Artificial Intelligence 228 (2015) 1–44

Contents lists available at ScienceDirect

Artificial Intelligence

www.elsevier.com/locate/artint

A new probabilistic constraint logic programming language based on a generalised distribution semantics [☆]

Steffen Michels [a,*], Arjen Hommersom [a], Peter J.F. Lucas [a], Marina Velikova [b]

[a] Radboud University, Institute for Computing and Information Sciences, Nijmegen, The Netherlands
[b] Embedded Systems Innovation by TNO, The Netherlands

ARTICLE INFO

Article history:
Received 7 April 2014
Received in revised form 2 May 2015
Accepted 26 June 2015
Available online 4 July 2015

Keywords:
Probabilistic logic programming
Imprecise probabilities
Continuous probability distributions
Exact probabilistic inference

ABSTRACT

Probabilistic logics combine the expressive power of logic with the ability to reason with uncertainty. Several probabilistic logic languages have been proposed in the past, each of them with their own features. We focus on a class of probabilistic logic based on *Sato's distribution semantics*, which extends logic programming with probability distributions on binary random variables and guarantees a unique probability distribution. For many applications binary random variables are, however, not sufficient and one requires random variables with arbitrary ranges, e.g. real numbers. We tackle this problem by developing a *generalised distribution semantics* for a new *probabilistic constraint logic programming* language. In order to perform exact inference, imprecise probabilities are taken as a starting point, i.e. we deal with sets of probability distributions rather than a single one. It is shown that given any continuous distribution, conditional probabilities of events can be approximated arbitrarily close to the true probability. Furthermore, for this setting an inference algorithm that is a generalisation of weighted model counting is developed, making use of SMT solvers. We show that inference has similar complexity properties as precise probabilistic inference, unlike most imprecise methods for which inference is more complex. We also experimentally confirm that our algorithm is able to exploit local structure, such as determinism, which further reduces the computational complexity.

1. Introduction

Probabilistic logics are gaining popularity as modelling and reasoning tools, since they combine the power of logic to represent knowledge with the ability of probability theory to deal with uncertainty. In addition, in the field of *statistical relational learning* (SRL) [1], powerful machine learning methods have been developed using probabilistic logical languages as their basis.

The need for those methods emerges from the fact that in many areas more and more data become available, which does not only imply uncertainty, but often provides rich structure in terms of relations between entities. Probabilistic logics and SRL methods have been applied to a wide range of problem domains. Examples include: link and node prediction in

☆ This publication was supported by the Dutch national program COMMIT. The research work was carried out as part of the Metis project under the responsibility of Embedded Systems Innovation by TNO, with Thales Nederland B.V. as the carrying industrial partner.

* Corresponding author.

E-mail addresses: s.michels@science.ru.nl (S. Michels), arjenh@cs.ru.nl (A. Hommersom), peterl@cs.ru.nl (P.J.F. Lucas), marina.velikova@tno.nl (M. Velikova).

http://dx.doi.org/10.1016/j.artint.2015.06.008
0004-3702/© 2015 Elsevier B.V. All rights reserved.

metabolic networks [2], discovering interactions between genes [3], dealing with a potentially unknown number of relations between multiple objects by a robot [3] and fusing information about vessels on the North Sea by modelling relations between those objects and heterogeneous pieces of information [4].

Combining logic and probability theory is challenging and involves dealing with a trade-off between expressivity and efficiency of inference. The research described in this article focuses on probabilistic logics adhering to *Sato's distribution semantics* [5]. This semantics guarantees a single unambiguously defined probability distribution and supports efficient inference. In this semantic framework, logic programming (LP) is used to define a probability distribution over a set of binary facts. Examples of languages based on that semantics are ProbLog [6], PRISM [7], ICL [8], and CP-logic [9]. The choice for this kind of semantics is motivated by the fact that it allows one to use probabilities with local meaning, which provides the modularity needed for knowledge representation, similar to the widely used *Bayesian networks* [10]. Some alternatives, in particular the popular *Markov Logic Networks*, make use of weights that only can be interpreted in the context of the entire theory [11].

For many real-world knowledge-representation problems binary random variables are not convenient or not sufficient; very often, random variables taking values within arbitrary ranges are needed, e.g. integers and real numbers. In fact, virtually all deterministic real-world models include such variables. Examples are: data models involving the age or height of persons, temporal models that use integer or real-valued time, and physical models expressing most quantities as real numbers. Since in all such domains uncertainty is present, the provision of a probabilistic representation of such models is essential. Furthermore, most domains are typically relational. An example involving relations and uncertainty, as well as real-valued variables, is inferring a map of the indoor environment based on observations by robots [12].

While finite-ranged discrete random variables can be represented by sets of binary random variables, random variables with an infinite number of values, such as continuous variables, have significant impact on the semantics of probabilistic logics and the complexity of the inference problem. Some previous attempts to extend probabilistic logic programming with real-valued variables heavily restrict the use of such variables [13], such that they are not powerful enough to model many real world problems. Other approaches resort to approximate methods for inference, such as sampling [14,12]. For many problems current sampling techniques are effective, but such approaches may fail and guarantees about the result's quality are weak. This is especially problematic for problems in which wrong decisions have a huge impact.

In this article, we propose an alternative to using exact inference and sampling with some inherent advantages; we can represent distributions for which exact inference is infeasible and at the same time provide more guarantees than offered by existing approximation methods. We provide a powerful, general theoretic foundation for probabilistic logic programs, which we refer to as a *generalised distribution semantics*, and a practically useful language based on the semantic framework. The theory supports approximating probability distributions with arbitrarily-ranged random variables, both continuous and discrete, and is equipped with an efficient method for inference.

The main contribution of our work is an expressive logical language that defines events in terms of constraints on the random variable's values, e.g. inequalities on real-valued random variables. The logical part of the language is inspired by *constraint logic programming* (CLP) [15]. To allow exact computation of probabilities, we make probabilities imprecise by introducing *credal sets* on top of the generalised distribution semantics. Given these credal sets, it is ensured that the bounds on marginals can be computed exactly. Moreover, by virtue of the distribution semantics, this new framework also ensures that there is always at least one consistent distribution, in contrast to some other probabilistic logics with imprecisions, e.g. the probabilistic logic of Nilsson [16].

Based on this semantics we introduce a new *probabilistic constraint logic programming* (PCLP) language, in which independent probability distributions of random variables are defined by means of a set of probability-constraint pairs. Similar to deterministic *constraint logic programming* (CLP) [15], PCLP is a family of languages allowing one to use arbitrary constraint theories, unlike most formalisms which are restricted to certain kinds of random variables, e.g. finite discrete and real-valued ones. This is possible thanks to the general semantic foundation. As examples we discuss discrete constants (PCLP(D)) and real numbers (PCLP(R)). To our knowledge this is the first usable framework that combines imprecise probabilities with continuous variables. An earlier version of the language was already presented at a previous occasion [17]. In the present paper we place the language in the context of the general semantic foundation and provide proofs of more general properties.

We finally present an inference algorithm that is a generalisation of recently emerged probabilistic inference methods based on translating inference problems to *weighted model counting* (WMC) problems. This has been shown to be an efficient inference method for propositional formalisms such as Bayesian networks [18]. In addition, it has been shown to be applicable to probabilistic logics based on distribution semantics [19]. We show that exact inference in our new language can be dealt with by a generalisation of this method.

This paper also shows that inference in our framework has similar complexity characteristics as precise probabilistic inference. In fact, we show that it is in the same complexity class as precise inference, unlike most other imprecise approaches. Furthermore, the complexity can be bounded in terms of the problem structure, similar to WMC. Our generalised framework can also exploit additional structure, as determinism, being the main advantage of WMC compared to classical inference algorithms [18]. We experimentally confirm that determinism can be exploited in our generalised setting. To our knowledge using ideas from WMC offers a novel method to approximating continuous distributions. A basic version of the algorithm was also described in earlier work [17]; here we provide an improved, more general description and additional experimental results.

This article is organised as follows. Motivating examples of how the novel PCLP language can be used are given in Section 2. We then provide the necessary background (Section 3) for our theoretical results, which is the generalised distribution semantics (Section 4) and a way to deal with credal sets in the context of this semantics (Section 5). The PCLP language is based on this theory and described in Section 6; inference for the language is explored in Section 7. We finally discuss related work in Section 8 and conclude the article in Section 9.

2. Motivating examples

To illustrate the expressive power of the new language PCLP, a few typical examples are presented.

2.1. Fruit selling

We present an example concerning the likelihood that consumers will buy a certain kind of fruit, based on [20]. Since we have a first-order formalism, this generalises easily to an arbitrary number of kinds (in the example: apples and bananas). The main goal is to show how PCLP can deal with continuous distributions.

Yield of fruit is clearly relevant for its price. We model the yield of fruit with normally distributed random variables (denoted by a string starting with an upper case letter and in bold):

$$\begin{aligned}\mathbf{Yield}(apple) &\sim \mathcal{N}(12\,000.0, 1000.0)\\ \mathbf{Yield}(banana) &\sim \mathcal{N}(10\,000.0, 1500.0)\end{aligned}$$

The price is also influenced by government support, which is modelled by discrete random variables:

$$\begin{aligned}\mathbf{Support}(apple) &\sim \{0.3{:}\,yes, 0.7{:}\,no\}\\ \mathbf{Support}(banana) &\sim \{0.5{:}\,yes, 0.5{:}\,no\}\end{aligned}$$

The basic price linearly depends on the yield, which is expressed as a deterministic logic fact:

$$\begin{aligned}basic\_price(apple, 250 &- 0.007 \cdot \mathbf{Yield}(apple))\\ basic\_price(banana, 200 &- 0.006 \cdot \mathbf{Yield}(banana))\end{aligned}$$

In case the price is supported it is raised by a fixed amount:

$$\begin{aligned}&price(Fruit, BPrice + 50) \leftarrow basic\_price(Fruit, BPrice),\ \langle \mathbf{Support}(Fruit) = yes\rangle\\ &price(Fruit, BPrice) \leftarrow basic\_price(Fruit, BPrice),\ \langle \mathbf{Support}(Fruit) = no\rangle\end{aligned}$$

Fruit is a logical variable (not denoted in bold) which can take kinds of fruit, e.g. *apple* and *banana*, as possible instantiations. Here we make use of the special predicate $\langle\rangle$, which represents probabilistic events; for example, $\langle \mathbf{Support}(Fruit) = yes\rangle$ is true in case the random variable $\mathbf{Support}(Fruit)$ takes the value *yes*. In constraint logic programming (CLP) [15] a similar predicate is used to represent constraints.

At which maximum price customers still buy a certain fruit is modelled by a gamma distribution:

$$\begin{aligned}\mathbf{Max\_price}(apple) &\sim \Gamma(10.0, 18.0)\\ \mathbf{Max\_price}(banana) &\sim \Gamma(12.0, 10.0)\end{aligned}$$

Thus, a customer is willing to buy in case the price is equal to or less than the maximum price, which can be expressed by the following first-order rule:

$$buy(Fruit) \leftarrow price(Fruit, P),\ \langle P \le \mathbf{Max\_price}(Fruit)\rangle$$

The interesting question to ask given this knowledge base is whether customers buy a certain fruit. As it is uncertain which of the events, specified by the occurrences of $\langle\rangle$, actually occur, the only answer we can give is how likely such statements are, e.g. $P(buy(apple))$ or $P(buy(apple) \vee buy(banana))$. Another possible question is what the probability is that customers buy apples given that we know what the least maximum yield will be, e.g. $P(buy(apple) \mid \langle \mathbf{Yield}(apple) \ge 10\,000.0\rangle)$.

Note that such probabilities cannot be computed straightforwardly, as they require computing probabilities of linear inequalities between different kinds of continuous distributions, which is in general not computable. Our framework however allows one to determine an approximation with known errors, for example:

$$\begin{aligned}P(buy(apple)) &\approx 0.464 \pm 0.031\\ P(buy(banana)) &\approx 0.162 \pm 0.031\\ P(buy(apple) \vee buy(banana)) &\approx 0.552 \pm 0.054\end{aligned}$$

The actual probabilities are guaranteed to be within the computed maximum error determined by the used approximation scheme and can be made arbitrarily small as explained in Section 5.6.

2.2. Diabetes mellitus

The next, medical, example shows how PCLP can be used to model problems involving continuous distributions as well as imprecise probabilities. The latter means one uses bounds rather than precise probability estimates, which is a way to handle situations concerned with insufficient data to reliably estimate probabilities. Such situations frequently occur in clinical research. A possible approach in such cases is to express uncertainty about probabilities by yet another probability distribution, i.e. using second-order probability distributions. In contrast, the approach of imprecise probabilities assumes that all possible probabilities within the specified range are possible, but furthermore expresses complete ignorance of what the actual probability is. So imprecise probabilities relieve from specifying a second-order distribution, which requires knowledge or an amount of data not always available, at the price of making a hard choice of which probabilities are possible and which are not.

The example concerns diabetes mellitus type 2, which is a complex disorder in which several metabolic control mechanisms are disturbed. A first step in its treatment is to regulate glucose metabolism, as in diabetic patients glucose, although present in abundance in the extracellular fluid with the exception of the cerebrospinal fluid, it is unable to cross the cellular membrane and cells therefore lack their usual energy resource (often called "starvation amidst abundance"). A standard test to check the quality of glucose control is the measurement of fasting blood glucose levels. Furthermore, the levels of glycated hemoglobin (HbA1c) offer insight into the effectiveness of long-term (8 to 12 weeks) glucose control. Clearly, the fasting blood glucose and HbA1c measurements are related, although only on average.

While type 2 diabetes is mostly related to lifestyle-related factors, recent biomedical research indicates that various genetic factors play a role in its onset. In [21], familial risk of type 2 diabetes was classified as average, moderate, or high. In the US population, 69.8% were in the average, 22.7% in the moderate, and 7.5% in the high familial risk group. In PCLP this can be represented as follows:

$$\mathbf{Predisposition} \sim \{0.698{:}\, \mathit{average}, 0.227{:}\, \mathit{moderate}, 0.075{:}\, \mathit{high}\}$$

According to [21], the crude prevalences of diabetes within each risk category was between 5.4% and 6.6% in the average risk group, between 13.1% and 16.7% in the moderate risk group, and between 26.6% and 33.6% in the high risk group.

$$\begin{aligned}\mathbf{DM\_if\_AverageRisk} &\sim \{0.054{:}\, \mathit{yes}, 0.934{:}\, \mathit{no}\}\\ \mathbf{DM\_if\_ModerateRisk} &\sim \{0.131{:}\, \mathit{yes}, 0.833{:}\, \mathit{no}\}\\ \mathbf{DM\_if\_HighRisk} &\sim \{0.266{:}\, \mathit{yes}, 0.664{:}\, \mathit{no}\}\end{aligned}$$

The uncertainty in these conditional probabilities is encoded by leaving part of the probability mass unspecified; e.g. for the high risk group, at least 0.266 of the probability mass is in the *yes* state, 0.664 is in the *no* state, and the remainder is unspecified.

An atom *dm* can be defined to indicate whether a patient suffers from diabetes by defining logical clauses for each case of **Predisposition**:

$$\begin{aligned}&\mathit{dm} \leftarrow \langle \mathbf{DM\_if\_AverageRisk} = \mathit{yes}\rangle, \quad \langle \mathbf{Predisposition} = \mathit{average}\rangle\\ &\mathit{dm} \leftarrow \langle \mathbf{DM\_if\_ModerateRisk} = \mathit{yes}\rangle, \langle \mathbf{Predisposition} = \mathit{moderate}\rangle\\ &\mathit{dm} \leftarrow \langle \mathbf{DM\_if\_HighRisk} = \mathit{yes}\rangle, \qquad \langle \mathbf{Predisposition} = \mathit{high}\rangle\end{aligned}$$

We can now compute, for example, a probability range for the probability of diabetes which yields: $0.087379 \leq P(\mathit{dm}) \leq 0.109177$.

To illustrate a combination with continuous variables, suppose the level of glucose is represented by two normal distributions $\mathcal{N}(\mu, \sigma^2)$, where μ and σ denote the mean and standard deviation for the cases where the patient either has or does not have diabetes:

$$\begin{aligned}\mathbf{Gluc\_if\_DM} &\sim \mathcal{N}(7.5, 3.8)\\ \mathbf{Gluc\_if\_not\_DM} &\sim \mathcal{N}(5.79, 0.98)\end{aligned}$$

Another continuous variable can be used to represent the level of HbA1c, which we may assume to linearly depend on the level of glucose plus some noise which depends on whether the patient is a diabetic. The noise variables can be modelled by two random variables, after which HbA1c can be defined:

$$\begin{aligned}\mathbf{Noise\_if\_DM} &\sim \mathcal{N}(0.0, 3.3)\\ \mathbf{Noise\_if\_not\_DM} &\sim \mathcal{N}(0.0, 0.3)\end{aligned}$$

$$\begin{aligned}\mathit{hba1c}(1.4 + 0.92 * \mathbf{Gluc\_if\_DM} + \mathbf{Noise\_if\_DM}) &\leftarrow \mathit{dm}\\ \mathit{hba1c}(0.6 + 0.9 * \mathbf{Gluc\_if\_not\_DM} + \mathbf{Noise\_if\_not\_DM}) &\leftarrow \mathit{not}(\mathit{dm})\end{aligned}$$

Using this representation, it is possible, for instance, to compute bounds on the probability of diabetes given that the level of HbA1c is larger than 7.2. The evidence can be encoded using the following clause:

$$e \leftarrow hba1c(H),\ \langle H > 7.2 \rangle$$

The following probability bounds can then be computed: $0.416 \le P(dm \mid e) \le 0.554$. Note that the imprecision results from the fact that we use imprecise probabilities, as well as from the fact that continuous distributions are approximated. Again, a better approximation can be found by investing more computation time.

2.3. Running example: fire on a ship

As a final example, we introduce the following short case description, which will be used to illustrate several concepts throughout this article. Suppose there is a fire in one compartment of a ship. The heat causes the hull of that compartment to warp and if the fire is not extinguished within 1.25 minutes the hull will breach. After 0.75 minutes the fire will spread to the compartment behind. This means that if the fire is extinguished within 0.75 minutes the ship is saved for sure:

$$saved \leftarrow \langle \mathbf{Time\_Comp}_1 < 0.75 \rangle$$

In the other compartment the hull will breach 0.625 minutes after the fire breaks out. In order to reach the second compartment the fire in the first one has to be extinguished. So both fires have to be extinguished within $0.75 + 0.625 = 1.375$ minutes. Additionally, the fire in the first compartment has to be extinguished within 1.25 minutes, because otherwise the hull breaches there. The second compartment is however more accessible, such that four fire-fighters can extinguish the fire at the same time, which means they can work four times faster:

$$saved \leftarrow \langle \mathbf{Time\_Comp}_1 < 1.25 \rangle,\ \langle \mathbf{Time\_Comp}_1 + 0.25 \cdot \mathbf{Time\_Comp}_2 < 1.375 \rangle$$

Finally, assume exponential distributions for both time durations available to extinguish the fires:

$$\mathbf{Time\_Comp}_1 \sim \mathrm{Exp}(1)$$
$$\mathbf{Time\_Comp}_2 \sim \mathrm{Exp}(1)$$

The interesting question here is how likely it is that the ship is saved, i.e. $P(saved)$ is required.

3. Background

In this section we discuss the theoretical background on which we build our theory.

3.1. Logic programming

Logic programming (LP) is based on the idea of using predicate logic as a programming language [22]. *Prolog* is by far the most well-known example of such a language, although there are several other LP languages available. Since LP is based on first-order logic, the language is also very suitable as a basis for knowledge representation and reasoning.

A logic program $\mathbf{L}$ consists of a set of *rules*, also called *clauses*. Rules are (implicitly universally quantified) expressions of the form:

$$h \leftarrow l_1, \ldots, l_n$$

where h is called the *head* and the collection of *literals* $l_1, \ldots, l_n$ form the *body* of the rule and represent a conjunction. The head h is an *atom*, i.e. an expression of the form $p(t_1, \ldots, t_m)$ with p a *predicate* and $t_1, \ldots, t_n$ are *terms*. If all the terms t_i are constants, the atom $p(t_1, \ldots, t_m)$ is called *ground*. The literals of a body are atoms (e.g. a) or negated atoms (e.g. $not(a)$). *Facts*, also called *unit clauses*, are clauses without a body, assumed to be always true. In the following, we use the convention that constants are denoted by lower-case letters (e.g. *apple*, *banana*, ...), while variables start with upper-case letters (e.g. *Fruit*, *BPrice*, ...).

