Probabilistic Inductive Logic Programming

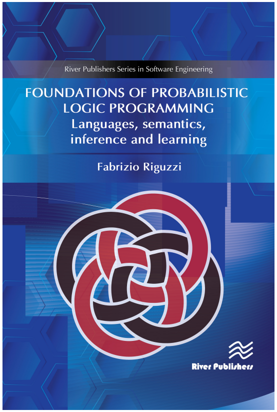
Fabrizio Riguzzi

MCS Deparment– University of Ferrara, Italy, fabrizio.riguzzi@unife.it

Dipartimento
di Matematica
e Informatica

Outline

- Probabilistic logic programming
- Parameter learning
    - PRISM
    - EMBLEM
    - LeProbLog
    - LFI-Problog
- Structure learning
    - SLIPCOVER
    - ProbFOIL+
    - LEMUR
- DPHIL


River Publishers Series in Software Engineering
FOUNDATIONS OF PROBABILISTIC LOGIC PROGRAMMING
Languages, semantics, inference and learning
Fabrizio Riguzzi
River Publishers

Dipartimento di Matematica e Informatica

Probabilistic Logic Programming

- Distribution Semantics [Sato ICLP95]
- A probabilistic logic program defines a probability distribution over normal logic programs (called instances or possible worlds or simply worlds)
- The distribution is extended to a joint distribution over worlds and interpretations (or queries)
- The probability of a query is obtained from this distribution

Dipartimento
di Matematica
e Informatica

Probabilistic Logic Programming (PLP) Languages under the Distribution Semantics

- Probabilistic Logic Programs [Dantsin RCLP91]
- Probabilistic Horn Abduction [Poole NGC93], Independent Choice Logic (ICL) [Poole AI97]
- PRISM [Sato ICLP95]
- Logic Programs with Annotated Disjunctions (LPADs) [Vennekens et al. ICLP04]
- ProbLog [De Raedt et al. IJCAI07]
- They differ in the way they define the distribution over logic programs

Dipartimento di Matematica e Informatica

PLP Online

- `http://cplint.eu`
  - Inference (knowledge compilation, Monte Carlo)
  - Parameter learning (EMBLEM)
  - Structure learning (SLIPCOVER, LEMUR)
- `https://dtai.cs.kuleuven.be/problog/`
  - Inference (knwoledge compilation, Monte Carlo)
  - Parameter learning (LFI-ProbLog)

Dipartimento
di Matematica
e Informatica

Logic Programs with Annotated Disjunctions

```
http://cplint.eu/e/sneezing_simple.pl
```

$$\begin{array}{l} sneezing(X) : 0.7 \; ; \; null : 0.3 \leftarrow flu(X). \\ sneezing(X) : 0.8 \; ; \; null : 0.2 \leftarrow hay\_fever(X). \\ flu(bob). \\ hay\_fever(bob). \end{array}$$

- Distributions over the head of rules
- *null* does not appear in the body of any rule
- Worlds obtained by selecting one atom from the head of every grounding of each clause

Dipartimento di Matematica e Informatica

Reasoning Tasks

- Inference: we want to compute the probability of a query given the model and, possibly, some evidence
- Weight learning: we know the structural part of the model (the logic formulas) but not the numeric part (the weights) and we want to infer the weights from data
- Structure learning we want to infer both the structure and the weights of the model from data

Dipartimento
di Matematica
e Informatica

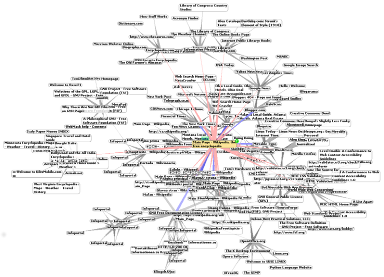
Applications

- Link prediction: given a (social) network, compute the probability of the existence of a link between two entities (UWCSE)

```
advisedby(X, Y) :0.7 :-
  publication(P, X),
  publication(P, Y),
  student(X).
```

Applications

- Classify web pages on the basis of the link structure (WebKB)

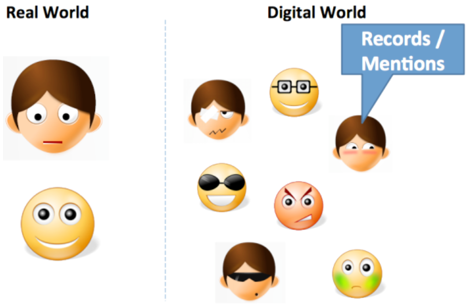


```
coursePage(Page1): 0.3 :- linkTo(Page2,Page1),coursePage(Page2).
coursePage(Page1): 0.6 :- linkTo(Page2,Page1),facultyPage(Page2).
...
coursePage(Page): 0.9 :- has('syllabus',Page).
...
```

Dipartimento
di Matematica
e Informatica

Applications

- Entity resolution: identify identical entities in text or databases

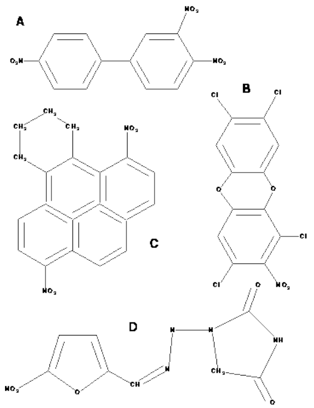


```
samebib(A,B):0.9 :-
samebib(A,C), samebib(C,B).
sameauthor(A,B):0.6 :-
  sameauthor(A,C), sameauthor(C,B).
sametitle(A,B):0.7 :-
  sametitle(A,C), sametitle(C,B).
samevenue(A,B):0.65 :-
  samevenue(A,C), samevenue(C,B).
samebib(B,C):0.5 :-
  author(B,D),author(C,E),sameauthor(D,E).
samebib(B,C):0.7 :-
  title(B,D),title(C,E),sametitle(D,E).
samebib(B,C):0.6 :-
  venue(B,D),venue(C,E),samevenue(D,E).
samevenue(B,C):0.3 :-
  haswordvenue(B,logic),
  haswordvenue(C,logic).
...
```

Dipartimento di Matematica e Informatica

Applications

- Chemistry: given the chemical composition of a substance, predict its mutagenicity or its carcenogenicity

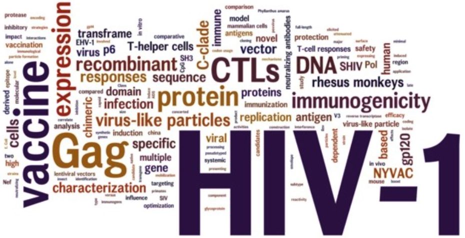


```
active(A):0.4 :-
   atm(A,B,c,29,C),
   gteq(C,-0.003),
   ring_size_5(A,D).
active(A):0.6:-
   lumo(A,B), lteq(B,-2.072).
active(A):0.3 :-
   bond(A,B,C,2),
   bond(A,C,D,1),
   ring_size_5(A,E).
active(A):0.7 :-
   carbon_6_ring(A,B).
active(A):0.8 :-
   anthracene(A,B).
...
```

Dipartimento di Matematica e Informatica

Applications

- Medicine: diagnose diseases on the basis of patient information (Hepatitis), influence of genes on HIV, risk of falling of elderly people



PRISM

- Hidden Markov model: a dynamical system that, at each time point t, is in a state S and emits one symbol O
- $P(O|S)$ and $P(NextS|S)$ are independent of time.
- The states are hidden: the task is to obtain information on them from the sequence of output symbols.
- Speech recognition.

Dipartimento
di Matematica
e Informatica

PRISM

$values(tr(\_), [s1, s2]).$
$values(out(\_), [a, b]).$
$\leftarrow set\_sw(tr(s1), [0.2, 0.8]).$
$\leftarrow set\_sw(tr(s2), [0.8, 0.2]).$
$\leftarrow set\_sw(out(s0), [0.5, 0.5]).$
$\leftarrow set\_sw(out(s1), [0.6, 0.4]).$
$hmm(Os) \leftarrow hmm(s1, Os).$
$hmm(\_S, []).$
$hmm(S, [O|Os]) \leftarrow$
$\quad msw(out(S), O), msw(tr(S), NextS), hmm(Next, Os).$

- $P(hmm(Os))$: probability that the sequence of symbols Os is emitted.
- No memoing.

PRISM

Definition (PRISM parameter learning problem)

Given a PRISM program $\mathcal{P}$ and a set of examples $E = \{e_1, \ldots, e_T\}$ which are ground atoms, find the parameters $\mathbf{\Pi}$ of *msw* fact so that the *likelihood* of the atoms

$$L = \prod_{t=1}^{T} P(e_t)$$

is maximized.
Equivalently, find the parameters of *msw* fact so that the log likelihood of the atoms

$$LL = \sum_{t=1}^{T} \log P(e_t)$$

is maximized.

PRISM Assumpions

1. the probability of a conjunction (A, B) is computed as the product of the probabilities of A and B (*independent-and assumption*),
2. the probability of a disjunction $(A; B)$ is computed as the sum of the probabilities of A and B (*exclusive-or assumption*).

Dipartimento
di Matematica
e Informatica

Example

$values(tr(\_), [s1, s2]).$
$values(out(\_), [a, b]).$
$\leftarrow set\_sw(tr(s1), [0.2, 0.8]).$
$\leftarrow set\_sw(tr(s2), [0.8, 0.2]).$
$\leftarrow set\_sw(out(s0), [0.5, 0.5]).$
$\leftarrow set\_sw(out(s1), [0.6, 0.4]).$
$hmm(Os) \leftarrow hmm(s1, Os).$
$hmm(\_S, []).$
$hmm(S, [O|Os]) \leftarrow$
    $msw(out(S), O), msw(tr(S), NextS), hmm(Next, Os).$

- $P(hmm(Os))$: probability that the sequence of symbols Os is emitted.

Example

- Query $hmm([a, b, b])$
- 8 explanations

$$
\begin{aligned}
E_1 = {} & m(out(s1), a), m(tr(s1), s1), m(out(s1), b), m(tr(s1), s1),\\
& m(out(s1), b), m(tr(s1), s1),\\
E_2 = {} & m(out(s1), a), m(tr(s1), s1), m(out(s1), b), m(tr(s1), s1),\\
& m(out(s1), b), m(tr(s1), s2),\\
E_3 = {} & m(out(s1), a), m(tr(s1), s1), m(out(s2), b), m(tr(s1), s2),\\
& m(out(s2), b), m(tr(s2), s1),\\
\ldots & \\
E_8 = {} & m(out(s1), a), m(tr(s1), s2), m(out(s2), b), m(tr(s2), s2),\\
& m(out(s2), b), m(tr(s2), s2)
\end{aligned}
$$

Dipartimento di Matematica e Informatica

Example

- If the query q has the explanations $E_1 \ldots, E_n$:

$$q \Leftrightarrow E_1 \vee \ldots \vee E_n$$

- $P(q) = \sum_{i=1}^{n} P(E_i)$
- $P(E_i)$ is the product of the probability of each atom
- Because of the assumptions

Dipartimento di Matematica e Informatica

Example

```
values(gene,[a,b,o]).
bloodtype(P) :-
  genotype(X,Y),
  ( X=Y -> P=X
  ; X=o -> P=Y
  ; Y=o -> P=X
  ; P=ab
  ).
genotype(X,Y) :- msw(gene,X),msw(gene,Y).
```

How a person's blood type is determined by his genotype, formed by a pair of two genes (a, b or o).

