Probabilistic Logic Programming
Hybrid Programs

Elena Bellodi

Department of Engineering
University of Ferrara, Italy

University of Udine, PhD Course, December 11th-12th 2018

Outline

Hybrid Probabilistic Logic Programs

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

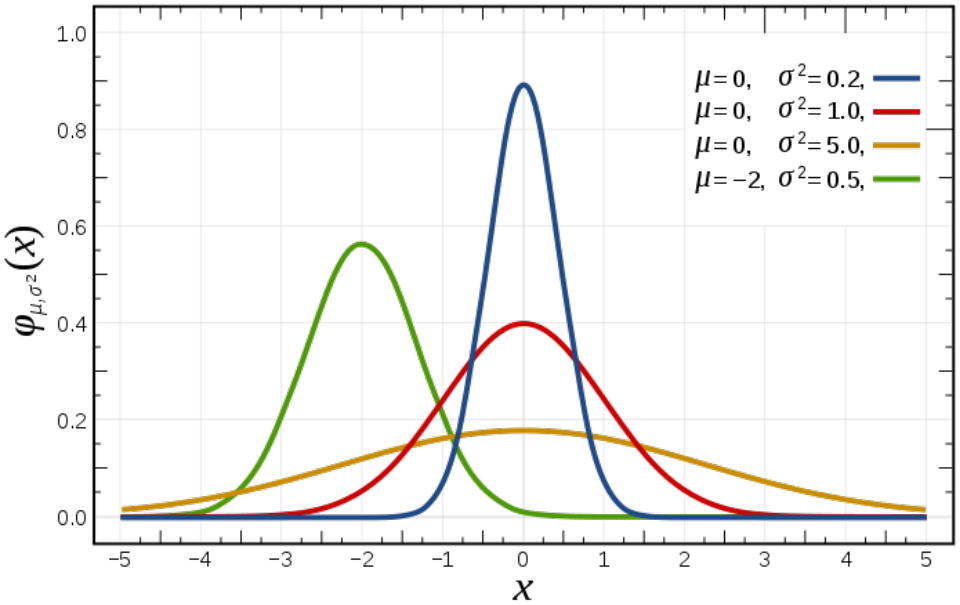
  $heads : 0.6; tails : 0.4.$
  $g(X) : gaussian(X, 0, 1).$
  $h(X) : gaussian(X, 5, 2).$
  $mix(X) \leftarrow heads, g(X).$
  $mix(X) \leftarrow tails, h(X).$

  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

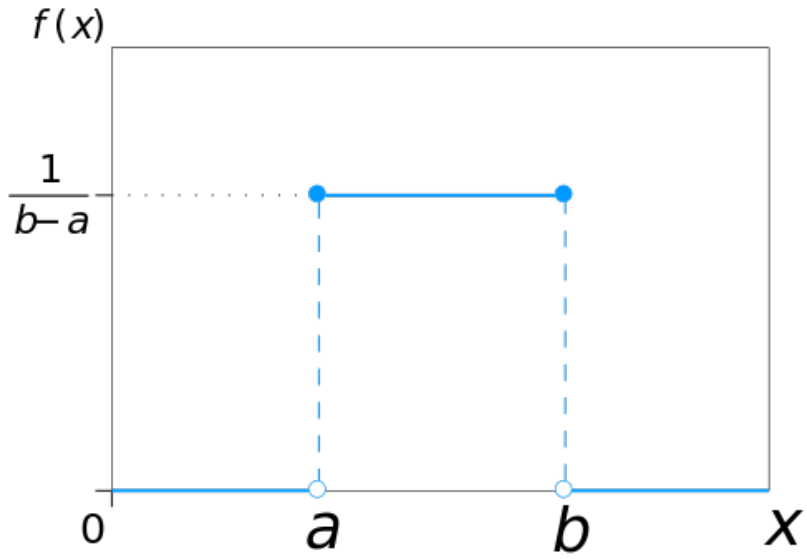
- *gaussian*(*Var*, *M*, *V*): *Var* follows a Gaussian distribution with mean *M* and variance *V*. The distribution can be multivariate if *M* is a list and *V* a list of lists representing the mean vector and the covariance matrix
- A Gaussian distribution is often used in the natural and social sciences 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

