Efficient inference in discrete and continuous domains for PLP languages under the Distribution Semantics

Elena Bellodi

Department of Engineering, University of Ferrara, Italy

PLP 2019, September 21st 2019

University of Ferrara

Outline

Handling real world's uncertainty

- Problem: real world is **uncertain and complex**
- Logic does not handle well uncertainty
- Graphical models do not handle well relationships among entities
- Solution: combine the two
- Many approaches proposed in the areas of Logic Programming, Machine Learning, Databases, Knowledge Representation

Probabilistic Logic Programming (PLP)

- Distribution Semantics [Sato ICLP 95]
- A probabilistic logic program defines a probability distribution over normal logic programs (called instances or possible worlds or simply textitworlds)
- The distribution is extended to a joint distribution over worlds and interpretations (or queries)
- The probability of a query is obtained from this distribution

Probabilistic Logic Programming 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

Logic Programs with Annotated Disjunctions (LPADs)

$$sneezing(X) : 0.7 \vee null : 0.3 \leftarrow flu(X).$$
$$sneezing(X) : 0.8 \vee null : 0.2 \leftarrow hay\_fever(X).$$
$$flu(bob).$$
$$hay\_fever(bob).$$

- **Distributions over the head of rules**
- *null* does not appear in the body of any rule, and is usually omitted

Worlds for the "sneezing domain" (LPAD syntax)

$sneezing(bob) \leftarrow flu(bob).$
$sneezing(bob) \leftarrow hay\_fever(bob).$
$flu(bob).$
$hay\_fever(bob).$
$\mathbf{P(w_1) = 0.7 \times 0.8}$

$null \leftarrow flu(bob).$
$sneezing(bob) \leftarrow hay\_fever(bob).$
$flu(bob).$
$hay\_fever(bob).$
$\mathbf{P(w_2) = 0.3 \times 0.8}$

$sneezing(bob) \leftarrow flu(bob).$
$null \leftarrow hay\_fever(bob).$
$flu(bob).$
$hay\_fever(bob).$
$\mathbf{P(w_3) = 0.7 \times 0.2}$

$null \leftarrow flu(bob).$
$null \leftarrow hay\_fever(bob).$
$flu(bob).$
$hay\_fever(bob).$
$\mathbf{P(w_4) = 0.3 \times 0.2}$

- $sneezing(bob)$ is true in 3 worlds out of 4
- $P(sneezing(bob)) = 0.7 \times 0.8 + 0.3 \times 0.8 + 0.7 \times 0.2 = 0.94$

Inference under the Distribution Semantics

- Let $\mathbf{At}$ be the set of all ground (probabilistic and derived) atoms in a given LPAD program
- Assume that we are given a set $\mathbf{E} \subset \mathbf{At}$ of observed atoms (*evidence* atoms)
  - a vector $\mathbf{e}$ with their observed truth values means that the evidence is $\mathbf{E} = \mathbf{e}$
- We are also given a set $\mathbf{Q} \subset \mathbf{At}$ of atoms of interest (*query* atoms)
- q indicates a single ground atom of interest
- Different inference tasks are possible

Inference under the Distribution Semantics

- Unconditional inference: computing the unconditional probability $P(\mathbf{e})$, the probability of evidence
- Conditional or marginal inference: computing the conditional probability distribution of every query atom given the evidence, i.e., computing $P(q|\mathbf{E} = \mathbf{e})$, for each $q \in \mathbf{Q}$
- Computing the probability distribution or density of the non-ground arguments of a conjunction of literals $\mathbf{Q} = \mathbf{q}$

Inference under the Distribution Semantics

Example

```
heads(Coin):1/2; tails(Coin):1/2:- toss(Coin),\+biased(Coin).
heads(Coin):0.6; tails(Coin):0.4:- toss(Coin),biased(Coin).
fair(Coin):0.9; biased(Coin):0.1.
toss(coin).
```

- Unconditional inference: $P(heads(coin))$, or $P(tails(coin))$
- Conditional inference: $P(heads(coin)|biased(coin))$

Inference under the Distribution Semantics

The inference problem is #**P-hard** and for large models is intractable.
Possible **solutions**:

1. Exact inference
   - **Knowledge Compilation**: compile the probabilistic logic program to an *intermediate representation* and then compute the probability by Weighted Model Counting
   - **Bayesian Network (BN) based**: convert the probabilistic logic program into a BN and apply BN inference algorithms (Meert et al. (2010))
   - **Lifted** inference (Poole (2003)): exploits symmetries in the model to speed up inference
2. Approximate inference

Knowledge Compilation

A probabilistic logic program can be compiled into the following intermediate representations:

- **Binary Decision Diagrams** (BDD): De Raedt et al. (2007), `cplint` (Riguzzi (2007),Riguzzi (2009)), `PITA` (Riguzzi and Swift (2010a))
- deterministic-Decomposable Negation Normal Form circuits **(d-DNNF)**: `ProbLog2`, Fierens et al. (2015)
- **Sentential Decision Diagrams** (SDD), Darwiche (2011)

Knowledge Compilation to BDD

1. **Assign Boolean random variables** (r.v.) to the probabilistic rules
2. Given a query Q, compute its **explanations**, i.e. assignments to the random variables that are sufficient for entailing the query
3. Let K be the set of all possible explanations
4. Build a Boolean formula f_K representing K
5. **Build a BDD** encoding f_K
6. **Compute the probability of evidence** from the BDD (unconditional inference)

Knowledge Compilation to BDD

(2,3,4) Example

- For the (LPAD) program:

$$sneezing(X) : 0.7 \leftarrow flu(X).$$
$$sneezing(X) : 0.8 \leftarrow hay\_fever(X).$$
$$flu(bob).$$
$$hay\_fever(bob).$$

  a set of covering explanations for $Q = sneezing(bob)$ is $K = \{\kappa_1, \kappa_2\}$
- $\kappa_1 = \{(C_1, \{X/bob\}, 1)\} \quad \kappa_2 = \{(C_2, \{X/bob\}, 1)\}$
- A composite choice κ is an explanation for a query Q if Q is true in every world compatible with κ

Knowledge Compilation to BDD

(2,3,4) Example

- $\theta_1 = X/bob$
- $X_{C_1\theta_1} \rightarrow X_{11}$ From $\kappa_1, X_{11} = 1$
- $X_{C_2\theta_1} \rightarrow X_{21}$ From $\kappa_2, X_{21} = 1$
- $f_K(\mathbf{X}) = (X_{11} = 1) \vee (X_{21} = 1)$
- $P(Q) = P(f_K(\mathbf{X})) = P(X_{11} \vee X_{21}) = P(X_{11}) + P(X_{21}) - P(X_{11})P(X_{21})$

Knowledge Compilation to BDD

And Now What?

- Worlds are mutually exclusive, and in theory we could compute $P(Q)$ as $\sum_{w \in W_T : w \models Q} P(w)$,
  - BUT in practice, it is unfeasible to find all the worlds w where the query is true
- It's easier to find explanations for the query
  - BUT explanations might not be mutually exclusive
  - **they must be made mutually exclusive in order to sum up probabilities**
  - Binary Decision Diagrams (BDD): they split paths on the basis of the values of binary variables, so **the branches are mutually disjoint**

Knowledge Compilation to BDD

(3,4) Building a formula for the explanations

- In order to use BDD, a multi-valued random variable X_{ij} with n_i values must be converted into $n_i - 1$ Boolean variables $X_{ij1}, ..., X_{ijn_i-1}$
- $X_{ij} = k$ for $k = 1, ..., n_i - 1$ is represented by the conjunction $\overline{X_{ij1}} \wedge \ldots \wedge \overline{X_{ijk-1}} \wedge X_{ijk}$
- $X_{ij} = n_i$ by $\overline{X_{ij1}} \wedge \ldots \wedge \overline{X_{ijn_i-1}}$

Knowledge Compilation to BDD

(3,4) Example

- For both C_1 and C_2 : $n_i = 2$
- (multi-valued) $X_{11} = 1 \rightarrow$ (Boolean) X_{111}
- (multi-valued) $X_{11} = 2 \rightarrow$ (Boolean) $\overline{X_{111}}$
- (multi-valued) $X_{21} = 1 \rightarrow$ (Boolean) X_{211}
- (multi-valued) $X_{21} = 2 \rightarrow$ (Boolean) $\overline{X_{211}}$
- $f'_K(\mathbf{X}) = X_{111} \vee X_{211}$

