An introduction to PRISM

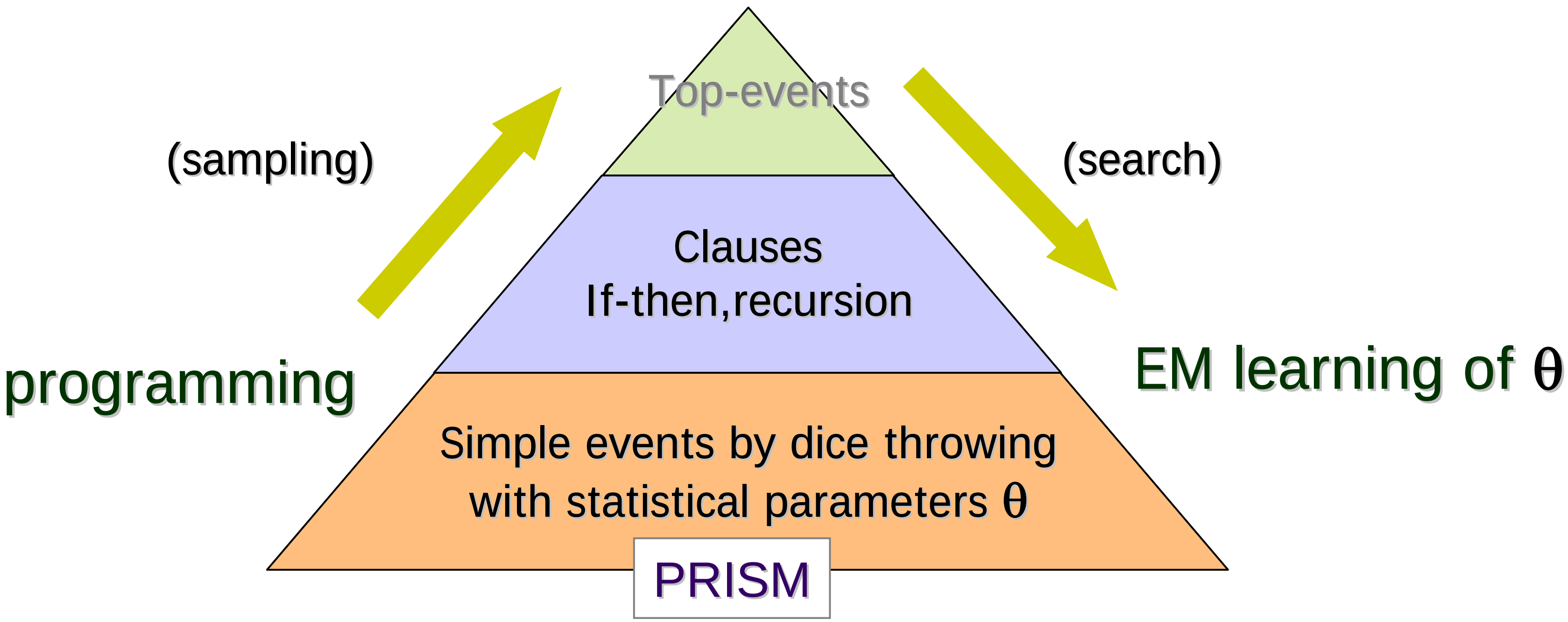
- What is PRISM?
  - acronym of PRogramming In Statistical Modeling
  - programming language for symbolic-statistical modeling
  - downloadable at http://mi.cs.titech.ac.jp/prism/
- Modeling targets
  - complex phenomena governed by rules and probabilities
  - gene-inheritance, stochastic NLP, consumer-behavior,...
- Features
  - programs as statistical models
  - probabilities and most likely paths computed
  - parameter learning by the EM algorithm
- See [Sato '01] for theoretical background

Development of PRISM

1995	Distribution semantics	First probabilistic semantics for (logic) programs as statistical models	
1997	PRISM	First programming language with EM learning ability	
2000	graphical EM	Fast EM algorithm based on AND-OR graphs	PRISM1.4
2003	Tabled search	Fast all solution search by linear tabling	PRISM1.6
2004	Negation	First parameter learning from negative goals	PRISM1.8