Although the syntax of LP is a subset of *first-order logic* (FOL), the semantics of LP and FOL differ. The difference emerges from the semantics of negations in LP, which, similarly to the *closed world assumption* (CWA), states that a certain conclusion is false if it cannot be derived from the knowledge base. In contrast, in FOL a statement is only proved to be false if the negation of this statement is implied by the knowledge base, which means that some statements cannot be decided to be true or false. This difference allows LP to express non-ground inductive definitions, such as transitive closures, which is not possible in FOL [23].

For the remainder of this paper where we combine LP with probability theory, we require that knowledge bases have unique (2-valued) models. Programs $\mathbf{L}$ without negation are characterised by their smallest model $M(\mathbf{L})$, called the *least Herbrand model*, consisting of ground atoms entailed by the logic program $\mathbf{L}$, i.e. $a \in M(\mathbf{L})$ iff $\mathbf{L} \models a$. Similarly, logic programs with negations which are *stratified*, meaning that the program disallows certain combinations of recursion and negations, have a unique Herbrand model [24].

Non-stratified programs may still have a unique model in the form of non-partial *well-founded models* [25] or *stable models* [26]. We abstract from which class of programs and which declarative semantics of LP are used in the generalised distribution semantics; it is only required that each program has a unique model. Thus, in the following we use M to denote models given the chosen semantics, although not necessarily the least Herbrand model.

3.2. Probability theory

Probability theory offers a widely used and well-founded basis for representing and reasoning with uncertainty. The likelihood of *events* is expressed in terms of probabilities between 0 and 1, where 0 means that the event *(almost) certainly*[1] does not occur and 1 the opposite. Probability theory provides well defined ways to combine probabilities and makes use of partial knowledge about the state of the world by conditioning on *evidence*.

Often *random variables* are used to represent probability distributions and in this article we denote them with bold upper-case letters, e.g. **X**, **Y**. We assume that each random variable $\mathbf{V}_i$ has a range $\mathbf{Range}_i$ of values. Sometimes sets of random variables are indicated by an index set: $\mathbf{V}_I = \{\mathbf{V}_i \mid i \in I\}$, where I is an index set. In the case of a finite number of random variables with finite ranges, one can uniquely define a joint probability distribution P by assigning a probability number to each joint assignment of values $v_i \in \mathbf{Range}_i, i = 1, \dots, n$, to random variables $P(\mathbf{V}_1 = v_1, \dots, \mathbf{V}_n = v_n)$. From probability distributions one can compute the probability of partial assignments using *marginalisation*:

$$P(\mathbf{V}_K = v_K) = \sum_{v_J, J = I \setminus K} P(\mathbf{V}_K = v_k, \mathbf{V}_J = v_J), \tag{1}$$

where $I = \{i \mid 1 \le i \le n\}$.

Example 1

Assume we only have the random variable **Predisposition** of the example in Section 2.2 and view the atom *dm* as binary random variable (**DM**). The definition does not exactly define a probability distribution as it contains imprecise knowledge, but we can define a probability distribution consistent with the definition for those two random variables by the following assignments:

$$P(\mathbf{Predisposition} = \mathit{average}, \mathbf{DM} = \mathit{false}) = 0.652$$
$$P(\mathbf{Predisposition} = \mathit{average}, \mathbf{DM} = \mathit{true}) = 0.046$$
$$P(\mathbf{Predisposition} = \mathit{moderate}, \mathbf{DM} = \mathit{false}) = 0.189$$
$$P(\mathbf{Predisposition} = \mathit{moderate}, \mathbf{DM} = \mathit{true}) = 0.038$$
$$P(\mathbf{Predisposition} = \mathit{high}, \mathbf{DM} = \mathit{false}) = 0.05$$
$$P(\mathbf{Predisposition} = \mathit{high}, \mathbf{DM} = \mathit{true}) = 0.025$$

From this one can compute the probability of for instance $P(\mathbf{DM} = \mathit{true})$:

$$\begin{aligned} P(\mathbf{DM} = \mathit{true}) = \;& P(\mathbf{Predisposition} = \mathit{average}, \mathbf{DM} = \mathit{true}) \\ &+ P(\mathbf{Predisposition} = \mathit{moderate}, \mathbf{DM} = \mathit{true}) \\ &+ P(\mathbf{Predisposition} = \mathit{high}, \mathbf{DM} = \mathit{true}) \\ =\;& 0.109 \end{aligned}$$

This works well for the discrete case, but the continuous case, i.e. if random variables have uncountable ranges, requires more sophisticated techniques. In the case the range of random variables are the real numbers, probability measures for intervals are often defined in terms of *probability density functions* (PDFs). If f is a PDF of a random variable $\mathbf{V}_i$ then the probability that $\mathbf{V}_i$ takes a value within the interval $[a, b]$ is defined as:

$$P\big(\mathbf{V}_i \in [a, b]\big) = \int_a^b f(x)\,\mathrm{d}x \tag{2}$$

Generally, random variables offer an intuitive way to look at probability distributions, although they are not always suited to construct complex probability distributions. This is especially true for our purpose, as we want to deal with infinitely

[1] Note that the difference between *certainly* and *almost certainly* has to be made for continuous distributions. For instance, the probability that the temperature is exactly 20 °C is 0.0, but theoretically possible. Therefore, we cannot say that this event does "certainly" not occur, but have to say that it does "almost certainly" not occur.

many random variables, with different ranges of which some are countable and others uncountable. For this reason, we use a measure-theoretic approach to probability theory, specifically Kolmogorov's probability theory [27]. Probability spaces form the basis of modelling uncertain processes in this theory. A *probability space* $(\Omega, \mathcal{A}, P)$ formally consists of:

1. A *sample space* Ω: An arbitrary set of all possible states the model can be in.
2. An *event space* $\mathcal{A}$: A subset of the sample space's powerset $(\mathcal{A} \subseteq \wp(\Omega))$ which is a σ-algebra, meaning that it (i) contains the empty set $(\emptyset \in \mathcal{A})$, is closed under (ii) complement $(e \in \mathcal{A} \Longrightarrow (\Omega \setminus e) \in \mathcal{A})$ and (iii) countable union $(e_1, e_2 \in \mathcal{A} \Longrightarrow (e_1 \cup e_2) \in \mathcal{A})$. A consequence of properties (i) and (ii) is that the entire sample space Ω is always part of the event space. Elements of $\mathcal{A}$ are called *events* and probabilities are only assigned to these events.
3. A *probability measure* P: A function assigning a number from the closed interval $[0, 1]$ to any event. P must be countably additive, which means that the probability of the union of countably many pairwise disjoint events must equal the sum of the probabilities of those events $(P(e_1) + P(e_2) = P(e_1 \cup e_2))$. Additionally, P must assign 1 to the entire sample space $(P(\Omega) = 1)$.

In this setting, random variables can still provide a convenient view on the probability distribution and are actually functions mapping the sample space to the variable's range: $\mathbf{V}_i : \Omega \to \mathbf{Range}_i$. This has to be done in such a way that the probability measure P of the original probability space assigns a probability to each event concerning the random variable $\mathbf{V}_i$ as well.

In this article we use a simple mapping from sample spaces to random variables. We represent the random variables' values as tuples of which each element represents a single random variable's state. The random variables are therefore just functions mapping a tuple to one specific element.

Example 2

The probability distribution of Example 1 can be represented as the following probability space. The sample space consists of all six possible assignments to both random variables:

$$\Omega = \{ \ (\mathit{false}, \mathit{average}), (\mathit{true}, \mathit{average}), (\mathit{false}, \mathit{moderate}), \\ (\mathit{true}, \mathit{moderate}), (\mathit{false}, \mathit{high}), (\mathit{true}, \mathit{high}) \ \}$$

The random variables map the corresponding element of the tuple: $\mathbf{DM}\big((\omega_1, \omega_2)\big) = \omega_1$ and $\mathbf{Predisposition}\big((\omega_1, \omega_2)\big) = \omega_2$. The event space is the powerset of Ω, which is always a σ-algebra for countable sample spaces. The probability assignments from Example 1 translate to the following probability assignments of events:

$$\begin{aligned} P\big((\mathit{false}, \mathit{average})\big) &= 0.652 & P\big((\mathit{true}, \mathit{average})\big) &= 0.046 \\ P\big((\mathit{false}, \mathit{moderate})\big) &= 0.189 & P\big((\mathit{true}, \mathit{moderate})\big) &= 0.038 \\ P\big((\mathit{false}, \mathit{high})\big) &= 0.05 & P\big((\mathit{true}, \mathit{high})\big) &= 0.025 \end{aligned}$$

This uniquely defines a probability measure P, i.e. establishes probabilities for all other events in the event space, given the properties of a probability measure. The probability of diabetes can be computed as: $P\big(\{(\mathit{true}, \mathit{average}), (\mathit{true}, \mathit{moderate}), (\mathit{true}, \mathit{high})\}\big) = 0.109$.

Imprecise probability theory is a generalisation of probability theory. It avoids using crisp probabilities and therefore allows one to express ignorance concerning probability distributions. The theory is used if it is not possible to obtain precise probabilities, either due to insufficient availability of data to estimate the probabilities, or because probabilities are estimated by a number of experts, thus providing a range of probabilities.

There are different approaches to imprecise probabilities with varying expressiveness [28]. In this article we consider convex sets of probability distributions, referred to as *credal sets* [29]. This setting makes it possible to express probabilities of events by a lower and upper bound.

Formally, we denote a credal set as $\mathbf{P}$, assume it has an associated sample and event space and consists of a collection of probability measures consistent with Kolmogorov's axioms. For each event e there exists a distribution in $\mathbf{P}$ for which the probability attains a minimum and maximum. We denote these probabilities by $\underline{P}(e)$ and $\overline{P}(e)$, respectively.

3.3. Probabilistic logic programming

Combining logic with probability theory is a subject that has gained increased interest during the past few years, and various approaches, often with an associated software implementation, have been explored. We focus in our work on Sato's distribution semantics, which extends LP with probabilities [5].

The purpose of the distribution semantics is to extend the semantics of LP towards a probabilistic semantics by guaranteeing the existence of a unique probability distribution consistent with Kolmogorov's probability axioms. The key property here is the fact that logic programs always have a unique model. Sato uses the traditional least model semantics, but the concept can be generalised to all programs for which some kind of unique model can be defined.

Suppose we split the program $\mathbf{L}$ into rules R and facts F, i.e. $\mathbf{L} = F \cup R$. Facts are rules with an empty body and we assume that facts never occur as the head of a rule. Facts are considered as being binary random variables, taking values

Table 1
Distribution semantics example.

$\omega_F = (high\_pred, dm\_hp, dm\_no\_hp)$	$M_{\mathbf{L}}(\omega_F)$	$P_{\mathbf{L}}(\omega_F)$
$(false, false, false)$	$\{\}$	0.71
$(false, false, true)$	$\{dm\_no\_hp, dm\}$	0.03
$(false, true, false)$	$\{dm\_hp\}$	0.06
$(false, true, true)$	$\{dm\_hp, dm\_no\_hp, dm\}$	0.02
$(true, false, false)$	$\{high\_pred\}$	0.02
$(true, false, true)$	$\{high\_pred, dm\_no\_hp\}$	0.05
$(true, true, false)$	$\{high\_pred, dm\_hp, dm\}$	0.1
$(true, true, true)$	$\{high\_pred, dm\_hp, dm\_no\_hp, dm\}$	0.01

true or *false*. A program is called *grounded* if each variable is replaced by all possible ground terms. A grounded program potentially includes an infinite, but countable, number of ground facts. For instance, a probabilistic process could include random variables representing the weather (a fact) on an infinite number of future days. Looking upon each ground fact as a random variable, we have a sample space Ω_F, where each element is for example an infinite sequence of Boolean values indicating the weather on each day. The truth value of each literal in $\mathbf{L}$ is determined by such a truth assignment, because logic programs have unique models, as discussed in Section 3.1. We can therefore speak of the model associated with an element $\omega_F \in \Omega_F$ and denote it by $M_{\mathbf{L}}(\omega_F)$.

Given a probability measure P_F defined on the event space, which is the powerset of Ω_F, it is possible to extend this distribution to all atoms occurring in the program in a unique way. We refer to the resulting distribution as $P_{\mathbf{L}}$. The used sample space $\Omega_{\mathbf{L}}$ is defined similarly as the event space of the facts, but includes all atoms. The event space is again the sample space's powerset.

To construct a measure on an infinite event space, we construct finite measures restricted to the first n atoms, which can then be extended to a measure on the infinite event space using the extension theorem of probability measures [30, III.3]. A finite measure on the first n atoms is defined as:

$$P^n_{\mathbf{L}}(\omega_{\mathbf{L}}) \overset{\text{def}}{=} P_F\big(\{\omega_F \in \Omega_F \mid M_{\mathbf{L}}(\omega_F) \models \omega_{\mathbf{L}}\}\big) \tag{3}$$

Here we use an element $\omega_{\mathbf{L}}$ of the sample space as a logical formula representing a conjunction that determines for each atom whether it is true or false.

The probability measure defined on all atoms makes it possible to compute the probability of any arbitrary query sentence q by $P_{\mathbf{L}}\big(\{\omega_{\mathbf{L}} \in \Omega_{\mathbf{L}} \mid \omega_{\mathbf{L}} \models q\}\big)$. Probabilities of queries with finite grounding can be computed exactly, since finite measures can be used then.

Example 3

We use a simplified and slightly changed version of the example in Section 2.2, as the original distribution semantics can only deal with precise probabilities and discrete distributions. For convenience, we do not consider all three states of **Predisposition**, but only distinguish whether it is high or not high. Consider the following rules, indicating whether a patient has diabetes mellitus (dm) depending on the predisposition:

$$dm \leftarrow dm\_high\_pred, high\_predisposition$$

$$dm \leftarrow dm\_no\_high\_pred, not(high\_predisposition)$$

Assume we are given a probability measure P_F. Table 1 shows element of the event space of F, together with models and probabilities associated with this element.

We can now compute, for instance, the probability of $P(dm)$ by summing over all cases in which dm is included in the model: $P(dm) = 0.03 + 0.02 + 0.1 + 0.01 = 0.16$. In the same way we can compute the probability of other logical sentences. Examples are $P(dm \wedge high\_predisposition) = 0.1 + 0.01 = 0.11$ and $P(dm \vee high\_predisposition) = 0.03 + 0.02 + 0.02 + 0.05 + 0.1 + 0.01 = 0.23$.

4. A generalised distribution semantics

In this section we introduce a generalised distribution semantics, extending Sato's original distribution semantics to random variables with arbitrary, possibly infinite ranges. While the original distribution semantics is defined for binary probabilistic facts only, it can easily be generalised to random variables with finite ranges, e.g. the implementation of the PRISM language supports such random variables [31]. However, as soon as we deal with infinite ranges the generalisation is semantically far from straightforward, in particular when the ranges are uncountable. For example, grounding a real variable would lead to an uncountable number of ground atoms, for which the original distribution semantics does not provide a well-defined probability distribution.

We tackle this problem by augmenting our logical formalism with special constraints, an approach also adopted by deterministic *constraint logic programming* (CLP) [15] and *satisfiability modulo theories* (SMT) solvers. This leads to an expressive language where for many queries there does not exist a closed form expression to compute marginal probabilities, i.e. exact inference is not possible. In the second part of this section, we therefore also discuss sufficient conditions under which exact inference is possible in this extended language.

Note that the results of this section are not fundamentally different from known solutions as offered by, for example, *Hybrid ProbLog* [13]. However, for the first time we formalise this approach in a general way. As will become clear, this more general theory will act as a basis for the more advanced work discussed in the remainder of the paper.

4.1. Probability distributions on constraint logic programs

Whereas Sato's distribution semantics assigns a joint probability distribution to the ground atoms of a logic program using probabilistic facts, in the generalised distribution semantics we make use of constraints to define this joint distribution. To represent constraints, we define a general probabilistic constraint logic. Subsequently, a generalised distribution semantics for this constraint logical language is defined.

4.1.1. Constraint logic theories with probability measures

The basic idea of the language is to have countably many random variables $\mathbf{V} = \{\mathbf{V}_1, \mathbf{V}_2, \dots\}$ with ranges that are sets of elements with arbitrary properties, for example discrete constants and real numbers. Hence, the number of random variables can be infinite, similar to the original distributions semantics. Note that probabilities of events involving an infinite number of variables may not be computable, but we separate the semantics from the issue of performing inference in order to not unnecessarily restrict the generality of the semantics.

Constraints φ will be looked upon as predicates on the state of the random variables, i.e. φ is a function from $\mathbf{V}_1 = v_1, \mathbf{V}_2 = v_2, \dots$ to $\{true, false\}$ where v_i is an element in the range of $\mathbf{V}_i$. Equivalently, each constraint can be seen as a predicate on the sample space, as it corresponds to the random variable's states. The subset of the sample space where a constraint holds, is called the *solution space* of the constraint.

Definition 1 *(Constraint solution space).* The solution space of a constraint φ given sample space Ω is defined as:

$$\mathrm{CSS}(\varphi) \overset{\text{def}}{=} \{\omega \in \Omega \mid \mathbf{V}(\omega) = \mathbf{v}, \varphi(\mathbf{v})\}$$

where $\mathbf{V}(\omega) = \mathbf{v}$ is shorthand for $\mathbf{V}_i(\omega) = \mathbf{v}_i$ for all i.

In the logical language, we will use rules similar to the ones in logic programming, except that in the body we may use both literals and constraints of the form $\langle\varphi(\mathbf{V})\rangle$. The brackets indicate that this is not a regular predicate, but a constraint, which makes it syntactically similar to normal CLP [15].[2] Formally, we define constraint logic theories as follows.

Definition 2 *(Probabilistic constraint logic theory).* A probabilistic constraint logic theory $\mathbf{T}$ is a tuple

$$(\mathbf{V}, \Omega_{\mathbf{V}}, \mathcal{A}_{\mathbf{V}}, P_{\mathbf{V}}, \mathbf{Constr}, \mathbf{L})$$

where

- $\mathbf{V} = \{\mathbf{V}_1, \mathbf{V}_2, \dots\}$ is a countable set representing random variables with associated non-empty ranges $\{\mathbf{Range}_1, \mathbf{Range}_2, \dots\}$ (we fix the enumeration such that each random variable has an index);
- $\Omega_{\mathbf{V}}$ is the sample space of the random variables $\mathbf{V}$ defined as the Cartesian product of the random variables' ranges:

  $$\Omega_{\mathbf{V}} \overset{\text{def}}{=} \mathbf{Range}_1 \times \mathbf{Range}_2 \times \cdots;$$

- $\mathcal{A}_{\mathbf{V}}$ is an event space representing events concerning the random variables $\mathbf{V}$;
- $P_{\mathbf{V}}$ is a probability measure on the space defined above, thus the tuple $(\Omega_{\mathbf{V}}, \mathcal{A}_{\mathbf{V}}, P_{\mathbf{V}})$ is a probability space;
- $\mathbf{Constr}$ is a set of constraints, closed under conjunction, disjunction and negation, such that:

  $$\{\mathrm{CSS}(\varphi) \mid \varphi \in \mathbf{Constr}\} \subseteq \mathcal{A}_{\mathbf{V}}$$

  i.e. the constraints correspond to events;
- $\mathbf{L}$ is a set of logic programming rules with constraints:

  $$h \leftarrow l_1, \dots, l_n, \langle\varphi_1(\mathbf{V})\rangle, \dots, \langle\varphi_m(\mathbf{V})\rangle$$

  where $\varphi_i \in \mathbf{Constr}$, $1 \le i \le m$.

[2] We use $\langle\rangle$ rather than $\{\}$ as used in CLP, to avoid confusion with set notation in mathematical definitions.

In the remainder of this section, we abstract from the actual constraint language used. In the examples given in this section, the language used to define constraints is only meant as an illustration. For example, if all $\mathbf{V}_i$ are continuous variables, then we may write $\langle \forall i\, \mathbf{V}_{i+1} \geq \mathbf{V}_i \rangle$ to express that $\mathbf{V}_i$ increases with i. If we interpret these constraints as simple ground atoms, then these rules are to be interpreted as a normal logic program with some associated logic programming semantics, as discussed in Section 3.1.

Note that since the sample space is defined by the Cartesian product of the ranges, the random variables are simple projections of single tuple elements of the sample space. So the solution space of an arbitrary constraint φ equals $\{\omega \in \Omega \mid \varphi(\omega)\}$.

Example 4

Consider the running example of Section 2.3:

$$\mathbf{Time\_Comp}_1 \sim \mathrm{Exp}(1)$$

$$\mathbf{Time\_Comp}_2 \sim \mathrm{Exp}(1)$$

$$saved \leftarrow \langle \mathbf{Time\_Comp}_1 < 0.75 \rangle$$

$$saved \leftarrow \langle \mathbf{Time\_Comp}_1 < 1.25 \rangle, \langle \mathbf{Time\_Comp}_1 + 0.25 \cdot \mathbf{Time\_Comp}_2 < 1.375 \rangle$$

Suppose that $\mathbf{Time\_Comp}_1$ and $\mathbf{Time\_Comp}_2$ are the only two random variables in the enumeration. The range of both variables are the real numbers: $\mathbf{Range}_1 = \mathbf{Range}_2 = \mathbb{R}$. The used constraint language includes at least linear inequalities and the probability measure is such that the first two variables are independent and distributed according to an exponential distribution.