Knowledge Compilation to BDD

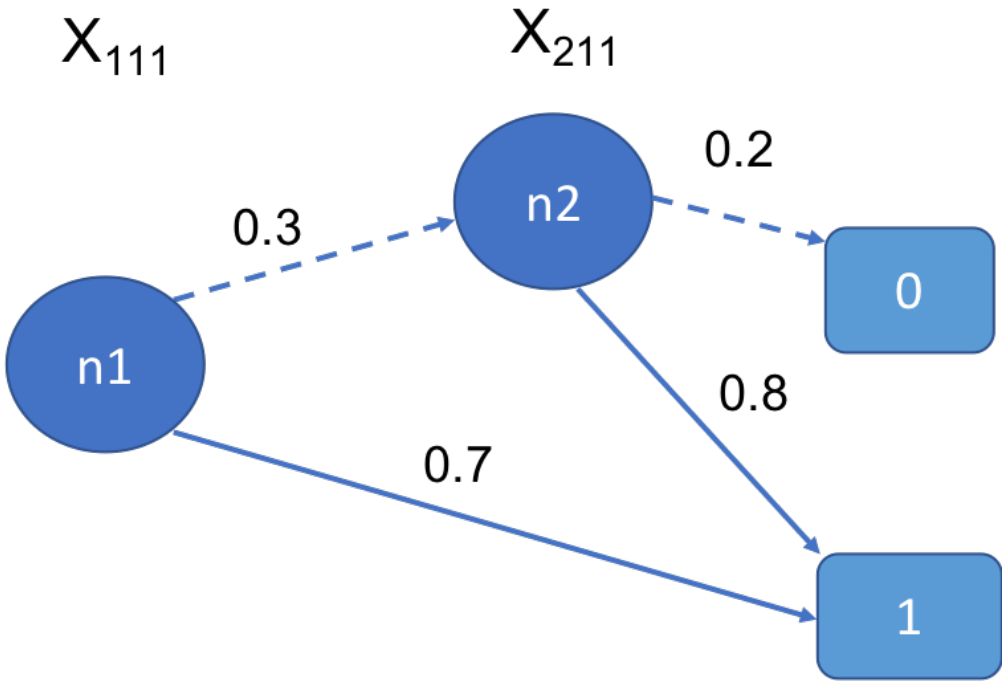
(6) Computing the probability from a BDD

Algorithm 4 Function PROB: Computation of the probability of a BDD.

```
1:  function PROB(node)
2:      if node is a terminal then
3:          return 1
4:      else
5:          if Table(node) ≠ null then
6:              return Table(node)
7:          else
8:              p0 ←PROB(child_0(node))
9:              p1 ←PROB(child_1(node))
10:             let π be the probability of being true of var(node)
11:             Res ← p1 · π + p0 · (1 − π)
12:             add node → Res to Table
13:             return Res
14:         end if
15:     end if
16: end function
```

Knowledge Compilation to BDD

(6) Example



$$P(n_2) = 0 \cdot 0.2 + 1 \cdot 0.8 = 0.8$$
$$P(n_1) = P(root) = 1 \cdot 0.7 + 0.3 \cdot 0.8 = 0.94 = P(Q) = P(sneezing(bob))$$

Tabling-based Inference

- Idea: **maintain in a *table* both subgoals encountered in a query evaluation and answers to these subgoals**
- If a subgoal is encountered more than once, **the evaluation reuses information from the table** rather than re-performing resolution against program clauses
- Tabling can be used to evaluate programs with negation
- Tabling integrates closely with Prolog: a predicate p/n is evaluated using SLDNF by default; the predicate can be made to use tabling by the directive `:- table p/n` that is added by the user or compiler

Tabling-based Inference

PITA: *Probabilistic Inference with Tabling and Answer subsumption*

- Introduced by Riguzzi and Swift (2010b)
- PITA computes the probability of queries from LPADs with tabling
- PITA builds all explanations for every subgoal encountered during a derivation of the query
- Explanations for subgoals are stored with tabling
- Explanations are compactly represented using BDDs that also allow an efficient computation of the probability
- Subgoals (ground atoms) have an extra argument storing a BDD that represents the explanations for their answers
- When an answer $q(\mathbf{x}, bdd)$ is found, bdd represents the explanations for $q(\mathbf{x})$

Tabling-based Inference

Answer Subsumption

- A feature of tabling in XSB and SWI Prolog
- **Combine different answers for the same goal**
- E.g `:-table path(X,Y,lattice(or/3))` means that, if two explanations `path(a,b,bdd0)` and `path(a,b,bdd1)` are found, the single answer `path(a,b,bdd)` will be stored in the table where `or(bdd0,bdd1,bdd)`

Tabling-based Inference

PITA: Program Transformation

- The first step of the PITA algorithm is to apply a program transformation to an LPAD to create a normal logic program that contains calls for manipulating BDDs
- Prolog interface to the CUDD C library

Tabling-based Inference

PITA Example

```
:- table path(X,Y,lattice(or/3)),edge(X,Y,lattice(or/3)).
```

LPAD

```
path(X,X).
path(X,Y):- path(X,Z),edge(Z,Y).
edge(a,b):0.3.
....
```

PITA Transformation

```
path(X,X,One):- one(One).
path(X,Y,BDD):- one(One),path(X,Z,BDD0),and(One,BDD0,BDD1),
                edge(Z,Y,BDD2),and(BDD1,BDD2,BDD).
edge(a,b,BDD):- one(One),get_var_n(3,[],[0.3,0.7],Var),
                equality(Var,1,BDD0),and(BDD0,One,BDD).
...
```

Tabling-based Inference

PITA Example

Query: `path(a,b)`

```
:- init,
   path(?HGNC_620?,?HGNC_983?,BDD),
   ret_prob(BDD,P),
   end.
```

Tabling-based Inference

PITA Results

- PITA was compared with CVE (Meert et al. (2010)) and ProbLog1 (Kimmig et al. (2008a)).
  - CVE transforms an LPAD into an equivalent Bayesian network and then performs inference on the network using the variable elimination algorithm
  - ProbLog employs BDDs for efficient inference
- The algorithm was able to successfully solve more complex queries than the other algorithms in most cases and it was also almost always faster

Inference Systems based on BDDs: `cplint`

- Suite of programs for reasoning with LPADs
- Inference and learning
- Versions for Yap Prolog and SWI-Prolog
- Distributed as a pack of SWI-Prolog. To install it, use
  `?- pack_install(cplint).`
- Available in the web application *cplint on SWISH*: http://cplint.eu/
- Exact Inference: module `PITA`

Hands-on: Coin example

- http://cplint.eu/example/inference/coin.pl
- Unconditional inference:
  - What is the probability that *coin* lands heads? `prob(heads(coin),Prob).`
  - What is the probability that *coin* lands tails? `prob(tails(coin),Prob).`
- Conditional inference:
  - What is the probability that *coin* lands heads, given that I know it is biased? `prob(heads(coin),biased(coin),Prob).`

Knowledge Compilation to d-DNNF

- A tractable logical form known as Deterministic, Decomposable Negation Normal Form, which **permits some generally intractable logical queries to be computed in time polynomial** in the form size (Darwiche (2004))
- Superset of OBDDs (Ordered BDD): BDDs with a defined variable ordering
- Allows to perform unconditional, conditional and MPE inference in **ProbLog2**

Knowledge Compilation to d-DNNF

- A negation normal form (NNF) is a rooted directed acyclic graph in which each leaf node is labeled with a literal, *true* or *false*, and each internal node is labeled with a conjunction or disjunction

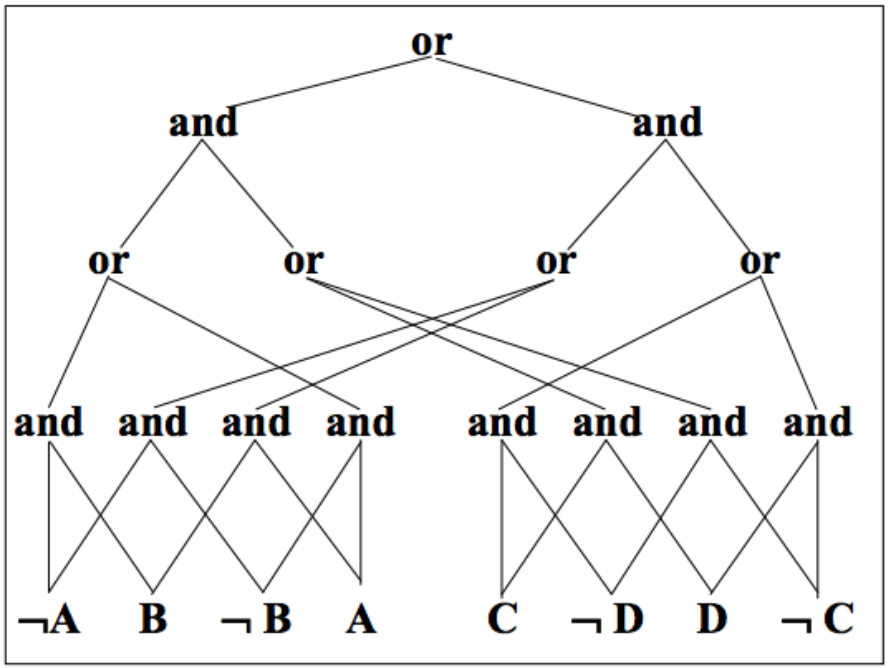


Figure 1. A negation normal form.

