Reasoning with Probabilistic Logic Languages

Fabrizio Riguzzi

Outline

Reasoning Tasks

- Inference: we want to compute the probability or an explanation 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

Inference Tasks

- Computing the (conditional) probability of a ground query given the model and, possibly, some evidence
- Finding the most likely state of a set of query atoms given the evidence (Maximum A Posteriori/Most Probable Explanation inference)
  - In Hidden Markov Models, the most likely state of the state variables given the observations is the Viterbi path, its probability the Viterbi probability
- Finding the (k) most probable explanation(s)
- Finding the distribution of variable substitutions for a non-ground query.
- Finding the most probable variable substitution for a non-ground query.

Weight Learning

- Given
  - model: a probabilistic logic model with unknown parameters
  - data: a set of interpretations
- Find the values of the parameters that maximize the probability of the data given the model
- Discriminative learning: maximize the conditional probability of a set of outputs (e.g. ground instances for a predicate) given a set of inputs
- Alternatively, the data are queries for which we know the probability: minimize the error in the probability of the queries that is returned by the model

Structure Learning

- Given
  - language bias: a specification of the search space
  - data: a set of interpretations
- Find the formulas and the parameters that maximize the likelihood of the data given the model
- Discriminative learning: again maximize the conditional likelihood of a set of outputs given a set of inputs

Inference for PLP under DS

- Computing the probability of a query (no evidence)
- Explanation based:
  - find explanations for queries
  - make the explanations mutually exclusive
    - by means of an iterative splitting algorithm (Ailog2 [Poole, 2000])
    - by means of Binary Decision Diagrams (ProbLog [De Raedt et al., 2007], `cplint` [Riguzzi, 2007, Riguzzi, 2009], PITA [Riguzzi and Swift, 2010])
- Bayesian Network based:
  - Convert to BN
  - Use BN inference algorithms (CVE [Meert et al., 2009])
  - Lifted inference

ProbLog

$$sneezing(X) \leftarrow flu(X), flu\_sneezing(X).$$
$$sneezing(X) \leftarrow hay\_fever(X), hay\_fever\_sneezing(X).$$
$$flu(bob).$$
$$hay\_fever(bob).$$
$$C_1 = 0.7 :: flu\_sneezing(X).$$
$$C_2 = 0.8 :: hay\_fever\_sneezing(X).$$

- Distributions over facts

Definitions

- Composite choice κ: consistent set of atomic choices (C_i, θ_j, l) with $l \in \{1, 2\}$
- Set of worlds compatible with κ: $\omega_\kappa = \{w_\sigma | \kappa \subseteq \sigma\}$
- Explanation κ for a query Q: Q is true in every world of ω_κ
- A set of composite choices K is covering with respect to Q: every world w in which Q is true is such that $w \in \omega_K$ where $\omega_K = \bigcup_{\kappa \in K} \omega_\kappa$
- Example:

$$K_1 = \{\{(C_1, \{X/bob\}, 1)\}, \{(C_2, \{X/bob\}, 1)\}\} \quad (1)$$

is covering for $sneezing(bob)$.

Finding Explanations

- All explanations for the query are collected
- ProbLog: source to source transformation for facts, use of dynamic database
- `cplint`: meta-interpretation
- PITA: source to source transformation, addition of an argument to predicates

Explanation Based Inference Algorithm

- $K =$ set of explanations found for Q, the probability of Q is given by the probability of the formula

$$f_K(\mathbf{X}) = \bigvee_{\kappa \in K} \bigwedge_{(C_i,\theta_j,l)\in\kappa} (X_{C_i\theta_j} = l)$$

  where $X_{C_i\theta_j}$ is a random variable whose domain is $1, 2$ and $P(X_{C_i\theta_j} = l) = P_0(C_i, l)$
- Binary domain: we use a Boolean variable X_{ij} to represent $(X_{C_i\theta_j} = 1)$
- $\neg X_{ij}$ represents $(X_{C_i\theta_j} = 2)$