4.1.2. Extending probability spaces to the entire theory

While a theory **T** defines a probability distribution over the random variables, the probabilities of the events in the logical language are not specified directly. As in the original distribution semantics, a probability distribution over the random variables can uniquely be extended to a distribution over the entire program. Next it is shown how to extend the sample space, the event space and the probability measure to the entire theory **T**.

We generalise the distribution semantics by looking upon probabilistic facts occurring in rules as a special kind of constraint, i.e. the probabilistic fact *pf* in a rule represents the constraint that *pf* is true. In order to define this semantics for arbitrary constraints, we extend the sample space by considering all atoms appearing in the logical theory **L**. We assume there is a countable number of atoms, treat them as random variables taking values *true* or *false* and denote the set of all those atoms with **A**. As for variables in **V** we fix the enumeration and define the sample space $\Omega_{\mathbf{L}}$ being the Cartesian product of the values of all atoms:

$$\Omega_{\mathbf{L}} \overset{\text{def}}{=} \prod_{i=1}^{|\mathbf{A}|} \{true, false\} \tag{4}$$

For the event space of the logic part of the theory $\mathcal{A}_{\mathbf{L}}$, we can take the sample space's powerset, since the sample space is countable:

$$\mathcal{A}_{\mathbf{L}} \overset{\text{def}}{=} \wp(\Omega_{\mathbf{L}}) \tag{5}$$

The sample space for the entire theory $\Omega_{\mathbf{T}}$ is the Cartesian product of the sample spaces for the random variables and the logical theory:

$$\Omega_{\mathbf{T}} \overset{\text{def}}{=} \Omega_{\mathbf{V}} \times \Omega_{\mathbf{L}} \tag{6}$$

The event space of the entire theory $\mathcal{A}_{\mathbf{T}}$ is built as well from the event spaces of the random variables and logical theory. Concretely, it is their *tensor-product σ-algebra*:

$$\mathcal{A}_{\mathbf{T}} \overset{\text{def}}{=} \mathcal{A}_{\mathbf{V}} \otimes \mathcal{A}_{\mathbf{L}} \tag{7}$$

The tensor-product σ-algebra $\mathcal{A}_{\mathbf{V}} \otimes \mathcal{A}_{\mathbf{L}}$ is the smallest σ-algebra generated by the products of elements of $\mathcal{A}_{\mathbf{V}}$ and $\mathcal{A}_{\mathbf{L}}$: $\sigma_{\Omega_{\mathbf{V}} \times \Omega_{\mathbf{L}}}\big(\{e_{\mathbf{V}} \times e_{\mathbf{V}} \mid e_{\mathbf{V}} \in \mathcal{A}_{\mathbf{L}}, e_{\mathbf{L}} \in \mathcal{A}_{\mathbf{L}}\}\big)$. We cannot simply use the product of both spaces, as such product is not always a σ-algebra.

Example 5

Consider the following event spaces:

$$\mathcal{A}_{\mathbf{V}} = \big\{\emptyset, \{a\}, \{b, c\}, \{a, b, c\}\big\}$$

$$\mathcal{A}_{\mathbf{L}} = \big\{\emptyset, \{true\}, \{false\}, \{true, false\}\big\}$$

The product $\{e_{\mathbf{V}} \times e_{\mathbf{L}} \mid e_{\mathbf{V}} \in \mathcal{A}_{\mathbf{V}}, e_{\mathbf{L}} \in \mathcal{A}_{\mathbf{L}}\}$ is then:

$$\begin{aligned}\{\ & \emptyset, \{(a, \mathit{true})\}, \{(a, \mathit{false})\}, \{(a, \mathit{true}), (a, \mathit{false})\},\\ & \{(b, \mathit{true}), (c, \mathit{true})\}, \{(b, \mathit{false}), (c, \mathit{false})\}, \{(b, \mathit{true}), (b, \mathit{false}), (c, \mathit{true}), (c, \mathit{false})\},\\ & \{(a, \mathit{true}), (b, \mathit{true}), (c, \mathit{true})\}, \{(a, \mathit{false}), (b, \mathit{false}), (c, \mathit{false})\},\\ & \{(a, \mathit{true}), (a, \mathit{false}), (b, \mathit{true}), (b, \mathit{false}), (c, \mathit{true}), (c, \mathit{false})\}\ \}\end{aligned}$$

This product is no σ-algebra as for instance $\{(a, \mathit{false})\} \cup \{(b, \mathit{true}), (c, \mathit{true})\}$ is not included. In the tensor product the minimal number of elements are added to the product, such that it becomes a σ-algebra.

Finally, the probability measure $P_{\mathbf{V}}$ is extended to a probability measure on the entire theory yielding $P_{\mathbf{T}}$. The way this is achieved in the distribution semantics builds upon the observation that determining truth values of the probabilistic facts uniquely determines the truth values of all atoms in the entire theory. In our generalised setting we similarly observe that determining which constraints are true uniquely determines the truth values of all atoms in the entire theory.

To formalise this notion, we will make use of the set **satisfiable**$(\omega_{\mathbf{V}})$, which includes all constraints which are satisfiable given a valuation $\omega_{\mathbf{V}}$ of the random variables. We can interpret this set as a partial logic program, by assuming that each constraint occurs as instantiation of the predicate $\langle\rangle$.

Example 6

Suppose that $\omega_{\mathbf{V}} = (0, \ldots)$, i.e. the first random variable $\mathbf{V}_1$ takes value 0. Then **satisfiable**$(\omega_{\mathbf{V}})$ does not include $\langle \mathbf{V}_1 > 0\rangle$, but does include for instance $\langle \mathbf{V}_1 > -1\rangle$.

An element of the sample space $\omega_{\mathbf{V}}$ therefore yields a logic theory $\mathbf{L} \cup$ **satisfiable**$(\omega_{\mathbf{V}})$, which we assume to have a unique model, as discussed in Section 3.1. This model is denoted by $M_{\mathbf{L}}(\omega_{\mathbf{V}})$.

Example 7

Consider again the clauses of the running example (Section 2.3):

$$\mathit{saved} \leftarrow \langle \mathbf{Time\_Comp}_1 < 0.75\rangle$$

$$\mathit{saved} \leftarrow \langle \mathbf{Time\_Comp}_1 < 1.25\rangle, \langle \mathbf{Time\_Comp}_1 + 0.25 \cdot \mathbf{Time\_Comp}_2 < 1.375\rangle$$

The definition of the atom *saved* implies that it is true if and only if the constraint $\mathbf{Time\_Comp}_1 < 0.75$ or both constraints $\mathbf{Time\_Comp}_1 < 1.25$ and $\mathbf{Time\_Comp}_1 + 0.25 \cdot \mathbf{Time\_Comp}_2 < 1.375$ are satisfied.

No model $M_{\mathbf{L}}\big((1,2)\big)$ includes $\langle \mathbf{Time\_Comp}_1 < 0.75\rangle$ and $\langle \mathbf{Time\_Comp}_1 + 0.25 \cdot \mathbf{Time\_Comp}_2 < 1.375\rangle$, since these constraints are not in **satisfiable**$\big((1,2)\big)$. $\langle \mathbf{Time\_Comp}_1 < 1.25\rangle$ is included in those models, but given the clauses above, *saved* is not included in $M_{\mathbf{L}}\big((1,2)\big)$. In contrast, $M_{\mathbf{L}}\big((0.5,1)\big)$ includes $\langle \mathbf{Time\_Comp}_1 < 0.75\rangle$ and therefore *saved* as well.

Given these notions, a probability measure $P_{\mathbf{T}}$ can be defined on the extended event space. The idea is to uniquely derive this measure from $P_{\mathbf{V}}$ by mapping elements of the entire event space to elements of the random variable's event space, such that truth values of logical atoms correspond to the unique models $M_{\mathbf{L}}(\omega_{\mathbf{V}})$, given by valuations of random variables $\omega_{\mathbf{V}}$.

Definition 3 *(Entire theory probability measure).* The probability measure on the entire theory **T**'s event space is defined as:

$$P_{\mathbf{T}}(e) \overset{\text{def}}{=} P_{\mathbf{V}}\big(\{\omega_{\mathbf{V}} \mid (\omega_{\mathbf{V}}, \omega_{\mathbf{L}}) \in e,\ M_{\mathbf{L}}(\omega_{\mathbf{V}}) \models \omega_{\mathbf{L}}\}\big)$$

Here we use elements $\omega_{\mathbf{L}}$ of the logical theory's sample space as logical formulas, where they represent conjunctions determining for each atom whether it is true or not. For example, $\omega_{\mathbf{L}} = (0, 1, 0, \ldots)$ means $\neg a \wedge b \wedge \neg c \wedge \cdots$, where $a, b, c, \ldots$ are the atoms of **L**.

It is ensured that events in this definition are in $\mathcal{A}_{\mathbf{V}}$, since the restriction $M_{\mathbf{L}}(\omega_{\mathbf{V}}) \models \omega_{\mathbf{L}}$ is based on compositions of events from $\mathcal{A}_{\mathbf{V}}$. We then extend this to the probability of a query atom q given a probability measure $P_{\mathbf{T}}$ as follows.

Definition 4 *(Query probability).* The probability of query q is defined as:

$$P(q) \overset{\text{def}}{=} P_{\mathbf{T}}\big(\{(\omega_{\mathbf{V}}, \omega_{\mathbf{L}}) \in \Omega_{\mathbf{T}} \mid \omega_{\mathbf{L}} \models q\}\big)$$

Note that there is no need for a restriction on the values of random variables in $\omega_{\mathbf{V}}$ in Definition 4, since Definition 3 ensures that valuations of random variables for which q does not hold do not contribute to the probability. We further know that the event defined by the equation above is an element of the event space $\mathcal{A}_{\mathbf{T}}$, since we do not put any restrictions on values of random variables and the event space concerning the logic atoms is defined as the powerset of the sample space (Equation (5)) thus each subset of the sample space is in the event space.

Example 8

Consider again the clauses of the running example (Section 2.3):

$$saved \leftarrow \langle \mathbf{Time\_Comp}_1 < 0.75 \rangle$$

$$saved \leftarrow \langle \mathbf{Time\_Comp}_1 < 1.25 \rangle, \langle \mathbf{Time\_Comp}_1 + 0.25 \cdot \mathbf{Time\_Comp}_2 < 1.375 \rangle$$

Suppose that *saved* corresponds to the first dimension in $\Omega_{\mathbf{L}}$, such that $\omega_{\mathbf{L}} \models saved$ requires the first element of each sample to be *true*. Then by applying Definition 4 we obtain:

$$P(saved) = P_{\mathbf{T}}\big(\{(\omega_{\mathbf{V}}, (saved, \ldots)) \in \Omega_{\mathbf{T}} \mid saved = true\}\big)$$

Then by applying Definition 3 we see that:

$$P(saved) = P_{\mathbf{V}}\big(\{(\omega_1, \ldots) \mid ((\omega_1, \ldots), (saved, \ldots)) \in \Omega_{\mathbf{T}},$$
$$M_{\mathbf{L}}(\omega_1, \ldots) \models (saved, \ldots), saved = true\}\big)$$

Given the clauses above and assuming $\mathbf{Time\_Comp}_1$ and $\mathbf{Time\_Comp}_2$ correspond to the first two random variables, $M_{\mathbf{L}}(\omega_1, \omega_2, \ldots)$ entails *saved* if and only if $\omega_1 < 0.75 \vee (\omega_1 < 1.25 \wedge \omega_1 + 0.25 \cdot \omega_2 < 1.375)$:

$$\begin{aligned} P(saved) &= P_{\mathbf{V}}\big(\{(\omega_1, \ldots) \mid ((\omega_1, \ldots), (saved, \ldots)) \in \Omega_{\mathbf{T}}, \\ &\qquad \omega_1 < 0.75 \vee (\omega_1 < 1.25 \wedge \omega_1 + 0.25 \cdot \omega_2 < 1.375)\}\big) \\ &= P_{\mathbf{V}}\big(\{(\omega_1, \ldots) \in \Omega_{\mathbf{V}} \mid \omega_1 < 0.75 \vee (\omega_1 < 1.25 \wedge \omega_1 + 0.25 \cdot \omega_2 < 1.375)\}\big) \end{aligned}$$

As $\mathbf{Time\_Comp}_1, \mathbf{Time\_Comp}_2 \sim \mathrm{Exp}(1)$, the probability $P(saved)$ can be computed as follows. We denote the associated density function with f and the cumulative distribution function with F:

$$P_{\mathbf{V}}(\{(\omega_1, \ldots) \in \Omega_{\mathbf{V}} \mid \omega_1 < 0.75\}) = F(0.75) = 1 - e^{-0.75} \approx 0.53$$

$$\begin{aligned} &P_{\mathbf{V}}(\{(\omega_1, \ldots) \in \Omega_{\mathbf{V}} \mid \omega_1 < 0.75 \wedge \omega_1 + 0.25 \cdot \omega_2 < 1.375\}) \\ &\quad = \int_0^{0.75} f(x) P(\mathbf{Time\_Comp}_2 < 4 \cdot 1.375 - 4x) \mathrm{d}x \\ &\quad = \int_0^{0.75} f(x) F(5.5 - 4x) \mathrm{d}x = \int_0^{0.75} e^{-x} - e^{3x-5.5} \mathrm{d}x \approx 0.52 \end{aligned}$$

$$P_{\mathbf{V}}(\{(\omega_1, \ldots) \in \Omega_{\mathbf{V}} \mid \omega_1 < 1.25 \wedge \omega_1 + 0.25 \cdot \omega_2 < 1.375\}) = \int_0^{1.25} e^{-x} - e^{3x-5.5} \mathrm{d}x \approx 0.66$$

To summarise:

$$P_{\mathbf{V}}(\{(\omega_1, \ldots) \in \Omega_{\mathbf{V}} \mid \omega_1 < 0.75 \vee (\omega_1 < 1.25 \wedge \omega_1 + 0.25 \cdot \omega_2 < 1.375)\}) \approx 0.53 + 0.66 - 0.52 = 0.67$$

By combining Definitions 3 and 4 we can determine a single event in the random variables' probability space with the same probability as a query q. Such an event is referred to in the following as the *solution event*. In this way, a separation is achieved of the symbolic part – determining in which cases a statement is true by logical reasoning – and the probabilistic part – determining the probability that one of those cases occurs.

Definition 5 *(Solution event).* The solution event of a query q is defined as:

$$\mathrm{SE}(q) \stackrel{\text{def}}{=} \{\omega_{\mathbf{V}} \in \Omega_{\mathbf{V}} \mid M_{\mathbf{L}}(\omega_{\mathbf{V}}) \models q\}$$

Lemma 1. *The probability of a query q as defined by Definition 4 can be computed using the solution event:*

$$P(q) = P_{\mathbf{V}}\big(\mathrm{SE}(q)\big)$$

All proofs are provided in Appendix A.

Definition 5 formulates the solution event in terms of samples. A different formulation in terms of constraints would, however, be more suited to the subsequent formulation of conditions under which exact inference is feasible. Such a formulation is provided by the following lemma.

Lemma 2. *The solution event of a query q can be expressed as:*

$$\mathrm{SE}(q) = \mathrm{CSS}\Big(\bigvee_{\substack{M_{\mathbf{L}}[\Phi] \models q \\ \Phi \subseteq \mathbf{Constr}}} \bigwedge_{\varphi \in \Phi} \varphi \Big)$$

where Φ denotes subsets of all constraints and $M_{\mathbf{L}}[\Phi]$ the model of the theory $\mathbf{L} \cup \{\langle\varphi\rangle \mid \varphi \in \Phi\}$.

Intuitively, the disjunction represents the collection of all possible models in which q is true in a logic sense, without dealing with the meaning of the construct $\langle\rangle$. All constraints in each Φ must be true at the same time in order to make q true as well. Thus using conjunctions, the sets of constraints are combined.

Example 9

Consider again the clauses of the running example (Section 2.3):

$$saved \leftarrow \langle \mathbf{Time\_Comp}_1 < 0.75\rangle$$

$$saved \leftarrow \langle \mathbf{Time\_Comp}_1 < 1.25\rangle, \langle \mathbf{Time\_Comp}_1 + 0.25 \cdot \mathbf{Time\_Comp}_2 < 1.375\rangle$$

Given these two clauses, there are two ways to prove *saved*. Each model including *saved* therefore has to include all constraints enforced by the first or second clause, which corresponds to the disjunction in Lemma 2. The second clause requires two constraints to hold, so they are combined using the conjunction in Lemma 2. The solution event of *saved* therefore is $\{(\omega_1, \omega_2, \ldots) \in \Omega_{\mathbf{V}} \mid \omega_1 < 0.75 \vee (\omega_1 < 1.25 \wedge \omega_1 + 0.25 \cdot \omega_2 < 1.375)\}$. Note that the event is the same as used to compute the query probability in Example 8.

4.2. Exact inference conditions

The semantics introduced above is very general and powerful, but the computation of event probabilities is in general infeasible. Practically useful languages always demand finding the proper balance between expressivity and feasibility of inference. We therefore discuss ways to restrict the general semantics in such a way that exact inference becomes feasible. The result provides a basis for analysing in a structured way which properties allow exact inference for different languages.

Proposition 1. *The probability of an arbitrary query can be computed exactly if the following conditions hold:*

1. ***finite-relevant-constraints condition**: There are only finitely many constraint predicates ($\langle\rangle$) relevant for determining truthfulness of each query atom. Formally, a constraint predicate $\langle\varphi\rangle$ is relevant for a query atom q if there exists a set of constraint predicates Φ, such that $q \in M(\Phi \cup \mathbf{L}) \not\Leftrightarrow q \in M(\{\langle\varphi\rangle\} \cup \Phi \cup \mathbf{L})$. Intuitively, there exists a set of constraint predicates for which it matters whether $\langle\varphi\rangle$ is included in the program or not, meaning $\langle\varphi\rangle$ is relevant. We also assume that finding such relevant constraints predicates and entailment checking can be done in finite time.*
2. ***finite-dimensional-constraints condition**: Constraints occurring in clauses as argument of the construct $\langle\rangle$ only concern a finitely-dimensional subset of the sample space. This means the constraint's solution spaces have the form $\{(\omega_1, \ldots, \omega_n, \omega_{n+1}, \ldots) \in \Omega_{\mathbf{V}} \mid cond(\omega_1, \ldots, \omega_n)\}$ where cond is an arbitrary predicate with n arguments, i.e. the constraint puts a condition only on a finite number of variables.*
3. ***computable-measure condition**: The probability of finite-dimensional events, in the sense of the previous condition, are computable.*

The computable-measure condition implies that one can exactly compute finite-dimensional integrals over employed PDFs, which is only possible under very strong assumptions.

The conditions stated here are sufficient, but not strictly necessary, as we do not restrict the kind of continuous distributions employed, for instance to Gaussian distributions, which is often done in other work, as discussed in Section 8. These conditions generalise the restrictions typically enforced by other approaches based on the distribution semantics, as will be discussed below.

The following example illustrates the exact inference conditions.

Example 10

Consider again the clauses of the running example (Section 2.3):

$$saved \leftarrow \langle \mathbf{Time\_Comp}_1 < 0.75\rangle$$

$$saved \leftarrow \langle \mathbf{Time\_Comp}_1 < 1.25\rangle, \langle \mathbf{Time\_Comp}_1 + 0.25 \cdot \mathbf{Time\_Comp}_2 < 1.375\rangle$$

As shown in Example 9 the solution event can be derived in finite time (finite-relevant-constraints condition) and is finite-dimensional, since all constraints in the program above are finite-dimensional as well (finite-dimensional-constraints condition). Assume we can, as in Example 8, exactly compute that the probability of $\{(\omega_1, \omega_2, \ldots) \in \Omega_{\mathbf{V}} \mid \omega_1 < 0.75 \vee (\omega_1 < 1.25 \wedge \omega_1 + 0.25 \cdot \omega_2 < 1.375)\}$ is 0.67 (computable-measure condition), thus $P(saved) = 0.67$.

An example of a problem that does not fulfil the exact inference condition is:

$$forever\_sun(X) \leftarrow \langle \mathbf{Weather}_X = sunny \rangle,\ forever\_sun(X+1)$$

The predicate $forever\_sun(X)$ intuitively represents that it is sunny forever from day X on, assuming there are infinitely many days in the future. The probability of $forever\_sun(0)$ cannot be computed in finite time, since an infinite number of days have to be considered, which means the finite-relevant-constraints condition is violated. One could usually say the limit of the probability of $forever\_sun(0)$ would be 0, but this is not true for all possible probability measures. Only assuming the computable-measure condition for the probability measure one cannot draw that conclusion.

