A speech about Generative Datalog and Non-measurable Sets

Mario Alviano, Arnel Zamayla

Department of Mathematics and Computer Science
University of Calabria
Rende (CS), IT 87036

Abstract

Generative Datalog is the first component of PPDL (short for Probabilistic-Programming Datalog), a recently proposed probabilistic programming language. Specifically, generative Datalog provides constructs to refer to parameterized probability distribution, and is used for the specification of stochastic processes. Possible outcomes of such a stochastic process are possibly filtered according to logical constraints, which constitute the second component of PPDL. This speech is about generative Datalog, and hints on the possibility to represent non-measurable sets by combining generative Datalog constructs with addition over real numbers and a single, atomic, ground constraint.

Keywords

Datalog, probabilistic reasoning, non-measurable sets, stable model semantics

1. Introduction

Many probabilistic programming languages extend a deterministic programming language with primitive constructs for expressing random choices [1, 2]. *Generative Datalog* [3, 4] is not an exception and extends Datalog with Δ-*terms*, primitive constructs for representing *parametrized probability distributions*. Semantically, generative Datalog programs are mapped to existential Datalog programs, so to represent the uncertainty of Δ-terms. In such existential Datalog programs the outcome of Δ-terms is encoded by means of auxiliary predicates, so that only ground Δ-terms that are involved in the computation of inferred facts are eventually represented in models. Hence, the evaluation of a generative Datalog program is seen as a probabilistic process, and as such the semantics of generative Datalog can be formalized in terms of *probability spaces*, whose existence is claimed by Theorem 3.8 in [4].

On the other hand, PPDL (short for *Probabilistic-Programming Datalog*) [3, 4] adds logical constraints to generative Datalog, and assumes that the set of filtered possible outcomes is a measurable set (see Definition 5.3 in [4]). While in the finite case measurability of such a set is clear, the general case is non-trivial. In fact, non-measuable sets can be represented by combining generative Datalog constructs with addition over real numbers and a single, atomic,

ASPOCP 2021: 14th Workshop on Answer Set Programming and Other Computing Paradigms, September 20–27, 2021, Porto, Portugal
mario.alviano@unical.it (M. Alviano); zamayla@mat.unical.it (A. Zamayla)
https://alviano.net/ (M. Alviano)
0000-0002-2052-2063 (M. Alviano); 0000-0002-6822-0574 (A. Zamayla)
© 2021 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).
CEUR Workshop Proceedings (CEUR-WS.org)

CEUR-WS.org/Vol-2970/aspocpinvited2.pdf

ground constraint. (Details on the construction are given in the invited speech.) This fact has implications on any extension of generative Datalog with constructs that allow for expressing constraints, among them default negation under stable model semantics [5]: if real addition is supported by the language, there exists no probability space for models of the general case.

Many other probabilistic logic languages exists in the literature. Some of them attach probabilities to database facts [6, 7, 8, 9, 10], or to rules [11, 12, 13, 14] and other provides constructs similar to Δ-terms [15, 16, 17, 18, 19, 20]. A comparison between these languages is out of the scope of this speech, and the reader is referred to [4] for details.

2. Probability Spaces and Parametrized Probability Distribution

A *probability space* is a triple $(\Omega, \mathcal{F}, P)$ satisfying the following conditions:

- Ω is the *sample space*, a nonempty set comprising all possible outcomes (of the modeled probabilistic process).
- $\mathcal{F} \subseteq 2^\Omega$ is the *event space*, a collection of all the *events* to consider, where an event is a set of possible outcomes. $\mathcal{F}$ must be a σ-algebra, ie.
  - $\mathcal{F}$ contains the sample space: $\Omega \in \mathcal{F}$;
  - $\mathcal{F}$ is closed under complement: if $E \in \mathcal{F}$, then $(\Omega \setminus E) \in \mathcal{F}$;
  - $\mathcal{F}$ is closed under countable unions: if $E_i \in \mathcal{F}$ for $i \in \mathbb{N}$, then $\left(\bigcup_{i\in\mathbb{N}} A_i\right) \in \mathcal{F}$.
- $P : \mathcal{F} \to [0, 1]$ is the *probability measure*, a function on events such that
  - P is countably additive: if $E_i \in \mathcal{F}$ (for all $i \in \mathbb{N}$) are pairwise disjoint sets, then $P(\bigcup_{i\in\mathbb{N}} E_i) = \sum_{i\in\mathbb{N}} P(E_i)$.
  - the measure of the sample space is equal to one: $P(\Omega) = 1$.