Dipartimento di Matematica e Informatica

Example

```
?- learn([count(bloodtype(a),40),count(bloodtype(b),20),
    count(bloodtype(o),30),count(bloodtype(ab),10)]).
```

where `count(At,N)` denotes the repetition of atom `At` `N` times.

```
?- show_sw.
Switch gene: unfixed: a (0.292329558535712)
b (0.163020241540856)
o (0.544650199923432)
```

PRISM

- PRISM looks for the maximum likelihood parameters of the *msw* atoms.
- These are not observed in the dataset, which contains only derived atoms.
- Relative frequency cannot be used
- Expectation Maximization

Dipartimento
di Matematica
e Informatica

PRISM

- Associate a random variable X_i with values $D = \{x_{i1}, \ldots, x_{in_i}\}$ to the ground switch name i of $msw(i, x)$ with domain D
- PRISM alternates between the two phases:
  - Expectation: compute $\mathbf{E}[c_{ik}|e]$ for all examples e, switches $msw(i, x)$ and $k \in \{1, \ldots, n_i\}$, where c_{ik} is the number of times variable X_i takes value x_{ik}

$$\mathbf{E}[c_{ik}|e] = P(X_i = x_{ik}|e).$$

  - Maximization: compute Π_{ik} for all $msw(i, x)$ and $k = 1, \ldots, n_i - 1$ as

$$\Pi_{ik} = \frac{\sum_{e \in E} \mathbf{E}[c_{ik}|e]}{\sum_{e \in E} \sum_{k=1}^{n_i} \mathbf{E}[c_{ik}|e]}$$

Dipartimento di Matematica e Informatica

PRISM

- If the program satisfies the exclusive-or assumption, $P(X_i = x_{ik}|e)$ can be computed as

$$P(X_i = x_{ik}|e) = \frac{P(X_i = x_{ik}, e)}{P(e)} = \frac{\sum_{\kappa \in K_e, msw(i,x_{ik}) \in e} P(\kappa)}{P(e)}$$

  where K_e is the set of explanations of e
- Each explanation κ is a set of *msw* atoms of the form $msw(i, x_{ik})$.

Naive PRISM

```
1:  function PRISM-EM-Naive($E, \mathcal{P}, \epsilon$)
2:      $LL = -inf$
3:      repeat
4:          $LL_0 = LL$
5:          for all $i, k$ do                                        ▷ Expectation step
6:              $\mathbf{E}[c_{ik}] \leftarrow \sum_{e \in E} \frac{\sum_{\kappa \in K_e, msw(i, x_{ik}) \in e} P(\kappa)}{P(e)}$
7:          end for
8:          for all $i, k$ do                                        ▷ Maximization step
9:              $\Pi_{ik} \leftarrow \frac{\mathbf{E}[c_{ik}]}{\sum_{k'=1}^{n_i} \mathbf{E}[c_{ik'}]}$
10:         end for
11:         $LL \leftarrow \sum_{e \in E} \log P(e)$
12:     until $LL - LL_0 < \epsilon$
13:     return $LL, \Pi_{ik}$ for all $i, k$
14: end function
```

PRISM

- There can be exponential numbers of explanations
- More efficient dynamic programming algorithm
- Tabling is used to find formulas of the form

$$g_i \Leftrightarrow S_{i1} \vee \ldots \vee S_{is_i}$$

- The g_is are subgoals that can be ordered as $\{g_1, \ldots, g_m\}$ such that $e = g_1$ and each S_{ij} contains only *msw* atoms and subgoals from $\{g_{i+1}, \ldots, g_m\}$.
- Linear number of formulas rather than exponential
- Acyclic support condition, true if tabling succeeds in evaluating q, i.e., if it doesn't go into a loop.

Dipartimento
di Matematica
e Informatica

Example

- For $hmm([a,b,b])$, PRISM builds the formulas

$$
\begin{array}{l}
hmm([a,b,b]) \Leftrightarrow hmm(s1,[a,b,b]) \\
hmm(s1,[a,b,b]) \Leftrightarrow m(out(s1),a), m(tr(s1),s1), hmm(s1,[b,b]) \vee \\
\quad m(out(s1),a), m(tr(s1),s2), hmm(s2,[b,b]) \\
hmm(s1,[b,b]) \Leftrightarrow m(out(s1),b), m(tr(s1),s1), hmm(s1,[b]) \vee \\
\quad m(out(s1),b), m(tr(s1),s2), hmm(s2,[b]) \\
hmm(s2,[b,b]) \Leftrightarrow m(out(s2),b), m(tr(s2),s1), hmm(s1,[b]) \vee \\
\quad m(out(s2),b), m(tr(s2),s2), hmm(s2,[b]) \\
hmm(s1,[b]) \Leftrightarrow m(out(s1),b), m(tr(s1),s1), hmm(s1,[]) \vee \\
\quad m(out(s1),b), m(tr(s1),s2), hmm(s2,[]) \\
hmm(s2,[b]) \Leftrightarrow m(out(s2),b), m(tr(s2),s1), hmm(s1,[]) \vee \\
\quad m(out(s2),b), m(tr(s2),s2), hmm(s2,[]) \\
hmm(s1,[]) \Leftrightarrow true \\
hmm(s2,[]) \Leftrightarrow true
\end{array}
$$

Outside probabilities

- We can divide the explanations for e into two sets, K_{e1}, that includes the explanations containing $msw(i, x_k)$, and K_{e2}, that includes the other explanations.
- $P(e) = P(K_{e1}) + P(K_{e2})$
- $P(X_{ij} = x_{ik}, e) = P(K_{e1})$.
- Each explanation in K_{e1} takes the form $\{\{g_i, W_1\}, \ldots, \{g_i, W_s\}\}$ and

$$P(K_{e1}) \quad = \quad \sum_{\{g_i, W\} \in K_{e1}} P(g_i) P(W) = P(g_i) \sum_{\{g_i, W\} \in K_{e1}} P(W)$$

Outside probabilities

- So we obtain

$$\begin{aligned} P(X_{ij} = x_{ik}, e) &= P(g_i) \sum_{\{g_i, W\} \in K_{e1}} P(W) = \\ & \frac{\partial P(K_e)}{\partial P(g_i)} P(g_i) = \\ & \frac{\partial P(e)}{\partial P(g_i)} P(g_i) = Q(g_i) P(g_i) \end{aligned} \tag{1}$$

- If $g_i = msw(i, x_k)$, then

$$P(X_i = x_{ik}, e) = Q(g_i)P(g_i) = Q(g_i)\Pi_{ik}.$$

Dipartimento di Matematica e Informatica

PRISM

- Inside probability: $P(g_i)$
- Outside probability: $Q(g_i)$
- PRISM generalizes the Inside-Outside algorithm for PCFG.
- It also generalizes the forward-backward algorithm for parameter learning in HMM by the Baum-Welch algorithm

Dipartimento di Matematica e Informatica

Get-Inside-Probs

```
1: procedure Get-Inside-Probs($q$)
2:     for all $i, k$ do
3:         $P(msw(i, v_k)) \leftarrow \Pi_{ik}$
4:     end for
5:     for $i \leftarrow m \rightarrow 1$ do
6:         $P(g_i) \leftarrow 0$
7:         for $j \leftarrow 1 \rightarrow s_i$ do
8:             Let $S_{ij}$ be $h_{ij1}, \ldots, h_{ijo}$
9:             $P(S_{ij}) \leftarrow \prod_{l=1}^{o} P(h_{ijl})$
10:            $P(g_i) \leftarrow P(g_i) + P(S_{ij})$
11:        end for
12:    end for
13: end procedure
```

Dipartimento
di Matematica
e Informatica

Outside probabilities

- Defined as

$$Q(g_i) = \frac{\partial P(e)}{\partial P(g_i)}$$

- Suppose g_i appears in the ground program as

$$\begin{array}{ccc} b_1 \leftarrow g_i, W_{11} & \ldots & b_1 \leftarrow g_i, W_{1i_1} \\ & \ldots & \\ b_K \leftarrow g_i, W_{K1} & \ldots & b_K \leftarrow g_i, W_{Ki_K} \end{array}$$

- Then

$$\begin{array}{c} P(b_1) = P(g_i, W_{11}) + \ldots + P(g_i, W_{1i_1}) \\ \ldots \\ P(b_K) = P(g_i, W_{K1}) + \ldots + P(g_i, W_{Ki_K}) \end{array}$$

Outside probabilities

- $Q(g_1) = 1$ as $e = g_1$.
- For $i = 2, \ldots, m$, $Q(g_i)$ by the chain rule knowing that $P(e)$ is a function of $P(b_1), \ldots, P(b_K)$

$$
\begin{aligned}
Q(g_i) &= \frac{\partial P(q)}{\partial P(b_1)} \frac{\partial P(b_1)}{\partial P(g_1)} + \ldots + \frac{\partial P(q)}{\partial P(b_K)} \frac{\partial P(b_K)}{\partial P(g_1)} = \\
& \frac{\partial P(q)}{\partial P(b_1)} \frac{\partial P(g_i, W_{11})}{\partial P(g_1)} + \ldots + \frac{\partial P(q)}{\partial P(b_K)} \frac{\partial P(g_i, W_{Ki_K})}{\partial P(g_1)} = \\
& Q(b_1) P(g_i, W_{11}) / P(g_i) + \ldots + P(g_i, W_{Ki_K}) / P(g_i)
\end{aligned}
$$

- Recursive formula

$$
\begin{aligned}
Q(g_1) &= 1 \\
Q(g_i) &= Q(b_1) \sum_{s=1}^{i_1} \frac{P(g_i, W_{1s})}{P(g_i)} + \ldots + Q(b_K) \sum_{s=1}^{i_K} \frac{P(g_i, W_{Ks})}{P(g_i)}
\end{aligned}
$$

- To be evaluated top-down from $q = g_1$ down to g_m.