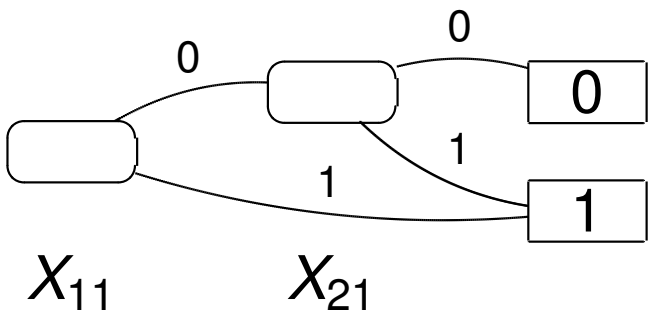
Example

A set of covering explanations for $sneezing(bob)$ is $K = \{\kappa_1, \kappa_2\}$
$\kappa_1 = \{(C_1, \{X/bob\}, 1)\}$
$\kappa_2 = \{(C_2, \{X/bob\}, 1)\}$
$K = \{\kappa_1, \kappa_2\}$
$f_K(\mathbf{X}) = (X_{C_1\{X/bob\}} = 1) \vee (X_{C_2\{X/bob\}} = 1).$
$X_{11} = (X_{C_1\{X/bob\}} = 1)$
$X_{21} = (X_{C_2\{X/bob\}} = 1)$
$f_K(\mathbf{X}) = X_{11} \vee X_{21}.$
$P(f_K(\mathbf{X})) = P(X_{11} \vee X_{21})$
$P(f_K(\mathbf{X})) = P(X_{11}) + P(X_{21}) - P(X_{11})P(X_{21})$

- In order to compute the probability, we must make the explanations mutually exclusive
- [De Raedt et al., 2007]: Binary Decision Diagram (BDD)

Binary Decision Diagrams

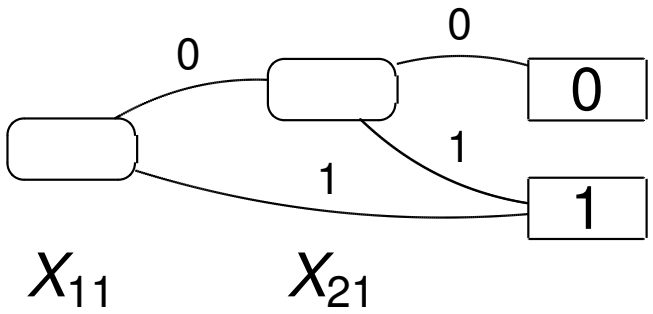
- A BDD for a function of Boolean variables is a rooted graph that has one level for each Boolean variable
- A node *n* in a BDD has two children: one corresponding to the 1 value of the variable associated with *n* and one corresponding the 0 value of the variable
- The leaves store either 0 or 1.
- A BDD can be used to compute the value of the formula by traversing the graph starting from the root and returning the value associated to the leaf that is reached.



Binary Decision Diagrams

- BDDs can be built by combining simpler BDDs using Boolean operators
- While building BDDs, simplification operations can be applied that delete or merge nodes
- Merging is performed when the diagram contains two identical sub-diagrams
- Deletion is performed when both arcs from a node point to the same node
- A reduced BDD often has a much smaller number of nodes with respect to the original BDD

Binary Decision Diagrams



$$f_K(\mathbf{X}) = X_{11} \times f_K^{X_{11}}(\mathbf{X}) + \neg X_{11} \times f_K^{\neg X_{11}}(\mathbf{X})$$

$$P(f_K(\mathbf{X})) = P(X_{11})P(f_K^{X_{11}}(\mathbf{X})) + (1 - P(X_{11}))P(f_K^{\neg X_1}(\mathbf{X}))$$

$$P(f_K(\mathbf{X})) = 0.7 \cdot P(f_K^{X_{11}}(\mathbf{X})) + 0.3 \cdot P(f_K^{\neg X_{11}}(\mathbf{X}))$$