An alternative definition of the problem would be:

$$forever\_sun \leftarrow \langle \underset{i \in \mathbb{N}}{\forall} \mathbf{Weather}_i = sunny \rangle$$

Similarly, the probability of *forever_sun* might be computable with further assumptions on the probability space, but generally this is not possible, since the finite-dimensional-constraints condition is violated.

We briefly discuss those conditions in the context of existing approaches. The finite-relevant-constraints condition seems a very reasonable condition for allowing exact inference and cannot be avoided. In Sato's original semantics, a condition called finite-support condition is required for probabilistic facts, which is similar to the finite-relevant-constraints condition, although restricted to positive programs. Similarly, the finite-dimensional-constraints condition is enforced in Sato's semantics, if we interpret probabilistic facts in the program as constraints concerning only a single variable, i.e. a probabilistic fact is required to be true or false; dependencies are expressed by the structure of rules. So actually a stronger variant of the finite-dimensional-constraints condition is enforced, as constraints only concern single variables. Finally, the computable-measure condition depends on how the probability distribution is defined in a concrete language. Most languages based on the distribution semantics satisfy this property by assuming that all random variables are (mutually) independent. This means that the probability measure is defined in terms of a single probability per variable and consequently the probability of events consisting of only a finite number of variables can be computed in finite time. Again this is done without loss of generality as dependencies can be introduced by the structure of the logic program. The situation is more complex when continuous distributions are considered. The most relevant language where exact inference is possible is *Hybrid ProbLog* [13] which extends Sato's semantics for continuous variables. As in Sato's semantics, *Hybrid ProbLog* only allows constraints on single variables. This means that for instance $\langle \mathbf{V}_1 > 0 \rangle$ is allowed, but $\langle \mathbf{V}_1 \geq \mathbf{V}_2 \rangle$ is not. This restriction ensures that the computable-measure condition is fulfilled, under the assumption that we can compute CDFs of the employed continuous distributions. While in the binary case the restriction to constraints on single variables does not restrict expressiveness of the language, in the continuous case it does. The main theory to overcome these restrictions are discussed next.

5. Relaxing the exact inference conditions using credal sets

The aim of this section is to provide a new theory that supports exact inference also covering problems involving constraints with multiple variables. As a consequence, only the general finite-dimensional-constraints condition is required, as introduced in the previous section. For example, we wish to allow the comparison of real-valued random variables, while at the same time avoiding severely restricting the kind of distributions to fulfil the computable-measure condition. This problem is tackled by introducing *credal sets*, i.e. a set of probability distributions. We show that this idea makes it possible to compute bounds on the probabilities of a query under conditions that are less strict than those assumed before. We finally discuss an important application of the theory: the approximation of precise, continuous distributions.

5.1. Assigning probability mass to events

We first give an overview of the basic idea. The goal is to define joint probability distributions in such a way that probabilities can still be computed as finite sums for queries with finite proofs. We achieve this by assigning probability masses to a number of arbitrary events rather than exhaustively to all atomic events. As a consequence, we deal with credal sets rather than a single probability distribution. However, at the same time, we will guarantee that this set is non-empty. Furthermore, the ability to express imprecise probabilities by credal sets is a useful feature if the probabilistic knowledge that is available is imprecise.

We define the meaning of the assignment of probability mass to a set of values of a random variable such that the probability mass is somehow distributed over those values. As there are multiple possible ways in which the probability mass can be distributed, this implies that this does not define a unique probability distribution, but a credal set, as illustrated by the following example.

Example 11

Assume we have a single random variable with the real numbers as range and assign some probability mass to the set of all values between 1 and 3. Fig. 1 depicts some possible ways of how the probability mass can be distributed: it can be distributed uniformly over the entire set (Fig. 1a) or only parts of it (Figs. 1b and 1c) or distributed in a more complex manner (Fig. 1d). These are just a few examples; there are actually uncountably many ways to distribute the probability mass over that set.

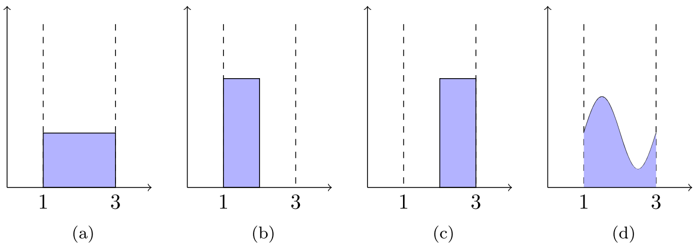

Fig. 1. Examples of possible distributions of probability mass over all real values between 1 and 3.

Defining a set of distributions is still useful, since for each query a lower bound and an upper bound for the success probability given the set of distributions can be computed. We provide some intuition on how the bounds can be computed through the following example before tackling the problem formally.

Example 12

Consider the following program:

$$q \leftarrow \langle \mathbf{V}_1 \geq 0 \rangle$$

Assume we define the probability distribution of $\mathbf{V}_1$ by assigning a probability mass of 0.1 to the set of all values smaller than -1 (Set 1), 0.3 to the closed interval $[-1, 1]$ (Set 2) and 0.6 to the set of all values larger than 1 (Set 3). It is clear that no matter how the probability mass is distributed over the values in Set 1 the probability that q can be derived is always 0.0. For Set 2 the probability could be 0.0 as well in case all probability mass is distributed to values below 0.0, or 0.3 in the opposite case. For Set 3 the query can always be derived no matter how the probability mass is distributed. We conclude that the probability of q is at least 0.6 and at most 0.9.

5.2. Credal set specifications

As discussed before, the basic idea is to assign probability masses to sets of values. Sets of values correspond to events and consequently credal sets are defined in terms of probability-event pairs. We refer to this kind of definitions as *credal set specifications*. Such specifications have the desirable property that they are guaranteed to define non-empty *credal sets*, i.e. sets of probability measures. *Credal set specifications* introduced in this section are between the purely semantic level of *credal sets* and the concrete syntactic level which is discussed later.

Since the number of random variables in our semantics can be infinite, we would have to allow potentially infinite sets of probability-event pairs. Such specifications would be hard to define directly and it is not clear how to construct probability distributions consistent with such specifications. Therefore, we define credal set specifications by means of a sequence of countably, potentially infinite number of specifications each defining finite-dimensional credal sets with increasing dimensionality. We have to make sure such specifications do not contradict each other, which we ensure by a property we call *compatibility*. This allows us to use an existing construction theorem to construct infinite-dimensional probability distributions, consistent with a credal set specification given.

Before we formally define the concept of credal set specifications we introduce the concept of *event projections*. We denote the i-th event projection of event e, where i is a natural number, with e^i and define it as:

$$\pi_i(e) \overset{\text{def}}{=} \left\{(\omega_1, \ldots, \omega_i) \mid (\omega_1, \ldots, \omega_i, \ldots) \in \Omega\right\} \tag{8}$$

Event projections are also similarly defined for finite events.

Definition 6 *(Credal set specification).* A credal set specification $\mathbf{C}$ consists of a countable number of finite-dimensional specifications $\mathbf{C}_1, \mathbf{C}_2, \ldots$. Each $\mathbf{C}_k$ has the form of a finite collection of probability-event pairs $(p_1, e_1), (p_2, e_2), \ldots, (p_n, e_n)$ such that for each $\mathbf{C}_k$:

1. The events are in a finite-dimensional event space $\mathcal{A}^k_{\mathbf{V}}$ over the sample space $\Omega^k_{\mathbf{V}} \overset{\text{def}}{=} \mathbf{Range}_1 \times \mathbf{Range}_2 \times \cdots \times \mathbf{Range}_k$.
2. The sum of the probabilities is 1.0: $\sum_{(p,e) \in \mathbf{C}_k} p = 1.0$.
3. The events must not be the empty set: $\forall_{(p,e) \in \mathbf{C}_k}\, e \neq \emptyset$.

Additionally, all finite $\mathbf{C}_k$ must be compatible, i.e. for all k: $\mathbf{C}_k = \pi_k(\mathbf{C}_{k+1})$, where $\pi_l(\mathbf{C}_k)$ is defined as:

$$\pi_l(\mathbf{C}_k) \overset{\text{def}}{=} \Big\{\big(\sum_{\substack{\pi_l(e)=e' \\ (p,e) \in \mathbf{C}_k}} p, e'\big) \mid e' \in \{\pi_l(e) \mid (p,e) \in \mathbf{C}_k\}\Big\}$$

One can look upon these credal set specifications as a way to split the probability mass into portions which are assigned to specific non-empty events given a finite set of random variables. As said, compatibility is used to inductively construct consistent distributions over all random variables by ensuring that specifications of different dimensionality do not contradict each other.

Example 13

Assume the first two random variables in the sample space have as range the set $\{sun, rain\}$ and $\mathbf{C}_1 = \Big\{\big(0.2, \{sun, rain\}\big), \big(0.8, \{sun\}\big)\Big\}$. Suppose this means the probability that there is sun tomorrow is at least 80%.

Consider the following specification, concerning not only the weather of tomorrow, but as well the weather of the day after tomorrow:

$$\mathbf{C}_2 = \Big\{\big(0.2, \{(sun, sun), (sun, rain)\}\big), \big(0.8, \{(sun, sun)\}\big)\Big\}$$

Both specifications are not compatible, as $\mathbf{C}_2$ fixes the probability that there is sun tomorrow to 1.0, which conflicts with $\mathbf{C}_1$, technically $\pi_1(\mathbf{C}_2) = \Big\{\big(1.0, \{sun\}\big)\Big\} \neq \mathbf{C}_1$. In contrast, an example of a compatible specification is:

$$\mathbf{C}'_2 = \Big\{\big(0.2, \{(sun, sun), (sun, rain), (rain, sun)\}\big), \big(0.8, \{(sun, sun)\}\big)\Big\}$$

Each $\mathbf{C}_k$ defines a set of probability measures, given by the following definition.

Definition 7 *(Finite credal sets).* The set of all probability measures $\mathbf{P}^k_{\mathbf{V}}$ defined by $\mathbf{C}_k$ includes all probability measures of $\mathcal{A}^k_{\mathbf{V}}$ consistent with Kolmogorov's probability axioms, for which additionally the following condition holds. A probability measure $P_{\mathbf{V}}$ is in $\mathbf{P}^k_{\mathbf{V}}$ if for each event $e \in \mathcal{A}^k_{\mathbf{V}}$:

$$\sum_{\substack{d \subseteq e \\ (p,d) \in \mathbf{C}_k}} p \leq P_{\mathbf{V}}(e) \leq \sum_{\substack{d \cap e \neq \emptyset \\ (p,d) \in \mathbf{C}_k}} p$$

The probabilities contributing to the lower bound are related to events which are subsets of e, and therefore certainly have to contribute to the probability of e as well. In contrast, the probabilities contributing to the upper bound relate to events which are not disjoint with e, and therefore can possibly contribute to the probability of e.

Example 14

For illustration we give a two-dimensional specification with two variables with range $\{sun, rain\}$:

$$\mathbf{C}_2 = \Big\{\big(0.2, \{(sun, sun), (sun, rain)\}\big), \big(0.8, \{(sun, sun), (sun, rain), (rain, sun)\}\big)\Big\}$$

Some distributions which are element of the resulting credal set are:

	(sun, sun)	$(sun, rain)$	$(rain, sun)$	$(rain, rain)$
$P^1_{\mathbf{V}}$	1.0	0.0	0.0	0.0
$P^2_{\mathbf{V}}$	0.0	1.0	0.0	0.0
$P^3_{\mathbf{V}}$	0.5	0.5	0.0	0.0
$P^4_{\mathbf{V}}$	0.2	0.3	0.5	0.0
$P^5_{\mathbf{V}}$	0.1	0.1	0.8	0.0
...	...	...	...	...

With a more strict specification we restrict the possible measures further:

$$\mathbf{C}_2 = \Big\{\big(0.2, \{(sun, sun), (sun, rain)\}\big), \big(0.8, \{(sun, sun)\}\big)\Big\}$$

In this case for all i: $P^i_{\mathbf{V}}\big((rain, sun)\big) = P^i_{\mathbf{V}}\big((rain, rain)\big) = 0.0$, $0.0 \leq P^i_{\mathbf{V}}\big((sun, rain)\big) \leq 0.2$ and $P^i_{\mathbf{V}}\big((sun, sun)\big) = 1 - P^i_{\mathbf{V}}\big((sun, rain)\big)$.

Next, we show that a credal set specification $\mathbf{C}$ defines a credal set for the entire, potentially infinite-dimensional, random variable's sample space and show that this set is convex and non-empty. To prove this fundamental property we show all $\mathbf{C}_k$ of $\mathbf{C}$ define a non-empty set of probability spaces over the entire event space $\mathcal{A}_{\mathbf{V}}$ which satisfy Kolmogorov's probability axioms.

Lemma 3. *For each credal set specification $\mathbf{C} = \{\mathbf{C}_1, \mathbf{C}_2, \ldots\}$ there exists a non-empty credal set of probability measures $\mathbf{P}_{\mathbf{V}}$ on the entire space $(\Omega_{\mathbf{V}}, \mathcal{A}_{\mathbf{V}})$, such that all measures $P_{\mathbf{V}}$ in $\mathbf{P}_{\mathbf{V}}$ agree with $(\mathbf{C}_1, \mathbf{C}_2, \ldots)$ in the sense that for all events e and natural numbers k:*

$$\sum_{\substack{d \subseteq e \\ (p,d) \in \mathbf{C}_k}} p \le P_{\mathbf{V}}\big(\pi_k(e)\big) \le \sum_{\substack{d \cap e \neq \emptyset \\ (p,d) \in \mathbf{C}_k}} p$$

We require the additional condition that the event space is chosen such that it is possible to construct an infinite dimensional probability measure from an infinite number of finite ones with increasing dimensionality.

The technical condition which enables the construction of probability measures poses no practical restriction. For instance, the condition is fulfilled in case the event space for the random variables is built from *Borel σ-algebras* of *Polish topological spaces* [30, III.3]. This includes virtually all possible event spaces relevant for practical applications, as all subsets of spaces with discrete topology, such as integers or other sets of countably many constants, and all closed and open intervals with rational bounds in the real numbers.

Moreover, the credal set of probability measures on the random variables can be extended to a credal set on the entire theory, by extending each element of the credal set as shown in Section 4.1.2.

Theorem 1. *Each credal set specification* $\mathbf{C}$ *defines a non-empty credal set of probability measures on the entire theory* $\mathbf{P_T}$ *and a set of corresponding query probability distributions* $\mathbf{P}$.

5.3. Probability bounds

We have shown that a theory defines a potentially infinite set of probability spaces, which means that for a query one can compute a potentially infinite number of probabilities. Instead, we are typically interested in the lower and upper bounds on the probability of a query.

Definition 8 *(Probability bounds).* We define the lower and upper probability bounds of a query q as follows:

$$\underline{P}(q) = \min_{P \in \mathbf{P}} P(q)$$
$$\overline{P}(q) = \max_{P \in \mathbf{P}} P(q)$$

We introduce formulas for computing those bounds for finite-dimensional queries.

Proposition 2. *The lower and upper probability bounds of a query q, fulfilling the finite-dimensional-constraints condition and putting constraints only on the first k dimensions, can be computed by:*

$$\underline{P}(q) = \sum_{\substack{e \subseteq \mathrm{SE}(q) \\ (p,e) \in \mathbf{C}_k}} p$$
$$\overline{P}(q) = \sum_{\substack{e \cap \mathrm{SE}(q) \neq \emptyset \\ (p,e) \in \mathbf{C}_k}} p$$

Example 15

Consider again the clauses of the running example (Section 2.3):

$$saved \leftarrow \langle \mathbf{Time\_Comp}_1 < 0.75 \rangle$$

$$saved \leftarrow \langle \mathbf{Time\_Comp}_1 < 1.25 \rangle, \langle \mathbf{Time\_Comp}_1 + 0.25 \cdot \mathbf{Time\_Comp}_2 < 1.375 \rangle$$

We give a finite credal set specification for the two variables occurring in the clauses. This is done in such a way that it roughly matches the exponential distributions used in the example: $\mathbf{Time\_Comp}_1, \mathbf{Time\_Comp}_2 \sim \mathrm{Exp}(1)$. (How to generate finite credal sets exactly matching continuous distributions in general is discussed in Section 5.6.)

$$\begin{aligned}\mathbf{C}_2 = \Big\{ & \big(0.49, \{(\omega_1, \omega_2) \mid 0 \le \omega_1 \le 1,\ 0 \le \omega_2 \le 1\}\big),\\ & \big(0.14, \{(\omega_1, \omega_2) \mid 1 \le \omega_1 \le 2,\ 0 \le \omega_2 \le 1\}\big),\\ & \big(0.07, \{(\omega_1, \omega_2) \mid 2 \le \omega_1 \le 3,\ 0 \le \omega_2 \le 1\}\big),\\ & \big(0.14, \{(\omega_1, \omega_2) \mid 0 \le \omega_1 \le 1,\ 1 \le \omega_2 \le 2\}\big),\\ & \big(0.04, \{(\omega_1, \omega_2) \mid 1 \le \omega_1 \le 2,\ 1 \le \omega_2 \le 2\}\big),\\ & \big(0.02, \{(\omega_1, \omega_2) \mid 2 \le \omega_1 \le 3,\ 1 \le \omega_2 \le 2\}\big),\end{aligned}$$

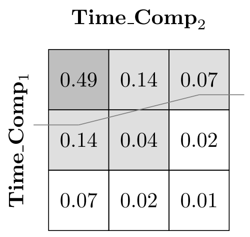

Fig. 2. Two-dimensional sample space with events and solution event.

$$\begin{aligned}&\big(0.07, \{(\omega_1,\omega_2) \mid 0 \le \omega_1 \le 1,\ 2 \le \omega_2 \le 3\}\big),\\ &\big(0.02, \{(\omega_1,\omega_2) \mid 1 \le \omega_1 \le 2,\ 2 \le \omega_2 \le 3\}\big),\\ &\big(0.01, \{(\omega_1,\omega_2) \mid 2 \le \omega_1 \le 3,\ 2 \le \omega_2 \le 3\}\big)\Big\}\end{aligned}$$

The solution event of *saved* is $\{(\omega_1,\omega_2,\ldots) \in \Omega_{\mathbf{V}} \mid \omega_1 < 0.75 \vee (\omega_1 < 1.25 \wedge \omega_1 + 0.25 \cdot \omega_2 < 1.375)\}$, as shown in Example 9. The sample space together with the events defined in the credal set specification is visualised in Fig. 2. The solution event is shown as well, where everything above the line is inside the solution event.

The first event in $\mathbf{C}_2$ – shown in the left upper corner in Fig. 2 – is a subset of the solution event. This intuitively means that no matter how we distribute the probability mass of 0.49 inside the event, all of it will be inside the solution event. Therefore $\underline{P}(saved) = 0.49$. All events represented by grey areas in Fig. 2 are not disjoint with the solution event. This means it is possible to distribute the probability mass in such a way that all of it is inside the solution event. We can conclude $\overline{P}(saved) = 0.49 + 0.14 + 0.07 + 0.14 + 0.04 = 0.88$.

We can finally provide formulas for the probability bounds for the general case that constraints are not finite-dimensional.

Theorem 2. *The lower and upper probability bounds of a query q are determined by:*

$$\underline{P}(q) = \lim_{k\to\infty} \sum_{\substack{e \subseteq \mathrm{SE}(q)\\ (p,e)\in \mathbf{C}_k}} p$$

$$\overline{P}(q) = \lim_{k\to\infty} \sum_{\substack{e \cap \mathrm{SE}(q) \neq \emptyset\\ (p,e)\in \mathbf{C}_k}} p$$

We finally present a result about the relation between lower and upper bound, which follows from this theorem and the sum rule for limits.

Corollary 1. *The lower bound can be expressed in terms of the upper bound and vice versa.*

$$\underline{P}(q) = 1 - \overline{P}(\neg q)$$

$$\overline{P}(q) = 1 - \underline{P}(\neg q)$$

5.4. Dealing with evidence

Making use of evidence is crucial for probabilistic reasoning. Taking evidence into account means to exclude parts of the event space and renormalise the probability measure such that probabilities sum up to one. For a single probability distribution the probability of a query q given evidence e denoted by $P(q \mid e)$ can be expressed in terms of non-conditional probabilities as $P(q \wedge e)/P(e)$. In case of a credal set we have to do this for all corresponding conditional probabilities and find the minimum and maximum.