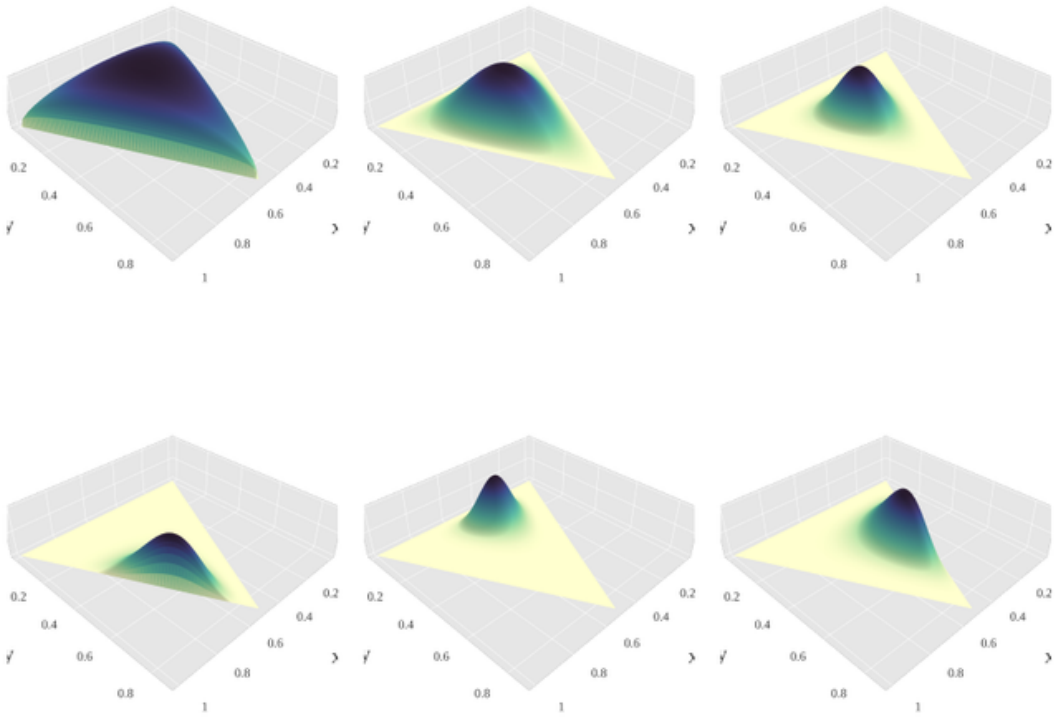
- *uniform*(*Var*, *L*, *U*): *Var* is uniformly distributed in [*L*, *U*]
- It represents a situation where all outcomes in a range between a minimum and maximum value are equally likely