Probability from a BDD

- Dynamic programming algorithm [De Raedt et al., 2007]

1: **function** PROB(n)
2: **if** n is a terminal note **then**
3: return $value(n)$
4: **else**
5: return
$\text{PROB}(child_1(n)) \times p(v(n)) + \text{PROB}(child_0(n)) \times (1 - p(v(n)))$
6: **end if**
7: **end function**

Logic Programs with Annotated Disjunctions

$$
\begin{array}{lll}
C_1 = & strong\_sneezing(X) : 0.3 \vee moderate\_sneezing(X) : 0.5 & \leftarrow \quad flu(X). \\
C_2 = & strong\_sneezing(X) : 0.2 \vee moderate\_sneezing(X) : 0.6 & \leftarrow \quad hay\_fever(X). \\
C_3 = & flu(bob). & \\
C_4 = & hay\_fever(bob). &
\end{array}
$$

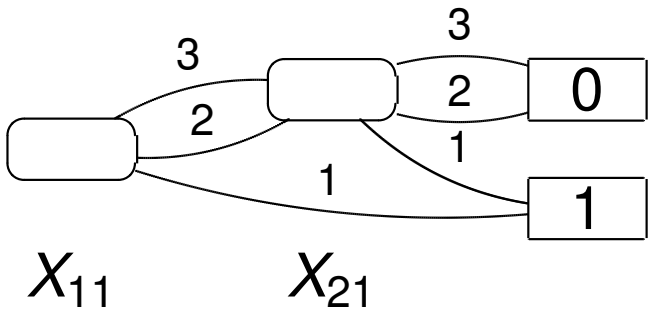
- Distributions over the head of rules
- More than two head atoms

Example

A set of covering explanations for *strong_sneezing*(*bob*) is
$K = \{\kappa_1, \kappa_2\}$
$\kappa_1 = \{(C_1, \{X/bob\}, 1)\}$
$\kappa_2 = \{(C_2, \{X/bob\}, 1)\}$
$K = \{\kappa_1, \kappa_2\}$
$X_{11} = X_{C_1\{X/bob\}}$
$X_{21} = X_{C_2\{X/bob\}}$
$f_K(\mathbf{X}) = (X_{11} = 1) \vee (X_{21} = 1)$.
$P(f_X) = P(X_{11} = 1) + P(X_{21} = 1) - P(X_{11} = 1)P(X_{21} = 1)$

- To make the explanations mutually exclusive: Multivalued Decision Diagram (MDD)

Multivalued Decision Diagrams



$$f_K(\mathbf{X}) = \bigvee_{l \in |X_{11}|} (X_{11} = l) \wedge f_K^{X_{11}=l}(\mathbf{X})$$

$$P(f_K(\mathbf{X})) = \sum_{l \in |X_{11}|} P(X_{11} = l) P(f_K^{X_{11}=l}(\mathbf{X}))$$

$$f_K(\mathbf{X}) = (X_{11} = 1) \wedge f_K^{X_{11}=1}(\mathbf{X}) + (X_{11} = 2) \wedge f_K^{X_{11}=2}(\mathbf{X}) + (X_{11} = 3) \wedge f_K^{X_{11}=3}(\mathbf{X})$$

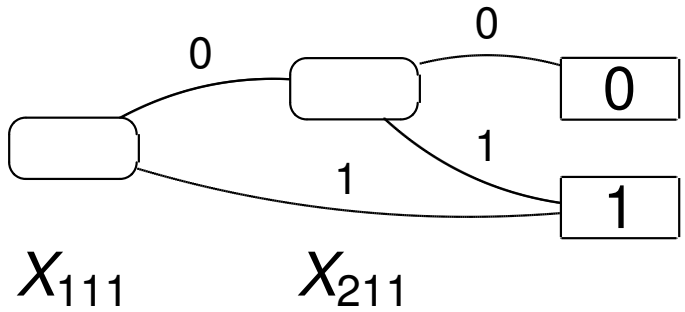
$$f_K(\mathbf{X}) = 0.3 \cdot P(f_K^{X_{11}=1}(\mathbf{X})) + 0.5 \cdot P(f_K^{X_{11}=2}(\mathbf{X})) + 0.2 \cdot P(f_K^{X_{11}=3}(\mathbf{X}))$$