Get-Outside-Probs

```
1: procedure Get-Outside-Probs($q$)
2:     $Q(g_1) \leftarrow 1.0$
3:     for $i \leftarrow 2 \rightarrow m$ do
4:         $Q(g_i) \leftarrow 0.0$
5:         for $j \leftarrow 1 \rightarrow s_i$ do
6:             Let $S_{ij}$ be $h_{ij1}, \ldots, h_{ijo}$
7:             for $l \leftarrow 1 \rightarrow o$ do
8:                 $Q(h_{ijl}) \leftarrow Q(h_{ijl}) + Q(g_i)P(S_{ij})/P(h_{ijl})$
9:             end for
10:        end for
11:    end for
12: end procedure
```

Dipartimento
di Matematica
e Informatica

PRISM-EM

```
1: function PRISM-EM($E, \mathcal{P}, \epsilon$)
2:     $LL = -inf$
3:     repeat
4:         $LL_0 = LL$
5:         $LL =$ Expectation($E$)
6:         for all $i$ do
7:             $Sum \leftarrow \sum_{k=1}^{n_i} \mathbf{E}[c_{ik}]$
8:             for $k = 1$ to $n_i$ do
9:                 $\Pi_{ik} = \frac{\mathbf{E}[c_{ik}]}{Sum}$
10:            end for
11:        end for
12:    until $LL - LL_0 < \epsilon$
13:    return $LL, \Pi_{ik}$ for all $i, k$
14: end function
```

Dipartimento di Matematica e Informatica

PRISM-Expectation

```
function PRISM-Expectation($E$)
    $LL = 0$
    for all $e \in E$ do
        Get-Inside-Probs($e$)
        Get-Outside-Probs($e$)
        for all $i$ do
            for $k = 1$ to $n_i$ do
                $\mathbf{E}[c_{ik}] = \mathbf{E}[c_{ik}] + Q(msw(i, x_k))\Pi_{ik}/P(e)$
            end for
        end for
        $LL = LL + \log P(e)$
    end for
    return $LL$
end function
```

Dipartimento di Matematica e Informatica

Complexity

- PRISM has the same time complexity for programs encoding HMM and PCFG as the specific parameter learning algorithms: the Baum-Welch algorithm and the Inside-Outside algorithm

Dipartimento di Matematica e Informatica

Parameter Learning for ProbLog and LPADs

- [Thon et al. ECML 2008] proposed an adaptation of EM for CPT-L, a simplified version of LPADs
- The algorithm computes the counts efficiently by repeatedly traversing the BDDs representing the explanations
- [Ishihata et al. ILP 2008] independently proposed a similar algorithm
- LFI-ProbLog [Gutamnn et al. ECML 2011]: EM for ProbLog on BDDs
- EMBLEM [Riguzzi & Bellodi IDA 2013] adapts [Ishihata et al. ILP 2008] to LPADs

Dipartimento
di Matematica
e Informatica

EMBLEM

Definition (EMBLEM learning problem)

Given an LPAD $\mathcal{P}$ with unknown parameters and two sets $E^+ = \{e_1, \dots, e_T\}$ and $E^- = \{e_{T+1}, \dots, e_Q\}$ of ground atoms (positive and negative examples), find the value of the parameters $\mathbf{\Pi}$ of $\mathcal{P}$ that maximize the likelihood of the examples, i.e., solve

$$\arg\max_{\mathbf{\Pi}} P(E^+, \sim E^-) = \arg\max_{\mathbf{\Pi}} \prod_{t=1}^{T} P(e_t) \prod_{t=T+1}^{Q} P(\sim e_t).$$

Predicates for the atoms in E^+ and E^-: target because the objective is to be able to better predict the truth value of atoms for them.

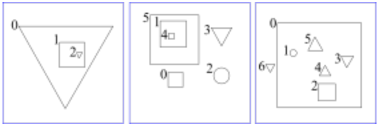
Parameter Learning

- Typically, the LPAD $\mathcal{P}$ has two components:
  - a set of rules, annotated with parameters
  - a set of certain ground facts, representing background knowledge on individual cases of a specific world
- Useful to provide information on more than one world: a background knowledge and sets of positive and negative examples for each world
- Description of one world: *mega-interpretation* or *mega-example*
- Positive examples encoded as ground facts of the mega-interpretation and the negative examples as suitably annotated ground facts (such as $neg(a)$ for negative example a)
- The task then is maximizing the product of the likelihood of the examples for all mega-interpretations.

Dipartimento di Matematica e Informatica

Example: Bongard Problems

- Introduced by the Russian scientist M. Bongard
- Pictures, some positive and some negative
- Problem: discriminate between the two classes.
- The pictures contain shapes with different properties, such as small, large, pointing down, ... and different relationships between them, such as inside, above, ...



Data

Each mega-examle encodes a single picture

```
begin(model(2)).
pos.
triangle(o5).
config(o5,up).
square(o4).
in(o4,o5).
circle(o3).
triangle(o2).
config(o2,up).
in(o2,o3).
triangle(o1).
config(o1,up).
end(model(2)).

begin(model(3)).
neg(pos).
circle(o4).
circle(o3).
in(o3,o4).
....
```

Dipartimento di Matematica e Informatica

Program

Theory for parameter learning and background

```
pos:0.5 :-
  circle(A),
  in(B,A).
pos:0.5 :-
  circle(A),
  triangle(B).
```

The task is to tune the two parameters

EMBLEM

- The interpretations record the truth value of ground atoms, not of the random variables
- Unseen data: relative frequency can't be used
- Expectation-Maximization algorithm:
  - Expectation step: the distribution of the unseen variables in each instance is computed given the observed data
  - Maximization step: new parameters are computed from the distributions using relative frequency
  - End when likelihood does not improve anymore

Dipartimento di Matematica e Informatica

EMBLEM

- EM over Bdds for probabilistic Logic programs Efficient Mining [Bellodi and Riguzzi IDA 2013]
- Input: an LPAD; logical interpretations (data); *target* predicate(s)
- All ground atoms in the interpretations for the target predicate(s) correspond to as many queries
- BDDs encode the explanations for each query
- Expectations computed with two passes over the BDDs

Dipartimento di Matematica e Informatica

EMBLEM

- EMBLEM encodes multi-valued random variable with Boolean random variables
- Variable X_{ij} associated with grounding θ_j of clause C_i having n values.
- Encoding using $n-1$ Boolean variables $X_{ij1}, \dots, X_{ijn-1}$.
- Equation $X_{ij} = k$ for $k = 1, \dots n-1$ represented by

$$\overline{X_{ij1}} \wedge \dots \wedge \overline{X_{ijk-1}} \wedge X_{ijk}$$

- Equation $X_{ij} = n$ represented by

$$\overline{X_{ij1}} \wedge \dots \wedge \overline{X_{ijn-1}}.$$

- Parameters:

$$\begin{aligned} P(X_{ij1}) &= P(X_{ij} = 1) \\ &\dots \\ P(X_{ijk}) &= \frac{P(X_{ij} = k)}{\prod_{l=1}^{k-1}(1 - P(X_{ijk-1}))} \end{aligned}$$

EMBLEM

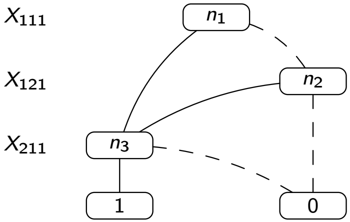
- Let X_{ijk} for $k = 1, \ldots, n_i - 1$ and $j \in g(i)$ be the Boolean random variables associated with grounding $C_i\theta_j$ of clause C_i of $\mathcal{P}$ where n_i is the number of head atoms of C_i and $g(i)$ is the set of indices of grounding substitutions of C_i.

Dipartimento di Matematica e Informatica

Example

$$\begin{aligned}
C_1 &= \mathit{epidemic} : 0.6 \; ; \; \mathit{pandemic} : 0.3 \leftarrow \mathit{flu}(X), \mathit{cold}.\\
C_2 &= \mathit{cold} : 0.7.\\
C_3 &= \mathit{flu}(\mathit{david}).\\
C_4 &= \mathit{flu}(\mathit{robert}).
\end{aligned}$$

- Clause C_1: two groundings, first: X_{111} and X_{112}, latter: X_{121} and X_{122}.
- C_2: single grounding, random variable X_{211}.



EMBLEM

- EMBLEM alternates between the two phases:
  - Expectation: compute $\mathbf{E}[c_{ik0}|e]$ and $\mathbf{E}[c_{ik1}|e]$ for all examples e, rules C_i in $\mathcal{P}$ and $k = 1, \ldots, n_i - 1$, where c_{ikx} is the number of times a variable X_{ijk} takes value x for $x \in \{0, 1\}$, with j in $g(i)$.

$$\mathbf{E}[c_{ikx}|e] = \sum_{j \in g(i)} P(X_{ijk} = x|e).$$

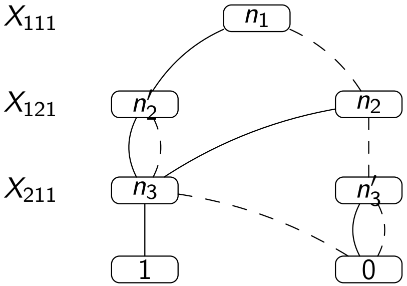
  - Maximization: compute π_{ik} for all rules C_i and $k = 1, \ldots, n_i - 1$.

$$\pi_{ik} = \frac{\sum_{e \in E} \mathbf{E}[c_{ik1}|e]}{\sum_{q \in E} \mathbf{E}[c_{ik0}|e] + \mathbf{E}[c_{ik1}|e]}$$

Dipartimento
di Matematica
e Informatica

EMBLEM

- $P(X_{ijk} = x|e)$ is given by $P(X_{ijk} = x|e) = \frac{P(X_{ijk}=x,e)}{P(e)}$.
- Consider a BDD for an example e built by applying only the merge rule



EMBLEM

- $P(e)$ is given by the sum of the probabilities of all the paths in the BDD from the root to a 1 leaf
- To compute $P(X_{ijk} = x, e)$ we need to consider only the paths passing through the x-child of a node n associated with variable X_{ijk} so

$$P(X_{ijk} = x, e) \quad = \quad \sum_{n \in N(X_{ijk})} \pi_{ikx} F(n) B(child_x(n)) = \sum_{n \in N(X_{ijk})} e^x(n)$$

  - $F(n)$ is the *forward probability*, the probability mass of the paths from the root to n,
  - $B(n)$ is the *backward probability*, the probability mass of paths from n to the 1 leaf.

Dipartimento
di Matematica
e Informatica

EMBLEM

- BDD obtained by also applying the deletion rule: paths where there is no node associated with X_{ijk} can also contribute to $P(X_{ijk} = x, e)$.
- Suppose the BDD was obtained deleting node m child of n associated with variable X_{ijk}
- Outgoing edges of m both point to $child_0(n)$.
- The probability mass of the two paths that were merged was $e^0(n)(1 - \pi_{ik})$ and $e^0(n)\pi_{ik}$ for
- The first quantity contributes to $P(X_{ijk} = 0, e)$, the latter to $P(X_{ijk} = 1, e)$.