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

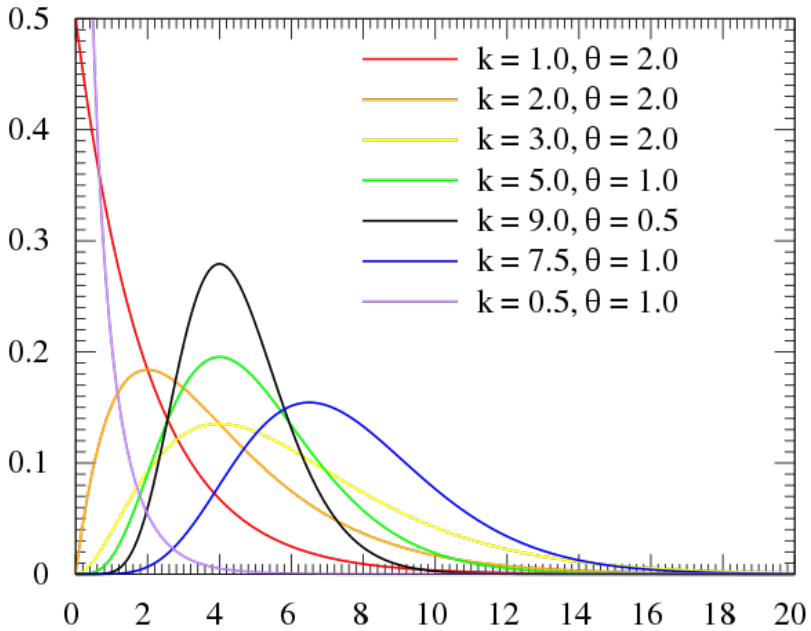


Dirichlet distributions with different α vectors

cplint Hybrid Programs

Probability density functions

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

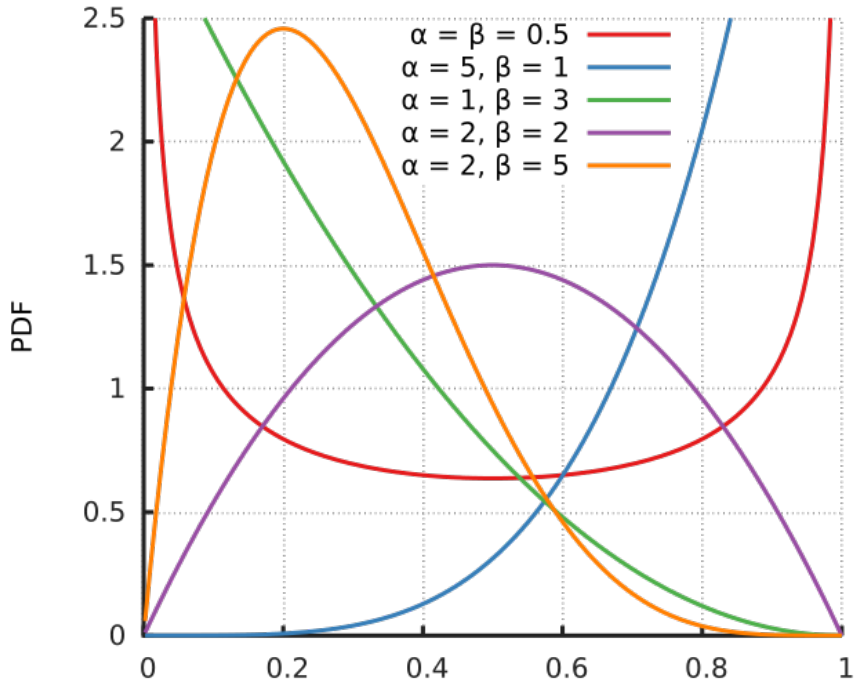


- k: shape parameter; θ: scale parameter

cplint Hybrid Programs

Probability density functions

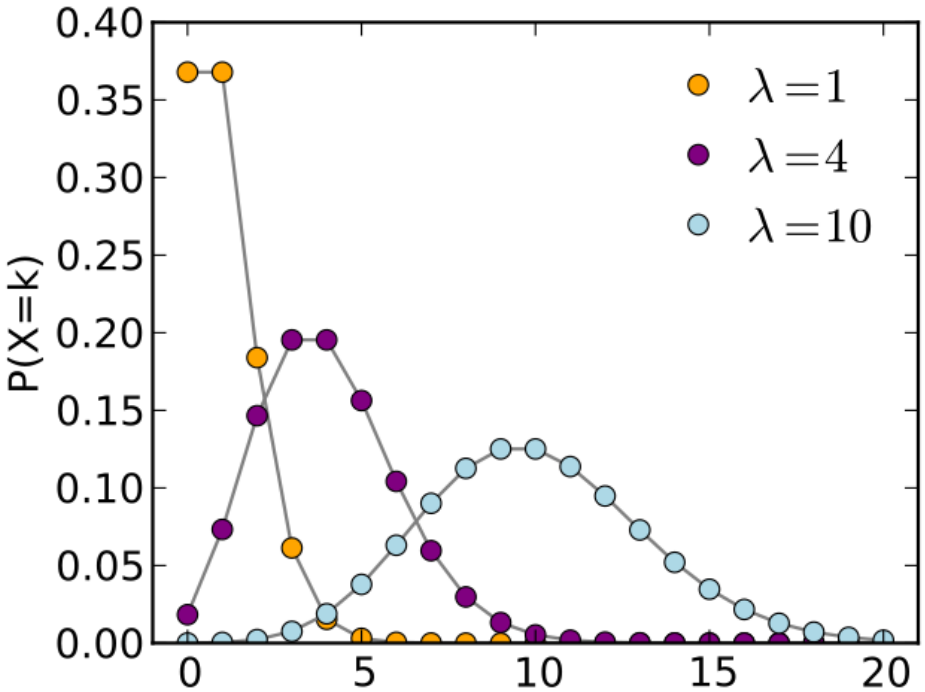
- *beta*(*Var*, α, β): *Var* follows a beta distribution with parameters α and β; it's defined on the interval [0, 1]