Example 1. Throwing a (6-faces) die is a classical example of probabilistic process. In this case, the sample space Ω is $\{1, 2, 3, 4, 5, 6\}$ (or $[1..6]$ for a more compact notation), ie. the possible outcome of the probabilistic process is one of the six faces of the die. The event space $\mathcal{F}$ can be the powerset of Ω, hence comprising elementary events such as $\{1\}$ (the die lands on 1) and complex events such as $\{1, 3, 5\}$ (the die lands on an odd number) — note that the event space can also be smaller, if there is no need to consider all the events. As for the probability measure P, assuming that the die is unbiased, it just divides the cardinality of the event by 6, ie. $P : E \mapsto \frac{|E|}{6}$. So, $P(\{1\}) = \frac{1}{6}$ and $P(\{1, 3, 5\}) = \frac{3}{6} = \frac{1}{2}$. ■

When Ω is countable, and all elementary events (ie. singletons) belongs to $\mathcal{F}$, the simpler notion of *discrete probability distribution* can be employed, ie. a function $P : \Omega \to [0, 1]$ such that $\sum_{o\in\Omega} P(o) = 1$. Let $\mathcal{P}_\Omega$ denote the set of all discrete probability distributions over Ω. A *parametrized probability distribution* over Ω is a function $\delta : \mathbb{R}^k \to \mathcal{P}_\Omega$, that is, $\delta(\mathbf{p})$ is a discrete probability distribution over Ω for every parameter instantiation $\mathbf{p} \in \mathbb{R}^k$.

Example 2 (Continuing Example 1)**.** The constant function $P = \{i \mapsto \frac{1}{6} \mid i \in [1..6]\}$ is a discrete probability distribution characterizing the throw of an unbiased die. Biased dice can be represented by the parametrized probability distribution $\mathsf{Die} : \mathbb{R}^6 \to \mathcal{P}_{[0..6]}$ satisfying the following conditions:

- if $p_i \in [0,1]$ for all $i \in [1..6]$ and $\sum_{i=1}^{6} p_i = 1$, then $\mathsf{Die}(p_1,\ldots,p_6)(0) = 0$ and $\mathsf{Die}(p_1,\ldots,p_6)(i) = p_i$ for all $j \in [1..6]$;
- otherwise, $\mathsf{Die}(p_1,\ldots,p_6)(0) = 1$ and $\mathsf{Die}(p_1,\ldots,p_6)(i) = 0$ for all $j \in [1..6]$.

Note that outcome 0 is associated with incorrect instantiations of the parameters. ■

3. Syntax and Semantics of Generative Datalog

A Δ-*term* is an expression of the form $\delta(\mathbf{P};\mathbf{S})$, where $\delta : \mathbb{R}^k \to \mathcal{P}_\Omega$ is a parametrized probability distribution over some Ω, each element in $\mathbf{P}$ and $\mathbf{S}$ is a constant or a variable, and $|\mathbf{P}| = k$. Elements in $\mathbf{S}$ contribute to a *signature* to distinguish different runs of the probabilistic process associated with $\delta(\mathbf{p})$, for any grounding $\mathbf{p}$ of $\mathbf{P}$. Intuitively, each of those runs returns a possible outcome from Ω, according to the probability distribution $\delta(\mathbf{p})$.