Knowledge Compilation to d-DNNF

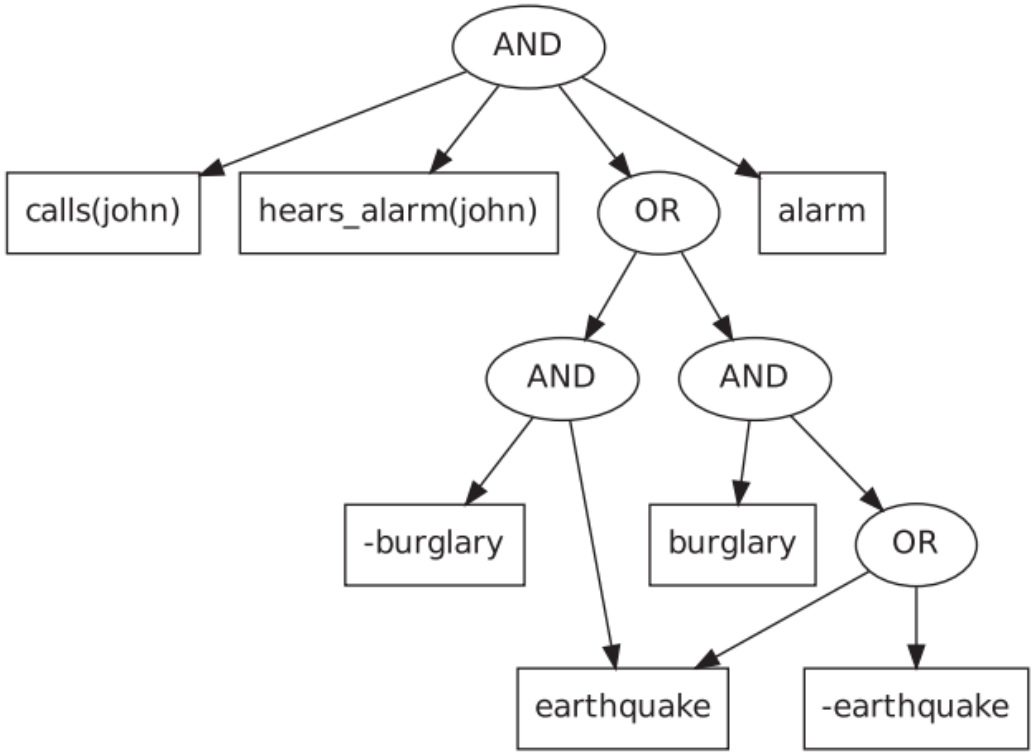
Procedure

1. Ground the probabilistic logic program and convert it into an equivalent Boolean formula ϕ_r
2. Build ϕ_e, the Boolean formula for the evidence
3. Rewrite $\phi = \phi_r \wedge \phi_e$ in CNF
4. Construct a *weighted* Boolean formula for ϕ
5. CNF formula $\rightarrow$ d-DNNF formula (#P hard step)
6. d-DNNF formula $\rightarrow$ aritmetic circuit (AC)
7. Compute the probability of evidence from the AC

Knowledge Compilation to d-DNNF

(5) Convert the CNF into a d-DNNF

- Conversion is made by compilers: c2d (Darwiche (2004)), DSHARP (Muise et al. (2012)), irrespective of the weighting function



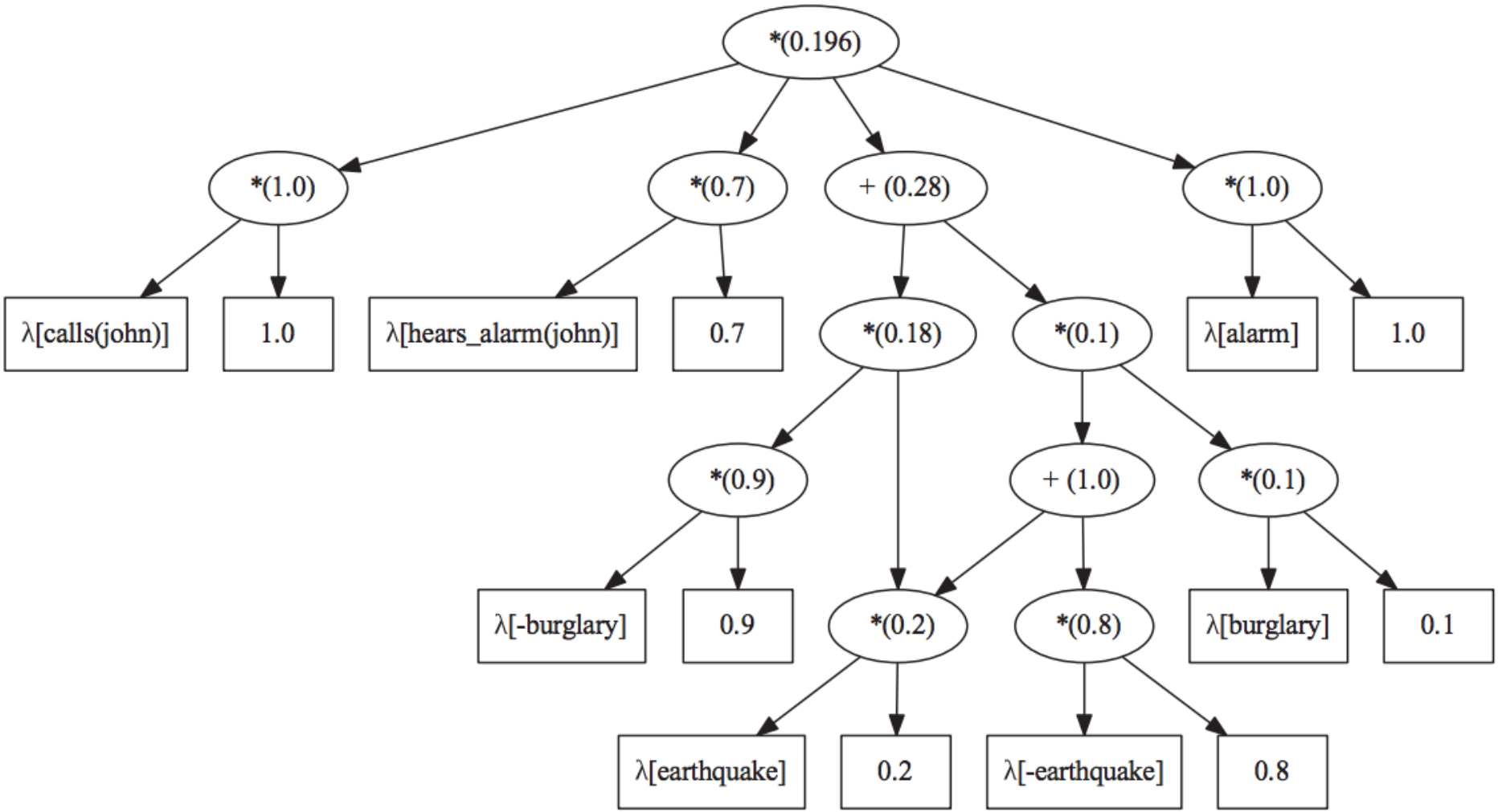
Knowledge Compilation to d-DNNF

(6) Convert the d-DNNF formula into an aritmetic circuit

1. Replace all conjunctions in the internal nodes of the d-DNNF by multiplications, and all disjunctions by summations
2. Replace every leaf node involving a literal l by a subtree consisting of a multiplication node having two children, namely, a leaf node with an indicator variable for the literal l and a leaf node with the weight of l according to the weighted formula
3. Numbers in parentheses represent results of the intermediate computations

Knowledge Compilation to d-DNNF

(6) Convert the d-DNNF formula into an aritmetic circuit



Knowledge Compilation to d-DNNF

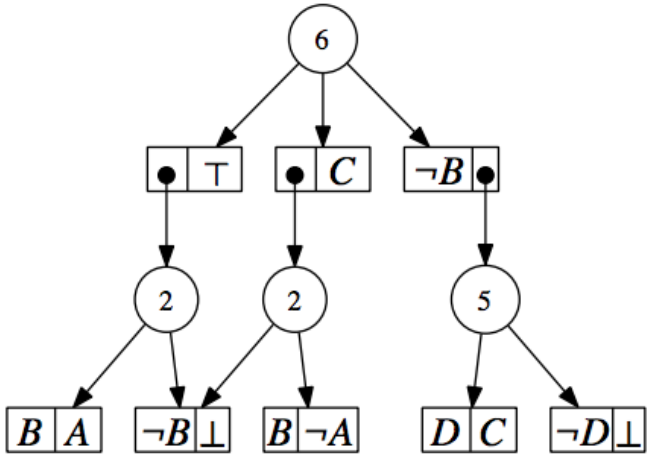
BDD vs d-DNNF

- A BDD is a special kind of d-DNNF, namely, one that satisfies the additional properties of ordering and decision
- In the approach seen earlier, we can replace a d-DNNF compiler with a BDD compiler
- Computing the probability of evidence can then be done by either operating directly on the BDD, or by converting the BDD to an arithmetic circuit and evaluating the circuit
- d-DNNFs outperform BDDs (Darwiche (2004))

Knowledge Compilation by SDD

- An SDD (Vlasselaer et al. (2014); Darwiche (2011)) contains two types of nodes
- *Decision nodes* (circles): **disjunctions** over mutually exclusive sentences
- *Elements* (paired boxes [p|s]): **conjunctions** between the two children
  - p: "prime"; s: "sub"
  - Elements are decision nodes' children and each box in an element can contain a pointer to a decision node or a terminal node, either a literal or the constants 0 or 1
- A decision node with children $[p_1|s_1], \ldots, [p_n|s_n]$ represents the function $(p_1 \wedge s_1) \vee \ldots \vee (p_n \wedge s_n)$
- Primes must form a partition: $p_i \neq 0$ (primes are consistent), $p_i \wedge p_j = 0$ for $i \neq j$ (every pair of distinct primes are mutually exclusive), and $p_1 \vee \ldots \vee p_n = 1$ (the disjunction of all primes is valid)

Knowledge Compilation by SDD



(b) Graphical depiction of an SDD

$$f = (A \wedge B) \vee (B \wedge C) \vee (C \wedge D).$$

The top level decomposition has three elements, with primes representing $A \wedge B, \neg A \wedge B, \neg B$ and corresponding subs representing *true*, C, and $C \wedge D$.