cplint Hybrid Programs

Probability mass functions

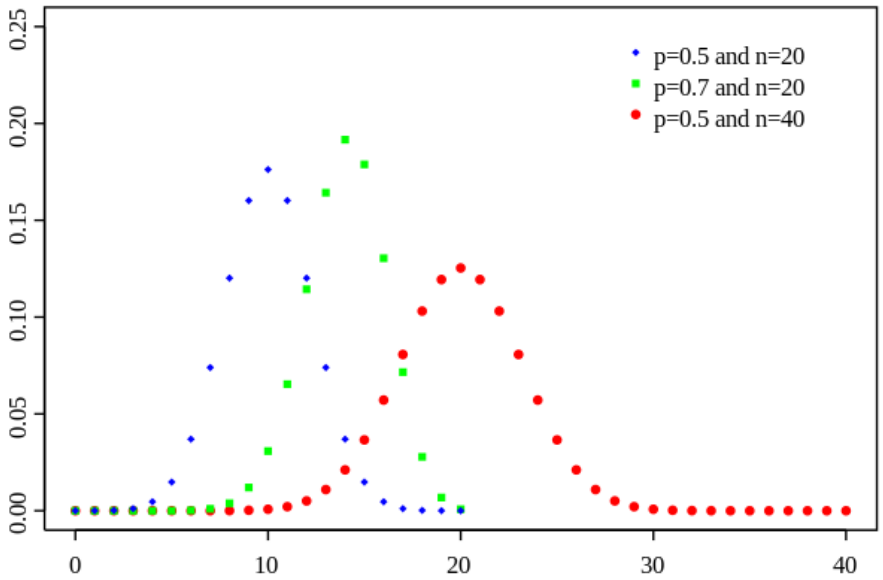
- *poisson*(*Var*, λ): *Var* follows a Poisson distribution with parameter λ
- Discrete probability distribution that expresses the probability of a given number of events occurring in a fixed interval of time if these events occur with a known constant rate (λ) and independently of the time since the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



cplint Hybrid Programs

Probability mass functions

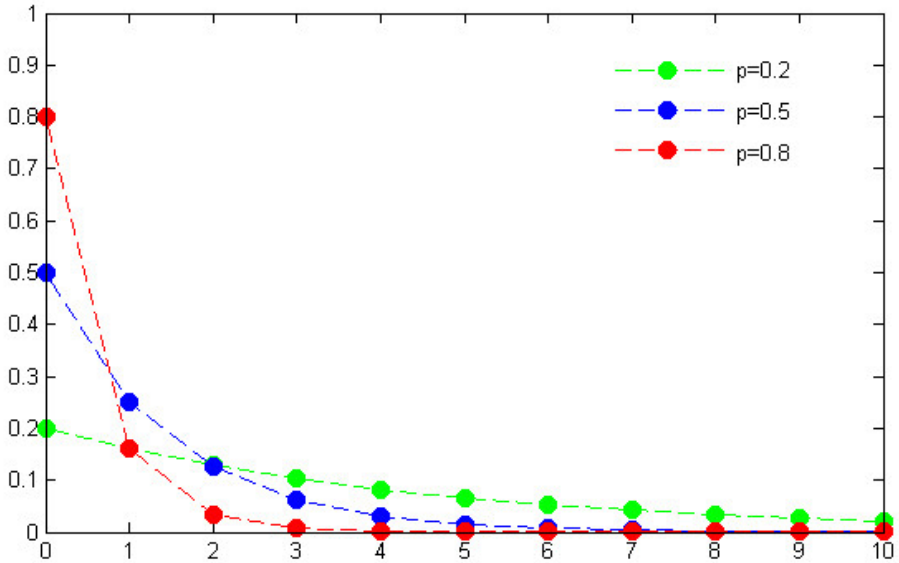
- *binomial*(*Var*, *N*, *P*): *Var* follows a binomial distribution with parameters *N* and *P*
- Discrete probability distribution of the number of successes in a sequence of *N* independent experiments, each asking a yes-no question; *P* is the success probability of a single experiment (trial)