Example 3 (Continuing Example 2)**.** Each throw of a (possibly biased) die may result in a different outcome, which can be represented by the Δ-term $\mathsf{Die}(p_1,\ldots,p_6;id)$, where id is an identifier for a specific throw of the die — eg. $\mathsf{Die}(\frac{1}{6},\ldots,\frac{1}{6};1)$ and $\mathsf{Die}(\frac{1}{6},\ldots,\frac{1}{6};2)$ to represent two throws of an unbiased die. Note that two Δ-terms with different values for the parameters $p_1,\ldots,p_6$ are necessarily associated with different throws, as they must either refer to different dice or to a die that is altered between the two throws — eg. $\mathsf{Die}(\frac{1}{6},\ldots,\frac{1}{6};1)$ and $\mathsf{Die}(\frac{1}{6}+\epsilon,\frac{1}{6}-\epsilon,\frac{1}{6},\frac{1}{6},\frac{1}{6},\frac{1}{6};1)$, for some $\epsilon \in]0,\frac{1}{6}]$, must refer to different throws. ■

Generative Datalog programs are Datalog programs whose head atoms possibly contains Δ-terms — to simplify the presentation, at most one Δ-term per rule. Their semantics is defined via an existential Datalog program obtained by replacing every rule $H(\mathbf{X}) \leftarrow B(\mathbf{X})$ containing a Δ-term $\delta(\mathbf{P};\mathbf{S})$ with the following rules:

$$\exists Y\ \mathit{Result}^\delta(\mathbf{P},\mathbf{S},Y) \leftarrow B(\mathbf{X}) \tag{1}$$

$$H(\mathbf{X}') \leftarrow B(\mathbf{X}), \mathit{Result}^\delta(\mathbf{P},\mathbf{S},Y) \tag{2}$$

where Result^δ is a fresh predicate name, Y a fresh variable, and $\mathbf{X}'$ is obtained from $\mathbf{X}$ by replacing $\delta(\mathbf{P};\mathbf{S})$ with Y. Intuitively, rule (1) and predicate Result^δ materialize the function describing the actual outcomes of the probabilistic processes associated with δ and involved in the computation of inferred facts. Such outcomes are then propagated in the correct place by rule (2). Let $\Pi_\exists$ denote the existential Datalog program associated with a generative Datalog program Π.

Example 4 (Continuing Example 3)**.** The following generative Datalog program Π encodes the repeated throw of an unbiased die until it lands on an even number:

$$\mathit{Odd}(1). \quad \mathit{Odd}(3). \quad \mathit{Odd}(5). \quad \mathit{Iteration}(1). \tag{3}$$

$$\mathit{Seen}\left(I, \mathsf{Die}\left(\frac{\mathbf{1}}{\mathbf{6}}; I\right)\right) \leftarrow \mathit{Iteration}(I) \tag{4}$$

$$\mathit{Iteration}(I+1) \leftarrow \mathit{Seen}(I,X), \mathit{Odd}(X). \tag{5}$$

Program $\Pi_\exists$ is obtained from Π by replacing rule (4) with the following rules:

$$\exists Y\ \mathit{Result}^{\mathsf{Die}}\left(\frac{\mathbf{1}}{\mathbf{6}}, I, Y\right) \leftarrow \mathit{Iteration}(I)$$

$$\mathit{Seen}(I, Y) \leftarrow \mathit{Iteration}(I), \mathit{Result}^{\mathsf{Die}}\left(\frac{\mathbf{1}}{\mathbf{6}}, I, Y\right).$$

Note that Π and $\Pi_\exists$ use integer addition (rule 5), which could be also expressed by Δ-terms (see Example 8). ■

The probability space associated with a generative Datalog program Π is the triple $(\Omega_\Pi, \mathcal{F}_\Pi, P_\Pi)$, defined next. A possible outcome of a generative Datalog program Π is a minimal model I of $\Pi_\exists$ such that $\delta(\mathbf{p})(y) > 0$ for every $\mathit{Result}^\delta(\mathbf{p}, \mathbf{s}, y) \in I$. Let Ω_Π be the sample space associated with Π, ie. the set of possible outcomes of Π.