Manipulating Multivalued Decision Diagrams

- Use an MDD package
- Convert to BDD, use a BDD package: BDD packages more developed, more efficient
- Conversion to BDD
  - Log encoding
  - Binary splits: more efficient

Transformation to a Binary Decision Diagram

- For a variable X_{ij} having n values, we use $n-1$ Boolean variables $X_{ij1}, \ldots, X_{ijn-1}$
- $X_{ij} = l$ for $l = 1, \ldots n-1$: $\overline{X_{ij1}} \wedge \overline{X_{ij2}} \wedge \ldots \wedge \overline{X_{ijl-1}} \wedge X_{ijl}$,
- $X_{ij} = n$: $\overline{X_{ij1}} \wedge \overline{X_{ij2}} \wedge \ldots \wedge \overline{X_{ijn-1}}$.
- Parameters: $P(X_{ij1}) = P(X_{ij} = 1) \ldots P(X_{ijl}) = \frac{P(X_{ij}=l)}{\prod_{m=1}^{l-1}(1-P(X_{ijm}))}$.



Approximate Inference

- Inference problem is #P hard
- For large models inference is intractable
- Approximate inference
  - Monte Carlo: draw samples of the truth value of the query
  - Iterative deepening: gives a lower and an upper bound
  - Compute only the best k explanations: branch and bound, gives a lower bound

Monte Carlo

- The disjunctive clause
  $C_r = H_1 : \alpha_1 \vee \ldots \vee H_n : \alpha_n \leftarrow L_1, \ldots, L_m.$
  is transformed into the set of clauses $MC(C_r)$
  $MC(C_r, 1) = H_1 \leftarrow L_1, \ldots, L_m, sample\_head(n, r, VC, NH), NH = 1.$
  $\ldots$
  $MC(C_r, n) = H_1 \leftarrow L_1, \ldots, L_m, sample\_head(n, r, VC, NH), NH = n.$
- Sample truth value of query `Q`:

```
...
  (call(Q)->  NT1 is NT+1 ; NT1 =NT),
...
```

Monte Carlo

- The proportion of successes in a Bernoulli trial process is in the binomial proportion confidence interval

$$\hat{p} \pm z_{1-\alpha/2}\sqrt{\frac{\hat{p}\,(1-\hat{p})}{n}}$$

- Algorithm:
- $n := 0$, $nt := 0$
- Repeat
  - Test query n' times, nt' successes
  - $n := n + n'$, $nt := nt + nt'$, $\hat{p} = nt/n$
  - Compute interval size s
- until $s < \delta$
- return $\hat{p}, s$

Approximate Inference

- Iterative deepening: build the derivation tree only up to a certain depth,
- Completed derivations give a lower bound, completed plus incomplete derivations an upper bound
- Best-k explanations: each time an explanation is found, update the set of explanations
- Cut a derivation if its probability falls below that of the k-th best explanation

Inference by Conversion to Bayesian Networks

- Convert the program to a BN, perform inference on the BN with belief propagation, variable elimination, etc.
- Problem: grounding the program
- With function symbols, infinite grounding
- Even without function symbols, the grounding can be huge (exponential size)
- Most of the network is irrelevant to the query
- Grounding:
  - Use a lifted inference algorithm
  - Build only the relevant network and apply an inference algorithm
  - Combination of the two approaches

Lifted Belief Propagation

- Belief propagation: nodes exchange messages, at convergence the marginal probability of each node can be extracted
- Correct for polytrees, approximate for general DAGs
- Lifted Belief Propagation: exploit the symmetries in the network to group nodes that exchange equal or similar messages into super nodes
- Perform belief propagation between super nodes taking into account the cardinalities of the messages