cplint Hybrid Programs

Probability mass functions

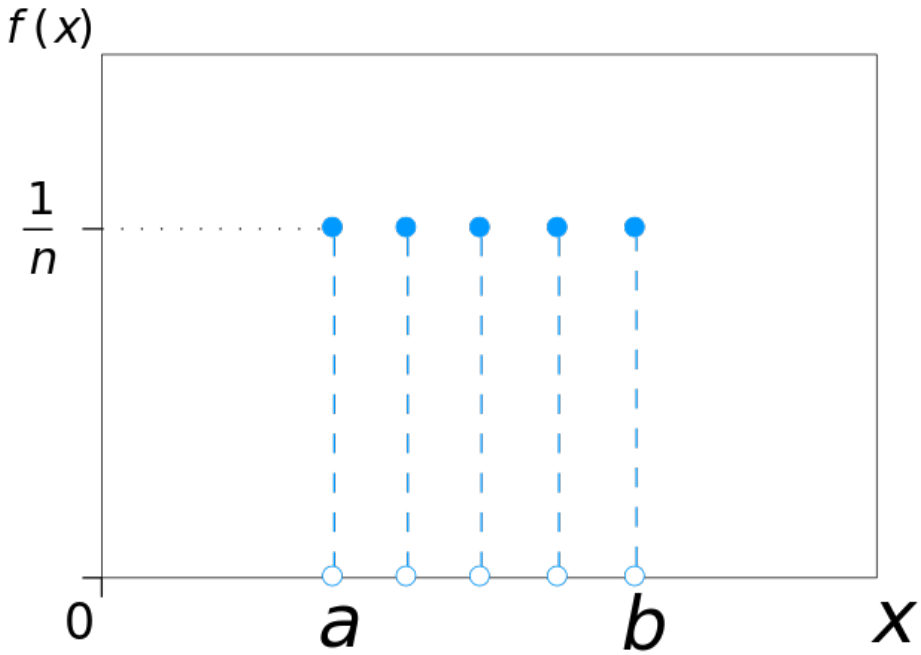
- *geometric*(*Var*, *P*): *Var* follows a geometric distribution with parameter *P*
- Discrete probability distribution which gives the probability that the first occurrence of success requires k independent trials, each with success probability *P*



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

cplint Hybrid Programs

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

cplint Hybrid Programs

- 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

cplint Hybrid Programs

- 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

cplint Hybrid Programs

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 position(P) 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
- if $N = 2$, there will be 2 independent random variables position(1) and position(2) with distribution $uniform(0, 20)$

Semantics

cplint Hybrid Programs

- {N = 2, pos(1,3.1), pos(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 (2013))
- 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
  $L = [[-1.649529159850351] - 1, [-1.1835705080559966] - 1, [-0.392696376975151] - 1, [-0.3539071368497263 6] - 1, [1.1232140506496011] - 1, [4.313659663263742] - 1, [4.662640966272441] - 1, [4.68700033893297] - 1, [6.050170140267 1895] - 1, [6.694841213586896] - 1]$

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.

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 II

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

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