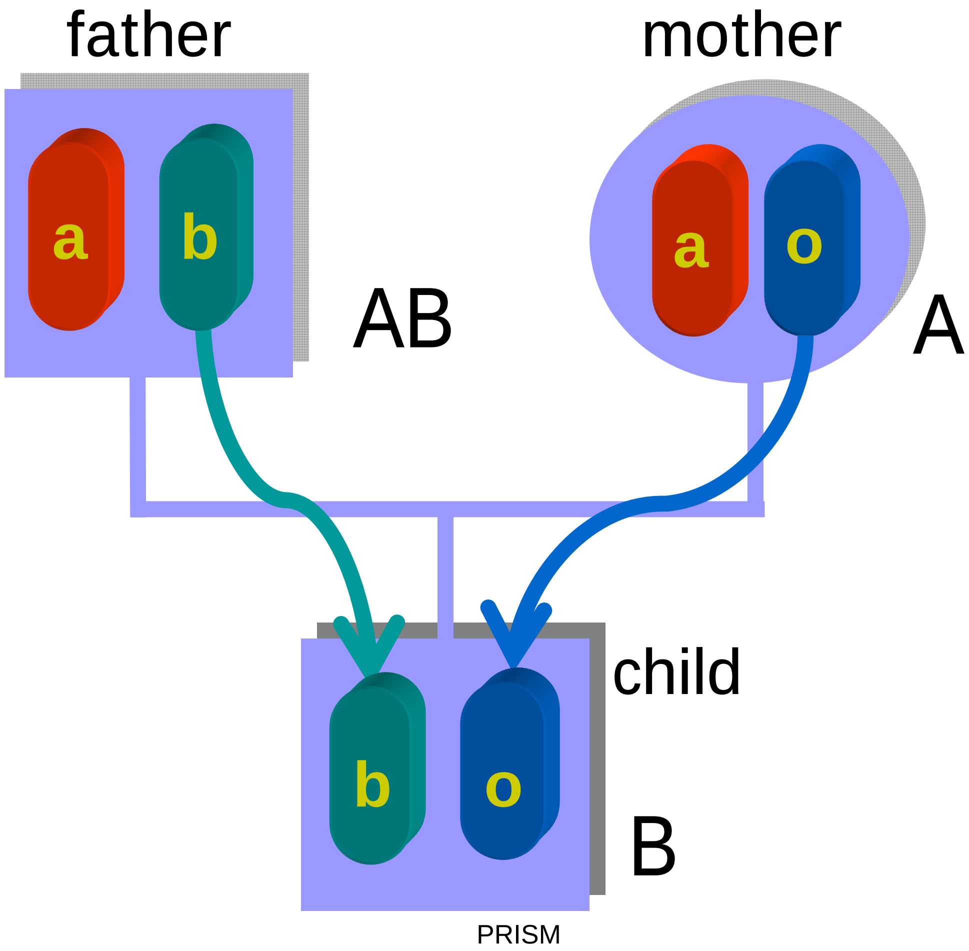
Basic idea

- PRISM = Prolog + probability + parameter learning



An example of PRISM modeling

Gene inheritance



Program DB = rules + distribution P_F over msws

```
btype(X):- pg_table(X,[Gf,Gm]),gtype(Gf,Gm).
pg_table(X,GT):-
 ((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),mws(abo,Gm).
```

(basic random switch)

$$P_F(\texttt{msw(abo, a)}) = \boxed{\theta_{(\texttt{abo,a})} = 0.3}$$ (parameter)

$$\cdots$$

$$\Rightarrow P_{DB}(\texttt{btype(a)}) = \boxed{0.4}$$ (computed prob.)

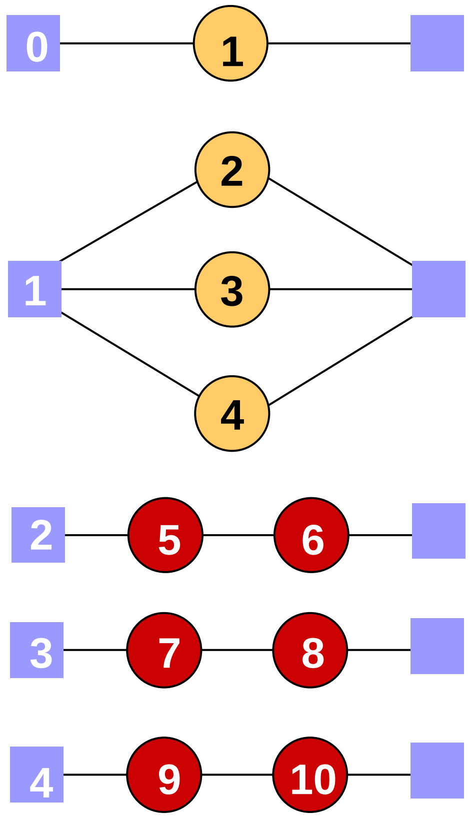
Observation ➡ Explanation search ➡ Prob. computation

observation (top-event)

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

explanation

Tabled search



PRISM Explanation graph

$$\boxed{P_{DB}(\mathtt{btype(a)})}$$

Inside prob.

Dynamic programming

Three modes of execution

Prob. computation:

```
| ?- prob(btype(a)).
The probability of btype(a) is: 0.4
```

Search for explanation graph:

```
| ?- probf(btype(a)).
  btype(a)    <=> gtype(a,a) v gtype(a,o) v gtype(o,a)
  gtype(a,a)  <=> msw(gene,a) & msw(gene,a)
  gtype(a,o)  <=> msw(gene,a) & msw(gene,o)
  gtype(o,a)  <=> msw(gene,o) & msw(gene,a)
```

Sampling:

```
| ?- sample(btype(X)).
X = a?
```

Learning parameters

```
| ?- Gs =[btype(a),btype(o),btype(ab),btype(o),...],learn(Gs).
.
Finished learning
        Number of iterations: 5.
        Final likelihood:-12.800480
        Total learning time: 0.0 seconds.
        All solution search time: 0.0 seconds.

| ?- show_sw(abo).
Switch abo:  a (0.292380)  b (0.163031)  o (0.544588)
```

observed data

parameter learning by the EM algorithm

learned parameters

Statistical modeling

500 samples

```
btype(a) :195
btype(b) : 97
btype(o) :159
btype(ab): 49
```

EM learning

Learned parameters

```
prob(msw(abo,a))=0.283
prob(msw(abo,b))=0.158
prob(msw(abo,o))=0.558
```

Calculation by the model

```
prob(btype(a)) =0.397
prob(btype(b)) =0.201
prob(btype(o)) =0.311
prob(btype(ab))=0.089
```

Fitting test

$\chi^2_1 = 0.65$
(accepted, level=0.05)

Declarative semantics

Parameterized logic programs

- Program $DB = F \cup R$
  - R definite clauses (rules)
  - F probabilistic facts with $P_F(\cdot \mid \theta)$

Prob. measure over the Herbrand interpretations of *F* parameterized with θ

- Distribution semantics

Prob. measure over the Herbrand interpretations of *DB*

  - $P_F(\cdot \mid \theta)$ is extended by DB to $P_{DB}(\cdot \mid \theta)$
  - $P_{DB}(\cdot \mid \theta)$ is the denotation of DB
  - θ is set manually or learned from data

Distribution semantics (1) [Sato 95]

- Ground atom = random variable taking on $\{1, 0\}$
- Program represents a set of ground clauses

$$\begin{aligned} DB &= F \cup R \\ &= \{A_1, A_2, \ldots\} \cup \{B_1 \leftarrow W_1, \ldots\} \end{aligned}$$

- Why semantic difficulty?
  - infinite symbols ➡ infinite Herbrand universe
  - Recursion ➡ infinitely many random variables
  - D-semantics allows for recursion and infinite domains, and unconditionally definable (even for looping programs)

Distribution semantics (2)