Knowledge Compilation by SDD

Procedure

1. Ground the probabilistic logic program and convert it into an equivalent propositional formula
2. **Compile directly the formula into an SDD** OR convert it into a CNF Boolean formula to be compiled into an SDD
3. Compute the probability of evidence from the SDD

Knowledge Compilation by SDD

Comparison of languages

- Succinctness: size of the smallest compiled circuit for every Boolean formula
  - $d - DNNF < SDD \leq OBDD$
  - There exists a Boolean formula whose smallest SDD representation is exponentially larger than its smallest d-DNNF representation, but the smallest OBDD for any formula is at least as big as its smallest SDD
- SDDs are special cases of d-DNNFs: if one replaces circle-nodes with or-nodes, and paired-boxes with and-nodes, one obtains a d-DNNF, with additional properties (structured decomposability and strong determinism)
- SDDs outperform d-DNNFs

Approximate Inference

- Approximate inference aims at computing the results of inference (probability of evidence $P(\mathbf{e})$) in an approximate way so that the process is cheaper than the exact computation of the results
- Two approaches: those that modify an exact inference algorithm and those based on sampling

- Iterative deepening
- *k-best*
- Monte Carlo

Approximate Inference

Iterative deepening

- De Raedt et al. (2007); Kimmig et al. (2008b)
- Input: an error bound ϵ, a depth bound d, and a query q
- Construct an SLD tree for q up to depth d
- Build two sets of explanations
  - K_l: set of composite choices corresponding to the successful proofs present in the tree
  - K_u: set of composite choices corresponding to the successful and still open proofs present in the tree
- $P(K_l)$: lower bound
- $P(K_u)$: upper bound of the exact probability

Approximate Inference

Iterative deepening

- If the difference $P(K_u) - P(K_l)$ is smaller than ϵ, this means that a solution with a satisfying accuracy has been found and the interval $[P(K_l), P(K_u)]$ is returned
- Otherwise, the depth bound is increased and a new SLD tree is built up to the new depth bound
- Stops when the difference $\leq \epsilon$

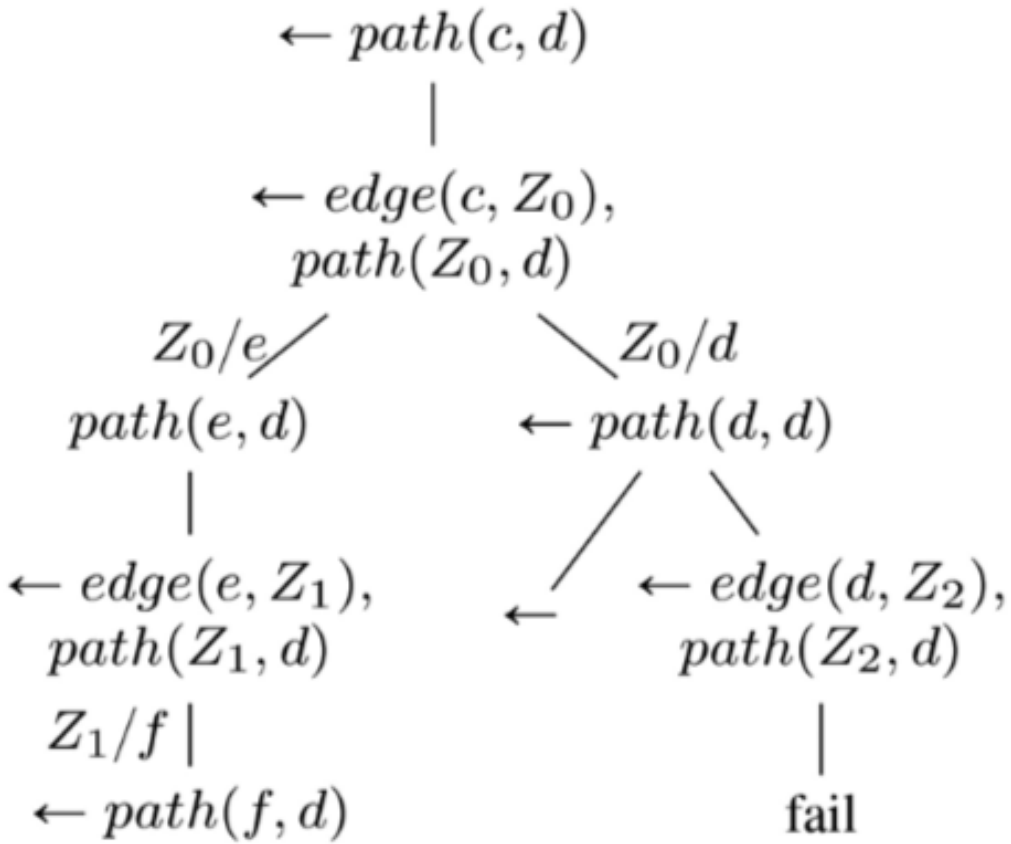
Approximate Inference

Iterative deepening - ProbLog syntax

$$path(X, X).$$
$$path(X, Y) \leftarrow edge(X, Z), path(Z, Y).$$
$$0.8 :: edge(a, c).$$
$$0.7 :: edge(a, b).$$
$$0.8 :: edge(c, e).$$
$$0.6 :: edge(b, c).$$
$$0.9 :: edge(c, d).$$
$$0.625 :: edge(e, f).$$
$$0.8 :: edge(f, d).$$

Approximate Inference

Iterative deepening



SLD tree up to depth 4 for the query $path(c, d)$ from the program

Approximate Inference

k-best

- Uses a fixed number k of proofs to obtain a lower bound of the probability of the query: the larger the k, the better the bound
- Given k, the best k proofs are found, corresponding to the set of best k explanations K_k
- $P(K_k)$ is a (lower) estimate of the probability of the query
- **Best** is intended in terms of probability: an explanation is better than another if its probability is higher
- **Branch and bound** approach: prune a derivation if its probability *falls below* that of the k-th best explanation

Approximate Inference

Monte Carlo

- Iterative procedure, until convergence
  1. Sample a world, by sampling each ground probabilistic fact/clause in turn
  2. Check whether the query is true in the world
  3. Compute the probability $\hat{p}$ of the query as the fraction of samples where the query is true
- Convergence is reached when the size of the confidence interval of $\hat{p}$ drops below a user-defined threshold δ
- If the width of the interval $< \delta$, it stops and returns $\hat{p}$

Approximate Inference

Monte Carlo

- The approach is not efficient on large programs, as proofs are often short while the generation of a world requires sampling many probabilistic facts
- Idea: generate samples lazily, by sampling probabilistic facts/clauses only when required by a proof
- In fact, it is not necessary to sample facts not needed by a proof, as any value for them would do

Approximate Inference

Monte Carlo Implementations

- **ProbLog1**: (Kimmig et al. (2011))
- **MCINTYRE** (Monte Carlo INference wiTh Yap REcord) (Riguzzi (2013a)): applies the Monte Carlo approach of ProbLog1 to LPADs using the YAP internal database for storing all samples and using tabling for speeding up inference
- Also available in SWI-Prolog (included in the `cplint` suite)

Approximate Inference

MCINTYRE: *Monte Carlo INference wiTh Yap REcord*

Example: The following LPAD models the development of an epidemic or a pandemic:

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

Clause C_1 is transformed in:

$$
\begin{aligned}
MC(C_1, 1) = \; & epidemic : -flu(X), cold,\\
& sample\_head([0.6, 0.3, 0.1], 1, [X], NH), NH = 1.\\
MC(C_1, 2) = \; & pandemic : -flu(X), cold,\\
& sample\_head([0.6, 0.3, 0.1], 1, [X], NH), NH = 2.
\end{aligned}
$$

Approximate Inference

MCINTYRE: *Monte Carlo INference wiTh Yap REcord*

- If $Q = epidemic$, resolution matches the goal with the head of clause $MC(C_1, 1)$.
- Suppose $flu(X)$ succeeds with $X/david$ and $cold$ succeeds as well.
- Then

$$sample\_head([0.6, 0.3, 0.1], 1, [david], NH)$$

  is called.
- Since clause 1 with X replaced by $david$ was not yet sampled, a number between 1 and 3 is sampled according to the distribution $[0.6, 0.3, 0.1]$ and stored in NH.
- If $NH = 1$, the derivation succeeds and the goal is true in the sample, if $NH = 2$ or $NH = 3$ then the derivation fails and backtracking is performed.