Building the Relevant Network

- Bayes Ball [Shachter, 1998]: algorithm for identifying the portion of a network that is relevant to query and evidence
- First-Order Bayes Ball [Meert et al., 2010]: lifted version of Bayes Ball
- Then apply a (lifted) inference algorithm

Parameter Learning

- Problem: given a set of interpretations and a program, find the parameters maximizing the likelihood of the interpretations (or of instances of a target predicate)
- Exploit the equivalence with BN to use BN learning algorithms
- The interpretations record the truth value of ground atoms, not of the choice variables
- Unseen data: relative frequency can't be used
- An Expectation-Maximization algorithm must be used:
  - 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

Parameter Learning

- [Thon et al., 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., 2008] independently proposed a similar algorithm
- CoPrEM [Gutmann et al., 2010] is the adaptation of EM to ProbLog
- EMBLEM [Bellodi and Riguzzi, 2013] adapts [Ishihata et al., 2008] to LPADs

EMBLEM

- EM over Bdds for probabilistic Logic programs Efficient Mining
- 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 disjunction of explanations for each query Q

EM Algorithm

- *Expectation step* (synthesis)

  1. Expectations $\mathbf{E}[c_{ik0}]$ and $\mathbf{E}[c_{ik1}]$ where c_{ikx} is the number of times a Boolean variable X_{ijk} takes value x for all C_is, $k = 1, \ldots, n_i - 1$

  $$\mathbf{E}[c_{ikx}] = \sum_{Q} \mathbf{E}[c_{ikx}|Q]$$

  2. Expected counts per query $\mathbf{E}[c_{ikx}|Q]$, for all queries Q and $x \in \{0, 1\}$.

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

  $g(i) := \{j|\theta_j \text{ is a substitution grounding } C_i\}$

- *Maximization step*
  - Updates parameters π_{ik} representing $P(X_{ijk} = 1)$
  - π_{ik} = $E[c_{ik1}]$ / ($E[c_{ik0}]$ + $E[c_{ik1}]$)

Expectation Computation

- $P(X_{ijk} = x|Q) = \frac{P(X_{ijk}=x,Q)}{P(Q)}$
- $P(X_{ijk} = x, Q) = \sum_{n \in N(Q), v(n)=X_{ijk}} F(n)B(child_x(n))\pi_{ikx} = \sum_{n \in N(Q), v(n)=X_{ijk}} e^x(n)$
  - π_{ikx} is π_{ik} if $x = 1$ and $(1 - \pi_{ik})$ if $x = 0$
  - $F(n)$ 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
  - $F(n)$ and $B(n)$ are computed by two traversal of the BDD of rQ
  - $e^x(n)$ is the probability mass of paths from the root to the 1 leaf passing through the x branch of n

Structure Learning for LPADs

1. Given
   - model: a trivial LPAD or an empty one
   - data: a set of interpretations
2. *Find the model and the parameters* that maximize the probability of the data (log-likelihood)
3. Two algorithms:
   1. SLIPCASE: Structure LearnIng of ProbabilistiC logic progrAmS with Em over bdds [Bellodi and Riguzzi, 2012]
      Beam search in the space of probabilistic programs
   2. SLIPCOVER: Structure LearnIng of Probabilistic logic program by searching OVER the clause space
      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.
4. Both perform *parameter learning* by means of EMBLEM

SLIPCASE

- 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 $\rightarrow$ 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 refinements 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

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 *IBs*, one for each predicate appearing in a head declaration, is generated by SLIPCOVER by building a set of *bottom clauses* as in Progol [Muggleton, 1995]
- To genereate a bottom clause for a mode declaration $m = modeh(r, s)$, an input interpretation is selected and an answer h for the goal $schema(s)$ is selected, where $schema(s)$ denotes the literal obtained from s by variabilization
- The resulting ground clause $h :- b_1, \ldots, b_m$ is then processed by replacing each term in a + or - placemarker with a variable

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 $h :- b_1, \ldots, b_m$
- This process is repeated for a number *NInt* of input mega-examples and a number *NA* of answers, thus obtaining $NInt \cdot NA$ bottom clauses.
- 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.