- Assume a parameterized basic distribution $P_F(A_1 = x_1, A_2 = x_2, A_3 = x_3, \ldots \mid \theta)$ is given over $F = \{A_1, A_2, A_3, \ldots\}$ $(x_i = 0, 1)$
- Sample P_F and get $\langle A_1 = 1, A_2 = 0, A_3 = 1, \ldots \rangle$

↳ Pick up $F' = \{A_1, A_3, \ldots\}$ a set of true facts

↳ The least Herbrand model $\mathrm{M}(F' \cup R)$ is defined

↳ Every ground atom has a truth value depending on F' and hence considered as a random variable

↳ $P_{DB}(A_1 = x_1, B_1 = y_1, A_2 = x_2, B_2 = y_2, \ldots \mid \theta)$ is defined

Distribution semantics (3)

- $P_{DB}(\cdot \mid \theta)$ is a σ-additive probability measure on

  $$\Omega = \{\omega \mid \omega = \langle z_1, z_2, \ldots \rangle, z_i \in \{0, 1\}\}$$

  where ω corresponds to an Herbrand interpretation
- φ closed formula: $P_{DB}(\varphi) = P_{DB}(\{\omega \mid \omega \models \varphi\})$
  - Globally consistent probs are assigned to all closed formulas
- Continuity by σ-additivity

  $$P_{DB}(\exists x \varphi(x)) = \lim_{n \to \infty} P_{DB}(\varphi(t_1) \vee \ldots \vee \varphi(t_n))$$

Note

- $P_F(\cdot \mid \theta)$ is constructed from finite distributions

$$P_F^{(n)}(x_1,\ldots,x_n \mid \theta)\ (n=1,2,\ldots)$$
$$\left(=\sum\nolimits_{x_{n+1}} P_F^{(n+1)}(x_1,\ldots,x_n,x_{n+1} \mid \theta)\right)$$

- Prob. mass distributes only over the set of possible least H-models $\{\mathrm{M}(F' \cup R) \mid F' \subseteq F\}$
- Distribution semantics covers logic programming, discrete Bayesian net, HMMs, PCFGs,...
- Definable for any DB (unlike other approaches:-)

Tabled search

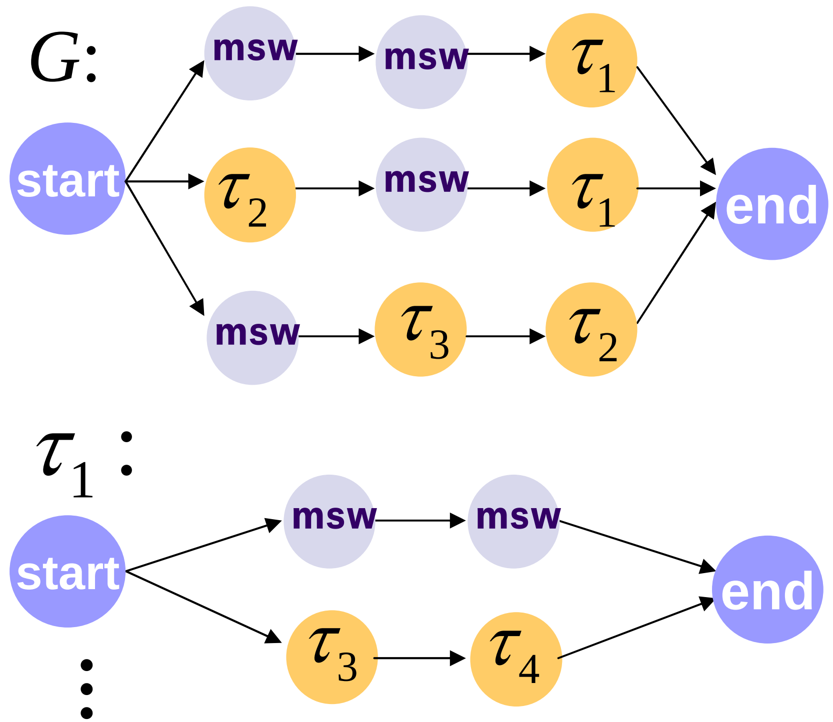
Explanation graphs

- We compute probabilities using explanation graphs which are a compact representation of statistical-logical dependency among events.
- In an explanation graph, subgraphs are partially ordered and shared by super-graphs.
- Sharing of subgraphs causes sharing of computations by dynamic programming.
- Thus efficient computation is achievable.

Computation sharing

- E-graph represents all explanations for G

$$
\begin{aligned}
G \Leftrightarrow\ & (\mathtt{msw} \wedge \mathtt{msw} \wedge \tau_1) \\
& \vee (\tau_2 \wedge \mathtt{msw} \wedge \tau_1) \\
& \vee (\mathtt{msw} \wedge \tau_3 \wedge \tau_2) \\
\tau_1 \Leftrightarrow\ & (\mathtt{msw} \wedge \mathtt{msw}) \\
& \vee (\tau_3 \wedge \tau_4) \\
\ldots & \\
P(G) =\ & P(\mathtt{msw})P(\mathtt{msw})P(\tau_1) \\
& + P(\tau_2)P(\mathtt{msw})P(\tau_1) \\
& + P(\mathtt{msw})P(\tau_3)P(\tau_2) \\
P(\tau_1) =\ & P(\mathtt{msw})P(\mathtt{msw}) \\
& + P(\tau_3)P(\tau_4)
\end{aligned}
$$