Example 5 (Continuing Example 4)**.** Among the minimal models of $\Pi_\exists$ there are those extending $I_1 = \{\mathit{Odd}(1), \mathit{Odd}(3), \mathit{Odd}(5), \mathit{Iteration}(1)\}$ as follows:

- $I_2 = I_1 \cup \{\mathit{Result}^{\mathsf{Die}}(\frac{\mathbf{1}}{\mathbf{6}}, 1, 2), \mathit{Seen}(1, 2)\}$;
- $I_3 = I_1 \cup \{\mathit{Result}^{\mathsf{Die}}(\frac{\mathbf{1}}{\mathbf{6}}, 1, 3), \mathit{Seen}(1, 3), \mathit{Iteration}(2), \mathit{Result}^{\mathsf{Die}}(\frac{\mathbf{1}}{\mathbf{6}}, 2, 6), \mathit{Seen}(2, 6)\}$;
- $I_4 = I_1 \cup \{\mathit{Result}^{\mathsf{Die}}(\frac{\mathbf{1}}{\mathbf{6}}, i, 3), \mathit{Seen}(i, 3), \mathit{Iteration}(i+1) \mid i \geq 1\}$.

Note that model I_4 comprises infinitely many facts. ■

A *derivation* wrt. a generative Datalog program Π is a finite sequence $I = [f_1, \ldots, f_n]$ of facts such that f_i is derived by some unsatisfied rule of $\Pi_\exists \cup \{f_1, \ldots, f_{i-1}\}$, for all $i \in [1..n]$. Let $\mathcal{I}_\Pi$ denote the set of derivations wrt. Π, and let $\Omega_\Pi^{\supseteq I}$ denote the set of possible outcomes of Π extending I, ie. $\{J \in \Omega_\Pi \mid J \supseteq I\}$. The event space associated with Π, denoted $\mathcal{F}_\Pi$, is the σ-algebra generated from $\{\Omega_\Pi^{\supseteq I} \mid I \in \mathcal{I}_\Pi\}$ (ie. its closure under complement and countable unions).

Example 6 (Continuing Example 5)**.** Recall minimal models I_2, I_3, I_4 from Example 5. Let I_1' be the derivation $[\mathit{Odd}(1), \mathit{Odd}(3), \mathit{Odd}(5), \mathit{Iteration}(1)]$. Hence, $\Omega_\Pi^{\supseteq I_1'}$ is the entire sample space Ω_Π. For $I_2' = I_1' \cup [\mathit{Result}^{\mathsf{Die}}(\frac{\mathbf{1}}{\mathbf{6}}, 1, 2), \mathit{Seen}(1, 2)]$, we have that $\Omega_\Pi^{\supseteq I_2'} = \{I_2\}$. On the other hand, for $I_3' = I_1' \cup [\mathit{Result}^{\mathsf{Die}}(\frac{\mathbf{1}}{\mathbf{6}}, 1, 3), \mathit{Seen}(1, 3)]$, we have that $\Omega_\Pi^{\supseteq I_3'}$ contains infinitely many possible outcomes, among them I_3 and I_4. Note that all finite elementary events belongs to the event space $\mathcal{F}_\Pi$ (eg. $\{I_2\}$), while infinite possible outcomes only occur in complex events (eg. $\{I_4\} \notin \mathcal{F}_\Pi$). ■

The probability measure associated with a generative Datalog program Π, denoted P_Π, is such that the probability of the set of possible outcomes of Π extending a derivation I is the product of the probabilities of the Δ-terms represented in I, ie.

$$P_\Pi(\Omega_\Pi^{\supseteq I}) = \prod_{\mathit{Result}^\delta(\mathbf{p}, \mathbf{s}, y) \in I} \delta(\mathbf{p})(y).$$

Example 7 (Continuing Example 6)**.** $P_\Pi(\Omega_\Pi^{\supseteq I_1'}) = 1$ because I_1' contains no $\mathit{Result}^{\mathsf{Die}}$ instance, while $P_\Pi(\Omega_\Pi^{\supseteq I_2'}) = P_\Pi(\Omega_\Pi^{\supseteq I_3'}) = \frac{1}{6}$ because $\mathit{Result}^{\mathsf{Die}}(\frac{\mathbf{1}}{\mathbf{6}}, 1, 2) \in I_2'$ and $\mathit{Result}^{\mathsf{Die}}(\frac{\mathbf{1}}{\mathbf{6}}, 1, 3) \in I_3'$ are the only $\mathit{Result}^{\mathsf{Die}}$ instances in I_2' and I_3', and $\mathsf{Die}(\frac{\mathbf{1}}{\mathbf{6}})(2) = \mathsf{Die}(\frac{\mathbf{1}}{\mathbf{6}})(3) = \frac{1}{6}$. ■