Dipartimento di Matematica e Informatica

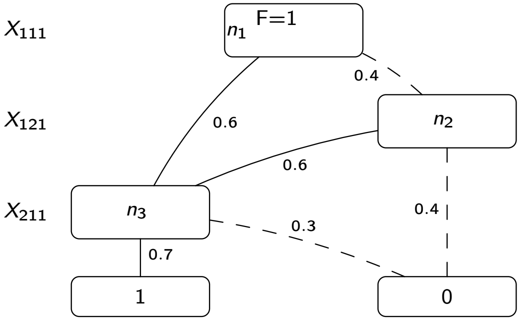
GetForward

```
procedure GetForward($root$)
    $F(root) = 1$
    $F(n) = 0$ for all nodes
    for $l = 1$ to $levels$ do          ▷ $levels$ is the number of levels of the BDD rooted at $root$
        $Nodes(l) = \emptyset$
    end for
    $Nodes(1) = \{root\}$
    for $l = 1$ to $levels$ do
        for all $node \in Nodes(l)$ do
            let $X_{ijk}$ be $v(node)$, the variable associated with $node$
            if $child_0(node)$ is not terminal then
                $F(child_0(node)) = F(child_0(node)) + F(node) \cdot (1 - \pi_{ik})$
                add $child_0(node)$ to $Nodes(level(child_0(node)))$
            end if
            if $child_1(node)$ is not terminal then
                $F(child_1(node)) = F(child_1(node)) + F(node) \cdot \pi_{ik}$
                add $child_1(node)$ to $Nodes(level(child_1(node)))$
            end if
        end for
    end for
end procedure
```

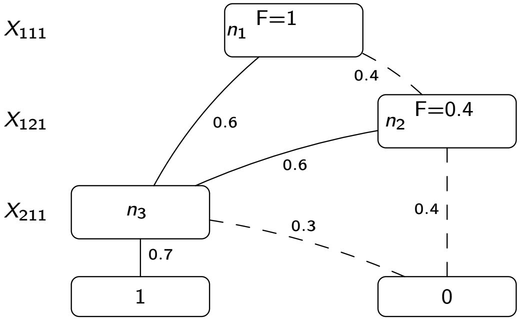
GetBackward

```
function GetBackward($node$)
    if $node$ is a terminal then
        return $value(node)$
    else
        let $X_{ijk}$ be $v(node)$
        $B(child_0(node))$ =GetBackward($child_0(node)$)
        $B(child_1(node))$ =GetBackward($child_1(node)$)
        $e^0(node) = F(node) \cdot B(child_0(node)) \cdot (1 - \pi_{ik})$
        $e^1(node) = F(node) \cdot B(child_1(node)) \cdot \pi_{ik}$
        $\eta^0(i,k) = \eta^0_t(i,k) + e^0(node)$
        $\eta^1(i,k) = \eta^1_t(i,k) + e^1(node)$
        take into account deleted paths
        return $B(child_0(node)) \cdot (1 - \pi_{ik}) + B(child_1(node)) \cdot \pi_{ik}$
    end if
end function
```

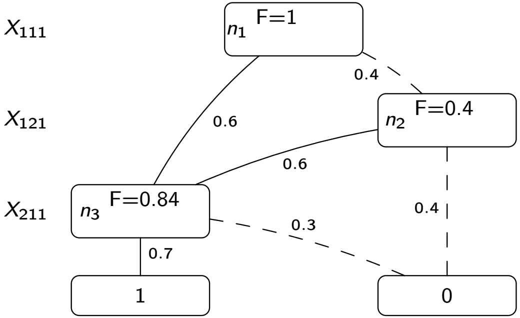
Example



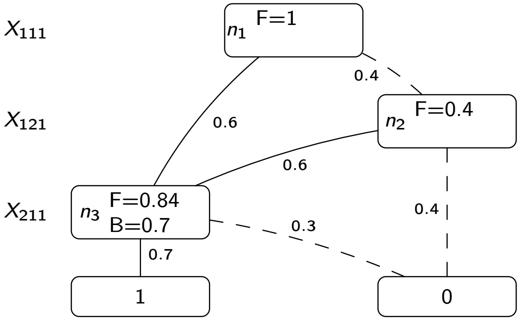
Example



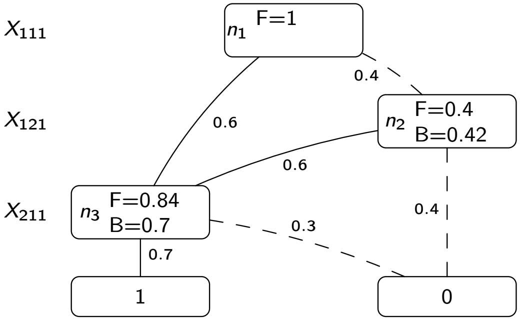
Example



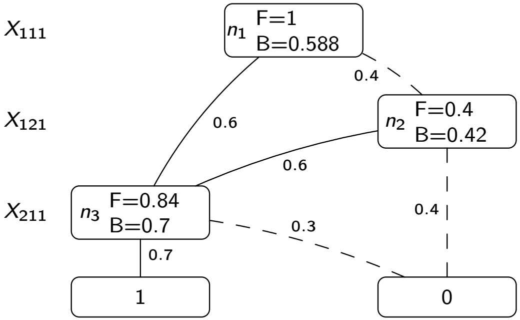
Example



Example



Example



X_{111}
X_{121}
X_{211}
n_1 F=1 B=0.588
n_2 F=0.4 B=0.42
n_3 F=0.84 B=0.7
0.6
0.4
0.6
0.4
0.3
0.7
1
0

LeProbLog

$$
\begin{array}{l}
0.1 :: burglary.\\
0.2 :: earthquake.\\
0.7 :: hears\_alarm(X) \leftarrow person(X).\\
alarm \leftarrow burglary.\\
alarm \leftarrow earthquake.\\
calls(X) \leftarrow alarm, hears\_alarm(X).\\
person(mary).\\
person(john).
\end{array}
$$

$q = burglary \;\; e = calls(john)$

LeProbLog

- LeProbLog [Gutmann et al PKDD 2008]

Definition (LeProbLog parameter learning problem)

Given a ProbLog program $\mathcal{P}$ and a set of training examples $E = \{(e_1, p_i), \ldots, (e_T, p_T)\}$ where e_t is a ground atom and $p_t \in [0, 1]$ for $t = 1, \ldots, T$, find the parameter of the program so that the mean squared error

$$MSE = \frac{1}{T} \sum_{t=1}^{T} (P(e_t) - p_t)^2$$

is minimized.

LeProbLog

- Gradient descent: it iteratively updates the parameters in the opposite direction of the gradient.
- Gradient
$$\frac{\partial MSE}{\partial \Pi_j} = \frac{2}{T}\sum_{t=1}^{T}(P(e_t) - p_t) \cdot \frac{\partial P(e_t)}{\partial \Pi_j}$$
- LeProbLog compiles queries to BDDs
- To compute $\frac{\partial P(e_t)}{\partial \Pi_j}$, it uses a dynamic programming algorithm that traverses the BDD bottom up

Dipartimento di Matematica e Informatica

LeProbLog

- If $f(\mathbf{X})$ is the Boolean function represented by the BDD:

$$\frac{\partial P(e_t)}{\partial \Pi_j} = \frac{\partial P(f(\mathbf{X}))}{\partial \Pi_j}$$

$$f(\mathbf{X}) = X_k \cdot f^{X_k}(\mathbf{X}) + \neg X_k \cdot f^{\neg X_k}(\mathbf{X})$$

$$P(f(\mathbf{X})) = \Pi_k \cdot P(f^{X_k}(\mathbf{X})) + (1-\Pi_k) \cdot P(f^{\neg X_k}(\mathbf{X}))$$

$$\frac{\partial P(f(\mathbf{X}))}{\partial \Pi_j} = P(f^{X_k}(\mathbf{X})) - P(f^{\neg X_k}(\mathbf{X}))$$

  if $k = j$, or

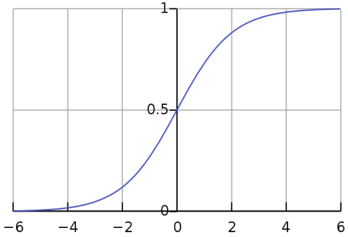
$$\frac{\partial P(f(\mathbf{X}))}{\partial \Pi_j} = \Pi_k \cdot \frac{\partial P(f^{X_k}(\mathbf{X}))}{\partial \Pi_j} + (1-\Pi_k) \cdot \frac{P(f^{\neg X_k}(\mathbf{X}))}{\partial \Pi_j}$$

  if $k \neq j$.
- If X_j does not appear in $\mathbf{X}$ $\frac{\partial P(f(\mathbf{X}))}{\partial \Pi_j} = 0$

Dipartimento di Matematica e Informatica

LeProbLog

- We have to ensure that the parameters remain in the $[0, 1]$ interval.
- Reparameterization by means of the sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$



- Each parameter is expressed as $\Pi_j = \sigma(a_j)$ and the a_js are used as the parameters
- Using the chain rule of derivatives

$$\frac{\partial P(e_t)}{\partial a_j} = \sigma(a_j) \cdot (1 - \sigma(a_j)) \frac{\partial P(f(\mathbf{X}))}{\partial \Pi_j}$$

ProbLog2

- ProbLog2 includes LFI-ProbLog [Gutmann et al PKDD 2011] that learns the parameters of ProbLog programs from partial interpretations.
- Partial interpretations specify the truth value of some but not necessarily all ground atoms.
- $\mathcal{I} = \langle I_T, I_F \rangle$: the atoms in I_T are true and those in I_F are false.
- $\mathcal{I} = \langle I_T, I_F \rangle$ can be associated with a conjunction $q(\mathcal{I}) = \bigwedge_{a \in I_T} a \wedge \bigwedge_{a \in I_F} \sim a$.

LFI-ProbLog

Definition (LFI-ProbLog learning problem)

Given a ProbLog program $\mathcal{P}$ with unknown parameters and a set $E = \{\mathcal{I}_1, \ldots, \mathcal{I}_T\}$ of partial interpretations (the examples), find the value of the parameters $\mathbf{\Pi}$ of $\mathcal{P}$ that maximize the likelihood of the examples, i.e., solve

$$\arg\max_{\mathbf{\Pi}} P(E) = \arg\max_{\mathbf{\Pi}} \prod_{t=1}^{T} P(q(\mathcal{I}_t))$$

Dipartimento di Matematica e Informatica

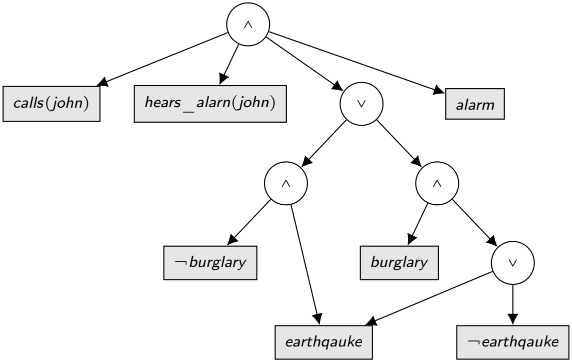
LFI-ProbLog

- EM algorithm
- A d-DNNF circuit for each partial interpretation $\mathcal{I} = \langle I_T, I_F \rangle$ by using the ProbLog2 inference algorithm with the evidence $q(\mathcal{I})$.
- A Boolean random variable X_{ij} is associated with each ground probabilistic fact $f_i\theta_j$.
- For each example $\mathcal{I}$, variable X_{ij} and $x \in \{0, 1\}$, LFI-ProbLog computes $P(X_{ij} = x|\mathcal{I})$.
- LFI-ProbLog computes $P(X_{ij} = x|\mathcal{I})$ by computing $P(X_{ij} = x, \mathcal{I})$ using Procedure CircP