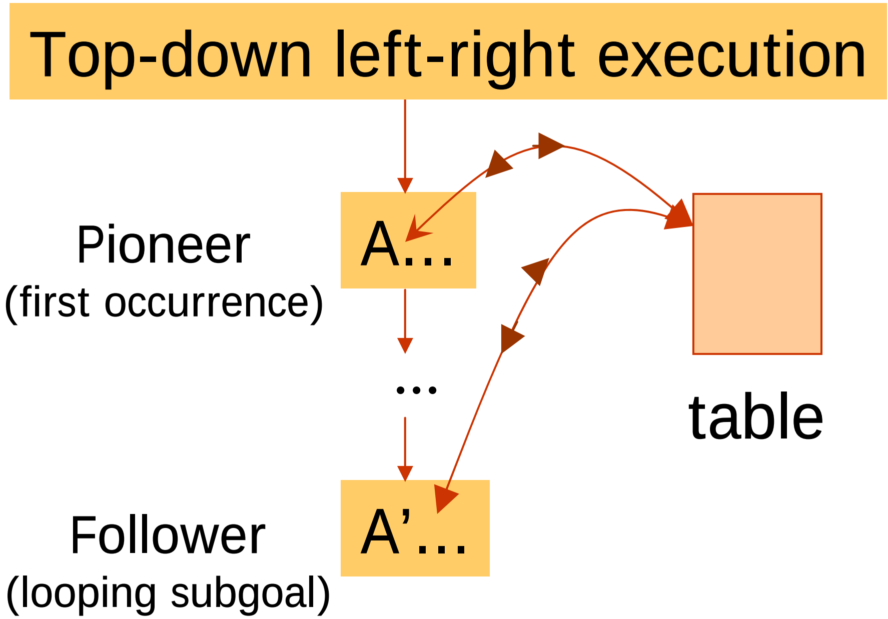


computation is shared!

Tabling

- An explanation graph for *G* is obtainable by searching for all explanations of *G* using tabling.
- Tabling remembers successful goals and reuses them to avoid recomputation of the same goal.
- There are two ways of tabling in logic programming;
  - Suspension & resumption of multiple processes
    - based on OLDT search, difficult to implement (see XSB)
  - Single process with iterative search
    - Based on linear-tabling, easy to implement (see B-Prolog)
    - Adopted in PRISM

Linear Tabling [Zhou 04]



- A' is a descendant of A but identical to A.
- A' immediately fails after consuming existing answers in the table

- Advantages
  - easy to implement
  - overhead-free
  - space efficient
  - cut is easy to handle
- Disadvantage
  - iterative computation
- Optimizations
  - subgoal optimization
  - semi-naïve optimization possible

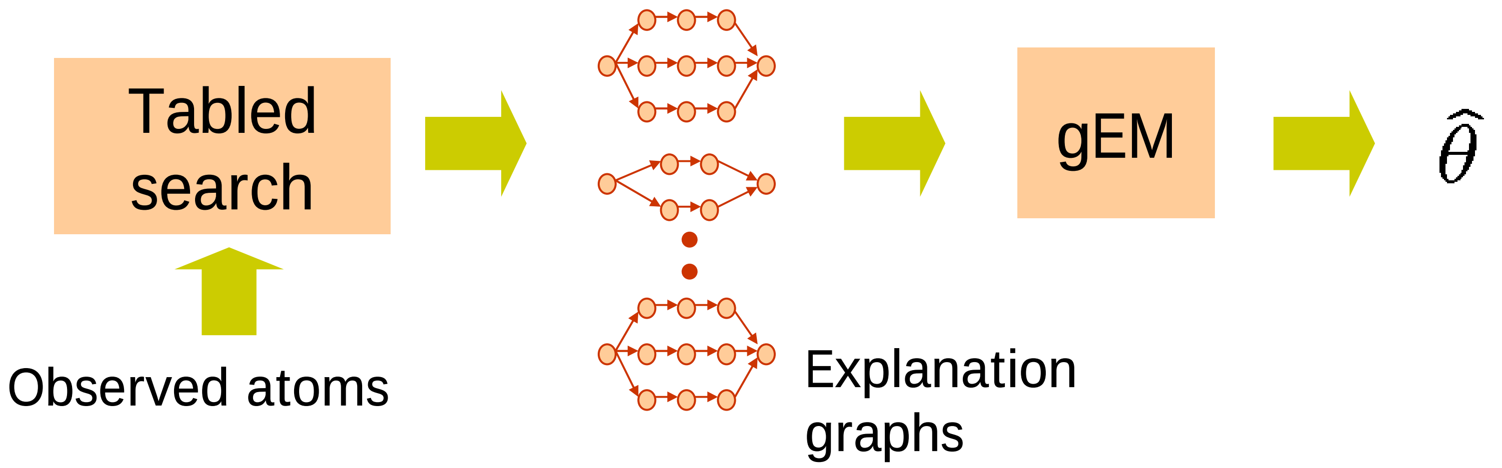
Learning parameters

gEM [Kemeya 00]

- To learn parameters in a program, we apply ML (maximum likelihood) estimation to observed data (top-goal G)
- Usually we do not know which of G's explanation is true one ➡ G is an incomplete data ➡ Use the EM algorithm.
- PRISM uses the gEM (graphical EM) algorithm which is a generic EM algorithm for PRISM programs unlike specialized ones such as the BW (Baum-Welch) algorithm and the IO (Inside-Outside) algorithm.
  - gEM is derived from distribution semantics.
  - gEM runs on explanation graphs in the manner of dynamic programming.
  - gEM achieves the same time complexity as BW and IO when OLDT search [Tamaki & Sato 86] is used for explanation graph construction.