4. Arithmetic and Non-measurable Sets

Example 4 concludes by suggesting that addition over integers can be expressed by Δ-terms without the need for an ad-hoc construct. The next example better clarify the idea.

Example 8. Integer addition can be represented by the Δ-term $\mathsf{Add} : \mathbb{R}^2 \to \mathcal{P}_{\mathbb{Z}\cup\{\bot\}}$ such that:

- if $i, j \in \mathbb{Z}$, then $\mathsf{Add}(i,j)(i+j) = 1$ and $\mathsf{Add}(i,j)(k) = 0$ for all $k \neq i+j$;
- otherwise, $\mathsf{Add}(i,j)(\bot) = 1$ and $\mathsf{Add}(i,j)(k) = 0$ for all $k \neq \bot$.

Accordingly, rule (5) in Example 4 can be rewritten as

$$\mathit{Iteration}(\mathsf{Add}(I,1)) \leftarrow \mathit{Seen}(I,X), \mathit{Odd}(X).$$

Similarly, Δ-terms can encode addition over rational numbers or any other countable set. ■

Essentially, $\mathsf{Add}(i,j)$ in the example above is a *Kronecker delta* (the discrete anolog of the Dirac delta function) assigning 1 to $i+j$, and 0 to all other numbers, for all integers i, j. The same idea can be adapted to represent any function over any countable set (eg. integer multiplication and addition over rational numbers), but also any relation over uncountable sets (eg. greater than or less than). On the other hand, real addition cannot be encoded in this way because Δ-terms by definition are associated with parametrized probability distributions over a countable sample space. (Such a restriction is overcome in [21], where parametrized probability distribution over uncountable sample spaces are supported.) Support for real addition in the language has implications on the existence of probability spaces for filtered models of generative Datalog programs, as hinted in the reminder of this section.

A *non-measurable set* is a set which cannot be assigned a meaningful "volume". In Zermelo-Fraenkel set theory, the axiom of choice entails that non-measurable subsets of $\mathbb{R}$ exist. Classical examples are *Vitali sets*, defined next.

Definition 1. A Vitali set V is a subset of $[0,1]$ such that, for each real number r, there is exactly one number $v \in V$ such that $v - r$ is a rational number.

We now hint on how to associate a Vitali set V with filtered-out possible outcomes of a generative Datalog program with real addition. Each possible outcome of such a program represents a different real number x in the unit interval $[0,1]$, together with all rational numbers being a truncation of the binary representation of x — eg. if $x = \frac{\sqrt{2}}{2}$ ($\simeq 0.101101_2$), its truncations are 0.1_2, 0.101_2, 0.1011_2, 0.101101_2, and so on. Intuitively, x and its truncations are produced by repeatedly appending to "0." a binary digit chosen at random; digits are appended by using real addition, and generated by a Δ-term selecting 0 or 2^{-n} with probability 0.5, for any integer parameter n.

It remains to filter-out possible outcomes containing members of the Vitali set V. For this purpose, the membership relation for V can be represented by a Δ-term employing a Kronecker delta. So, we can detect if the real number x in a possible outcome belongs to V, but we should avoid to recognize as a member of V any of its truncations. This is achieved by taking a Vitali set V containing the rational number 1, and therefore such that $V \setminus \{1\}$ is a set of irrational numbers (by Definition 1).

Summing up, the probability measure of the filtered-out possible outcomes would be the Lebesgue measure of the Vitali set V, which is known to be a non-measurable set. Details on the construction are given in the invited speech.