Approximate Inference

MCINTYRE: *Monte Carlo INference wiTh Yap REcord*

- This involves finding the solution $X/robert$ for $flu(X)$. $cold$ was sampled as true before so it succeeds again.
- Then

$$sample\_head([0.6, 0.3, 0.1], 1, [robert], NH)$$

is called to take another sample.

From discrete to continuous variables

- The languages presented up to now allowed the definition of *discrete random variables only*, i.e. variables taking values from a finite or countable set
- Probabilistic logic programs have been recently extended to deal with **hybrid relational domains**, involving both discrete and continuous random variables
- Hybrid ProbLog (Gutmann et al. (2010)), Distributional Clauses (DC) (Gutmann et al. (2011)), extended PRISM (Islam et al. (2012)), cplint Hybrid Programs (Alberti et al. (2017))
- The *continuous distributions* are defined over variables in the facts/clauses in the program, **variables that take values from uncountable sets** such as that of the real numbers

cplint Hybrid Programs (HP)

- *cplint* HP handle both discrete and continuous probability distributions
- Discrete probability distribution: applicable when the set of possible outcomes is discrete (a coin toss or a roll of dice) and can be encoded by a **discrete list of the probabilities of the outcomes**, known as a *probability mass function*
- Continuous probability distribution: applicable when the set of possible **outcomes can take on values in a continuous range** (e.g. real numbers, such as the temperature on a given day). Described by *probability density functions* (PDF), with the probability of any individual outcome being 0

cplint Hybrid Programs

- *cplint* allows the specification of **probability density/mass functions over an argument X of atoms in the head of rules**
- $a(A, B, ..., X) : Density \leftarrow Body$
  where *Density* is a special atom identifying a probability distribution on variable X and Body (optional) is a regular clause body
- *Example*

$$g(X) : gaussian(X, 0, 1).$$

  states that argument X of g(X) follows a (univariate) Gaussian distribution with mean $\mu = 0$ and variance $\sigma^2 = 1$

cplint Hybrid Programs

Gaussian distribution examples

- $g(X) : gaussian(X, [0, 0], [[1, 0], [0, 1]])$.
  states that argument X of $g(X)$ follows a Gaussian multivariate distribution with mean vector $[0, 0]$ and covariance matrix

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

  .
- $temp(D, X) : gaussian(X, 2, 8)$.
  states that the temperature for day D is Gaussian-distributed with mean 2 and standard deviation 8

cplint Hybrid Programs

Gaussian distribution examples

- Example of a Gaussian mixture model (a probabilistic model that assumes all the data points are generated from a mixture of a finite number of Gaussian distributions)

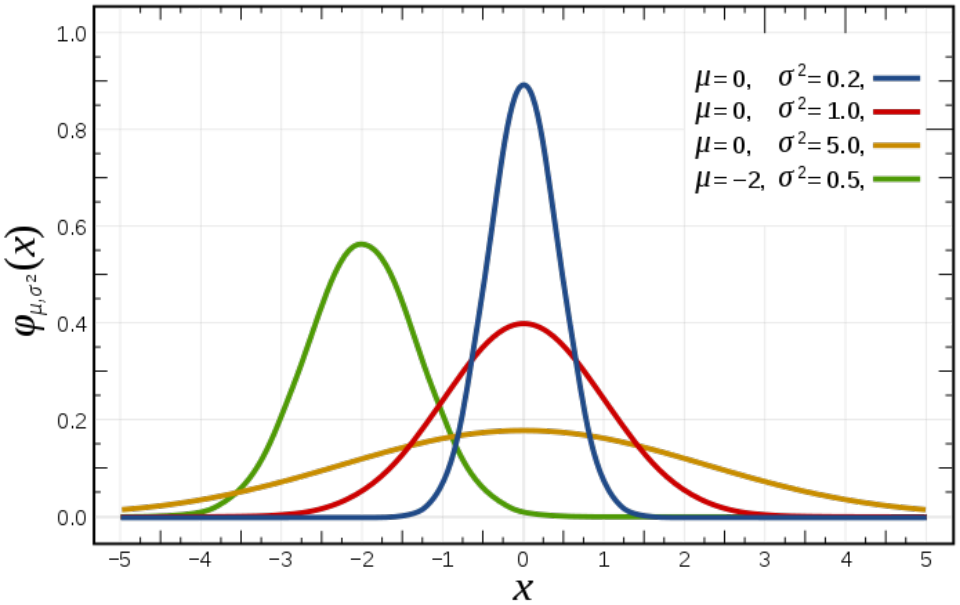
$$heads : 0.6; tails : 0.4.$$
$$g(X) : gaussian(X, 0, 1).$$
$$h(X) : gaussian(X, 5, 2).$$
$$mix(X) \leftarrow heads, g(X).$$
$$mix(X) \leftarrow tails, h(X).$$

The argument X of $mix(X)$ follows a distribution that is a mixture of two Gaussians, one with mean 0 and variance 1 *with probability 0.6* and one with mean 5 and variance 2 *with probability 0.4*.

cplint Hybrid Programs

Probability density functions

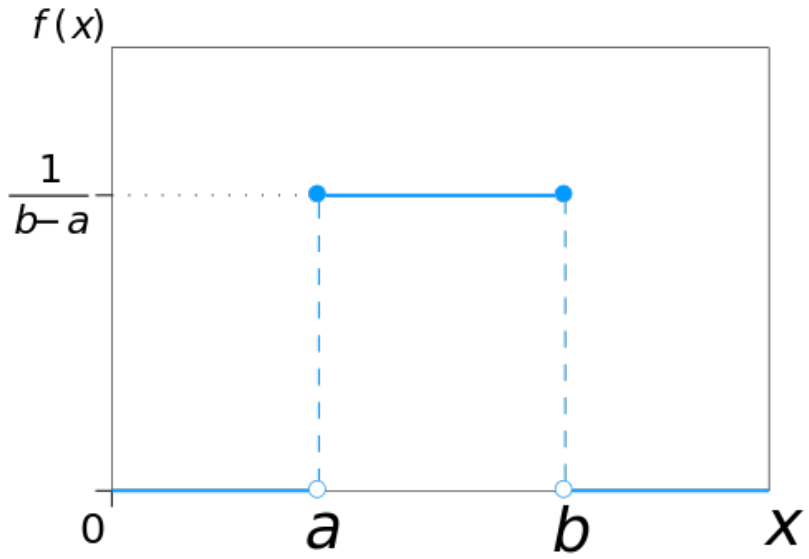
- *gaussian*(*Var*, *M*, *V*): *Var* follows a Gaussian distribution with mean *M* and variance *V*. The distribution can be multivariate if *M* is a list and *V* a list of lists representing the mean vector and the covariance matrix
- A Gaussian distribution is often used in the natural and social sciences to represent real-valued random variables whose distributions are not known and thery are assumed to concentrate around a mean



cplint Hybrid Programs

Probability density functions

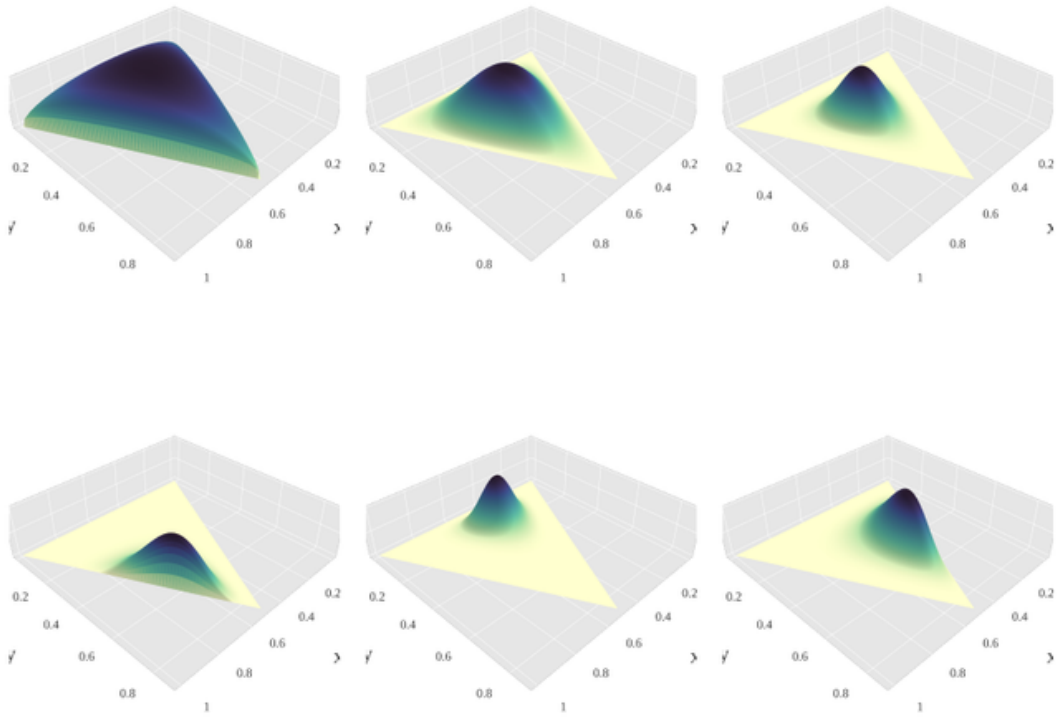
- *uniform*(*Var*, *L*, *U*): *Var* is uniformly distributed in [*L*, *U*]
- It represents a situation where all outcomes in a range between a minimum and maximum value are equally likely



cplint Hybrid Programs

Probability density functions

- *dirichlet*(*Var*, *Par*): it is a family of continuous multivariate probability distributions parameterized by a vector α of positive reals. *Par* specifies the α parameters