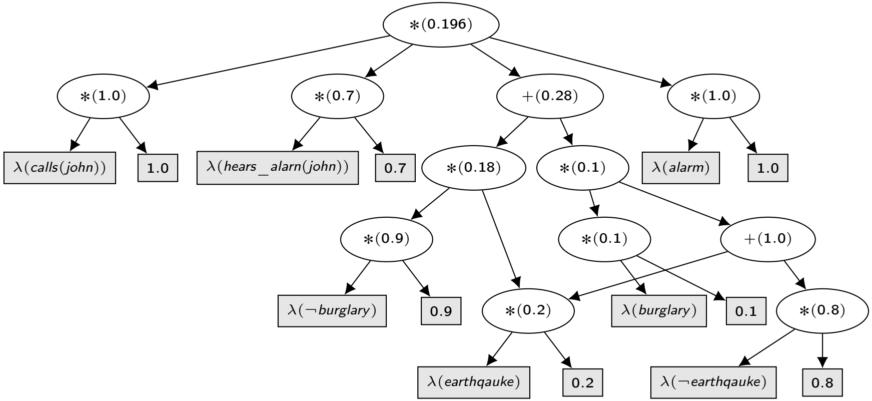
Dipartimento
di Matematica
e Informatica

Example of a d-DNNF Formula

$$q = burglary \;\; e = calls(john)$$



Example of a d-DNNF Circuit



Computing Expectations

$$WMC(\phi) = \sum_{\omega \in SAT(\phi)} \prod_{l \in \omega} w(l)\lambda_l = \sum_{\omega \in SAT(\phi)} \prod_{l \in \omega} w(l) \prod_{l \in \omega} \lambda_l$$

$$P(e) = \sum_{\omega \in SAT(\phi)} \prod_{l \in \omega} w(l)$$

- We want to compute $P(q|e)$ for all atoms $q \in Q$.
- Partial derivative $\frac{\partial f}{\partial \lambda_q}$ for an atom q:

$$\begin{aligned}\frac{\partial f}{\partial \lambda_q} &= \sum_{\omega \in SAT(\phi), q \in \omega} \prod_{l \in \omega} w(l) \prod_{l \in \omega, l \neq q} \lambda_l = \\ & \sum_{\omega \in SAT(\phi), q \in \omega} \prod_{l \in \omega} w(l) = \\ & P(e, q)\end{aligned}$$

Computing Expectations

- If we compute the partial derivatives of f for all indicator variables λ_q, we get $P(q, e)$ for all atoms q.
- $v(n)$: value of each node n
- $d(n) = \frac{\partial v(r)}{\partial v(n)}$.
- $d(r) = 1$
- By the chain rule of calculus, for an arbitrary non-root node n with p indicating its parents

$$d(n) = \sum_p \frac{\partial v(r)}{\partial v(p)} \frac{\partial v(p)}{\partial v(n)} = \sum_p d(p) \frac{\partial v(p)}{\partial v(n)}.$$

Dipartimento di Matematica e Informatica

Computing Expectations

- If p is a multiplication node with n' indicating its children

$$\frac{\partial v(p)}{\partial v(n)} = \frac{\partial v(n) \prod_{n' \neq n} v(n')}{\partial v(n)} = \prod_{n' \neq n} v(n').$$

- If parent p is an addition node with n' indicating its children

$$\frac{\partial v(p)}{\partial v(n)} = \frac{\partial v(n) + \sum_{n' \neq n} v(n')}{\partial v(n)} = 1.$$

- $+p$ an addition parent of n and $*p$ a multiplication parent of n:

$$d(n) = \sum_{+p} d(+p) + \sum_{*p} d(*p) \prod_{n' \neq n} v(n').$$

- If $v(n) \neq 0$.

$$d(n) = \sum_{+p} d(+p) + \sum_{*p} d(*p) v(*p)/v(n).$$

CircP

```
procedure CircP(circuit)
    assign values to leaves
    for all non-leaf node n with children c (visit children before parents) do
        if n is an addition node then
            $v(n) \leftarrow \sum_c v(c)$
        else
            $v(n) \leftarrow \prod_c v(c)$
        end if
    end for
    $d(r) \leftarrow 1$, $d(n) = 0$ for all non-root nodes
    for all non-root node n (visit parents before children) do
        for all parents p of n do
            if p is an addition parent then
                $d(n) = d(n) + d(p)$
            else
                $d(n) \leftarrow d(n) + d(p)v(p)/v(n)$
            end if
        end for
    end for
end procedure
```

Structure Learning for LPADs

- Given a set of interpretations (data)
- *Find the model and the parameters* that maximize the probability of the data (log-likelihood)
- SLIPCOVER: Structure LearnIng of Probabilistic logic program by searching OVER the clause space [Riguzzi & Bellodi TPLP 2015]
  1. Beam search in the space of clauses to find the promising ones
  2. Greedy search in the space of probabilistic programs guided by the LL of the data.
- *Parameter learning* by means of EMBLEM

Dipartimento di Matematica e Informatica

SLIPCOVER

- Cycle on the set of predicates that can appear in the head of clauses, either target or background
- For each predicate, beam search in the space of clauses
- The initial set of beams is generated by building a set of *bottom clauses* as in Progol [Muggleton NGC 1995]
- Bottom clause: most specific clause covering an example

Language Bias

- Mode declarations as in Progol
- Syntax

```
modeh(RecallNumber,PredicateMode).
modeb(RecallNumber,PredicateMode).
```

- `RecallNumber` can be a number or *. Usually *. Maximum number of answers to queries to include in the bottom clause

Mode Declarations

- `PredicateMode` template of the form:

```
p(ModeType, ModeType,...)
```

- Some examples:

```
modeb(1,mem(+number,+list)).
modeb(1,dec(+integer,-integer)).
modeb(1,mult(+integer,+integer,-integer)).
modeb(1,plus(+integer,+integer,-integer)).
modeb(1,(+integer)=(#integer)).
modeb(*,has_car(+train,-car))
```

Dipartimento di Matematica e Informatica

Mode Declarations

- `ModeType` can be:
  - Simple:
    - `+T` input variables of type `T`;
    - `-T` output variables of type `T`; or
    - `#T`, `-#T` constants of type `T`.
  - Structured: of the form `f(..)` where `f` is a function symbol and every argument can be either simple or structured. For example:

```
modeb(1,mem(+number,[+number|+list])).
```

Bottom Clause $\perp$

- Most specific clause covering an example e
- Form: $e \leftarrow B$
- B: set of ground literals that are true regarding the example e
- B obtained by considering the constants in e and querying the data for true atoms regarding these constants
- Values for output arguments are used as input arguments for other predicates
- A map from types to lists of constants is kept, it is enlarged with constants in the answers to the queries and the procedure is iterated a user-defined number of times
- `#T` arguments are instantiated in calls, `-#T` aren't and the values after the call are added to the list of constants
- `-#T` arguments can be used to retrieve values for `T`, `#T` can't

Dipartimento di Matematica e Informatica

Bottom Clause $\perp$

- Initialize to empty a map m from types to lists of values
- Pick a $modeh(r, s)$, an example e matching s, add to $m(T)$ the values of $+T$ arguments in e
- For $i = 1$ to d
    - For each $modeb(r, s)$

Dipartimento di Matematica e Informatica

Bottom Clause $\perp$

- For each possible way of building a query q from s by replacing $+T$ and $\#T$ arguments with constants from $m(T)$ and all other arguments with variables
  - Find all possible answers for q and put them in a list L
  - $L' := r$ elements sampled from L
  - For each $l \in L'$, add the values in l corresponding to $-T$ or $-\#T$ to $m(T)$

Dipartimento di Matematica e Informatica

Bottom Clause ⊥

- Example:

$e = father(john, mary)$
$BG = \{parent(john, mary), parent(david, steve),$
$parent(kathy, mary), female(kathy), male(john), male(david)\}$
$modeh(father(+person, +person)).$
$modeb(parent(+person, -person)).$
$modeb(parent(-\#person, +person)).$
$modeb(male(+person)). \quad modeb(female(\#person)).$
$e \leftarrow B = father(john, mary) \leftarrow parent(john, mary), male(john),$
$parent(kathy, mary), female(kathy).$

Dipartimento di Matematica e Informatica

Bottom Clause $\perp$

- The resulting ground clause $\perp$ is then processed by replacing each term in a + or - placemarker with a variable
- An input variable (+T) must appear as an output variable with the same type in a previous literal and a constant (#T or -#T) is not replaced by a variable.

$\perp = father(X, Y) \leftarrow$
$parent(X, Y), male(X), parent(kathy, Y), female(kathy).$

Dipartimento
di Matematica
e Informatica

Determination

```
determination(pred1/n1,pred2/n2).
```

- indicates that `pred2/n2` can appear in the body of clauses for predicate `pred1/n1`
- As in Progol

Head Declarations

- To generate clauses with more than two head atoms, head declarations of the form

$$modeh(r, [s_1, \ldots, s_n], [a_1, \ldots, a_n], [P_1/Ar_1, \ldots, P_k/Ar_k])$$

- $s_1, \ldots, s_n$ are schemas
- $a_1, \ldots, a_n$ are atoms such that a_i is obtained from s_i by replacing placemarkers with variables
- P_i/Ar_i are the predicates admitted in the body.
- $a_1, \ldots, a_n$ are used to indicate which variables should be shared by the atoms in the head.
- The generation of a bottom clause is the same except for the fact that the goal to call is composed of more than one atom.

Dipartimento di Matematica e Informatica

Head Declarations

- Goal $a_1, \ldots, a_n$ is called and r answers that ground all a_is are kept
- Resulting bottom clauses $a_1 \ ; \ \ldots \ ; \ a_n :- \ b_1, \ldots, b_m$
- The initial beam contains clauses with an empty body of the form

$$a_1 : \frac{1}{n+1} \ ; \ \ldots \ ; \ a_n : \frac{1}{n+1}.$$

SLIPCOVER

- The initial beam associated with predicate P/Ar of h will contain the clause with the empty body $h : 0.5.$ for each bottom clause $h :- b_1, \ldots, b_m$
- In each iteration of the cycle over predicates, it performs a beam search in the space of clauses for the predicate.
- The beam contains couples $(Cl, LIterals)$ where $Literals = \{b_1, \ldots, b_m\}$
- For each clause Cl of the form $Head :- Body$, the refinements are computed by adding a literal from $Literals$ to the body.

Dipartimento di Matematica e Informatica

SLIPCOVER

- The tuple $(Cl', Literals')$ indicates a refined clause Cl' together with the new set $Literals'$
- EMBLEM is then executed for a theory composed of the single refined clause.
- LL is used as the score of the updated clause $(Cl'', Literals')$.
- $(Cl'', Literals')$ is then inserted into a list of promising clauses.
- Two lists are used, TC for target predicates and BC for background predicates.
- These lists ave a maximum size

SLIPCOVER

- After the clause search phase, SLIPCOVER performs a greedy search in the space of theories:
  - it starts with an empty theory and adds a target clause at a time from the list TC.
  - After each addition, it runs EMBLEM and computes the LL of the data as the score of the resulting theory.
  - If the score is better than the current best, the clause is kept in the theory, otherwise it is discarded.
- Finally, SLIPCOVER adds all the clauses in BC to the theory and performs parameter learning on the resulting theory.