Definition 9 *(Conditional probability bounds).* We define the probability bounds given evidence as:

$$\underline{P}(q \mid e) \overset{\text{def}}{=} \min_{P \in \mathbf{P}} P(q \mid e)$$

$$\overline{P}(q \mid e) \overset{\text{def}}{=} \max_{P \in \mathbf{P}} P(q \mid e)$$

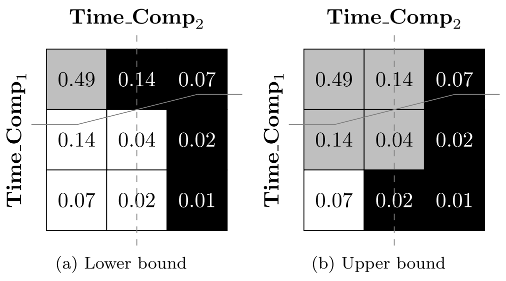

Fig. 3. Two-dimensional sample space with events, solution event (solid) and evidence (dashed).

Note that there are alternative definitions of conditional probabilities for imprecise probability distributions. Weichselberger argues that different notions of conditional probabilities should be used depending on the purpose they are used for [32]. We here restrict to our definition above, which corresponds to what Weichselberger calls the *intuitive concept*.

In contrast to the precise case we cannot define a normalisation factor – also called partition function – which only depends on the evidence, but also have to take the query into account. This is illustrated by the following example.

Example 16

Consider again the clauses of the running example (Section 2.3):

$$saved \leftarrow \langle \mathbf{Time\_Comp}_1 < 0.75 \rangle$$

$$saved \leftarrow \langle \mathbf{Time\_Comp}_1 < 1.25 \rangle, \langle \mathbf{Time\_Comp}_1 + 0.25 \cdot \mathbf{Time\_Comp}_2 < 1.375 \rangle$$

Consider an additional rule:

$$e \leftarrow \langle \mathbf{Time\_Comp}_2 < 1.5 \rangle$$

Suppose we would like to compute $\overline{P}(saved \mid e)$. This is illustrated in Fig. 3, where the events represented by black areas are excluded from the event space. All events in the right column certainly have to be excluded, since they are disjoint with the evidence. It depends on the probability measure chosen from the credal set whether to exclude or not the events in the middle column. The choice depends on whether we want to compute the lower or upper bound as is illustrated by the figure.

The probability mass of the upper middle event can only be included in the partition function (within e) if also completely within *saved*. So excluding it minimises the probability. The remaining events in the middle column would only contribute to the partition function of the lower bound. Excluding them would increase the result. The events are consequently not excluded for computing the lower bound, which is $\underline{P}(saved \mid e) = 0.49/(0.49 + 0.14 + 0.04 + 0.07 + 0.02) \approx 0.64$.

For the upper bound events contributing to the numerator and denominator of the probability have to be included, but events only contributing to the denominator have to be excluded, in order to obtain the maximal probability. The upper bound is thus $\overline{P}(saved \mid e) = (0.49 + 0.14 + 0.14 + 0.04)/(0.49 + 0.14 + 0.14 + 0.04 + 0.07) \approx 0.92$.

To compute conditional probabilities we have the following proposition that relates the joint probability to the conditional probability.

Proposition 3. *The probability bounds of a query q given evidence e, as defined by Definition 9, can be computed as follows:*

$$\underline{P}(q \mid e) = \frac{\underline{P}(q \wedge e)}{\underline{P}(q \wedge e) + \overline{P}(\neg q \wedge e)}$$

$$\overline{P}(q \mid e) = \frac{\overline{P}(q \wedge e)}{\overline{P}(q \wedge e) + \underline{P}(\neg q \wedge e)}$$

5.5. Exact inference conditions

We introduce a variant of the exact inference conditions (Proposition 1) concerning probability bounds instead of precise probabilities.

Theorem 3. *The probability bounds of an arbitrary query can be computed in finite time if the following conditions hold:*

1. ***finite-relevant-constraints condition****: There are only finitely many constraint predicates (⟨⟩) relevant for determining truthfulness of each query atom and finding such relevant constraints predicates and entailment checking can be done in finite time. The condition is the same in Proposition 1.*
2. ***finite-dimensional-constraints condition****: Events occurring in clauses as argument of ⟨⟩ only concern a finitely-dimensional subset of the sample space. The condition is the same in Proposition 1.*

3. ***disjoint-events-decidability condition****: For each two finite-dimensional events e_1 and e_2 in the event space $\mathcal{A}_{\mathbf{V}}$ one can decide whether they are disjoint or not ($e_1 \cap e_2 = \emptyset$).*

Note that the disjoint-events-decidability condition means we can also decide whether one event is a subset of another, since $e_1 \subseteq e_2$ is equivalent to $e_1 \cap (\Omega_{\mathbf{V}} \setminus e_2) = \emptyset$.

In the above condition we replaced the – in our view – too strict computable-measure condition with the disjoint-events-decidability condition. It is fulfilled for a wide class of possible ways to define events, e.g. linear inequalities on integers (excluding multiplication) [33] and inequalities on real numbers including multiplication [34]. Although we may not be able to define arbitrary distributions, we can always define a set of distributions which includes any such distribution. For instance, queries of programs consisting of linear constraints on real numbers distributed according to arbitrary continuous distributions can be approximated with known maximal error, as will be shown next. Additionally, it is possible to define imprecise distributions in case not enough knowledge is available about what the actual distribution is.

Example 17

Assume we have random variables $\mathbf{V}_1$ and $\mathbf{V}_2$ with the range of real numbers and the corresponding credal set specification $\{(0.25, \omega_1 < 0 \wedge \omega_2 < 0), (0.25, \omega_1 < 0 \wedge \omega_2 > 0), (0.25, \omega_1 > 0 \wedge \omega_2 < 0), (0.25, \omega_1 > 0 \wedge \omega_2 > 0)\}$. Here we use a shorthand notation for events, only denoting the condition on elements of the sample space. This credal set specification for instance includes the case that ω_1 and ω_2 are independent and normally distributed with mean 0.0 and arbitrary standard deviation.

Suppose we want to compute the probability of the event $2\omega_1 > \omega_2$. We can observe the following statements, since linear constraints on real-valued variables are decidable (disjoint-events-decidability condition):

$$(2\omega_1 > \omega_2) \not\supseteq (\omega_1 < 0 \wedge \omega_2 < 0)$$

$$(2\omega_1 > \omega_2) \not\supseteq (\omega_1 < 0 \wedge \omega_2 > 0)$$

$$(2\omega_1 > \omega_2) \supseteq (\omega_1 > 0 \wedge \omega_2 < 0)$$

$$(2\omega_1 > \omega_2) \not\supseteq (\omega_1 > 0 \wedge \omega_2 > 0)$$

From this we can compute $\underline{P}(2\omega_1 > \omega_2) = 0.25$. To determine the upper bound we observe the following:

$$(2\omega_1 > \omega_2) \cap (\omega_1 < 0 \wedge \omega_2 < 0) \neq \emptyset$$

$$(2\omega_1 > \omega_2) \cap (\omega_1 < 0 \wedge \omega_2 > 0) = \emptyset$$

$$(2\omega_1 > \omega_2) \cap (\omega_1 > 0 \wedge \omega_2 < 0) \neq \emptyset$$

$$(2\omega_1 > \omega_2) \cap (\omega_1 > 0 \wedge \omega_2 > 0) \neq \emptyset$$

From this we can compute $\overline{P}(2\omega_1 > \omega_2) = 0.75$.

5.6. *Approximating continuous distributions using credal sets*

The examples in Section 2 contain continuous distributions, which would translate to infinite credal set specifications, for which therefore exact inference would not be possible. Credal sets can however be used to approximate combinations of arbitrary continuous distributions with known *cumulative distribution function* (CDF). This is done by associating probabilities to intervals, defining a set of distributions including the actually intended one. To do that one can divide the sample space of a single variable $\mathbf{V}_i$ in n intervals $(l_1, u_1), \dots, (l_n, u_n)$ where for all j: $l_j \leq u_j$, l_j may be a real number or $-\infty$ and u_i may be a real number or ∞. We can now define the following one-dimensional credal set: $\{P(l_1 < \mathbf{V}_i < u_1) : l_1 < \mathbf{V}_i < u_1, \dots, P(l_n < \mathbf{V}_i < u_n) : l_n < \mathbf{V}_i < u_n\}$. To compute probabilities of the integrals, we make use of the fact that one-dimensional integrals over typical density functions can be computed with negligible error, for example for normal distributions such integrals can be computed using CDFs using the error function. The credal sets of independent random variables can be combined by taking the product of the credal sets, which means using all combinations of elements, taking the product of probabilities and the intersection of events. The same technique is also applicable for multivariate distributions by using for instance in two dimensions rectangles instead of intervals.

Providing a specification with more intervals restricts the possible distributions more, which means one can get a credal set arbitrarily close to an arbitrary single distribution. However, since the credal set specification has to be finite, it generally cannot be restricted to an arbitrary single distribution. The probability bounds of each query can be used to determine the maximum error of the approximation. The number of intervals determine the precision, i.e. the gap between the probability bounds one can compute. So the precision can be increased arbitrarily. Fig. 4 gives an example of a Gaussian distribution divided into five intervals with equal probability.

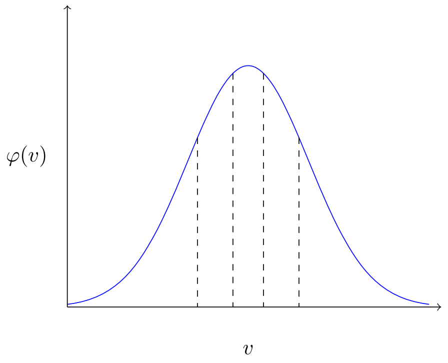

Fig. 4. Discretised PDF of a Gaussian distribution; $\varphi(v)$ is the density for the value v; the dashed lines indicate the intervals the distribution is discretised into.

6. A new probabilistic constraint logic programming language

After introducing an abstract semantics for probabilistic logic programs, this section deals with constructing an actual language, of which examples were given in Section 2. The structure of the rules is already fixed by the semantics; the issue to be solved is how to represent a countable collection of credal set specifications such that they are compatible in a concrete language. In summary, we will define credal sets for independent sets of random variables and represent events using constraints, similar to constraints occurring in rules. The resulting language is called *probabilistic constraint logic programming* (PCLP).

In the following, we will discuss constraint theories in context of PCLP. Then we will give the syntax of the language, which we will link to the semantics discussed. Finally, we give discuss inference tasks that can be performed using PCLP.

6.1. Constraint theories

As mentioned, PCLP can be based on arbitrary constraint theories under mild restrictions. First, the constraints must correspond to a subset of an event space fulfilling the requirement of Lemma 3, which is that infinite dimensional probability measures can be constructed from an infinite number of finite ones with increasing dimensionality. As discussed, a large class of practically useful constraints, such as inequalities on integers or real numbers, fulfil that requirement. Further, satisfiability of constraints must be decidable, which corresponds to deciding disjointness of events, as will be shown later (Section 6.3). The requirement therefore means that the disjoint-events-decidability condition holds. It is possible to use so-called incomplete solvers as well, but then exact bounds cannot be computed, as we will discuss later.

We refer to an instance of PCLP based on constraint language **Constr** as PCLP(**Constr**). In this article we make use of two constraint theories which were already used in the examples before: real numbers (R) and discrete constants (D). In the examples of Section 2 and further in this article we use a combination and refer to the language as PCLP(D, R). The combination of different theories is realised by assuming that a constraint theory is attached to each variable. Statements about variables can only be made using the constraint language of the corresponding constraint theory attached, for example real-valued random variables cannot be compared to discrete ones. Constraints of different theories can only be combined using logical connectives.

In the constraint theory D random variables take values of discrete constants. The basic building blocks of the language are set membership ($\in$), its negation ($\notin$) and their special cases equality ($=$) and inequality ($\neq$). The constraint theory is very similar to the *finite domain* constraint theory of CLP(FD) [35], with the difference that there is no requirement to explicitly define the range. The range of all variables are a countably-infinite number of constants, but one can effectively restrict the range by assigning probability mass to only a finite number of constants. Using D, one can represent, for instance, Bayesian networks [10], but also first-order formalisms such as CP-logic [9].

The R theory is basically the same as in CLP(R) [36]. Variables represent real numbers and constraints consist of linear equalities and inequalities using predicates such as $\{=, \neq, <, >, \leq, \geq\}$ and functions $\{+, -, \cdot\}$. One argument of the multiplication operator must be a constant to make constraints linear and guarantee decidability. This theory can be used to approximate a large class of continuous distributions, as shown in Section 5.6.

6.2. Syntax

An overview of the PCLP syntax is given in Table 2. Rules are built according to the LP syntax with additional $\langle\rangle$-constructs within which constraints are defined using the syntax of the used constraint languages, as discussed previously. New here is the definition of credal set specifications for a countable number of variables in the sense of Section 5.2. We define credal sets by a number of *random variable definitions*. Random variable definitions define credal sets for groups

Table 2
PCLP syntax overview.

Concept	Syntax
Random variable definition	$(\mathbf{V}_1,\dots,\mathbf{V}_n)(t_1,\dots,t_m)\sim\{p_1{:}\,\varphi_1,\dots,p_l{:}\,\varphi_l\}$
Rule	$a \leftarrow b_1,\dots,b_n$
Constraint φ_i	element of **Constr** (e.g. $\mathbf{V}\in\{a,b\}$, $\mathbf{V}\le\mathbf{W}$)
Body element b_i	a_i or $not(a_i)$ or $\langle\varphi_i\rangle$
Atom a_i	$d(t_1,\dots,t_n)$
Predicate d	some predicate name (starting with lower case)
Term t_i	some Prolog term (e.g. a, 3, $a(b,3)$), may contain logical variables (e.g. X, $a(X)$)
Logical variable X_i	some variable name (starting with upper case)
Random variable $\mathbf{V}_i$	some random variable name (starting with upper case, bold)
Probability p_i	$[0.0, 1.0]$

of finitely many random variables, such that the definition of each random variable depends only on a finite number of other ones. Definitions have the form:

$$(\mathbf{V}_1,\dots,\mathbf{V}_n)(X_1,\dots,X_m)\sim\{p_1{:}\,\varphi_1,\dots,p_l{:}\,\varphi_l\}$$

where each $\mathbf{V}_i$ is a random variable label, X_i is a parameter, p_i is a probability and φ_i is a constraint.

The random variable definitions define random variables

$$\mathbf{V}_1(X_1,\dots,X_m),\dots,\mathbf{V}_n(X_1,\dots,X_m)$$

for all groundings of $X_1,\dots,X_m$. All labels have the same parameters to make sure all X_i are ground in case a single ground instance of one of the defined random variables is used in the program. There must be at least one random variable label in the list and all labels must be distinct. If there is only one, the brackets may be left out.

Example 18

A single random variable could for instance be defined as:

$$\mathbf{Temperature}\sim\dots$$

We can also define random variables representing the temperature of all infinitely many future days as:

$$\mathbf{Temperature}(Day)\sim\dots$$

We can finally define the temperature and humidity together, which makes sense in case they are modelled by a multivariate distribution, i.e. the dependency between the random variables cannot be expressed by the logical structure:

$$(\mathbf{Temperature},\mathbf{Humidity})(Day)\sim\dots$$

In case multiple definitions define the same random variable, the definition occurring first in the program defines the variable. Labels occurring together in a definition can only occur in another definition in an identical list of labels, to make sure all random variables are always defined unambiguously.

Example 19

Suppose we want to specify distributions for temperature on future days, but want to make an exception for Day 0. We could do this as follows:

$$\mathbf{Temperature}(0)\sim\dots$$

$$\mathbf{Temperature}(Day)\sim\dots$$

The second line would define **Temperature**(0) as well, but it is not used in this case because the first line occurs first.

The following definition is invalid:

$$\mathbf{Temperature}(Day)\sim\dots$$

$$(\mathbf{Temperature},\mathbf{Humidity})(Day)\sim\dots$$

Temperature is already defined by the first line; the definition in the second line may contradict that definition.

A random variable definition $\{p_1{:}\,\varphi_1,\dots,p_l{:}\,\varphi_l\}$ represents a credal set specification defining an n-dimensional distribution on $\mathbf{V}_1(X_1,\dots,X_m),\dots,\mathbf{V}_n(X_1,\dots,X_m)$. Each p_i is a real number between 0 and 1 and $p_1+\dots+p_l=1$. The φ_i are constraint definitions using the $\mathbf{V}_i$ as placeholder for the random variables. Each φ_i must be consistent, to make sure all probability mass is assigned to non-empty events. The p_i and φ_i can be expressions using $X_1,\dots,X_m$.

Example 20

Consider the following valid definition:

$$\mathbf{Temperature}(Day) \sim \{0.2\colon \mathbf{Temperature} < 0, 0.8\colon \mathbf{Temperature} > 0\}$$

The definition could also depend on the value of the logical variables *Day*:

$$\mathbf{Temperature}(Day) \sim \{0.2\colon \mathbf{Temperature} < Day/1000, 0.8\colon \mathbf{Temperature} > Day/1000\}$$

The definition is only valid if groundings for *Day* are restricted to numbers. It actually expresses that the temperature in the future will increase on average.

Note that in the definitions in the examples (Section 2) we use syntactic sugar. For example consider this definition from the examples:

$$\mathbf{DM\_if\_HighRisk} \sim \{0.266\colon yes, 0.664\colon no\}$$

The idea of those kinds of definitions is that they leave part of the probability mass out to express imprecision. The above definition is actually an abbreviation for:

$$\mathbf{DM\_if\_HighRisk} \sim \{ \begin{array}{l} 0.266\colon \mathbf{DM\_if\_HighRisk} = yes, \\ 0.664\colon \mathbf{DM\_if\_HighRisk} = no, \\ 0.07\ \colon \mathbf{DM\_if\_HighRisk} \in \{yes, no\} \end{array} \}$$

Definitions can also be continuous distributions, like $\mathcal{N}(0.0, 1.0)$. This can be seen as infinite definitions. In such a case, only approximation of the result by finite definitions is possible, as discussed in Section 5.6.

6.3. Semantics

We start showing that a PCLP program defines a countable sequence of finite credal set specifications $\mathbf{C}_1, \ldots$ with the length of the number of random variables and therefore defines a credal set specification in the sense of Definition 6.

Definition 10 *(Credal set specification of a PCLP program).* We associate a single random variable definition with each random variable, which is the first matching definition occurring in the program. If there is no matching definition we associate the specification $\{1.0\colon true\}$. We fix the enumeration of random variables and denote the family of the definitions for the first n random variables as $\mathbf{D}_n$. From this we define the credal set specifications of the program $\mathbf{C}_n$ as follows:

$$\mathbf{C}_n = \pi_n\Big(\big\{(p, \mathrm{CSS}(\varphi)) \mid (p, \varphi) \in \hat{\prod}_{d \in \mathbf{D}_n} d\big\}\Big)$$

Here the product of two random variable definitions $d_1 \hat{\times} d_2$ is defined as:

$$d \hat{\times} e \overset{\mathrm{def}}{=} \big\{(p_d \cdot p_e, \varphi_d \wedge \varphi_e) \mid \big((p_d, \varphi_d), (p_e, \varphi_e)\big) \in d \times e\big\}$$

We take the product of the definitions, since we deal with independent probabilities, and map the constraints to events by making use of their solution space (CSS). The projection to n dimensions (π_n) is necessary, because random variables not in the first n are possibly included in case they are defined together in the same definition with one of the first n.