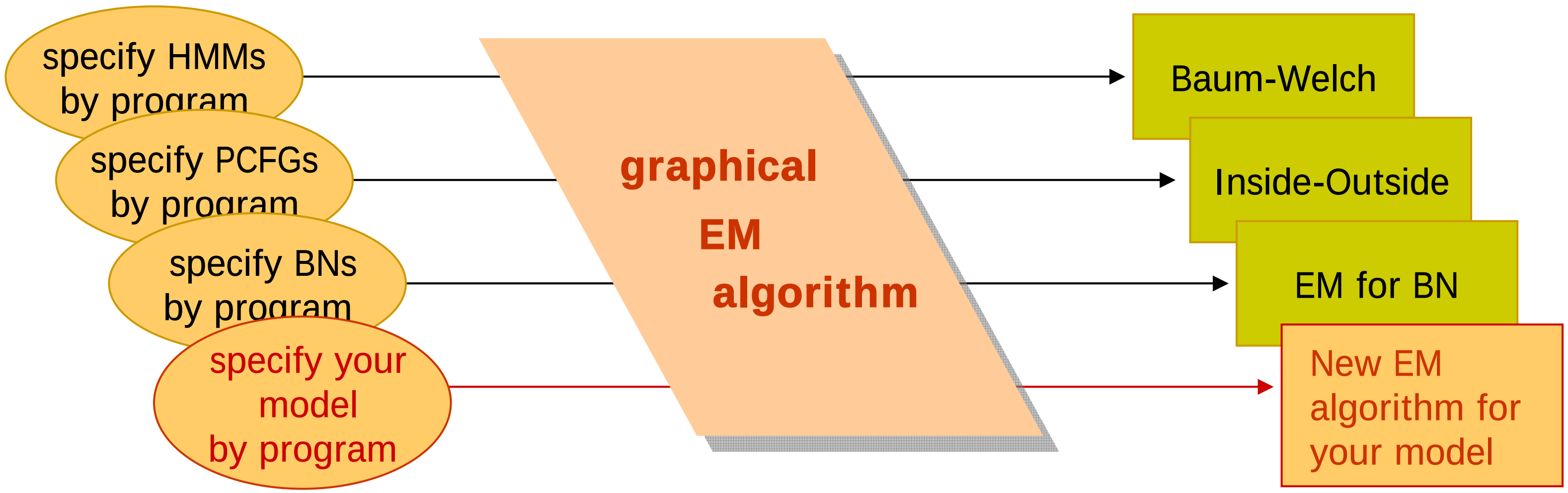
Search-and-learn schema with tabulation

- Tabled search + the graphical EM algorithm
  = efficient parameter learning



EM learning in PRISM

- Old approach:
  - Design a new EM algorithm for each application
- Our approach:
  - Write a PRISM program for each application



Time complexity of gEM + OLDT

- Total time = OLDT search time + iterations * time/iteration
- O(OLDT) $\geq$ O(explanation graphs) = O(updata-time/iteration)
- Equal to existing (specialized) EM algorithms

	OLDT	gEM	Specialized EM
HMMs	$O(N^2LT)$	$O(N^2LT)$	Baum-Welch
PCFGs	$O(N^3L^3T)$	$O(N^3L^3T)$	Inside-Outside
Singly connected Bayesian net	$O(\|V\|T)$	$O(\|V\|T)$	[Castillo et al. 97]
Pseudo PCSGs	$O(N^4L^3T)$	$O(N^4L^3T)$	[Charniak & Carroll 94]

N = #symbols, #states, L=sentence length, T = #data, |V| = #nodes

Conditions for fast EM learning

- Each observation has finitely many explanations:

$$\text{comp}(R) \vdash G \Leftrightarrow E_1 \vee \ldots \vee E_n$$
$$\text{where } E_i = \mathtt{msw}_1 \wedge \ldots \wedge \mathtt{msw}_k$$

- Exclusiveness of explanations:

$$P_{DB}(E_i \wedge E_j) = 0; (i \neq j)$$

- Uniqueness of observable goals:

$$P_{DB}(G_i \wedge G_j) = 0; (i \neq j) \text{ and } \sum_i P_{DB}(G_i) = 1$$

- Acyclicity:
  - caller-callee relation is partial ordering
- Independence:
  - atoms in an explanation are independent

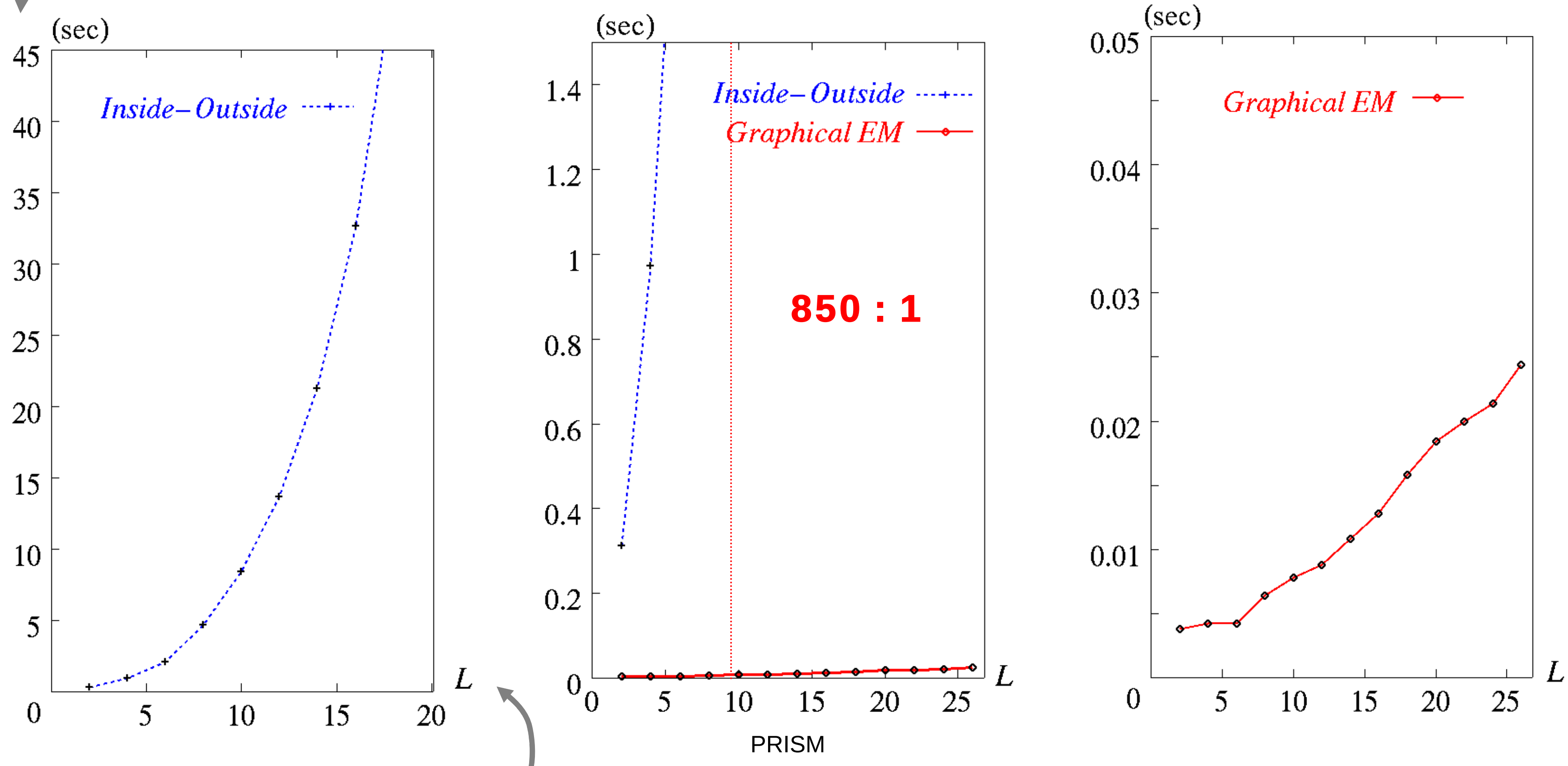
gEM vs. the Inside-Outside algorithm (1)