Acknowledgments

This speech is about some ongoing research with Matthias Lanzinger, Michael Morak, and Andreas Pieris. This work was partially supported by the projects PON-MISE MAP4ID (CUP B21B19000650008) and PON-MISE S2BDW (CUP B28I17000250008), by the LAIA lab (part of the SILA labs) and by GNCS-INdAM.

References

[1] N. D. Goodman, The principles and practice of probabilistic programming, in: R. Giacobazzi, R. Cousot (Eds.), The 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '13, Rome, Italy - January 23 - 25, 2013, ACM, 2013, pp. 399–402. URL: https://doi.org/10.1145/2429069.2429117. doi:10.1145/2429069.2429117.

[2] C. Jones, G. D. Plotkin, A probabilistic powerdomain of evaluations, in: Proceedings of the Fourth Annual Symposium on Logic in Computer Science (LICS '89), Pacific Grove, California, USA, June 5-8, 1989, IEEE Computer Society, 1989, pp. 186–195. URL: https://doi.org/10.1109/LICS.1989.39173. doi:10.1109/LICS.1989.39173.

[3] V. Bárány, B. ten Cate, B. Kimelfeld, D. Olteanu, Z. Vagena, Declarative probabilistic programming with datalog, in: W. Martens, T. Zeume (Eds.), 19th International Conference on Database Theory, ICDT 2016, Bordeaux, France, March 15-18, 2016, volume 48 of *LIPIcs*, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016, pp. 7:1–7:19. URL: https://doi.org/10.4230/LIPIcs.ICDT.2016.7. doi:10.4230/LIPIcs.ICDT.2016.7.

[4] V. Bárány, B. ten Cate, B. Kimelfeld, D. Olteanu, Z. Vagena, Declarative probabilistic programming with datalog, ACM Trans. Database Syst. 42 (2017) 22:1–22:35. URL: https://doi.org/10.1145/3132700. doi:10.1145/3132700.

[5] M. Gelfond, V. Lifschitz, Classical negation in logic programs and disjunctive databases, New Gener. Comput. 9 (1991) 365–386. URL: https://doi.org/10.1007/BF03037169. doi:10.1007/BF03037169.

[6] T. Sato, Y. Kameya, PRISM: A language for symbolic-statistical modeling, in: Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence, IJCAI 97, Nagoya, Japan, August 23-29, 1997, 2 Volumes, Morgan Kaufmann, 1997, pp. 1330–1339. URL: http://ijcai.org/Proceedings/97-2/Papers/078.pdf.

[7] D. Suciu, D. Olteanu, C. Ré, C. Koch, Probabilistic Databases, Synthesis Lectures on Data Management, Morgan & Claypool Publishers, 2011. URL: https://doi.org/10.2200/S00362ED1V01Y201105DTM016. doi:10.2200/S00362ED1V01Y201105DTM016.

[8] B. Kimelfeld, P. Senellart, Probabilistic XML: models and complexity, in: Z. Ma, L. Yan (Eds.), Advances in Probabilistic Databases for Uncertain Information Management, volume 304 of *Studies in Fuzziness and Soft Computing*, Springer, 2013, pp. 39–66. URL: https://doi.org/10.1007/978-3-642-37509-5_3. doi:10.1007/978-3-642-37509-5_3.

[9] T. Vieira, M. Francis-Landau, N. W. Filardo, F. Khorasani, J. Eisner, Dyna: toward a self-optimizing declarative language for machine learning applications, in: T. Shpeisman, J. Gottschlich (Eds.), Proceedings of the 1st ACM SIGPLAN International Workshop on Machine Learning and Programming Languages, MAPL@PLDI 2017, Barcelona, Spain, June 18, 2017, ACM, 2017, pp. 8–17. URL: https://doi.org/10.1145/3088525.3088562. doi:10.1145/3088525.3088562.

[10] F. G. Cozman, D. D. Mauá, The joy of probabilistic answer set programming: Semantics, complexity, expressivity, inference, Int. J. Approx. Reason. 125 (2020) 218–239. URL: https://doi.org/10.1016/j.ijar.2020.07.004. doi:10.1016/j.ijar.2020.07.004.