Example 21

In Example 15 the two-dimensional credal set specification of the variables $\mathbf{Time\_Comp}_1$ and $\mathbf{Time\_Comp}_2$ was given as:

$$\begin{aligned} \mathbf{C}_2 = \Big\{ & \big(0.49, \{(\omega_1, \omega_2) \mid 0 \le \omega_1 \le 1,\ 0 \le \omega_2 \le 1\}\big), \\ & \big(0.14, \{(\omega_1, \omega_2) \mid 1 \le \omega_1 \le 2,\ 0 \le \omega_2 \le 1\}\big), \\ & \big(0.07, \{(\omega_1, \omega_2) \mid 2 \le \omega_1 \le 3,\ 0 \le \omega_2 \le 1\}\big), \\ & \big(0.14, \{(\omega_1, \omega_2) \mid 0 \le \omega_1 \le 1,\ 1 \le \omega_2 \le 2\}\big), \\ & \big(0.04, \{(\omega_1, \omega_2) \mid 1 \le \omega_1 \le 2,\ 1 \le \omega_2 \le 2\}\big), \\ & \big(0.02, \{(\omega_1, \omega_2) \mid 2 \le \omega_1 \le 3,\ 1 \le \omega_2 \le 2\}\big), \\ & \big(0.07, \{(\omega_1, \omega_2) \mid 0 \le \omega_1 \le 1,\ 2 \le \omega_2 \le 3\}\big), \\ & \big(0.02, \{(\omega_1, \omega_2) \mid 1 \le \omega_1 \le 2,\ 2 \le \omega_2 \le 3\}\big), \\ & \big(0.01, \{(\omega_1, \omega_2) \mid 2 \le \omega_1 \le 3,\ 2 \le \omega_2 \le 3\}\big) \Big\} \end{aligned}$$

This credal set specification can be represented in PCLP as:

$$\begin{aligned}\mathbf{Time\_Comp}_1 \sim \{ \quad & 0.7{:}\, 0 \le \mathbf{Time\_Comp}_1 \le 1, 0.2{:}\, 1 \le \mathbf{Time\_Comp}_1 \le 2,\\ & 0.1{:}\, 2 \le \mathbf{Time\_Comp}_1 \le 3 \quad \}\\ \mathbf{Time\_Comp}_2 \sim \{ \quad & 0.7{:}\, 0 \le \mathbf{Time\_Comp}_2 \le 1, 0.2{:}\, 1 \le \mathbf{Time\_Comp}_2 \le 2,\\ & 0.1{:}\, 2 \le \mathbf{Time\_Comp}_2 \le 3 \quad \}\end{aligned}$$

Lemma 4. *A PCLP program defines a credal set over all random variables* $\mathbf{V}_1, \mathbf{V}_2, \ldots$ *occurring in it.*

To derive formulas for the probability bounds of a PCLP program, we first introduce the concept of *solution constraint*, which is the equivalent of the solution event expressed in terms of constraints (cf. Lemma 2).

Definition 11 *(Solution constraint).* The solution constraint of a query q is defined as:

$$\mathrm{SC}(q) \overset{\text{def}}{=} \bigvee_{\substack{M_{\mathbf{L}}[\Phi] \models q\\ \Phi \subseteq \mathbf{Constr}}} \bigwedge_{\varphi \in \Phi} \varphi$$

Example 22

Consider again the clauses of the running example (Section 2.3):

$$saved \leftarrow \langle \mathbf{Time\_Comp}_1 < 0.75 \rangle$$

$$saved \leftarrow \langle \mathbf{Time\_Comp}_1 < 1.25 \rangle, \langle \mathbf{Time\_Comp}_1 + 0.25 \cdot \mathbf{Time\_Comp}_2 < 1.375 \rangle$$

The solution constraint for query *saved* is:

$$\begin{aligned}\mathrm{SC}(saved) = {} & \mathbf{Time\_Comp}_1 < 0.75\\ & \vee (\mathbf{Time\_Comp}_1 < 1.25 \wedge \mathbf{Time\_Comp}_1 + 0.25 \cdot \mathbf{Time\_Comp}_2 < 1.375)\end{aligned}$$

The way to compute probabilities according to Proposition 2 requires deciding disjointness of events. We substitute this by deciding satisfiability of constraints. For this we introduce a function for checking satisfiability of constraints, which has to be filled in by a constraint checker for a concrete implementation. We also consider only partially decidable constraint theories. Non-linear constraints can for instance often be solved, but not in general.

Definition 12 *(Satisfiability check).* The function

$$check \colon \mathbf{Constr} \to \{sat, unsat, unknown\}$$

checks satisfiability of constraints. The possible values of the function have the following meaning:

sat:	The constraint is certainly satisfiable, i.e. there is a solution.
unsat:	The constraint is certainly unsatisfiable, i.e. there is no solution.
unknown:	Satisfiability could not be decided, i.e. nothing is said about the constraint.

In case the function never yields *unknown* for any constraint, we call the constraint theory *fully decidable*.

We can now express probability bounds of a query in a PCLP program using the satisfiability check function.

Proposition 4. *The lower and upper probability bounds of a query q, concerning only the first n random variables, in a PCLP program, fulfilling the exact inference conditions, can be computed as:*

$$\underline{P}(q) = \sum_{\substack{check(\varphi \wedge \neg \mathrm{SC}(q)) = unsat\\ (p,\varphi) \in \mathbf{C}_n}} p$$

$$\overline{P}(q) = \sum_{\substack{check(\varphi \wedge \mathrm{SC}(q)) = sat\\ (p,\varphi) \in \mathbf{C}_n}} p$$

Table 3
Overview of inference tasks.

	Precise: P(q)	Imprecise: $\underline{P}(q)$, $\overline{P}(q)$
Discrete	$P(q)$	$\underline{P}(q)$, $\overline{P}(q)$
Hybrid	$P(q) \pm \epsilon$ known, arbitrary ϵ	$\underline{P}(q) - \underline{\epsilon}$, $\overline{P}(q) + \overline{\epsilon}$ unknown, arbitrary $\underline{\epsilon}$, $\overline{\epsilon}$
Partially decidable constraints	$P(q) \pm \epsilon$ known, bounded ϵ	$\underline{P}(q) - \underline{\epsilon}$, $\overline{P}(q) + \overline{\epsilon}$ unknown, bounded $\underline{\epsilon}$, $\overline{\epsilon}$

Here **C** *is the credal set specification defined by the program (Lemma 4).*

Corollary 2. *For PCLP programs making use of partially decidable constraints, the following holds:*

$$\underline{P}(q) \geq \sum_{\substack{check(\varphi \wedge \neg SC(q)) = unsat \\ (p,\varphi) \in \mathbf{C}_n}} p$$

$$\overline{P}(q) \leq \sum_{\substack{check(\varphi \wedge SC(q)) \neq unsat \\ (p,\varphi) \in \mathbf{C}_n}} p$$

Corollary 3. *Exact inference in PCLP(***Constr***) is possible for queries with a finite number (finite-relevant-constraints condition) of finite-dimensional constraints (finite-dimensional-constraints condition) and in case satisfiability can be decided for the constraint language* **Constr***.*

6.4. Inference tasks

PCLP can be used for a number of inference tasks as summarised by Table 3. In case we just have discrete distributions we can compute the exact bounds of a query, which is a point probability for precise distributions, using Proposition 4.

In the hybrid case we are able to compute bounds using Proposition 4 as well. Such bounds are only approximations, since the random variable definitions we use are only approximations of the actual continuous distributions. For the precise case we know the bounds approximate a point probability and therefore know the maximal error of the approximation. In the imprecise case the difference between the lower and upper bound is partially explained by approximated continuous distributions and partially by the imprecision the distributions have anyway. We therefore do not know how good the approximation is. In both cases the approximation asymptotically equals the actual result in case of infinite computation time.

Finally, in case we have a partially decidable constraint theory, we can still determine approximations using Corollary 2. The achievable precision may however be bounded by constraints which are not decidable.

7. Inference by generalised weighted model counting

We have shown that under certain conditions we can perform exact inference for PCLP in two steps: determining the solution constraint (Definition 11) and computing its probability using Proposition 4. Both steps have exponential time complexity if done in a naive way. The solution constraint can be determined by considering all subsets of constraints in the support set of the query. Then, probabilities can be computed by summing over all possible choices of which there are exponentially many in the number of elements in independent random variable definitions.

Inference can often be done more efficiently by making use of the problem's structure. We show how this can be done for the first step (Section 7.1) and develop a novel algorithm for the second step (Section 7.2). We consider the theoretical properties in terms of complexity of the proposed algorithm as key result for showing its potential and discuss those results in Section 7.3. We further experimentally evaluate our theoretical claims in Section 7.4.

7.1. Determining the solution constraint

We only briefly describe how we derive the solution constraints, since it is a variation of existing techniques. The algorithm is based on SLD resolution, which is a well-known way to derive proofs of queries in LP. We deal with constraints in bodies the same way as the operational semantics of CLP does, i.e., we collect all constraints we encounter during an SLD derivation [37]. A derivation proves the query in case all collected constraints are true, i.e. if the conjunction of these constraints is true. This means a query can be derived if the conjunction of constraints derived from at least one of the derivations is true. The solution constraint is therefore the disjunction of those conjunctions, which relates to the definition of solution constraints (Definition 11). The main advantage of SLD resolution is that it restricts the subsets of constraints that need to be considered to only those that prove the query. Note that this coincides with the way that proofs are collected in the first implementation of ProbLog [6], where the constraints are probabilistic facts rather than general constraints.

Negation can be added in a straightforward way, but simple SLD resolution is not sufficient in case cycles are present. There are however solutions to the problem: either by translating cyclic rules to acyclic ones [38] or using tabled SLD resolution [39]. We do not go into detail here and assume it is possible to efficiently derive the solution constraint for each query. Possible optimisations applied for CLP, such as pruning derivations if the imposed constraints become inconsistent, are not discussed either.

7.2. Computing probabilities

The proposed algorithm for computing the probability of a solution constraint, is a generalisation of probabilistic inference by WMC. We first discuss this way of inference before introducing our generalisation.

7.2.1. Probabilistic inference by weighted model counting

Real problems often possess a lot of structure one can make use of to speed up inference. Various inference methods have been developed to exploit certain kinds of structure. Examples are *variable elimination* [40] and *recursive conditioning* [41]. We focus on performing probabilistic inference by translation to a WMC problem. This has been shown to be an efficient inference method for propositional formalisms such as *Bayesian networks* [18] and as well to be applicable to probabilistic logics based on distributions semantics [19]. The approach exploits not only topological structure, but also local structure such as *determinism* [42] and *context-specific independence* [43].

The problem of model counting basically means to find the number of models of a propositional knowledge base. WMC is a straightforward generalisation of the problem where each model has a weight. Those weights are defined in terms of weights attached to the different literals. The weight of a model is the product of the weights of all literals included in the model.

Efficient weighted model counting algorithms are based on the observation that counts of components sharing no variables can be computed independently [44]. Model counting then proceeds as follows. We assume the theory is expressed in *conjunctive normal form* (CNF). In case the disjunctions in the CNF can be split such that they share no variables, weights can be computed independently and the weight of the entire CNF is the product of those weights; this step is referred to as *decomposition*. Otherwise a case distinction has to be made for a single variable. Then one gets two CNFs: one with the assumption this variable is false and another one with the assumption it is true. The weight is then computed as the sum of the weights of both CNFs, which represent theories with disjoint models. It can happen that due to the structure of the problem, such as determinism, this choice leads to a theory which can be simplified and potentially includes fewer variables or even becomes *true* or *false*, which means determinism in the problem can be exploited. The choice of variable order is essential for the efficiency of the algorithm. Different possible heuristics for this choice are for instance discussed by Sang et al. [45, Section 3.2].

Example 23

In this example we illustrate the basic idea of binary model counting. The example will later be used to compare with the generalised WMC algorithm. We only show splitting and omit decomposition as the latter is identical in the generalised version.

We start with the solution constraint of Example 22 and convert it to CNF:

$$\begin{aligned}\mathrm{SC}(\mathit{saved}) &= \mathbf{Time\_Comp}_1 < 0.75\\ &\qquad \vee (\mathbf{Time\_Comp}_1 < 1.25 \wedge \mathbf{Time\_Comp}_1 + 0.25\cdot\mathbf{Time\_Comp}_2 < 1.375)\\ &= (\mathbf{Time\_Comp}_1 < 0.75 \vee \mathbf{Time\_Comp}_1 < 1.25)\\ &\qquad \wedge (\mathbf{Time\_Comp}_1 < 0.75 \vee \mathbf{Time\_Comp}_1 + 0.25\cdot\mathbf{Time\_Comp}_2 < 1.375)\end{aligned}$$

In the following, we abbreviate the primitive constraints as follows:

$$\begin{aligned}\varphi_1 &= \mathbf{Time\_Comp}_1 < 0.75\\ \varphi_2 &= \mathbf{Time\_Comp}_1 < 1.25\\ \varphi_3 &= \mathbf{Time\_Comp}_1 + 0.25\cdot\mathbf{Time\_Comp}_2 < 1.375\end{aligned}$$

Weighted model counting proceeds by choosing a variable to split on, which means to continue with two branches with the assumption the variable is true and false respectively. This is illustrated in Fig. 5.

Note that assuming the truth values of literals makes it always possible to immediately simplify the constraint, which is not the case in the generalised version as shown later. The computation stops in case *true* or *false* is derived.

To illustrate the binary version in comparison with the generalised one, we assume further in this example that φ_1, φ_2 and φ_3 represent independent binary random variables with probabilities 0.1, 0.2 and 0.3 respectively. To compute probabilities we put the probability corresponding to the choice at each edge and replace *true* by 1.0 and *false* by 0.0, as shown in Fig. 6. The probabilities of the remaining nodes are computed bottom-up by taking the sum of the probability associated to the children weighted by the probability associated to the edges. Then, the probability of the root node is the probability of the query q.

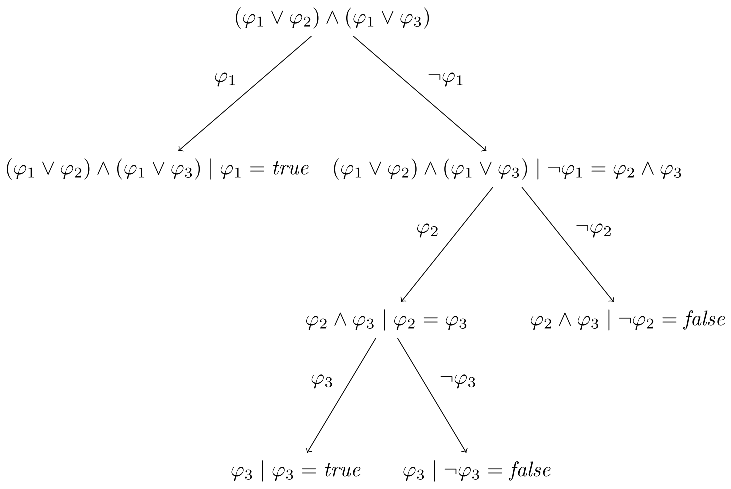
$(\varphi_1 \vee \varphi_2) \wedge (\varphi_1 \vee \varphi_3)$
φ_1
$\neg\varphi_1$
$(\varphi_1 \vee \varphi_2) \wedge (\varphi_1 \vee \varphi_3) \mid \varphi_1 = true$
$(\varphi_1 \vee \varphi_2) \wedge (\varphi_1 \vee \varphi_3) \mid \neg\varphi_1 = \varphi_2 \wedge \varphi_3$
φ_2
$\neg\varphi_2$
$\varphi_2 \wedge \varphi_3 \mid \varphi_2 = \varphi_3$
$\varphi_2 \wedge \varphi_3 \mid \neg\varphi_2 = false$
φ_3
$\neg\varphi_3$
$\varphi_3 \mid \varphi_3 = true$
$\varphi_3 \mid \neg\varphi_3 = false$

Fig. 5. Binary WMC example (deriving possible models).

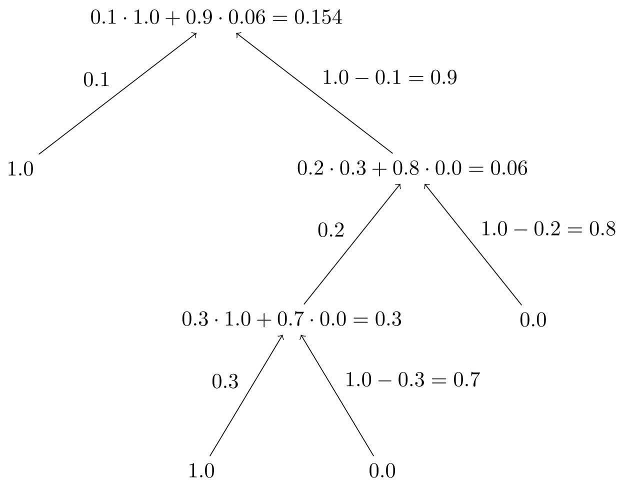
$0.1 \cdot 1.0 + 0.9 \cdot 0.06 = 0.154$
0.1
$1.0 - 0.1 = 0.9$
1.0
$0.2 \cdot 0.3 + 0.8 \cdot 0.0 = 0.06$
0.2
$1.0 - 0.2 = 0.8$
$0.3 \cdot 1.0 + 0.7 \cdot 0.0 = 0.3$
0.0
0.3
$1.0 - 0.3 = 0.7$
1.0
0.0

Fig. 6. Binary WMC example (computing probabilities).

7.2.2. Generalised weighted model counting involving constraints

For the general case, we no longer restrict to binary variables, which makes deciding whether constraints have solutions more complex. Rather we employ so-called satisfiability modulo theories solvers. The *satisfiability modulo theories* (SMT) problem is the problem of deciding whether a given FOL theory including additional background theories is satisfiable. *SMT solver* technology can be used to efficiency decide whether two events are disjoint or not, i.e. whether the conjunction of the constraints representing them is satisfiable or not. Very efficient SMT solvers for e.g. linear arithmetic are currently available, e.g. [46,47]. Since we use those solvers without modifications, we do not discuss the algorithms used to decide satisfiability.

An SMT solver is used as an implementation of the function *check* (Definition 12). Even if the constraint theory is theoretically decidable, in practice a decision procedure might not be implemented. We refer to solvers which cannot decide satisfiability for all instances, i.e. for some cases return *unknown*, as *incomplete solvers*. Incomplete solvers have the same consequences for inference as discussed for partially undecidable constraint theories in Section 6.4.

As a way to implement elements of credal set specifications we, instead, make use of *choices* for the random variables, which are defined as follows.

Definition 13 *(Choice).* A *choice* ψ is a function which selects for each random variable $\mathbf{V}_i$ a probability-constraint pair from its definition.

Without loss of generality, we can restrict ourselves to those random variables that actually occur in the solution constraint of a query, as all other random variables have no influence on the probability of that query.

Algorithm 1: Generalised weighted model counting algorithm (GWMC).

```
   Input: Constraint $\varphi$ in CNF and partial choice constraint $\Psi$
   Result: $(\underline{P}(\varphi \mid \Psi), \overline{P}(\varphi \mid \Psi))$
1  simplify $\varphi$ assuming $\Psi$
2  if $\varphi = false$ then return $(0.0, 0.0)$
3  else if $\varphi = true$ then return $(1.0, 1.0)$
4  else if there are independent CNFs $\varphi_1, \varphi_2$ in $\varphi$ then
5      return GWMC$(\varphi_1, \Psi) \cdot$ GWMC$(\varphi_2, \Psi)$
6  else if there is some random variable $\mathbf{V}_i$ in $\varphi$, not occurring in $\Psi$ then
7      return $\sum_{\psi(\mathbf{V}_i)=(p,\psi)} p \cdot$ GWMC$(\varphi, \Psi \wedge \psi)$
8  else return $(0.0, 1.0)$
```

We generalise the two steps of WMC, which are case distinction and decomposition. The resulting algorithm (Algorithm 1) can be used to compute the bounds of a query's q probability as GWMC(SC(q), *true*) and is discussed below. In case the SMT solver is incomplete we get bounds as given in Corollary 2.

Case distinction We first discuss how the process of distinguishing the cases that atoms are true or false is generalised. Instead of only two cases we have to consider all possible choices for the random variable at each level, as illustrated by the following example.

Example 24

We use the same solution constraint as in Example 23:

$$\begin{aligned}\mathrm{SC}(saved) = &(\mathbf{Time\_Comp}_1 < 0.75 \vee \mathbf{Time\_Comp}_1 < 1.25)\\ &\wedge (\mathbf{Time\_Comp}_1 < 0.75 \vee \mathbf{Time\_Comp}_1 + 0.25 \cdot \mathbf{Time\_Comp}_2 < 1.375)\end{aligned}$$

$$\varphi_1 = \mathbf{Time\_Comp}_1 < 0.75$$

$$\varphi_2 = \mathbf{Time\_Comp}_1 < 1.25$$

$$\varphi_3 = \mathbf{Time\_Comp}_1 + 0.25 \cdot \mathbf{Time\_Comp}_2 < 1.375$$

This time we treat φ_1, φ_2 and φ_3 as constraints instead of binary literals, which has as consequence that we cannot split on the two cases of the literal being true and false, but have to consider the choices given by the random variable definitions. We use the random variable definitions of Example 21:

$$\begin{aligned}\mathbf{Time\_Comp}_1 \sim \{ \quad &0.7{:}\, 0 \le \mathbf{Time\_Comp}_1 \le 1, 0.2{:}\, 1 \le \mathbf{Time\_Comp}_1 \le 2,\\ &0.1{:}\, 2 \le \mathbf{Time\_Comp}_1 \le 3 \quad \}\\ \mathbf{Time\_Comp}_2 \sim \{ \quad &0.7{:}\, 0 \le \mathbf{Time\_Comp}_2 \le 1, 0.2{:}\, 1 \le \mathbf{Time\_Comp}_2 \le 2,\\ &0.1{:}\, 2 \le \mathbf{Time\_Comp}_2 \le 3 \quad \}\end{aligned}$$

We denote the choices of $\mathbf{Time\_Comp}_1$ with ψ_{11}, ψ_{12} and ψ_{13} and the choices off $\mathbf{Time\_Comp}_2$ with ψ_{21}, ψ_{22} and ψ_{23}. We then at each level split on the choices of one variable as shown in Fig. 7.