- PCFG is a CFG with probs assigned to rules

  $\mathtt{NP} \rightarrow \mathtt{N}(0.3)$, $\mathtt{NP} \rightarrow \mathtt{Adj}\,\mathtt{N}(0.3)$, $\mathtt{NP} \rightarrow \mathtt{S}\,\mathtt{NP}(0.4)$
- ATR corpus (size=10,995 min=2 ave.=10 max=49)
- PCFG: 860 rules (NT 173, POS 441)
- Parser used for explanation graph construction: Generalized LR (Tomita) parser
- gEM is 850 times faster than IO per iteration

Comparing updating time
for sampled 100 sentences (ATR)

time per iteration



sentence length

gEM vs. the Inside-Outside algorithm (2)

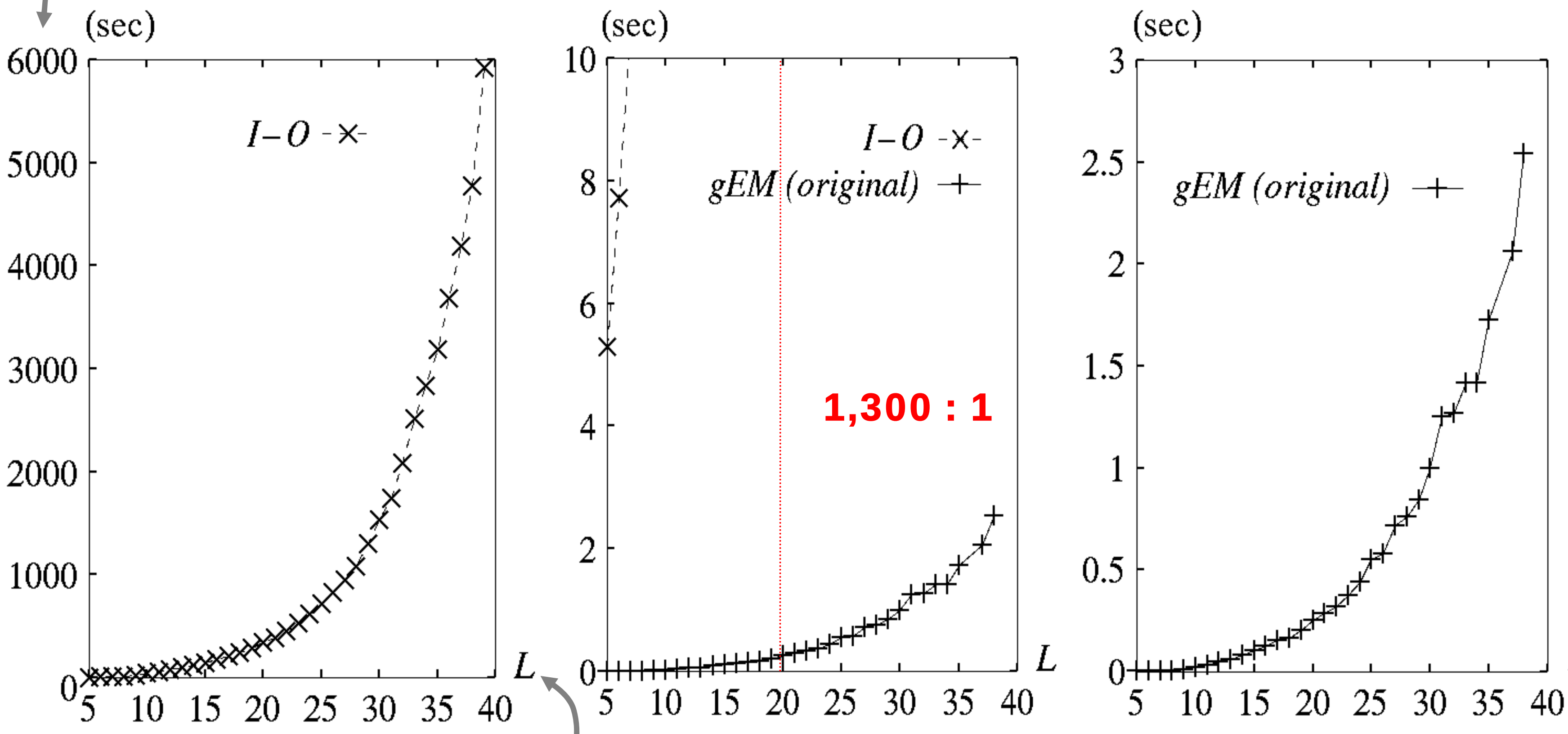
- EDR corpus (size=9,900 min=5 ave.=20 max=63)
- PCFG: 2,687 rules / 12,798 rules (CNF),

  3*10^8 parses/sentence at sentence length 20

  6.7*10^19 at 38
- Parser used for explanation graph construction: Generalized LR (Tomita) parser
- gEM is 1300 times faster than IO per iteration