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 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.
- The clause is inserted in *TC* if a target predicate appears in its head, otherwise in *BC*. The insertion is in order of LL.
- These lists have 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.
  - This is done for each clause in *TC*.
- Finally, SLIPCOVER adds all the clauses in *BC* to the theory and performs parameter learning on the resulting theory.

References I


Bellodi, E. and Riguzzi, F. (2012).
Learning the structure of probabilistic logic programs.
In *Inductive Logic Programming 21st International Conference, ILP 2011, London, UK, July 31 - August 3, 2011. Revised Papers*, volume 7207 of *LNCS*, pages 61–75, Heidelberg, Germany. Springer.
The original publication is available at `http://www.springerlink.com`.


Bellodi, E. and Riguzzi, F. (2013).
Expectation Maximization over binary decision diagrams for probabilistic logic programs.
Intelligent Data Analysis, 17(2).

References II


De Raedt, L., Kimmig, A., and Toivonen, H. (2007).
Problog: A probabilistic prolog and its application in link discovery.
In *International Joint Conference on Artificial Intelligence*, pages 2462–2467.


Gutmann, B., Kimmig, A., Kersting, K., and De Raedt, L. (2010).
Parameter estimation in ProbLog from annotated queries.
Technical Report CW 583, Department of Computer Science, Katholieke Universiteit Leuven, Belgium.


Ishihata, M., Kameya, Y., Sato, T., and Minato, S. (2008).
Propositionalizing the em algorithm by bdds.
In *Late Breaking Papers of the 18th International Conf. on Inductive Logic Programming*, pages 44–49.

References III


Meert, W., Struyf, J., and Blockeel, H. (2009).
CP-Logic theory inference with contextual variable elimination and comparison to bdd based inference methods.
In *ILP 2009*.


Meert, W., Taghipour, N., and Blockeel, H. (2010).
First-order bayes-ball.
In Balcázar, J. L., Bonchi, F., Gionis, A., and Sebag, M., editors, *Machine Learning and Knowledge Discovery in Databases, European Conference, ECML PKDD 2010, Barcelona, Spain, September 20-24, 2010, Proceedings, Part II*, volume 6322 of *Lecture Notes in Computer Science*, pages 369–384. Springer.


Muggleton, S. (1995).
Inverse entailment and progol.
New Generation Comput., 13(3&4):245–286.

References IV

Poole, D. (2000).
Abducing through negation as failure: stable models within the independent choice logic.
J. Log. Program., 44(1-3):5–35.

Riguzzi, F. (2007).
A top down interpreter for LPAD and CP-logic.
In *Congress of the Italian Association for Artificial Intelligence*, number 4733 in LNAI, pages 109–120. Springer.

Riguzzi, F. (2009).
Extended semantics and inference for the Independent Choice Logic.
Logic Journal of the IGPL.
to appear.

References V


Riguzzi, F. and Swift, T. (2010).
Tabling and Answer Subsumption for Reasoning on Logic Programs with Annotated Disjunctions.
In Hermenegildo, M. and Schaub, T., editors, *Technical Communications of the 26th Int'l. Conference on Logic Programming (ICLP'10)*, volume 7 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 162–171, Dagstuhl, Germany. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.


Shachter, R. D. (1998).
Bayes-ball: The rational pastime (for determining irrelevance and requisite information in belief networks and influence diagrams.
In *In Uncertainty in Artificial Intelligence*, pages 480–487. Morgan Kaufmann.

References VI


Thon, I., Landwehr, N., and Raedt, L. D. (2008).
A simple model for sequences of relational state descriptions.
In Daelemans, W., Goethals, B., and Morik, K., editors, *Machine Learning and Knowledge Discovery in Databases, European Conference, ECML/PKDD 2008, Antwerp, Belgium, September 15-19, 2008, Proceedings, Part II*, volume 5212 of *Lecture Notes in Computer Science*, pages 506–521. Springer.