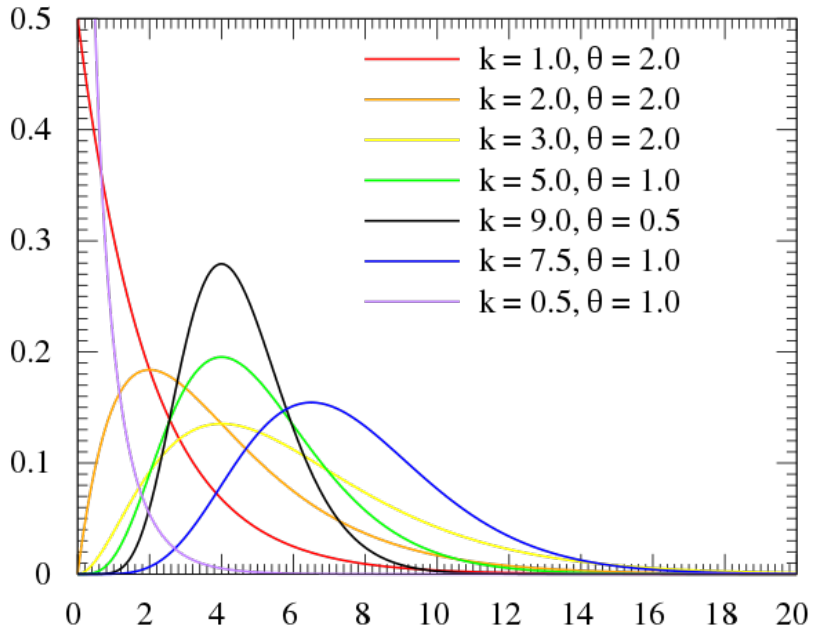


Dirichlet distributions with different α vectors

cplint Hybrid Programs

Probability density functions

- *gamma*(*Var*, *Shape*, *Scale*): *Var* follows a gamma distribution with shape parameter *Shape* and scale parameter *Scale*

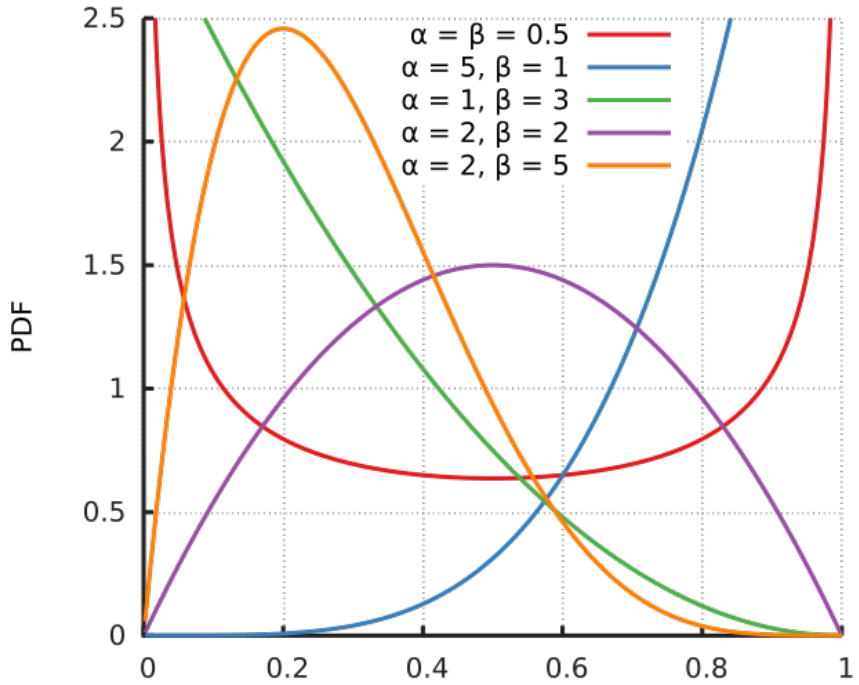


- k: shape parameter; θ: scale parameter

cplint Hybrid Programs

Probability density functions

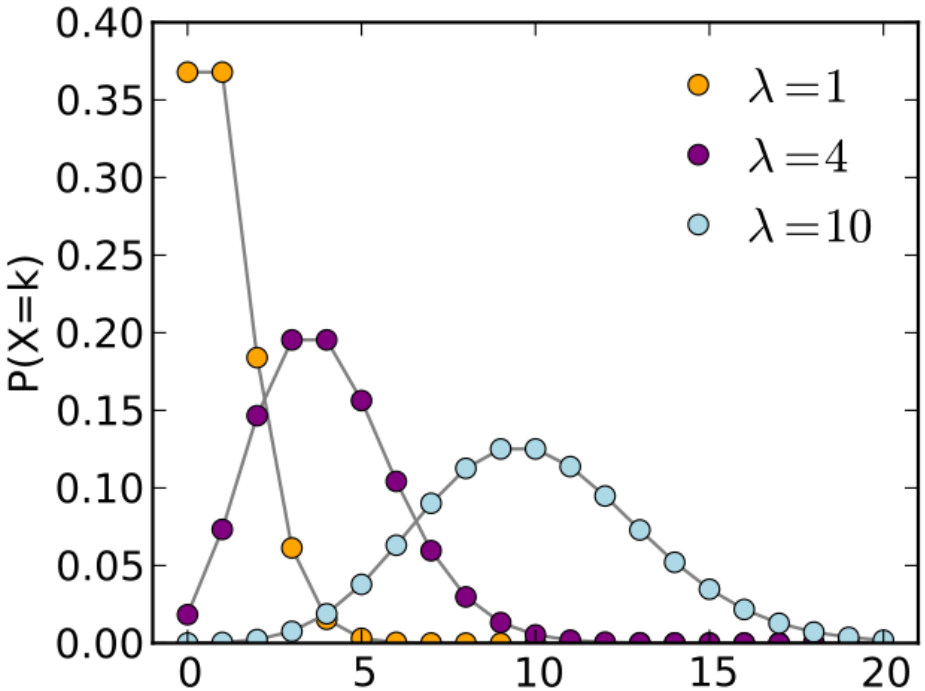
- *beta*(*Var*, α, β): *Var* follows a beta distribution with parameters α and β; it's defined on the interval [0, 1]



cplint Hybrid Programs

Probability mass functions

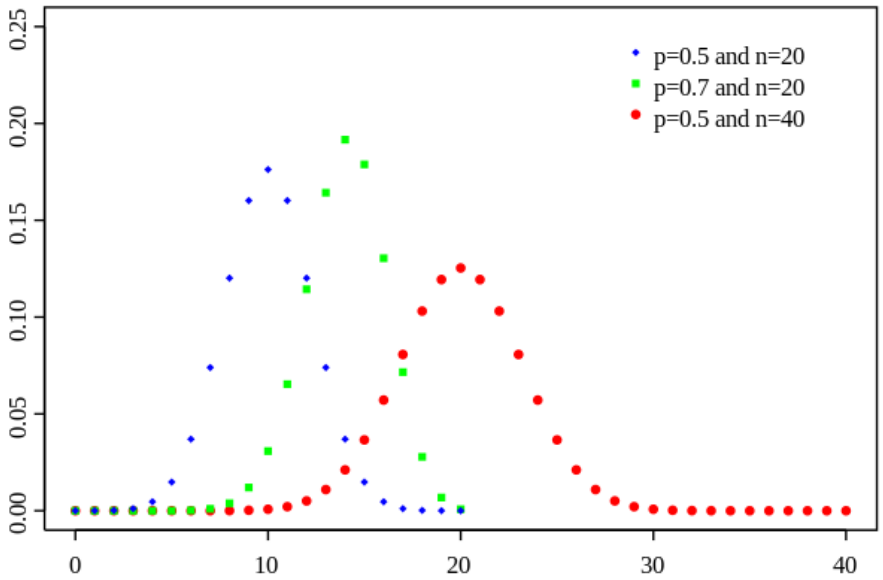
- *poisson*(*Var*, λ): *Var* follows a Poisson distribution with parameter λ
- Discrete probability distribution that expresses the probability of a given number of events occurring in a fixed interval of time if these events occur with a known constant rate (λ) and independently of the time since the last event