Comparing updating time
for sampled 100 sentences (EDR)

time per iteration



sentence length

PCFGs in PRISM

- Probabilistic LL(1) parser with $O(L^3)$

```
target(pdcg,1).              % we observe pdcg([boys,run]),...
values(vp,[[v],[v,np]]).     % vp has two rules {vp->v, vp->v np}
...                          % one of {msw(vp,[v]), msw(vp,[v,np])}
                             % is probabilistically chosen

pdcg(L):- start_symbol(C), pdcg2([C],L,[]).
pdcg2([Wd|R],[Wd|L0],L2):- terminal(Wd), pdcg2(R,L0,L2).
pdcg2([A|R],[Wd|L0],L2):-
    first(A,Wd),             % Wd is in first(A)
    msw(A,RHS),              % probabilistic choice
    pdcg2(RHS,[Wd|L0],L1), pdcg2(R,L1,L2).
pdcg2([],L1,L1).
```

Meta-interpreter

- Parameter learning of PDCG form + ATR corpus completes in 3 min by a PC (3.4Ghz,2GB)

Exploring diverse modeling and parameter learning

- Naïve Bayes
- Profile HMM
- Linkage analysis
- PCFGs (PDCG, PLC,PGLR(k))
- HPSGs
- Graph grammars (HR, NLC)
- Shogi palyer

one report so far

no report so far

Negation and probabilistic constraint modeling

Failure by constraints
[Sato 05]

- Generative models
  - simulate how observations are generated
  - no failure assumed (e.g.BNs,HMMs,PCFGs)
- Complex models use constraints
  - failure is inevitable (e.g.HPSGs)
  - Let's model probabilistic agreement in number

```
agree(A):-
    msw(coin0,A),
    msw(coin1,B),
    A=B.
```

agree(A) succeeds only
when A=B. O.w. fails

The fgEM algorithm

- Failure means loss of probability mass
  - gEM is not usable
  - Distribution is log-linear;

    $P(x \mid \mathtt{success}, \theta)$ where $P(\mathtt{success}) = \sum_{\mathtt{x}:proof} P(x \mid \theta)$
- EM learning of parameters is possible by fgEM
  - fgEM [Sato 04] = gEM [Kameya 00] + FAM [Cussens 01]
    - FAM computes average count of msws in a failed computation

$$E[\mathtt{msw}(i,v)|\mathtt{fail}] = \frac{\sum_{\chi(\mathtt{expl})=\mathtt{fail}} P_{DB}(\mathtt{expl} \mid \theta_k)\delta(\mathtt{msw}(i,v) \in \mathtt{expl})}{\sum_{\chi(\mathtt{expl})=\mathtt{fail}} p(\mathtt{expl} \mid \theta)}$$

  - fgEM requires a failure program

Failure program

- A failure program is one that explicitly describes how failure occurs.

```
agree(A):-
    msw(coin0,A),
    msw(coin1,B),
    A=B.
```

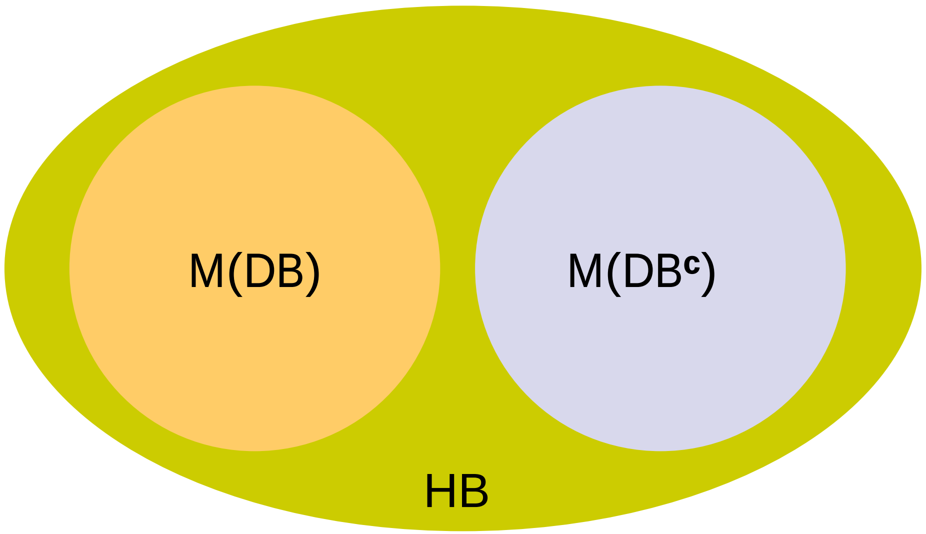
```
failure:-
    msw(coin0,A),
    msw(coin1,B),
    ¥+A=B.
```

failure
= no output generated
= not(∃X agree(X))

- PRISM1.8 uses FOC to automatically derive a failure program from the negation of a source program.

FOC (first-order compiler)

- Full automatic program synthesis for logic programs with negation [Sato 89]
- Compiled program DB^c positively computes the finite failure of DB



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

The program terminates for every ground goal

Negation elimination by FOC

Source program $DB_{\mathbf{even}}$

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

Compiled program $DB^{c}_{\mathbf{even}}$

```
even(0).
even(s(A)):- closure_even0(A,f0).
closure_even0(s(A),_):- even(A).
```

Extension

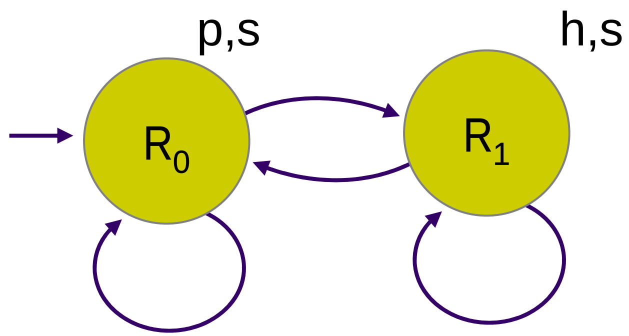
- Original FOC = for non-probabilistic programs
- Extended for PRISM programs containing negation

$$\neg \exists X (\mathtt{msw}(\mathtt{abo}, X) \wedge X = \mathtt{a}) \\ \Rightarrow \exists X \, \mathtt{msw}(\mathtt{abo}, X) \wedge (X \neq \mathtt{a}))$$

  - This transformation is meaning-preserving in view of a new distribution semantics (not included in slides)

Constrained HMMs and a dieting professor

- Constrained HMMs are an instance of probabilistic constraint modeling.
- They are HMMs with constrains that may fail.