The important difference with binary WMC is that the constraint cannot always immediately be simplified. Simplification is only possible in case part of the constraints or its negation is a consequence of the choices. Therefore, for some branches choices for all random variables have been made, but still it cannot be decided whether the constraint is true or false. This corresponds to probability mass contributing to the upper, but not to the lower bound.

On the other hand, as for binary WMC, there are cases for which examining all choices is not necessary, e.g. for the right-most branch beneath the root node in Fig. 7. The choice ψ_{13} imposes $2 \le \mathbf{Time\_Comp}_1 \le 3$ which implies that φ_1 ($\mathbf{Time\_Comp}_1 < 0.75$) and φ_2 ($\mathbf{Time\_Comp}_1 < 1.25$) are false and therefore that the entire solution constraint is false. This shows that the order in which variables are chosen matters and – more importantly – that in this generalised setting we can still make use of determinism. The tree in Fig. 7 would have 9 leaves if all choices were examined, but we can make use of the structure to reduce that to 7.

In general, if a subconstraint φ of the CNF is a consequence of the choices Ψ made so far ($\Psi \models \varphi$) it can be replaced by *true*. Similarly, if its negation is a consequence of the choices ($\Psi \models \neg\varphi$), it can be replaced by *false*. The SMT solver can be used to check the conditions by making use of the fact that $\Psi \models \neg\varphi$ is a consequence of $check(\varphi \wedge \Psi) = unsat$ and $\Psi \models \varphi$ a consequence of $check(\neg\varphi \wedge \Psi) = unsat$. A subconstraint φ could for instance be a single primitive constraint in the CNF or the constraint represented by the entire CNF. After substitution the CNF can be simplified by the usual logical rules. The simplified CNF may enable further decomposition, since random variables may be removed.

$$\begin{tikzcd}
& & & & (\varphi_1 \vee \varphi_2) \wedge (\varphi_1 \vee \varphi_3) \arrow[dlll, "\psi_{11}"'] \arrow[d, "\psi_{12}"] \arrow[drr, "\psi_{13}"] & & \\
& \begin{array}{c}(\varphi_1 \vee \varphi_2) \wedge (\varphi_1 \vee \varphi_3) \mid \psi_{11} \\ = \varphi_1 \vee \varphi_3 \mid \psi_{11}\end{array} \arrow[dl, "\psi_{21}"'] \arrow[d, "\psi_{22}"] \arrow[dr, "\psi_{23}"] & & & \begin{array}{c}(\varphi_1 \vee \varphi_2) \wedge (\varphi_1 \vee \varphi_3) \mid \psi_{12} \\ = \varphi_2 \wedge \varphi_3 \mid \psi_{12}\end{array} \arrow[dl, "\psi_{21}"'] \arrow[d, "\psi_{22}"] \arrow[dr, "\psi_{23}"] & & \begin{array}{c}(\varphi_1 \vee \varphi_2) \wedge (\varphi_1 \vee \varphi_3) \\ \mid \psi_{13} \\ = \mathit{false}\end{array} \\
\begin{array}{c}\varphi_1 \vee \varphi_3 \\ \mid \psi_{11}, \psi_{21} \\ = \mathit{true}\end{array} & \begin{array}{c}\varphi_1 \vee \varphi_3 \\ \mid \psi_{11}, \psi_{22}\end{array} & \begin{array}{c}\varphi_1 \vee \varphi_3 \\ \mid \psi_{11}, \psi_{23}\end{array} & \begin{array}{c}\varphi_2 \wedge \varphi_3 \\ \mid \psi_{12}, \psi_{21}\end{array} & \begin{array}{c}\varphi_2 \wedge \varphi_3 \\ \mid \psi_{12}, \psi_{22}\end{array} & \begin{array}{c}\varphi_2 \wedge \varphi_3 \\ \mid \psi_{12}, \psi_{23} \\ = \mathit{false}\end{array} &
\end{tikzcd}$$

Fig. 7. Generalised WMC example.

$$\begin{tikzcd}
& & & & (0.49, 0.88) & & \\
& (0.7, 1.0) \arrow[urrr, "0.7"] & & & (0.0, 0.9) \arrow[u, "0.2"'] & & (0.0, 0.0) \arrow[ull, "0.1"'] \\
(1.0, 1.0) \arrow[ur, "0.7"] & (0.0, 1.0) \arrow[u, "0.2"'] & (0.0, 1.0) \arrow[ul, "0.1"'] & (0.0, 1.0) \arrow[ur, "0.7"] & (0.0, 1.0) \arrow[u, "0.2"'] & (0.0, 0.0) \arrow[ul, "0.1"'] &
\end{tikzcd}$$

Fig. 8. Generalised WMC example (computing probabilities).

To compute probability bounds we proceed as for binary WMC, except that we use probability pairs to represent the lower and upper bound. Leaves corresponding to *true* contribute to both bounds and are substituted by $(1.0, 1.0)$, leaves corresponding to *false* contribute to neither and are substituted by $(0.0, 0.0)$ and finally the leaves for which it is undecided contribute only to the upper bound and are substituted by $(0.0, 1.0)$. Each bound is then computed similar to the binary case.

Example 25

We continue computing probability bounds using the tree of Example 24, as illustrated in Fig. 8. The result equals the bounds computed in Example 15, where we applied the semantic definition.

This idea is exploited in Algorithm 1 in Lines 1–3. We use Ψ to denote a partial choice, which is a conjunction of constraints chosen from random variable definitions, but possibly not from all relevant random variables. At Line 1 the current solution constraint is simplified as discussed above. Note that it is not efficient to always check all possible subconstraints at each step. There are several strategies for which subconstraints to check, e.g. only checking primitive constraints for which all choices have been made and only checking the entire solution constraint at certain steps. In the implementation we use for the experiments we check all primitive constraints for which all choices have been made at each step. This strategy, in combination with simple logical simplifications, is often already sufficient to prove the solution constraint to be *false* or *true*. Only if this is not the case and all variables in the solution constraint have been chosen, does our implementation check the entire solution constraint.

If the constraint can be simplified to *true* or *false*, the branch of the computation is finished (Lines 2, 3). If a case distinction has to be made (Lines 6, 7) the efficiency of the further computations depends on the order variables are selected at Line 6. The main objective is to eliminate random variables to enable further decomposition, which is the same for binary WMC. In the generalised setting, the order of variables also determines how well determinism is exploited. As GWMC terminates in case an inconsistency can be found, a simple heuristic is to order random variables such that $\mathbf{V}_i < \mathbf{V}_j$ if $\mathbf{V}_i$ occurs in an (in)equality constraint with fewer variables on average than $\mathbf{V}_j$. For example, in the constraint

Algorithm 2: Generalised weighted model counting algorithm with optimisations (OGWMC).

```
1  Assuming an initially empty global cache map: cache
   Input: Constraint $\varphi$ in CNF and partial choice constraint $\Psi$
   Result: $(\underline{P}(\varphi \mid \Psi), \overline{P}(\varphi \mid \Psi))$
2  simplify $\varphi$ assuming $\Psi$
3  if $(\varphi, \Psi)$ in cache then return cache$(\varphi, \Psi)$
4  else if $\varphi = false$ then return $(0.0, 0.0)$
5  else if $\varphi = true$ then return $(1.0, 1.0)$
6  else if there is a disjunction $\varphi'$ in $\varphi$ and all variables occurring in $\varphi'$ occur in $\Psi$ as well then
7      $lower, upper = \text{OGWMC}(\varphi, \Psi \wedge \varphi')$
8      $result = (0.0, upper)$
9  else if there is a primitive constraint $\varphi'$ occurring in all components of $\varphi$ and all variables occurring in $\varphi'$ occur in $\Psi$ as well then
10     $lower, upper = \text{OGWMC}(\varphi, \Psi \wedge \neg\varphi')$
11     $result = (lower, 1.0)$
12 else if there are independent CNFs $\varphi_1, \varphi_2$ in $\varphi$ then
13     $result = \text{OGWMC}(\varphi_1, \Psi) \cdot \text{OGWMC}(\varphi_2, \Psi)$
14 else if there is some random variable $\mathbf{V}_i$ in $\varphi$, not occurring in $\Psi$ then
15     $result = \sum_{\psi(\mathbf{V}_i)=(p,\psi)} p \cdot \text{OGWMC}(\varphi, \Psi \wedge \psi)$
16 else return $(0.0, 1.0)$
17 cache$(\varphi, \Psi) \hookleftarrow result$
18 return result
```

$\mathbf{X} > 0 \wedge \mathbf{X} + \mathbf{Y} > 0$ we first select choices for $\mathbf{X}$ as some of these choices might make the whole constraint inconsistent.

In the implementation we use for the experiments we prioritise exploiting determinism and use a simple counting heuristic for choosing variables. Concretely, we count the number of disjunctions in which a random variable occurs since, intuitively, eliminating variables occurring in more disjunctions gives more opportunities for decomposition. In case random variables have the same count we try to exploit determinism by choosing the variable occurring in a primitive constraint with the least number of other random variables together.

Decomposition In the spirit of the well-known RelSAT algorithm [44] for weighted model counting, we can also observe that in many cases the problem can be decomposed into subconstraints which do not share any random variables.

Example 26

Consider the solution constraint $\mathbf{V}_1 > 0 \wedge \mathbf{V}_2 > 0$ and let $|\Psi_{\mathbf{V}_i}|$ denote the number of choices for the random variable $\mathbf{V}_i$. In this case $\overline{P}(\mathbf{V}_1 \wedge \mathbf{V}_2) = \overline{P}(\mathbf{V}_1 > 0) \cdot \overline{P}(\mathbf{V}_2 > 0)$ and the same for the lower bound, which can be computed by examining $|\Psi_{\mathbf{V}_1}| + |\Psi_{\mathbf{V}_2}|$ choices only, whereas naively $|\Psi_{\mathbf{V}_1}| \cdot |\Psi_{\mathbf{V}_2}|$ would have to be examined.

In Algorithm 1 we decompose if possible (Line 4) and recursively compute the bounds for both CNFs (Line 5).

Other optimisations We discuss some further optimisation, resulting in the refined Algorithm 2. Some optimisations for binary WMC can straightforwardly be generalised. Consider for example caching. For binary WMC, in case the same CNF is encountered twice, it has the same count. So computing such subproblems multiple times can be avoided using caching. In our generalised setting the CNF may still contain random variables for which a choice is already made, which means the same CNF can have different counts. We can, however, still make use of caching by reusing results for equal CNFs in combination with equal choices made for all random variables occurring in the CNF. So we first check whether a result is already cached (Line 3), otherwise we compute the result and store it in the cache (Line 17).

The characteristics of our generalised problem require additional optimisations. The fact that after making choices for random variables, the truthfulness of constraints involving them may still be undetermined, can dramatically hinder opportunities for decomposition. To counter this we implemented two optimisations eliminating such undetermined constraints. First, in case all variables occurring in a disjunction of the CNF are chosen, but the disjunction is still undetermined, the lower bound of the CNF is 0.0: even if all other components of the CNF will reduce to *true*, this single disjunction will lead to an undetermined CNF. Conversely, for determining the upper bound we assume the disjunction holds, which is realised by adding it to the constraints imposed by the choices, as formalised in Lines 6–8 of Algorithm 2. This leads to elimination of the disjunction and, therefore, a simplified CNF. It can be easily shown that the upper bound for the original CNF equals the upper bound for the simplified CNF with the additional assumption included in Ψ. Secondly, in case all disjunctions in the CNF have an undetermined constraint in common, it follows that the upper bound is 1.0. This is because the constraint alone can make the solution constraint true and the constraint will never proven to be false, as the choices for all variables occurring in it are already made. In this case, the lower bound equals the lower bound of the CNF assuming the negation of the constraint. This is formalised in Lines 9–11 of Algorithm 2.

7.3. Complexity analysis

To obtain insight into the complexity of the inference problem of PCLP, we analyse the main source of computational complexity, which is the computation of probabilities from a solution constraint. Very often, imprecise probabilistic inference is in general much more complex than precise probabilistic inference. For instance, obtaining a bound on a marginal probability given a *credal network* is $\mathsf{NP}^{\mathsf{PP}}$-complete, while it is PP-complete for Bayesian networks [48]. In order to compare the complexity of PCLP to credal networks, we first show that the decision problem associated with PCLP inference is similar to Bayesian network inference for most practical constraint theories, i.e., it is in PP rather than $\mathsf{NP}^{\mathsf{PP}}$-hard. Of course, this reduced complexity comes at the cost of expressiveness, as discussed in Section 8. Subsequently, it is proven that PCLP also has complexity properties similar to Bayesian network inference and WMC if the treewidth of the problem is bounded.

7.3.1. Worst-case complexity

By Corollary 1 it is clear that computing the lower and upperbound of a PCLP query has the same complexity. Therefore, the problem that we consider is, given the solution constraint of a query q and any probability p, is $\underline{P}(q) > p$? We first show that, given the complexity of constraint checking C, this problem is in PP^{C}.

Theorem 4. *Given a PCLP program* **P** *and a solution constraint φ for a particular query q, determining a bound on $\underline{P}(q)$ is in PP^{C}, where* C *is the complexity of checking satisfiability of sets of constraints in* **P**.

If the complexity of SMT solving is polynomial, i.e., $\mathsf{C} = \mathsf{P}$, then the problem is in PP. Furthermore, proving PP-hardness is trivial as MAJSAT (deciding whether at least half of assignments to a propositional formula satisfy a formula) is PP-complete and can obviously be reduced to bound on the probability of a solution constraint. This shows that the main PCLP inference problem is as complex as precise, discrete probabilistic inference if checking constraints is tractable, even though we can deal with random variables of arbitrary domain and perform a particular kind of imprecise inference.

In the following, we will abstract from the complexity of SMT solving. For many interesting constraint domains such as equalities for discrete constraints and inequalities on sums and differences of real numbers, the complexity of SMT solving is polynomial [49,50]. In other theories it is often desirable to use incomplete solvers, e.g. polynomial algorithms for theories where the satisfiability problem is NP-complete (e.g. arithmetic with integers [33]), even harder (e.g. real number arithmetic including multiplication [34]) or undecidable in general (e.g. arithmetic with integers including multiplication [51]). Finally, note that for typical problems the size of constraints is small compared to the size of the entire probabilistic inference problem.

7.3.2. Parametrised complexity in terms of treewidth

Inference algorithms exploit the structure of problems. For example, the upper bounds of inference in Bayesian networks can more precisely be expressed in terms of the network's *treewidth* t [52], as $O(c^t)$, where c is the maximum cardinality of the variables. Instead of the cardinality of variables, we use the maximum number of choices d in the context of PCLP. The number of choices d is bounded by 2^{c^n}, where n is the number of variables. It can however be smaller as well, which is the strength of PCLP, especially in case c is infinite.

The treewidth measures how tree-like a graph is. In case a Bayesian network is a tree, inference can be done in linear time. A formal definition of treewidth is given below. In our context, we want to bound complexity in terms of a property of the input CNF. There are different kinds of graphs representing the structure of CNFs and algorithms for determining the model count of CNFs, of which the complexity is bounded in terms of the treewidth of different kinds of those graphs are known [53]. We here focus on the primal graph, which is the undirected graph with all CNF's atoms as nodes in which all nodes occurring together in a disjunction of the CNF are connected.

For the PCLP inference problem we slightly adapt the definition of a primal graph. First, we have to use random variables instead of atoms. Secondly, we also have to consider additional dependencies between random variables. Those dependencies emerge from the fact that it may not be possible to eliminate constraints, in spite of the fact that choices for all random variables occurring in it have been made, as discussed in Section 7.2.2.

We refer to *imprecise constraints* as constraints φ for which there is a choice Ψ for all variables occurring in φ, which cannot be used to simplify: neither $\Psi \models \varphi$ nor $\Psi \models \neg\varphi$.

Definition 14 *(Constraint primal graph).* The constraint primal graph of a CNF including constraints, is the undirected graph with all random variables occurring in the CNF as nodes. Two nodes **X** and **Y** are connected if and only if one of the following conditions hold:

1. There is a disjunction in the CNF including **X** and **Y**.
2. **X** occurs together in a disjunction with an imprecise constraint φ, **Y** occurs together in a disjunction with an imprecise constraint χ, and φ and χ share random variables.

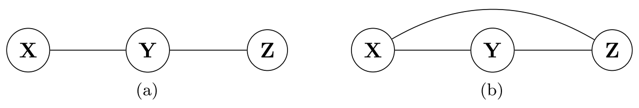

Fig. 9. Example constrained primal graphs.

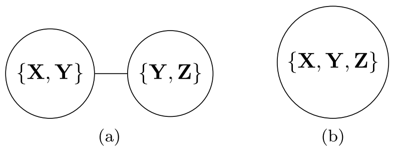

Fig. 10. Example tree decompositions.

Example 27

As an example consider the following CNF:

$$(\mathbf{X} > 0 \vee \mathbf{Y} > 0) \wedge \mathbf{Y} > \mathbf{Z}$$

The constrained primal graph for the case none of the constraints is imprecise is given in Fig. 9a. In case the constraint $\mathbf{Y} > \mathbf{Z}$ is imprecise, an additional edge has to be added as shown in Fig. 9b.

To measure the complexity of the constraint primal graph, we introduce the existing notions of *tree decomposition* and *treewidth*.

Definition 15 *(Tree decomposition).* (See [54].) A tree decomposition of an undirected graph $G = (V, E)$ is a tree $T = (W, H)$, with nodes $x_1, \ldots, x_n$, where each x_i is a subset of V, satisfying:

1. T includes only and all nodes of G.
2. All nodes connected by edges in G occur together in at least one node of T.
3. If two vertices of T contain a vertex of G, then all nodes of the tree in the path between those vertexes contain this vertex of G as well.

Definition 16 *(Treewidth).* (See [54].) The treewidth of a graph is the width of its tree decomposition with minimal width. The width of a tree decomposition is the size of its largest node minus 1.

Example 28

A tree decomposition with minimal width for the graph shown in Fig. 9a, is given in Fig. 10a. Therefore, the treewidth of this graph is 1. For the graph of Fig. 9b only one tree decomposition is possible, shown in Fig. 10b. The treewidth is increased to 2.

Determining the treewidth of a given graph is NP-complete [55]. However, if the treewidth is known a tree decomposition of this width can be constructed in linear time [56].

Finally, we present the main result of this section which relates the input CNF's constraint primal graph treewidth to the complexity of inference.

Theorem 5. *Given an input CNF of bounded treewidth t, the complexity of Algorithm 2 with proper variable order is $O(m\,t\,d^t)$, where m is the number of nodes of the tree decomposition and d is the maximal number of choices for a single variable probability mass is assigned to.*

We emphasize that this complexity is similar to the results for ordinary model counting, for instance the complexity in terms of the primal graph found by Samer and Szeider is $O(o\,m\,t\,2^t)$, where o is the maximum number of occurrences over all variables.

7.3.3. Making use of additional structure

Chavira et al. [18] argue that standard algorithms have complexity $\Theta(c^t)$, i.e. not only the worst-case, but as well the best-case complexity, is bounded exponentially by the treewidth. In contrast, WMC can make use local structure, such as *determinism* [42] and *context-specific independence* [43], and can therefore in some cases perform better. Superiority of WMC has experimentally been shown as well [57]. The same is true for our generalised WMC algorithm. Example 24 for instance shows that branches can be pruned, which means not all exponentially many choices have to be examined. This is further evaluated experimentally in the following.

7.4. Experiments

In this section, we provide some insight into the behaviour of the proposed algorithm. In particular, we show that our inference algorithm is competitive with existing approaches for discrete problems, i.e. the overhead of handling constraints and computing bounds instead of single probabilities is negligible. We further investigate scalability of the algorithm and show that our algorithm can make use of determinism. The implementation is available at http://www.steffen-michels.de/pclp. It makes use of *YAP Prolog* 6.2.2 and the SMT solver Yices 2.0.1 [47], which supports linear arithmetic. The experiments are run under Ubuntu 14.04 on a Laptop with an Intel Core i3 2.4 GHz processor and 4 GB of RAM. All runtimes are averaged over 10 runs.

7.4.1. Comparison with other implementations

We first compare our PCLP implementation with implementations based on similar inference algorithms. Such implementations are limited to precise distributions and mostly to discrete variables. We therefore use a precise, discrete problem to compare. The question we want to answer with the experiment is how scalability of our generalised inference algorithm compares to existing approaches. We expect some overhead for computing bounds instead of point probabilities, though the bounds will be equal for the precise problem we use, and handling constraints. However, scalability of the generalised algorithm should essentially be similar.