cplint Hybrid Programs

Probability mass functions

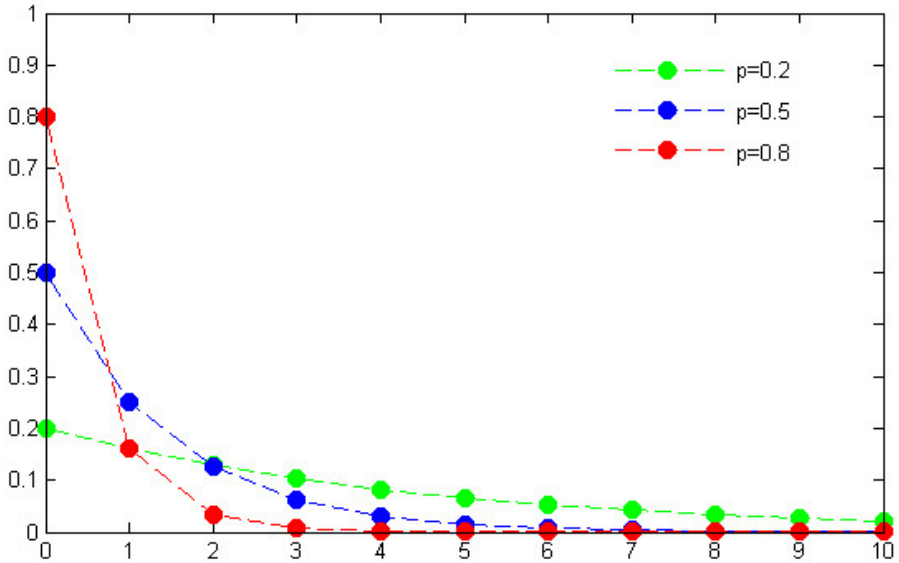
- *binomial*(*Var*, *N*, *P*): *Var* follows a binomial distribution with parameters *N* and *P*
- Discrete probability distribution of the number of successes in a sequence of *N* independent experiments, each asking a yes-no question; *P* is the success probability of a single experiment (trial)



cplint Hybrid Programs

Probability mass functions

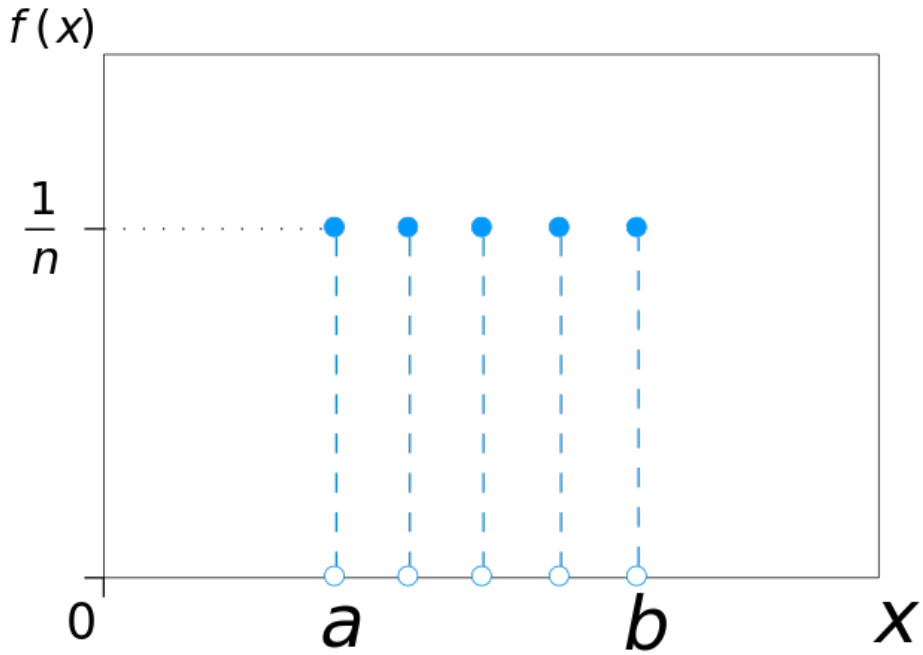
- *geometric*(*Var*, *P*): *Var* follows a geometric distribution with parameter *P*
- Discrete probability distribution which gives the probability that the first occurrence of success requires k independent trials, each with success probability *P*



cplint Hybrid Programs

Probability mass functions

- $uniform(Var, D)$
- Discrete probability distribution where a finite number of values (specified in the list D) are equally likely to be observed with probability $1/length(D)$



cplint Hybrid Programs

Probability mass functions

- *discrete*(*Var*, *D*) or *finite*(*Var*, *D*): *D* is a list of couples *Value* : *Prob* assigning probability *Prob* to *Value*

Semantics

- Both cplint Hybrid Programs and Distributional Clauses follow (Nitti et al. (2016)), where the semantics is based on the ST_P operator (S for stochastic), a generalization of the T_P operator for logic programming
- Let $\mathcal{P}$ be a definite program and I a Herbrand interpretation

$$T_P(I) = \{h\theta | h \leftarrow b_1, ..., b_n \in \mathcal{P}, \{b_1\theta, ..., b_n\theta\} \subseteq I\}$$

If the body of a rule is true in I, the head is in $T_P(I)$.

Semantics

Definition (ST_P operator - cplint)

Let $\mathcal{P}$ be a probabilistic logic program. Starting from an interpretation I:

$$
\begin{aligned}
ST_P(I) \quad = \quad & \{h' | h : Density \leftarrow b_1, \ldots, b_n \in P, \{b_1\theta, ..., b_n\theta\} \subseteq I \\
& s.t.\ h' = h\{Var/v\} \text{ with } Var \text{ the continuous variable of } h \\
& \text{and v is sampled from distribution } Density\} \\
\cup \\
& \{h | Dist \leftarrow b_1, \ldots, b_n \in P, \{b_1\theta, ..., b_n\theta\} \subseteq I \\
& \text{with } h \text{ sampled from discrete distribution } Dist\}
\end{aligned}
$$

Semantics

- For each probabilistic clause $h : Density \leftarrow b_1 \ldots, b_n$ whenever the body $b_1 \ldots, b_n$ is true in I with substitution θ, a value v for the continuous variable Var of h is sampled from the distribution $Density$ and the random variable $h\theta = h\{Var/v\}$ is added to the interpretation
- Similarly for discrete and deterministic clauses
- *cplint* HP can handle DC and Extended PRISM by translating clauses in these languages to cplint HP clauses
- *cplint* HP allow more expressivity freedom than DC and Extended PRISM

Semantics

- An hybrid program $\mathcal{P}$ is a set of clauses with continuous and/or discrete r.v. that defines a distribution $P(x)$ over possible worlds x
- The probability $P(q)$ of a query q can be estimated using Monte-Carlo methods: possible worlds are sampled from $P(x)$, and $P(q)$ is approximated as the ratio of samples in which the query q is true
- The process is based on sampling, thus 'world' is often replaced with *sample or particle*
- Note: a possible world may contain a countably infinite number of random variables in the case of programs with function symbols

Semantics

Example: People position

$n(N) : poisson(N, 6).$
$position(P, Y) : uniform(Y, 0, M) \leftarrow n(N), between(1, N, P), M \ is \ 10 * N.$
$left(A, B) \leftarrow position(A, PA), position(B, PB), PA < PB.$

- Clause (1): the number of people $n(N)$ is governed by a Poisson distribution with mean $\lambda = 6$
- Clause (2) models the position as a continuous random variable uniformly distributed from 0 to M = 10N (10 times the number of people) for each person P such that $1 \leq P \leq N$, and N unifies with the number of people

Semantics

- if $N = 2$, there will be 2 independent random variables $position(1)$ and $position(2)$ with distribution $uniform(0, 20)$
- {N = 2, position(1,3.1), position(2,4.5), left(1, 2)} is a possible (complete) world
- 3.1 and 4.5 are uniformly sampled in [0,20]
- After sampling a number of worlds, the fraction where $left(1, 2)$ is true is $P(left(1, 2))$

Inference

- Sampling full worlds is generally inefficient or may not even terminate as possible worlds can be infinitely large
- Solution: **approximate inference by sampling** (Nitti et al. (2016), Alberti et al. (2017))
- Inference on cplint HP can be performed by MCINTYRE (Monte Carlo INference wiTh Yap REcord) (Riguzzi (2013b))
- In presence of continuous random variables, allows to **sample arguments of goals representing continuous random variables**
- In this way one can build **a probability density of the sampled argument**

Inference on *cplint* Hybrid Programs

- Unconditional inference: Monte Carlo sampling and Gibbs sampling, a MCMC (Markov Chain Monte Carlo) method
- Conditional inference: when evidence is available on ground atoms that have continuous values as arguments, *likelihood weighting* or *particle filtering* (Nitti et al. (2016)) to obtain samples of continuous arguments of a goal

Inference on *cplint* Hybrid Programs

Unconditional Inference with Monte Carlo

- $mc\_sample\_arg(:Query:atom, +Samples:int, ?Arg:var, -Values:list)$.

  samples *Query* a number of *Samples* times.
- *Arg* should be a variable in *Query*
- The predicate returns in *Values* a list of couples $L - N$ where L is the list of values of *Arg* for which *Query* succeeds in a world sampled at random and N is the number of samples returning that list of values
- If L is empty, it means that for that sample the query failed
- If, in all couples L-N, L is a list with a single element, it means that the clauses in the program are mutually exclusive, i.e., that in every sample, only one clause for each subgoal has the body true