[11] P. M. Domingos, D. Lowd, Markov Logic: An Interface Layer for Artificial Intelligence, Synthesis Lectures on Artificial Intelligence and Machine Learning, Morgan & Claypool Publishers, 2009. URL: https://doi.org/10.2200/S00206ED1V01Y200907AIM007. doi:10.2200/S00206ED1V01Y200907AIM007.

[12] F. Niu, C. Ré, A. Doan, J. W. Shavlik, Tuffy: Scaling up statistical inference in markov logic networks using an RDBMS, Proc. VLDB Endow. 4 (2011) 373–384. URL: http://www.vldb.org/pvldb/vol4/p373-niu.pdf. doi:10.14778/1978665.1978669.

[13] F. Niu, C. Zhang, C. Ré, J. W. Shavlik, Deepdive: Web-scale knowledge-base construction using statistical learning and inference, in: M. Brambilla, S. Ceri, T. Furche, G. Gottlob (Eds.), Proceedings of the Second International Workshop on Searching and Integrating New Web Data Sources, Istanbul, Turkey, August 31, 2012, volume 884 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2012, pp. 25–28. URL: http://ceur-ws.org/Vol-884/VLDS2012_p25_Niu.pdf.

[14] J. Lee, S. Talsania, Y. Wang, Computing LPMLN using ASP and MLN solvers, Theory Pract. Log. Program. 17 (2017) 942–960. URL: https://doi.org/10.1017/S1471068417000400. doi:10.1017/S1471068417000400.

[15] V. S. Costa, D. Page, J. Cussens, Clp(*BN*): Constraint logic programming for probabilistic knowledge, in: L. D. Raedt, P. Frasconi, K. Kersting, S. Muggleton (Eds.), Probabilistic Inductive Logic Programming - Theory and Applications, volume 4911 of *Lecture Notes in Computer Science*, Springer, 2008, pp. 156–188. URL: https://doi.org/10.1007/978-3-540-78652-8_6. doi:10.1007/978-3-540-78652-8_6.

[16] D. Poole, The independent choice logic and beyond, in: L. D. Raedt, P. Frasconi, K. Kersting, S. Muggleton (Eds.), Probabilistic Inductive Logic Programming - Theory and Applications, volume 4911 of *Lecture Notes in Computer Science*, Springer, 2008, pp. 222–243. URL: https://doi.org/10.1007/978-3-540-78652-8_8. doi:10.1007/978-3-540-78652-8_8.

[17] C. Baral, M. Gelfond, J. N. Rushton, Probabilistic reasoning with answer sets, Theory Pract. Log. Program. 9 (2009) 57–144. URL: https://doi.org/10.1017/S1471068408003645. doi:10.1017/S1471068408003645.

[18] J. Vennekens, M. Denecker, M. Bruynooghe, Cp-logic: A language of causal probabilistic events and its relation to logic programming, Theory Pract. Log. Program. 9 (2009) 245–308. URL: https://doi.org/10.1017/S1471068409003767. doi:10.1017/S1471068409003767.

[19] B. Gutmann, I. Thon, A. Kimmig, M. Bruynooghe, L. D. Raedt, The magic of logical inference in probabilistic programming, Theory Pract. Log. Program. 11 (2011) 663–680. URL: https://doi.org/10.1017/S1471068411000238. doi:10.1017/S1471068411000238.

[20] D. Nitti, T. D. Laet, L. D. Raedt, Probabilistic logic programming for hybrid relational domains, Mach. Learn. 103 (2016) 407–449. URL: https://doi.org/10.1007/s10994-016-5558-8.

doi:10.1007/s10994-016-5558-8.

[21] M. Grohe, B. L. Kaminski, J. Katoen, P. Lindner, Generative datalog with continuous distributions, in: D. Suciu, Y. Tao, Z. Wei (Eds.), Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2020, Portland, OR, USA, June 14-19, 2020, ACM, 2020, pp. 347–360. URL: https://doi.org/10.1145/3375395.3387659. doi:10.1145/3375395.3387659.