Dipartimento di Matematica e Informatica

Execution Example

- UW-CSE dataset: 22 different predicates, such as advisedby/2, yearsinprogram/2 and taughtby/3.
- The aim is to predict the predicate advisedby/2
- The language bias includes

```
modeh(*,advisedby(+person,+person)).
modeh(*,[advisedby(+person,+person),tempadvisedby(+person,+person)],
  [advisedby(A,B),tempadvisedby(A,B)],
  [professor/1,student/1,hasposition/2,inphase/2,publication/2,
  taughtby/3,ta/3,courselevel/2,yearsinprogram/2]).

modeh(*,[student(+person),professor(+person)],
  [student(P),professor(P)],
  [hasposition/2,inphase/2,taughtby/3,ta/3,courselevel/2,
  yearsinprogram/2,advisedby/2,tempadvisedby/2]).

modeh(*,[inphase(+person,pre_quals),inphase(+person,post_quals),
  inphase(+person,post_generals)],
  [inphase(P,pre_quals),inphase(P,post_quals),inphase(P,post_generals)],
  [professor/1,student/1,taughtby/3,ta/3,courselevel/2,
  yearsinprogram/2,advisedby/2,tempadvisedby/2,hasposition/2]).
```

Dipartimento di Matematica e Informatica

Execution Example

- *modeb* declarations such as

```
modeb(*,courselevel(+course, -level)).
modeb(*,courselevel(+course, #level)).
```

Execution Example

- Example of a two-head bottom clause generated from the first *modeh* declaration

```
advisedby(A,B):0.5 :- professor(B),student(A),hasposition(B,C),
  hasposition(B,faculty),inphase(A,D),inphase(A,pre_quals),
  yearsinprogram(A,E),taughtby(F,B,G),taughtby(F,B,H),taughtby(I,B,J),
  taughtby(I,B,J),taughtby(F,B,G),taughtby(F,B,H),
  ta(I,K,L),ta(F,M,H),ta(F,M,H),ta(I,K,L),ta(N,K,O),ta(N,A,P),
  ta(Q,A,P),ta(R,A,L),ta(S,A,T),ta(U,A,O),ta(U,A,O),ta(S,A,T),
  ta(R,A,L),ta(Q,A,P),ta(N,K,O),ta(N,A,P),ta(I,K,L),ta(F,M,H).
```

Dipartimento
di Matematica
e Informatica

Execution Example

- Example of a multi-head bottom clause generated from the second *modeh* declaration

```
student(A):0.33; professor(A):0.33 :- inphase(A,B),
  inphase(A,post_generals),
  yearsinprogram(A,C).
```

Execution Example

- Example of a refinement from the first bottom clause is

  `advisedby(A,B):0.5 :- professor(B).`
- EMBLEM is applied to the theory, the only parameter is updated obtaining:

  `advisedby(A,B):0.108939 :- professor(B).`
- The clause is further refined to

  `advisedby(A,B):0.108939 :- professor(B),hasposition(B,C).`

Dipartimento di Matematica e Informatica

Execution Example

- Example of a refinement that is generated from the second bottom clause is

  ```
  student(A):0.33; professor(A):0.33 :- inphase(A,B).
  ```

- Updated refinement after EMBLEM

  ```
  student(A):0.5869;professor(A):0.09832 :- inphase(A,B).
  ```

Dipartimento di Matematica e Informatica

Execution Example

- When searching the *space of theories* for the target predicate `advisedby`, SLIPCOVER generates the program:

```
advisedby(A,B):0.1198 :- professor(B),inphase(A,C).
advisedby(A,B):0.1198 :- professor(B),student(A).
```

  with a LL of -350.01.

- After EMBLEM we get:

```
advisedby(A,B):0.05465 :- professor(B),inphase(A,C).
advisedby(A,B):0.06893 :- professor(B),student(A).
```

  with a LL of -318.17.

- Since the LL increased, the last clause is retained and at the next iteration a new clause is added:

```
advisedby(A,B):0.12032 :- hasposition(B,C),inphase(A,D).
advisedby(A,B):0.05465 :- professor(B),inphase(A,C).
advisedby(A,B):0.06893 :- professor(B),student(A).
```

Dipartimento di Matematica e Informatica

ProbFOIL+

- ProbFOIL+ [De Raedt et al IJCAI 2015] learn rules from probabilistic examples.

Definition (ProbFoil+ learning problem)

Given

1. a set of training examples $E = \{(e_1, p_1), \ldots, (e_T, p_T)\}$ where each e_i is a ground fact for a target predicate
2. a background theory $\mathcal{B}$ containing information about the examples in the form of a ProbLog program
3. a space of possible clauses $\mathcal{L}$

find a hypothesis $H \subseteq \mathcal{L}$ so that the absolute error $AE = \sum_{i=1}^{T} |P(e_i) - p_i|$ is minimized, i.e.,

$$\arg\min_{H \in \mathcal{L}} \sum_{i=1}^{T} |P(e_i) - p_i|$$

ProbFOIL+

- Form of clauses: $x :: h \leftarrow B$, with $x \in [0, 1]$.
- To be interpreted as
  $h \leftarrow B, prob(id).$
  $x :: prob(id).$
- Different from an LPAD $h : x \leftarrow B$, as this stands for the union of ground rules $h' : x \leftarrow B'$. obtained by grounding $h : x \leftarrow B$.

ProbFOIL+

- ProbFOIL+ generalizes mFOIL and FOIL
- Covering loop: one rule is added to the theory at each iteration.
- Clause search loop: builds the rule by iteratively adding literals to the body.
- The covering loop ends when a condition based on a global scoring function is satisfied.
- Clause search loop: beam search using a local scoring function as the heuristic.

Dipartimento di Matematica e Informatica

ProbFOIL+

```
function ProbFOIL+($target$)
    $H \leftarrow \emptyset$
    while true do
        $clause \leftarrow$ LearnRule($H, target$)
        if GScore($H$) $<$ GScore($H \cup \{clause\}$) $\wedge$ Significant($H, clause$) then
            $H \leftarrow H \cup \{clause\}$
        else
            return $H$
        end if
    end while
end function
```

ProbFOIL+

```
function LearnRule($H$, $target$)
    $candidates \leftarrow \{x :: target \leftarrow true\}$
    $best \leftarrow (x :: target \leftarrow true)$
    while $candidates \neq \emptyset$ do
        $next\_cand \leftarrow \emptyset$
        for all $x :: target \leftarrow body \in candidates$ do
            for all $(target \leftarrow bod, refinement) \in \rho(target \leftarrow body)$ do
                if not Reject($H, best, (x :: target \leftarrow body, refinement)$) then
                    $next\_cand \leftarrow next\_cand \cup \{(x :: target \leftarrow body, refinement)\}$
                    if LScore($H, (x :: target \leftarrow body, refinement)$) $>$ LScore($H, best$) then
                        $best \leftarrow (x :: target \leftarrow body, refinement)$
                    end if
                end if
            end for
        end for
        $candidates \leftarrow next\_cand$
    end while
    return $best$
end function
```

ProbFOIL+

- Global scoring function: accuracy over the dataset, given by

$$accuracy_H = \frac{TP_H + TN_H}{T}$$

  where T is number of examples and TP_H and TN_H are, respectively, the number of *true positives* and of *true negatives*

- Local scoring function: an *m-estimate* of the *precision*

$$m\text{-}estimate_H = \frac{TP_H + m\frac{P}{P+N}}{TP_H + FP_H + m}$$

Dipartimento di Matematica e Informatica

ProbFOIL+

- Each example e_i is associated with a probability p_i.
- An example (e_i, p_i) contributes a part p_i to the positive part of training set and $1 - p_i$ to the negative part: $P = \sum_{i=1}^{T} p_i$ and $N = \sum_{i=1}^{T}(1 - p_i)$.
- Hypothesis H assigns a probability $p_{H,i}$ to each example e_i
- The contribution $tp_{H,i}$ of example e_i to TP_H will be $p_{H,i}$ if $p_i > p_{H,i}$ and p_i otherwise, because if $p_i < p_{H,i}$ the hypothesis is overestimating e_i.
- The contribution $fp_{H,i}$ of example e_i to FP_H will be $p_{H,i} - p_i$ if $p_i < p_{H,i}$ and 0 otherwise, because if $p_i > p_{H,i}$ the hypothesis is underestimating e_i.
- $TP_H = \sum_{i=1}^{T} tp_{H,i}$, $FP_H = \sum_{i=1}^{T} fp_{H,i}$, $TN_H = N - FP_H$ and $FN_H = P - TP_H$

ProbFOIL+

- LScore$(H, x :: C)$ computes the local scoring function for the addition of clause $C(x) = x :: C$ to H
- The heuristic depends on the value of $x \in [0, 1]$.
- Find the value of x that maximizes the score

$$M(x) = \frac{TP_{H \cup C(x)} + mP/T}{TP_{H \cup C(x)} + FP_{H \cup C(x)} + m}.$$

- We need to compute $TP_{H \cup C(x)}$ and $FP_{H \cup C(x)}$, $tp_{H \cup C(x),i}$ and $fp_{H \cup C(x),i}$ as a function of x.

ProbFOIL+

- $M(x)$ is a piecewise function where each piece is of the form

$$\frac{Ax+B}{Cx+D}$$

  with A, B, C and D constants.
- The derivative of a piece is

$$\frac{dM(x)}{dx} = \frac{AD - BC}{(Cx+D)^2}$$

- It is either 0 or different from 0 everywhere in each interval so the maximum of $M(x)$ can only occur at the x_is values that are the endpoints of the intervals.
- Compute the value of $M(x)$ for each x_i and pick the maximum.
- Ordering the x_i values

Dipartimento di Matematica e Informatica

ProbFOIL+

- ProbFOIL+ prunes refinements when
  - they cannot lead to a local score higher than the current best,
  - they cannot lead to a global score higher than the current best or
  - they are not significant, i.e., when they provide only a limited contribution.
- By adding a literal to a clause, the true positives and false positives can only decrease, so we can obtain an upper bound of the local score by setting the false positives to 0 and computing the m-estimate.
- By adding a clause to a theory, the true positives and false positives can only increase, so if the number of true positives of $H \cup C(x)$ is not larger than the true positives of H, the refinement $C(x)$ can be discarded.
- *significance test* based on the *likelihood ratio statistics*.