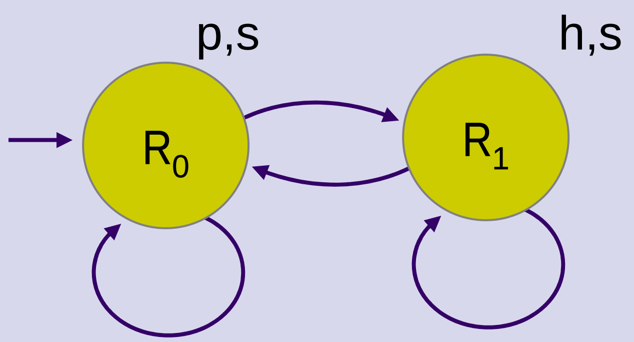
- Suppose a professor wishes to diet.
- There are two restaurants R0 and R1
- He visits them and orders pizza or sandwich at R0, and hamburger or sandwich at R1, probabilstically.
- He records lunches like [s,s,h,p,s,h,s].
- He tries to keep the total lunch calories in a week < 4000.
- Only successful records are kept.

- Given: we have a list of his successful records.
- Task: infer the failure probability.

Program for the dieting professor

```
failure:- not(success).
success:- success(_).
success(L):- diet(L,r0,0,7).

diet(L,R,C,N):-
   N>0,
   msw(lunch(R),D),    % order lunch
   ( R == r0,       % pizza or sandwich
      ( D = p, C2 is C+900 ; D = s, C2 is C+400 )
   ; R == r1,       % hamburger or sandwich
      ( D = h, C2 is C+400 ; D = s, C2 is C+500 ) ),
   L=[D|L2],
   N2 is N-1,
   msw(tr(R),R2),       % next restaurant
  diet(L2,R2,R2,N2).
diet([],_,C,0):- C < 4000.   % calorie constraint must be met
```



Failure program by FOC

```
failure:- closure_success0(f0).
closure_success0(A):- closure_chmm0(r0,0,7,A).
closure_chmm0(R,B,C,D):-
    ( C>0,
        msw(tr(R),R2), msw(lunch(R),F),
        ( R¥==r0
        ; R==r0,
            ( ¥+F=p ; F=p, G is B+900,H is C-1, closure_chmm0(R2,G,H,D)),
            ( ¥+F=s ; F=s,  I is B+400,J is C-1,closure_chmm0(R2,I,J,D))),
        ( R¥== r1
        ; R == r1,
            ( ¥+F=h ; F=h, K is B+400,L is C-1,closure_chmm0(R2,K,L,D)),
            ( ¥+F=s ; F=s, M is B+500,N is C-1,closure_chmm0(R2,M,N,D)))
    ; C=<0 ),
    ( ¥+C=0 ; C=0, B>=4000 ).
```

- tail recursive just like positive case
  → dynamic programming

fgEM learning

```
| ?- prismn('prof.psm'),set_sw,
                  generate_goals(500,Gs),learn([failure|Gs]).
success([s,s,p,p,s,h,s])
...
.........50..(Converged: -2905.412443514)
Finished learning
Number of iterations: 63
Total learning time: 0.2 seconds
All solution search time: 0.08 seconds

Switch lunch(r0): p  (0.4014) s  (0.5986)
Switch lunch(r1): h  (0.5339) s  (0.4661)
Switch tr(r0):    r0 (0.7190) r1 (0.2810)
Switch tr(r1):    r1 (0.7236) r0 (0.2764)

?- prob(failure).
The probability of failure is: 0.3448
```

sampled goal

original values

0.4 0.6
0.5 0.5
0.7 0.3
0.7 0.3
0.3486

- If failure is not assumed, the estimated failure probability deteriorates to 0.0823.

Note

- Failure programs can be obtainable by other methods
  - Manual derivation
    - traces all failed paths of computation by inspection and represent them as a program.
  - Negation technique [Sato 89]
    - gives better code than FOC but there are restrictions on applicability
- More complex probabilistic constraint modeling than constrained HMMs is possible.
  - Finite PCFGs = PCFGs with failure constraints [Sato 04]
  - HPSGs = unification based constraint grammar, approximated by PCFGs

References

- Please visit http://mi.cs.titech.ac.jp/prism/
  - Current version is PRISM1.8
- Papers ([Sato 01] is a most comprehensive paper)

[Sato 05] Sato, T., Kameya, Y. and Zhou, N.-F.: Generative modeling with failure in PRISM. IJCAI2005, to appear, 2005.

[Zhou 04] Zhou, N.-F., Shen, Y.-D. and Sato, T.: Semi-naive Evaluation in Linear Tabling, PPDP04, pp.90–97, 2004.

[Sato 01] Sato, T. and Kameya, Y.: Parameter learning of logic programs for symbolic-statistical modeling. JAIR, Vol.15, pp.391–454, 2001.

[Cussens 01] Cussens, J.: Parameter estimation in stochastic logic programs. *Machine learning*, Vol.44, Issue 3, pp.245–271, 2001.

[Kameya 00] Kameya, Y. and Sato, T.: Efficient EM learning with tabulation for parameterized logic programs. *CL2000*, LNAI, Vol.1861, pp.269–294, 2000.

[Sato 95] Sato, T.: A statistical learning method for logic programs with distribution semantics, ICLP95, Tokyo, pp.715–729, 1995.

Future plan

- More sophisticated learning
  - Conditional random fields
  - DAEM
  - Better tabled search
- More computer power
  - Parallel search on a grid machine
  - 64bit
- More applications
  - Graph grammars
  - User modeling