Inference on *cplint* Hybrid Programs

Unconditional Inference with Monte Carlo

- By querying the program
  $heads : 0.6; tails : 0.4.$
  $g(X) : gaussian(X, 0, 1).$
  $h(X) : gaussian(X, 5, 2).$
  $mix(X) : -heads, g(X).$
  $mix(X) : -tails, h(X).$

  with `?- mc_sample_arg(mix(X),10,X,L).`
- one gets
  $L = [[-1.649529159850351] - 1, [-1.1835705080559966] - 1, [-0.392969376975151] - 1, [-0.3539071368497263 6] - 1, [1.1232140506496011] - 1, [4.313659663263742] - 1, [4.662640966272441] - 1, [4.68700033893297] - 1, [6.050170140267 1895] - 1, [6.694841213586896] - 1]$

Inference on *cplint* Hybrid Programs

Unconditional Inference with Monte Carlo

- The query `mc_sample_arg(mix(X),1000,X,L),histogram(L,Chart,[nbins(40)]).` takes 1000 samples of X in $mix(X)$ and draws a histogram with 40 bins representing the probability density of X
- See http://cplint.eu/example/inference/gaussian_mixture.pl
- Other variants of `mc_sample_arg`, see the cplint manual (`http://cplint.eu/help/help-cplint.html#uncondq`)

Inference on *cplint* Hybrid Programs

Unconditional Inference with Gibbs sampling

- *mc_gibbs_sample*(: *Query* : *atom*, +*Samples* : *int*, −*Probability* : *float*, +*Options* : *list*)
- *mc_gibbs_sample_arg*(: *Query* : *atom*, +*Samples* : *int*, ?*Arg* : *var*, −*Values* : *list*, +*Options* : *list*)
- Used in the same way seen in the Monte Carlo case

Inference on *cplint* Hybrid Programs

Conditional Inference with likelihood weighting (LW)

- For each sample to be taken, likelihood weighting samples the query and then assigns a weight to the sample on the basis of evidence
- **The weight is computed by deriving the evidence backward in the same sample of the query starting with a weight of one**: each time a choice should be taken or a continuous variable sampled, if the choice/variable has already been sampled, the current weight is multiplied by the probability of the choice/by the density value of the continuous variable
- If the value has not been sampled, it takes a sample and records it, leaving the weight unchanged
- In this way, **each sample of the query is associated with a weight that reflects the influence of evidence**

Inference on *cplint* Hybrid Programs

Conditional Inference with likelihood weighting (LW)

- The probability of the query is computed as the sum of the weights of the samples where the query is true divided by the total sum of the weights of the samples
- *cplint* HP randomize the choice of clauses when more than one resolves the goal, in order to obtain an unbiased sample
- $mc\_lw\_sample(:Query:atom,:Evidence:atom,+Samples:int,-Prob:float)$ samples *Query* a number of *Samples* times given that *Evidence* (one or a conjunction of atoms) is true. *Prob* is the probability that the query is true
- $mc\_lw\_expectation(:Query:atom,Evidence:atom,+N:int,?Arg:var,-Exp:float)$ computes the expected value of *Arg* in *Query* given that *Evidence* is true. It takes *N* samples of *Arg* in *Query*, weighting each according to the evidence, and returns their weighted average

Hands On

- http://cplint.eu/example/figaro_coin.swinb
- Immagine you have found a coin of dubious origin: it might be biased or not so you don't know the probability that it'll lands heads on any given toss.
- Goal: estimating the bias of the coin and predicting consecutive coin tosses

Inference on *cplint* Hybrid Programs

Conditional Inference with Particle Filtering

- When you have a dynamic model and observations on continuous variables for a number of time points, or your evidence is represented by many atoms, **likelihood weighting has numerical stability problems**, as samples' weight may go rapidly to 0 due to floating point arithmetic
- Particle filtering (PF) periodically resamples the individual samples/particles so that their weight is reset to 1
- *Each sample constitutes a particle*
- In this case, *evidence is a list of literals*

Inference on *cplint* Hybrid Programs

Conditional Inference with Particle Filtering

1. **Prediction step**: samples a new set of N samples of the query, from the distribution
2. **Weighting step**: assigns to each sample $x(i), i = 1...N$, the weight $w(i)$ with the likelihood of the first element of the evidence list
3. **Resampling**: if the variance of the sample weights exceeds a certain threshold, resample with replacement from the sample set, with probability proportional to $w(i)$ and set the weights to 1
4. After resampling, the next element of the evidence is considered. A new weight for each particle is computed on the basis of the new evidence element and the process is repeated until the last evidence element

Inference on *cplint* Hybrid Programs

Conditional Inference with particle filtering

- *mc_particle_sample*(: *Query* : *atom*, : *Evidence* : *list*, +*Samples* : *int*, −*Prob* : *float*)
  samples *Query* a number of *Samples* times given that *Evidence* is true. Evidence is a list of goals. *Prob* is the probability that the query is true.

References I

Alberti, M., Bellodi, E., Cota, G., Riguzzi, F., and Zese, R. (2017). cplint on SWISH: probabilistic logical inference with a web browser. *Intelligenza Artificiale*, 11(1):47–64.

Darwiche, A. (2004). New advances in compiling cnf to decomposable negation normal form. In *Proceedings of the 16th European Conference on Artificial Intelligence*, ECAI'04, pages 318–322, Amsterdam, The Netherlands, The Netherlands. IOS Press.

Darwiche, A. (2011). SDD: A new canonical representation of propositional knowledge bases. In *IJCAI*, pages 819–826. IJCAI/AAAI.

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.

References II

Fierens, D., Van den Broeck, G., Renkens, J., Shterionov, D., Gutmann, B., Thon, I., Janssens, G., and De Raedt, L. (2015). Inference and learning in probabilistic logic programs using weighted Boolean formulas. *Theory and Practice of Logic Programming*, 15:358–401.

Gutmann, B., Jaeger, M., and De Raedt, L. (2010). Extending problog with continuous distributions. In Frasconi, P. and Lisi, F. A., editors, *Proceedings of the 20th International Conference on Inductive Logic Programming (ILP–10)*, Firenze, Italy. (Accepted).

Gutmann, B., Thon, I., Kimmig, A., Bruynooghe, M., and Raedt, L. D. (2011). The magic of logical inference in probabilistic programming. *TPLP*, 11(4-5):663–680.

Islam, M. a., Ramakrishnan, C. r., and Ramakrishnan, I. v. (2012). Inference in probabilistic logic programs with continuous random variables. *Theory Pract. Log. Program.*, 12(4-5):505–523.

References III

Janhunen, T. (2004). Representing normal programs with clauses. In de Mántaras, R. L. and Saitta, L., editors, *Proceedings of the 16th Eureopean Conference on Artificial Intelligence, ECAI'2004, including Prestigious Applicants of Intelligent Systems, PAIS 2004, Valencia, Spain, August 22-27, 2004*, pages 358–362. IOS Press.

Kimmig, A., Demoen, B., De Raedt, L., Costa, V. S., and Rocha, R. (2011). On the implementation of the probabilistic logic programming language ProbLog. 11(2-3):235–262.

Kimmig, A., Santos Costa, V., Rocha, R., Demoen, B., and De Raedt, L. (2008a). On the efficient execution of problog programs. In Garcia de la Banda, M. and Pontelli, E., editors, *Logic Programming*, pages 175–189, Berlin, Heidelberg. Springer Berlin Heidelberg.

Kimmig, A., Santos Costa, V., Rocha, R., Demoen, B., and De Raedt, L. (2008b). On the efficient execution of ProbLog programs. volume 5366 of *LNCS*, pages 175–189.

References IV

Lloyd, J. W. (1987). *Foundations of Logic Programming, 2nd Edition*.

Mantadelis, T. and Janssens, G. (2010). Dedicated tabling for a probabilistic setting. volume 7 of *LIPIcs*, pages 124–133.

Meert, W., Struyf, J., and Blockeel, H. (2010). Cp-logic theory inference with contextual variable elimination and comparison to bdd based inference methods. In De Raedt, L., editor, *Inductive Logic Programming*, pages 96–109, Berlin, Heidelberg. Springer Berlin Heidelberg.

Muise, C. J., McIlraith, S. A., Beck, J. C., and Hsu, E. I. (2012). Dsharp: Fast d-DNNF compilation with sharpSAT. volume 7310, pages 356–361.

Nitti, D., De Laet, T., and De Raedt, L. (2016). Probabilistic logic programming for hybrid relational domains. volume 103, pages 407–449. Springer Verlag.

References V

Poole, D. (2003). First-order probabilistic inference. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, IJCAI'03, pages 985–991, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

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*, 17(6):589–629.

Riguzzi, F. (2013a). MCINTYRE: A Monte Carlo system for probabilistic logic programming. 124(4):521–541.

Riguzzi, F. (2013b). MCINTYRE: A monte carlo system for probabilistic logic programming. *Fundam. Inform.*, 124(4):521–541.

References VI

Riguzzi, F. and Swift, T. (2010a). 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.

Riguzzi, F. and Swift, T. (2010b). Tabling and answer subsumption for reasoning on logic programs with annotated disjunctions. In *ICLP (Technical Communications)*, volume 7 of *LIPIcs*, pages 162–171. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik.

Vlasselaer, J., Renkens, J., Van den Broeck, G., and De Raedt, L. (2014). Compiling probabilistic logic programs into sentential decision diagrams. pages 1–10.