n,n,n,n,y,?,y,y,y,y
democrat,   n,y,y,n,y,y,n,n,n,n,n,n,y,y,y,y
democrat,   n,y,n,y,y,y,n,n,n,n,n,n,?,y,y,y
republican, n,y,n,y,y,y,n,n,n,n,n,n,y,y,?,y
    …                    …
republican, n,y,n,y,y,y,n,n,n,y,n,y,y,y,?,n
```

435 data

Naïve Bayes



$P(C,V_1,\ldots,V_{16}) =$
$P(C)P(V_1|C)..P(V_{16}|C)$
C = republican, democrat
V_i = y, n

Learn $P(V_1|C),..,P(V_{16}|C)$ from data

Predict C for unknown $V_1,\ldots,V_{16}$ by
$C = \text{argmax}\_c\ P(C|V_1,\ldots,V_{16})$

Estimate precision by cross-validation

Vote program

modeling part

```
values(class,[democrat,republican]).              % class labels
values(attr(_,_),[y,n]).  % all attributes have two values: y or n

nbayes(C,Vals):- msw(class,C),nbayes(1,C,Vals).
nbayes(_,_,[]):- !.
nbayes(J,C,[V|Vals]):-
    choose(J,C,V),
    J1 is J+1,!,  % cut is ok
    nbayes(J1,C,Vals).

choose(J,C,V):-
    ( nonvar(V) -> msw(attr(J,C),V)
    ; msw(attr(J,C),_) ).

%%%%% Utilities
vote_learn:-  load_data_file(Gs),  learn(Gs).

%% Batch routine for N-fold cross validation
vote_cv(N):-
    random_set_seed(81729),
    load_data_file(Gs0),    % Load the entire data
    random_shuffle(Gs0,Gs), % Randomly reorder the data
    numlist(1,N,Js),        % Get Js = [1,...,N] (B-Prolog built-in)
    maplist(J,Rate,vote_cv(Gs,J,N,Rate),Js,Rates),
    avglist(Rates,AvgRate),    % Get the avg. of the precisions
    maplist(J,Rate,format("Test #~d: ~2f%~n",[J,Rate*100]),
       Js,Rates),
    format("Average: ~2f%~n",[AvgRate*100]).
```

```
%% Subroutine for learning and testing for J-th split data (J = 1...N)
vote_cv(Gs,J,N,Rate):-
    format("<<<< Test #~d >>>>~n",[J]),
    separate_data(Gs,J,N,Gs0,Gs1),
    learn(Gs0),
    maplist(nbayes(C,Vs),R,(viterbig(nbayes(C0,Vs)),(C0==C->R=1;R=0)),Gs1,Rs),
avglist(Rs,Rate),
    format("Done (~2f%).~n~n",[Rate*100]).

separate_data(Data,J,N,Learn,Test):-
    length(Data,L),
    L0 is L*(J-1)//N,      % L0: offset of the test data (// - integer division)
    L1 is L*(J-0)//N-L0,   % L1: size of the test data
    splitlist(Learn0,Rest,Data,L0),   % Length of Learn0 = L0
    splitlist(Test,Learn1,Rest,L1),   % Length of Test = L1
    append(Learn0,Learn1,Learn).

load_data_file(Gs):-
    load_csv('UCI/house-votes-84.data',Gs0,[missing('?')]),
          % '?' in the data will be converted into an anonymous variable (_)
    maplist(csvrow([C|Vs]),nbayes(C,Vs),true,Gs0,Gs).
```

Let PRISM automatically estimate the precision of the model by cross-validation and paste it in the submitted paper!

utility part

Conclusion

- Logic and probability have been cross-fertilizing each other, in particular in PLL/SRL
- Their integration can make a powerful probabilistic modeling language with rigorous semantics
- In PRISM
  - the user encodes a probabilistic model as a program DB at predicate level using variables and relations
  - DB uniquely defines a prob. measure
  - The remaining tasks (prob. computation, parameter learning etc) are automatically carried out by the PRISM system