Dipartimento di Matematica e Informatica

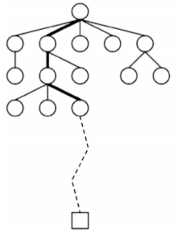
SLIPCASE: *Structure LearnIng of ProbabilistiC logic progrAmS with Em over bdds*

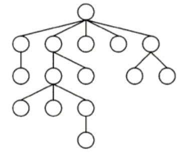
- Input: simple initial *Theory*
- Compute optimum parameters and log-likelihood *LL* of the data for *Theory* with EMBLEM
- best theory=*Theory*, best likelihood=*LL*
- Beam Search
  1. Beam: the *N* theories with the highest log-likelihood, initially *Theory*
  2. Remove the 1st theory from beam → theory refinements:
     - language bias with modeh/modeb declarations
     - *+/- literal in a clause and +/- clause*
  3. Estimate LL for each refinement with *Nmax* iterations of EMBLEM
  4. Update (best theory,best likelihood)
  5. Insert the refinement in the beam, ordered by likelihood
  6. Remove the refinements exceeding the size of the beam
- Stop search after *MaxSteps* iterations or if empty Beam
- EMBLEM over best theory

Dipartimento di Matematica e Informatica

Monte Carlo Tree Search

- MCTS: take random samples in the decision space and build a search tree in an incremental and asymmetric manner
- First a *tree policy* is used in order to find the most urgent node of the tree to expand
- Then a *simulation* phase is conducted from the selected node, by adding a new child node and using a *default policy* that suggests the sequence of actions ("simulation") to be chosen from this new node.
- Finally, the simulation result is *backpropagated* upwards to update the statistics of the nodes.





LEMUR: *LEarning with a Monte carlo Upgrade of tRee search*

- We consider each logic theory as a bandit problem, where each legal theory revision is an arm with unknown reward
- Tree policy: LEMUR selects one move, corresponding to a possible theory revision, according to a formula
- LEMUR descends to the selected child node and selects a new move until it reaches a leaf
- Then LEMUR starts the Monte Carlo simulation phase to score the theory at this leaf
- One random sequence of revisions is applied starting from the leaf theory until a *finite unknown horizon* is reached
- LEMUR stops the simulation after k steps, where k is a uniformly sampled random integer smaller than d, an input parameter.
- Once the horizon is reached, LEMUR produces a reward value Δ.

Dipartimento di Matematica e Informatica

LEMUR: *LEarning with a Monte carlo Upgrade of tRee search*

- The nodes visited in the tree policy are saved with their statistics: the visit count n_j, the average reward $\overline{X}_j$ and the score L_j
- In the simulation phase, all the visited nodes are scored by computing their log-likelihood using EMBLEM as in the tree policy, and the reward Δ corresponds to the maximum score obtained in this random descent.
- Δ is backpropagated up the sequence of nodes selected for this iteration to update the node statistics: for each node j, its visit count is incremented and its average reward $\overline{X}_j$ is updated according to Δ.

Dipartimento di Matematica e Informatica

Hierarchical PLP

- Learning probabilistic logic programs is expensive due to the high cost of inference.
- A restriction of the language of Logic Programs with Annotated Disjunctions called hierarchical PLP in which clauses and predicates are hierarchically organized.
- Inference is then much cheaper.

Hierarchical PLP

- We want to compute the probability of atoms for a predicate r: $r(\vec{t})$, where $\vec{t}$ is a vector of constants.
- $r(\vec{t})$ can be an example in a learning problem and r a target predicate.
- A specific form of an LPADs defining r in terms of the input predicates.
- The program defined r using a number of input and hidden predicates disjoint from input and target predicates.
- Each rule in the program has a single head atom annotated with a probability.
- The program is hierarchically defined so that it can be divided into layers.

Hierarchical PLP

- Each layer contains a set of hidden predicates that are defined in terms of predicates of the layer immediately below or in terms of input predicates.
- Extreme form of program stratification: stronger than acyclicity [Apt NGC91] because it is imposed on the predicate dependency graph, and is also stronger than stratification [Chandra, Harel JLP85] that allows clauses with positive literals built on predicates in the same layer.
- It prevents inductive definitions and recursion in general, thus making the language not Turing-complete.

Dipartimento di Matematica e Informatica

Hierarchical PLP

- Generic clause C:

$$C = p(\vec{X}) : \pi :- \phi(\vec{X}, \vec{Y}), b_1(\vec{X}, \vec{Y}), \ldots, b_m(\vec{X}, \vec{Y})$$

  where $\phi(\vec{X}, \vec{Y})$ is a conjunction of literals for the input predicates using variables $\vec{X}, \vec{Y}$.
- $b_i(\vec{X}, \vec{Y})$ for $i = 1, \ldots, m$ is a literal built on a hidden predicate.
- $\vec{Y}$ is a possibly empty vector of variables existentially quantified with scope the body.
- Literals for hidden predicates must use the whole set of variables $\vec{X}, \vec{Y}$.
- The predicate of each $b_i(\vec{X}, \vec{Y})$ does not appear elsewhere in the body of C or in the body of any other clause.

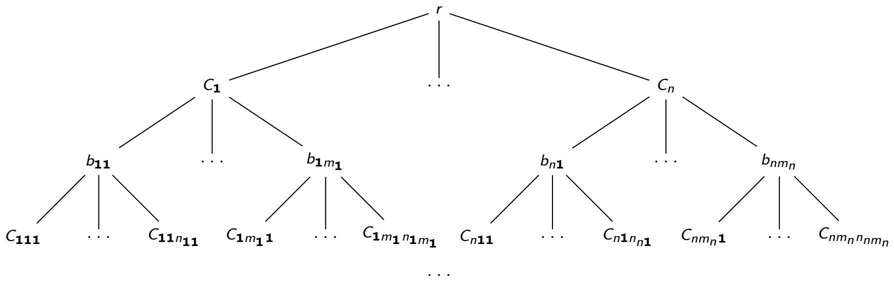
Dipartimento di Matematica e Informatica

Hierarchical PLP

- A generic program defining r is thus:

$$\begin{array}{rcl}
C_1 = r(\vec{X}) : \pi_1 & :- & \phi_1, b_{11}, \ldots, b_{1m_1} \\
 & \ldots & \\
C_n = r(\vec{X}) : \pi_n & :- & \phi_n, b_{n1}, \ldots, b_{nm_n} \\
C_{111} = r_{11}(\vec{X}) : \pi_{111} & :- & \phi_{111}, b_{1111}, \ldots, b_{111m_{111}} \\
 & \ldots & \\
C_{11n_{11}} = r_{11}(\vec{X}) : \pi_{11n_{11}} & :- & \phi_{11n_{11}}, b_{11n_{11}1}, \ldots, b_{11n_{11}m_{11n_{11}}} \\
 & \ldots & \\
C_{n11} = r_{n1}(\vec{X}) : \pi_{n11} & :- & \phi_{n11}, b_{n111}, \ldots, b_{n11m_{n11}} \\
 & \ldots & \\
C_{n1n_{n1}} = r_{n1}(\vec{X}) : \pi_{n1n_{n1}} & :- & \phi_{n1n_{n1}}, b_{n1n_{n1}1}, \ldots, b_{n1n_{n1}m_{n1n_{n1}}} \\
 & \ldots &
\end{array}$$

Program Tree



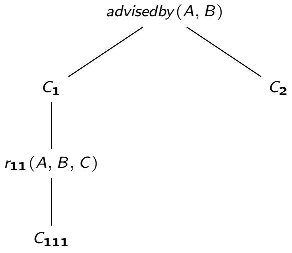
Example

$$
\begin{aligned}
C_1 &= \mathit{advisedby}(A,B) : 0.3 :- \\
&\quad \mathit{student}(A), \mathit{professor}(B), \mathit{project}(C,A), \mathit{project}(C,B), \\
&\quad r_{11}(A,B,C). \\
C_2 &= \mathit{advisedby}(A,B) : 0.6 :- \\
&\quad \mathit{student}(A), \mathit{professor}(B), \mathit{ta}(C,A), \mathit{taughtby}(C,B). \\
C_{111} &= r_{11}(A,B,C) : 0.2 :- \\
&\quad \mathit{publication}(D,A,C), \mathit{publication}(D,B,C).
\end{aligned}
$$

Dipartimento di Matematica e Informatica

Example

$$
\begin{array}{lcl}
C_1 & = & advisedby(A,B) : 0.3 :- \\
 & & \quad student(A), professor(B), project(C,A), project(C,B), \\
 & & \quad r_{11}(A,B,C). \\
C_2 & = & advisedby(A,B) : 0.6 :- \\
 & & \quad student(A), professor(B), ta(C,A), taughtby(C,B). \\
C_{111} & = & r_{11}(A,B,C) : 0.2 :- \\
 & & \quad publication(D,A,C), publication(D,B,C).
\end{array}
$$



Hierarchical PLP

- Writing programs in hierarchical PLP may be unintuitive for humans because of the need of satisfying the constraints and because the hidden predicates may not have a clear meaning.
- The structure of the program should be learned by means of a specialized algorithm
- Hidden predicates generated by a form of predicate invention.

Dipartimento
di Matematica
e Informatica

Inference

- Generate the grounding.
- Each ground probabilistic clause is associated with a random variable whose probability of being true is given by the parameter of the clause and that is independent of all the other clause random variables.
- Ground clause $C_{\vec{p}i} = a_{\vec{p}} : \pi_{\vec{p}i} :- b_{\vec{p}i1}, \dots, b_{\vec{p}im_{\vec{p}}}$. where $\vec{p}$ is a path in the program tree
- $P(b_{\vec{p}i1}, \dots, b_{\vec{p}im_{\vec{p}}}) = \prod_{i=k}^{m_{\vec{p}}} P(b_{\vec{p}ik})$ and $P(b_{\vec{p}ik}) = 1 - P(a_{\vec{p}ik})$ if $b_{\vec{p}ik} = \neg a_{\vec{p}ik}$.
- If a is a literal for an input predicate, then $P(a) = 1$ if a belongs to the example interpretation and $P(a) = 0$ otherwise.

Dipartimento di Matematica e Informatica

Inference

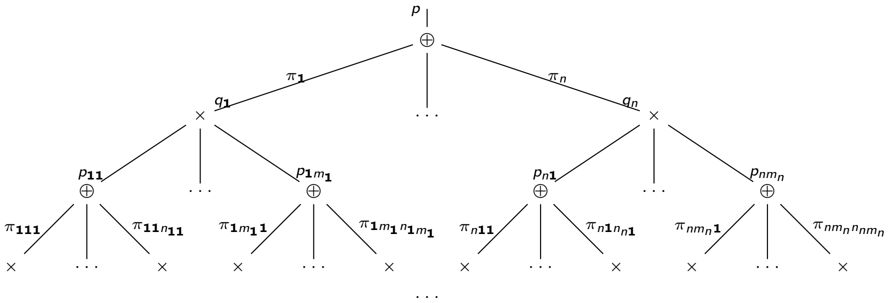
- Hidden predicates: to compute $P(a_{\vec{p}})$ we need to take into account the contribution of every ground clause for the predicate of $a_{\vec{p}}$.
- Suppose these clauses are $\{C_{\vec{p}1}, \ldots, C_{\vec{p}o_{\vec{p}}}\}$.
- If we have two clauses,
  $P(a_{\vec{p}i}) = 1 - (1 - \pi_{\vec{p}1} \cdot P(body(C_{\vec{p}1})) \cdot (1 - \pi_{\vec{p}2} \cdot P(body(C_{\vec{p}2})))$
- $p \oplus q \triangleq 1 - (1 - p) \cdot (1 - q)$.
- This operator is commutative and associative:

$$\bigoplus_i p_i = 1 - \prod_i (1 - p_i)$$

- The operators $\times$ and $\oplus$ are respectively the t-norm and t-conorm of the product fuzzy logic [Hajek 98]: product t-norm and probabilistic sum.

Inference

- If the probabilistic program is ground, the probability of the example atom can be computed with the arithmetic circuit:



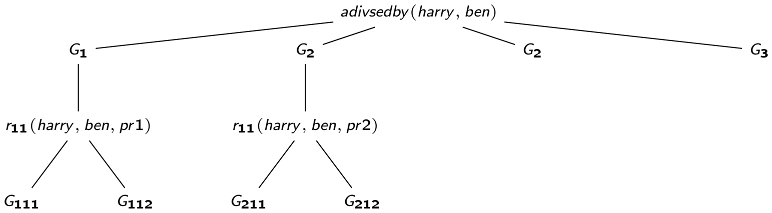
- The arithmetic circuit can be interpreted as a deep neural network where nodes have the activation functions $\times$ and $\oplus$

Example

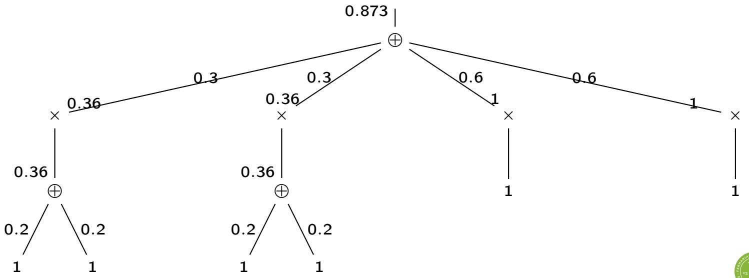
$$
\begin{array}{lcl}
G_1 & = & advisedby(harry, ben) : 0.3 :- \\
 & & \quad student(harry), professor(ben), project(pr1, harry), \\
 & & \quad project(pr1, ben), r_{11}(harry, ben, pr1). \\
G_2 & = & advisedby(harry, ben) : 0.3 :- \\
 & & \quad student(harry), professor(ben), project(pr2, harry), \\
 & & \quad project(pr2, ben), r_{11}(harry, ben, pr2). \\
G_3 & = & advisedby(harry, ben) : 0.6 :- \\
 & & \quad student(harry), professor(ben), ta(c1, harry), taughtby(c1, ben). \\
G_4 & = & advisedby(harry, ben) : 0.6 :- \\
 & & \quad student(harry), professor(ben), ta(c2, harry), taughtby(c2, ben). \\
G_{111} & = & r_{11}(harry, ben, pr1) : 0.2 :- \\
 & & \quad publication(p1, harry, pr1), publication(p1, ben, pr1). \\
G_{112} & = & r_{11}(harry, ben, pr1) : 0.2 :- \\
 & & \quad publication(p2, harry, pr1), publication(p2, ben, pr1). \\
G_{211} & = & r_{11}(harry, ben, pr2) : 0.2 :- \\
 & & \quad publication(p3, harry, pr2), publication(p3, ben, pr2). \\
G_{212} & = & r_{11}(harry, ben, pr2) : 0.2 :- \\
 & & \quad publication(p4, harry, pr2), publication(p4, ben, pr2).
\end{array}
$$

Dipartimento di Matematica e Informatica

Example



$adivsedby(harry, ben)$
G_1
G_2
G_2
G_3
$r_{11}(harry, ben, pr1)$
$r_{11}(harry, ben, pr2)$
G_{111}
G_{112}
G_{211}
G_{212}



0.873
⊕
0.3
0.3
0.6
0.6
0.36
0.36
1
1
×
×
×
×
0.36
0.36
⊕
⊕
1
1
0.2
0.2
0.2
0.2
1
1
1
1

Building the Network

- The network can be built by performing inference using tabling and answer subsumption
- PITA(IND,IND) [Riguzzi CJ14] is a program transformation that adds an extra argument to each subgoal of the program and of the query to store the probability of answers to the subgoal
- When a subgoal returns, the extra argument will be instantiated to the probability of the ground atom that corresponds to the subgoal without the extra argument.
- In programs of hierarchical PLP, when a subgoal returns the original arguments are guaranteed to be instantiated.
- PITA(IND,IND) adds literals to bodies that combine the extra arguments of the subgoals

Building the Network

- The contributions of multiple groundings of multiple clauses are combined by means of tabling with answer subsumption.
- Tabling: keep a store of the subgoals encountered in a derivation together with answers to these subgoals.
- If one of the subgoals is encountered again, its answers are retrieved from the store rather than recomputing them.
- Tabling reduces computation time and ensures termination for a large class of programs [Swift TPLP12].
- Answer subsumption [Swift TPLP12] is a tabling feature that, when a new answer for a tabled subgoal is found, combines old answers with the new one.
- In PITA(IND, IND) the combination operator is probabilistic sum.

Dipartimento di Matematica e Informatica

Parameter Learning

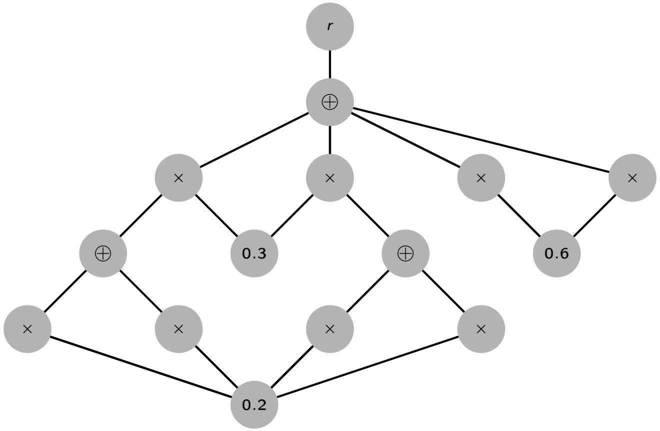
- Parameter learning by backpropagation or EM
- Inference has to be performed repeatedly on the same program with different values of the parameters.
- PITA(IND,IND) can build a representation of the arithmetic circuit, instead of just computing the probability.
- Extra argument used to store a term representing the circuit

Parameter Learning by Gradient Descent

- Deep Parameter learning for HIerarchical probabilistic Logic programs (DPHIL)
- Back-propagation.
- Build a representation of arithmetic circuits sharing parameters (using PITA(IND,IND)).
- Each AC is transformed as follows:
  - Parameters, π_i, labeling arcs from $\oplus$ to $\times$ nodes, are set as children leaves of $\times$ nodes.
  - Shared parameters are considered as individual leaves with many $\times$ parents.
  - Negative literals are represented by nodes of the form $not(a)$ with the single child a.

Dipartimento di Matematica e Informatica

Parameter Learning



Parameter Learning

- Given a Hierarchical PLP T with parameters Π, an interpretation I defining input predicates and a training set $E = \{e_1, \dots, e_M, \sim e_{M+1}, \dots, \sim e_N\}$ find the values of Π that maximize the log likelihood:

$$\arg\max_{\Pi} \sum_{i=1}^{M} \log P(e_i) + \sum_{i=M+1}^{N} \log(1 - P(e_i)) \tag{2}$$

  where $P(e_i)$ is the probability assigned to e_i by $T \cup I$.
- Maximizing the log likelihood can be equivalently seen as minimizing the sum of *cross entropy errors* err_i for all the examples

$$err_i = -y_i \log(p_i) - (1 - y_i) \log(1 - p_i) \tag{3}$$

  where $y_i = 1$ for positive example, $y_i = 0$ otherwise and p_i the probability that the atom is true.

Dipartimento di Matematica e Informatica

Parameter Learning

- Partial derivative of the error with respect to each node $v(n)$:

$$\frac{\partial err}{\partial v(n)} = \begin{cases} -\frac{1}{v(r)} d(n) & \text{if } e \text{ is positive,} \\ \frac{1}{1-v(r)} d(n) & \text{if } e \text{ negative.} \end{cases}$$

where

$$d(n) = \begin{cases} d(p)\frac{v(p)}{v(n)} & \text{if n is a } \bigoplus \textit{ node}, \\ d(p)\frac{1-v(p)}{1-v(n)} & \text{if n is a } \times \text{ node} \\ \sum_p d(p)v(p)(1-\Pi_i) & \text{if n is a leaf node } \Pi_i \\ -d(p) & p = not(n) \end{cases} \tag{4}$$

and $v(n)$, p are respectively the value and the parent of the node n.

Parameter Learning

- Build the ACs and initialize the parameters and the gradients.
- Perform the forward pass by computing the output of each node ($v(n)$) in the AC.
- Compute the gradient of the error w.r.t the output and back-propagate.
- Update the parameters using Adam optimizer.
- Until convergence or a certain condition is satisfied.

Dipartimento di Matematica e Informatica

Parameter Learning by EM

- Two passes over the AC, one bottom-up and one top-down, using message passing
- Bottom-up: compute $v(n)$, message to a node n from below
- Top-down: compute $t(n)$, message to node n from above

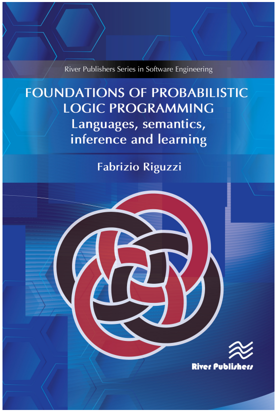
$$t(n) = \begin{cases} \frac{t(p)}{t(p)+v(p)\ominus v(n)t(p)+(1-v(p)\ominus v(n))(1-t(p))} & \text{if } p \text{ is a } \oplus \text{ node} \\ \frac{t(p)\frac{v(p)}{v(p)}+(1-t(p))\left(1-\frac{v(p)}{v(n)}\right)}{t(p)\frac{v(p)}{v(n)}+(1-t(p))\left(1-\frac{v(p)}{v(n)}\right)+(1-t(p))} & \text{if } p \text{ is a } \times \text{ node} \\ 1-t(p) & p = not(n) \end{cases}$$

$$v(p) \ominus v(n) = 1 - \frac{1-v(p)}{1-v(n)}$$

Dipartimento di Matematica e Informatica

Conclusions

- Exciting field!
- Much is left to do:
  - Structure learning search strategies
  - Learning programs with continuous variables
  - Combining Deep Learning with PILP


River Publishers Series in Software Engineering

FOUNDATIONS OF PROBABILISTIC LOGIC PROGRAMMING
Languages, semantics, inference and learning

Fabrizio Riguzzi

River Publishers

Dipartimento di Matematica e Informatica

THANKS FOR LISTENING

AND

ANY QUESTIONS ?

Dipartimento di Matematica e Informatica