We compare to *ProbLog 1* [6] and *ProbLog 2* [58]. The first version of *ProbLog* works similar to our implementation. First, all explanations are collected using SDD resolution, resulting in a formula similar to what we call solution constraint. The models of such formula are then counted by compilation to *binary-decision diagrams* (BDD). *ProbLog 2* uses a different approach. All grounded rules are translated to equivalent propositional formulas, which defined the WMC problem. This can yield more compact CNFs, but requires inclusion of all head atoms in the CNF, instead of only the probabilistic facts, as in the formerly mentioned proof-based approach. The CNF is then compiled to *deterministic decomposable negation normal form* (d-DNNF). Because the performance of this approach mainly depends on the d-DNNF compiler used, we provide results for two difference compilers: *C2D* [59] and the *DSHARP* compiler [60]. A fair comparison with *PRISM* is not possible, as it makes the strong assumption that bodies of the same head are exclusive. It therefore does not solve the difficult problem of computing probabilities of possibly overlapping bodies, according to the *inclusion-exclusion principle*, as the formerly mentioned approaches do.

For the experiment we use a version of the fruit selling problem, we introduced in Section 2.1. In the original version we compare the actual price with the price customers are willing to pay:

$$\mathit{buy}(\mathit{Fruit}) \leftarrow \mathit{price}(\mathit{Fruit}, P),\ \langle P \le \mathbf{Max\_price}(\mathit{Fruit})\rangle$$

We replace this with simple discrete random variables, indicating whether the customer is willing to pay the price with and without government support:

$$\mathbf{Buys\_with\_support}(\mathit{Fruit}) \sim \{0.3{:}\,\mathit{yes}, 0.7{:}\,\mathit{no}\}$$

$$\mathbf{Buys\_without\_support}(\mathit{Fruit}) \sim \{0.6{:}\,\mathit{yes}, 0.4{:}\,\mathit{no}\}$$

The definition of *buy* becomes then:

$$\mathit{buy}(\mathit{Fruit}) \leftarrow \langle \mathbf{Support}(\mathit{Fruit}) = \mathit{yes}\rangle,\ \langle \mathbf{Buys\_with\_support}(\mathit{Fruit}) = \mathit{yes}\rangle$$

$$\mathit{buy}(\mathit{Fruit}) \leftarrow \langle \mathbf{Support}(\mathit{Fruit}) = \mathit{no}\rangle,\ \langle \mathbf{Buys\_without\_support}(\mathit{Fruit}) = \mathit{yes}\rangle$$

In the experiment we query $\mathit{buy}(\mathit{fruit}_1) \vee \cdots \vee \mathit{buy}(\mathit{fruit}_n)$ with increasing n and evaluate inference time. Ideally, inference time should scale linearly, since the query is a disjunction of n independent formulas. This however requires an efficient caching and elimination mechanism.

The result is shown in Fig. 11. It clearly shows that PCLP is competitive, in spite of the overhead of calling the SMT solver and computing bounds (with are equal in this case) instead of point probabilities. However, we do not achieve a linear complexity as would be ideally expected. Only *C2D* scales linearly; it is superior for very large n, although for small n all other implementations perform better.

We adapt the problem and introduce a continuous random variable for the fruit's price. The price the customer is willing to pay is still not uncertain, which makes the constraint one-dimensional. This means it can be dealt with by Hybrid ProbLog [13], which is integrated in the *ProbLog 1* implementation we used in the former experiment. For our PCLP implementation we discretise continuous distributions at the points of those constants, which guarantees a precise distribution. All other implementations used in the previous comparison cannot deal with the problem.

The price of a kind of fruit is defined as in the example in Section 2.1:

$$\mathbf{Yield}(\mathit{Fruit}) \sim \mathcal{N}(12000.0, 1000.0)$$

$$\mathbf{Support}(\mathit{Fruit}) \sim \{0.3{:}\,\mathit{yes}, 0.7{:}\,\mathit{no}\}$$

$$\mathit{basic\_price}(\mathit{Fruit}, 250 - 0.007 \cdot \mathbf{Yield}(\mathit{Fruit}))$$

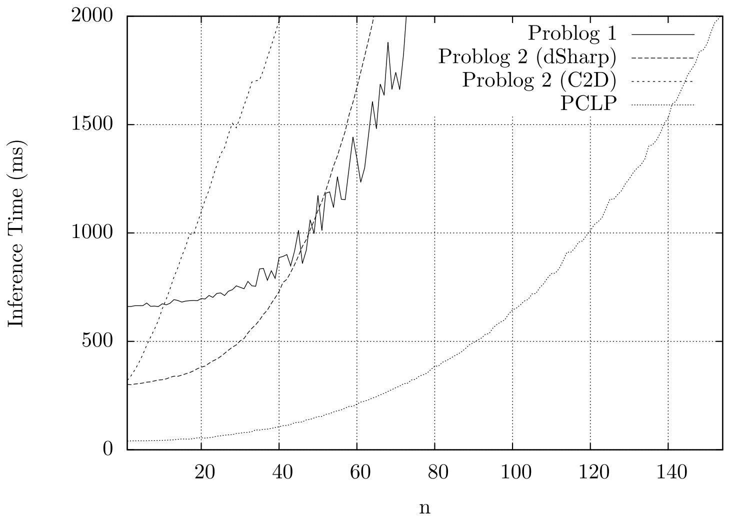

Fig. 11. Comparison of PCLP with other implementation for a discrete problem.

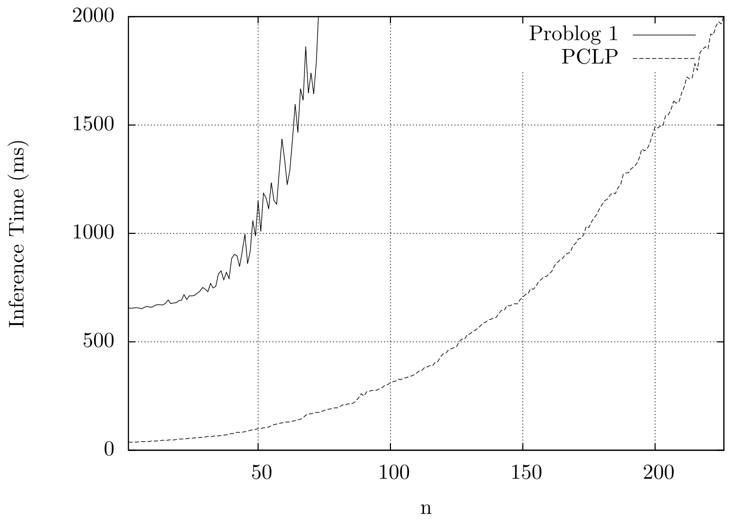

Fig. 12. Comparison of PCLP with *Hybrid ProbLog* for a continuous problem.

$$price(Fruit, BPrice + 50) \leftarrow basic\_price(Fruit, BPrice),\ \langle \mathbf{Support}(Fruit) = yes \rangle$$

$$price(Fruit, BPrice) \leftarrow basic\_price(Fruit, BPrice),\ \langle \mathbf{Support}(Fruit) = no \rangle$$

For the price customers are willing to pay we just use a constant. The rule defining *buy* becomes then:

$$buy(Fruit) \leftarrow price(Fruit, P),\ \langle P \leq 100.0 \rangle$$

We again evaluated inference time for query $buy(fruit_1) \vee \cdots \vee buy(fruit_n)$ with increasing n. The result in Fig. 12 shows again that the performance of PCLP is competitive. The fact that PCLP is superior confirms the better results in the previous experiment. As the time for handling of continuous distributions is negligible, the result reflects the higher overhead for this problem of the BDD package, used by *ProbLog*, compared to PCLP.

7.4.2. Scalability

We now consider the original problem as in Section 2.1, which means we use the following rule to model whether customers buy a certain kind of fruit:

$$buy(Fruit) \leftarrow price(Fruit, P),\ \langle P \leq \mathbf{Max\_price}(Fruit) \rangle$$

Inference for this problem cannot be performed by any of the methods we compared to previously, and to our knowledge PCLP is the only framework that can provide approximations with known error. To perform inference with PCLP, we discre-

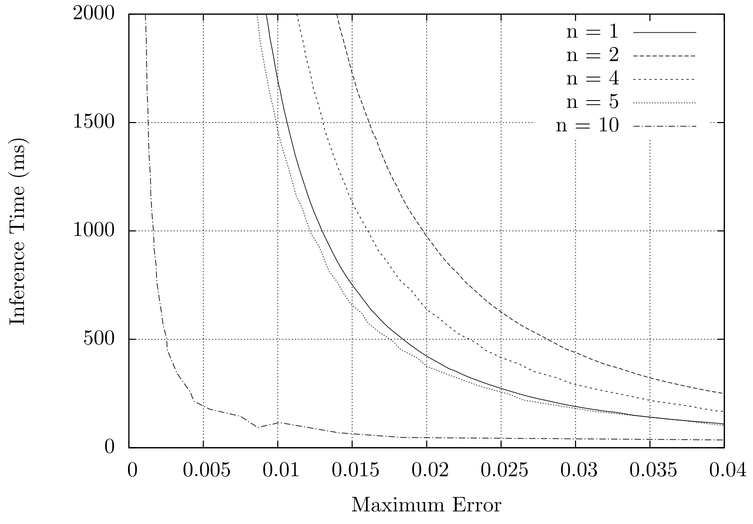

Fig. 13. Inference time vs maximum error for query $buy(fruit_1) \vee \cdots \vee buy(fruit_n)$.

tise continuous distributions as described in Section 5.6 by naively choosing n intervals with equal probability, using CDFs. In the experiments we increase n to decrease the maximal error of the approximation, which increases inference time. Note that this naive discretisation scheme is used on purpose, since we focus on evaluating the generalised WMC algorithm and showing its potential. It is expected that much better performance can be achieved by more sophisticated discretisation, which remains future work.

As before we compute $P(buy(fruit_1) \vee \cdots \vee buy(fruit_n))$, but this time determine approximations. The result is shown in Fig. 13, where we show the relationship between inference time and maximum error.

The result shows an interesting non-monotonic behaviour: while for $n = 2$, inference is more expensive than for $n = 1$, for larger n the efficiency of inference increases with increasing n. For $n = 10$ we achieve a substantially better performance than for small n.

The efficiency decreases between $n = 1$ and $n = 2$, because in general in higher dimensional spaces, a finer discretisation is necessary to achieve the same precision. So with a naive inference approach, the maximum error would always be higher with larger n given the same inference time. However, by making use of the problem's structure the inference time for the choice space is sub-exponential, in particular because subconstraints are independent. That effect dominates for larger n and the error can even decrease given the same inference time.

The experiment shows the potential of the inference approach, as, due to the structure of the problem, often sub-problems concerning continuous variables only have to be estimated with low precision in order to achieve a satisfiable precision for the entire problem.

7.4.3. Exploiting determinism

We finally show that the inference algorithm can make use of determinism by comparing inference times of the query $buy(apple) \wedge \langle \mathbf{Crop}(apple) > \mathrm{X}\rangle$ for various X. The topological structure of the problems is the same, but with larger X the query can already be determined to be unsatisfiable, only based on the choice for the random variable $\mathbf{Crop}(apple)$. The result is shown in Fig. 14, where we show again the relationship between inference time and maximum error.

For $\mathrm{X} = 10\,000$ the result is similar to the result for the query $buy(apple)$, as the probability of the excluded subspace $\mathbf{Crop}(apple) \leq 10\,000$ is very small. With higher X however, the inference efficiency drastically increases, i.e. to achieve a certain precision less inference time is needed. The experiment confirms that the inference algorithm exploits determinism, similar to WMC algorithms in the binary case.

8. Related work

This section is organised as follows. We will first link our work to languages related to PCLP. After that, we will discuss inference methods which have some relationship to the method presented in this paper.

8.1. Related languages

There are various paradigms for probabilistic programming, which are related to paradigms for non-probabilistic programming languages. They range from graphical approaches (e.g. *Bayesian networks* [10], *Multi-Entity Bayesian networks* [61]), imperative and object-oriented approaches (e.g. *FACTORIE* [62], *Figaro* [63]), purely functional approaches (e.g. *Church* [64]), logic programming approaches (e.g. *ICL* [8], ProbLog [6]), to other logic approaches (e.g. *BLOG* [65]). PCLP fits within the

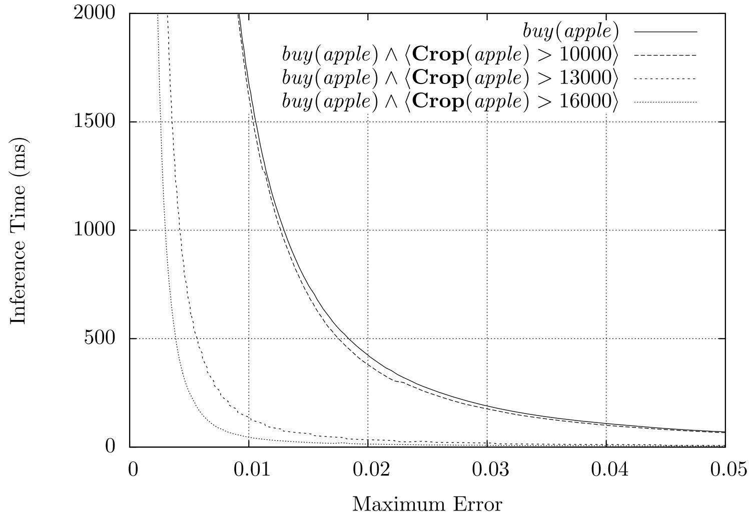

Fig. 14. Inference time vs maximum error for query $buy(apple) \wedge \langle \mathbf{Crop}(apple) > X \rangle$ with varying X.

Table 4
Related languages.

	Discrete	Hybrid
Precise	*Bayesian Networks* [10] *ProbLog* [6] *PRISM* [7] *ICL* [8]	*Conditional Linear Gaussian* [66] *Hybrid Problog* [13] *Distributional clauses* [14] *BLOG* [65]
Imprecise	*Locally Defined Credal Networks* [67] *Probabilistic logic programming* [68] *Imprecise Probabilistic Horn Clause Logic* [69]	PCLP

logic programming approach, which has been shown to be a powerful knowledge-representation tool for non-probabilistic problems. As a result, this is the first and arguably best studied first-order probabilistic programming paradigm, with a well-founded semantics.

In the following, rather than focusing on the underlying programming paradigm, we discuss the expressivity of related languages, i.e. which kinds of distributions can be represented. This is discussed along two dimensions: discrete versus hybrid distributions and precise versus imprecise distributions. The resulting classes of formalisms, together with some example formalisms, are shown in Table 4 and will guide the rest of this section. As the table illustrates, being able to represent imprecise hybrid distributions is, to our knowledge, a unique feature of PCLP.

8.1.1. *Precise discrete distributions*

PCLP clearly subsumes languages allowing one to define discrete and precise probability distributions including propositional ones, such as *Bayesian networks* [10], as well as first-order languages, e.g. probabilistic logics based on Sato's distributions semantics. Examples are *ICL* [8], PRISM [7], ProbLog [6] and CP-logic [9]. Approaches in which weights instead of probabilities are used, such as *Markov logic networks* (MLNs) [70], have a very different nature compared to the above approaches, in the sense that the parameters in such models do not necessarily have a direct probabilistic meaning.

A number of approaches use constraints to model discrete distributions. This is done with advantages for representation and learning in mind. There are a number of languages, which are, as PCLP, based on deterministic *constraint logic programming* (CLP). One example of such language is CLP($\mathcal{BN}$) [71], which does not use constraints to denote events and define probability distributions, but considers probability distributions as constraints themselves. The approach is quite different from approaches based on the distribution semantics, as illustrated by some examples by Costa and Paes [72]. Especially, dynamic models, as *Hidden Markov Models*, can be represented by PRISM very compactly and concisely. This observation similarly holds for other approaches based on the distributions semantics, including PCLP. An approach, similar to CLP($\mathcal{BN}$), but more general, is clp(pdf(y)) [73], which is however restricted to discrete random variables, too. Another approach by Riezler [74], sharing the name with our language, assigns probabilities to derivation choices. The approach is developed with natural language processing applications in mind and does not consider continuous distributions. Finally, Sato's CBPMs [75] are based on the idea that any discrete joint probability distribution can be expressed as independent binary random variables, constrained by arbitrary FOL formulas. Constraints in this setting are therefore purely logical, without additional theories, and the approach is restricted to discrete distributions, i.e. it cover at most countably infinite domains as integers.

A limitation of PCLP and most other logic programming approaches is that they do not directly support generative definitions, i.e. random variables cannot be defined in terms of the value of other random variables. For finite, discrete distributions, this can be circumvented by introducing distinct random variables, each related to a value of another random variable. However, this is not possible for infinite distributions. One example where generative definitions are useful is modelling an unknown number of objects, as shown by the work on *BLOG* [65]. The number of objects can for instance be modelled by a *Poisson* distribution, which is discrete, but not finite. Such generate process could be modelled in PCLP, yet the number of relevant constraints would not be finite, and therefore exact inference may not be possible in PCLP (Theorem 3).

8.1.2. Imprecise discrete distributions

Approaches dealing with imprecise probabilities can typically represent more complex imprecise distributions than PCLP, for example [76,68]. Specifically, PCLP cannot express qualitative relations between probabilities of events, e.g. express that the probability of event a is greater than the probability of event b. However, the restricted way of how imprecise probabilities are defined in PCLP guarantees that definitions cannot be inconsistent (Theorem 1). Also some languages that do guarantee consistency such as *locally defined credal networks* (LDCNs) [67] and relational variants [77] are more expressive than PCLP, since they can express conditional credal sets. Of course, this comes at the price that inference is more complex as well (cf. Section 7.3). PCLP can also not express open probability intervals, which is e.g. realised for *Imprecise Probabilistic Horn Clause Logic* [69] by using infinitesimal probabilities.

8.1.3. Precise hybrid distributions

All approaches we discuss are not as general as PCLP, in the sense that continuous distributions are restricted to real-valued random variables. The semantic foundation of PCLP makes use of a more general notion of continuous distributions, concerning random variables with arbitrary uncountable ranges. There are in general roughly two different ways to deal with continuous distributions: computing posterior continuous distributions of continuous random variables or computing probabilities of events.

Examples of the first approaches are *Conditional Linear Gaussian* (CLG) models [66] or the first-order approach in [78], which is based on the distribution semantics as well. The approach provides more information about the continuous distributions as possible with probabilities of events alone. This can for instance be used for computing expectations and decision making. However, the approach requires a way to represent resulting probability density functions of distributions. Those could be combinations of Gaussian distributions or other ways to represent functions as *mixtures of truncated basis functions* [79] and *piecewise polynomials* [80], which can approximate usually employed distributions, though no strong guarantees can be given that the approximation is close to the true distribution. Summarised, determining result distributions can only be achieved by severely restricting the distributions allowed (e.g. to conditional linear Gaussian distributions) or by using an approximate representation.

The second approach to deal with continuous distributions – as adopted by PCLP – focuses on computing probabilities of events involving continuous variables. It is still powerful enough for decision making in case of discrete decisions. Within this category, most probabilistic languages based on imperative, object-oriented and functional approaches support hybrid distributions. A straightforward hybrid extension of logic programming based approaches is *Hybrid ProbLog* [13], which allows real-valued random variables, but restricts their use to one dimensional constraints, for instance $\mathbf{X} > 0$. This framework is therefore less expressive than PCLP. One of the most general extensions of logical semantics are *Distributional clauses* (*DCs*) [14], which supports arbitrary constraints on real-valued random variables and generative definitions, e.g. the variance of a distribution could depend on the value of another one. Such generative definitions cannot be expressed by PCLP. However, incorporating generative definitions within the context of the distribution semantics complicates the semantics and puts a burden on the user of the language by requiring a number of very technical requirements for a program to be valid [14, Definition 3]. Our extension of the distribution semantics is – in our view – more close to the original idea and provides the necessary properties for our analysis of the exact inference conditions and our credal set extension, but is less expressive.

8.2. Related inference methods

The main contribution presented in this paper is the idea to define events in terms of decidable constraint theories and define credal sets based on those constraint theories to allow exact computation of bounds. In this subsection, we will consider related methods for probabilistic inference, subdivided by exact, approximate, and lifted inference.

8.2.1. Exact inference

There are various algorithms for exact probabilistic inference, most of them defined for *Bayesian Networks*. Examples are *variable elimination* [40] and *recursive conditioning* [41]. As discussed, we base our work on the approach of translating inference to WMC problems [18], since this method makes it possible to exploit structure as *determinism* [42] and *context-specific independence* [43], which often occurs in logical theories. Such exact inference is also possible for hybrid models with restricted types of distributions, for example *Conditional Linear Gaussian* (CLG) models [66]. In the probabilistic logic context, inference in *Hybrid ProbLog* [13] can be done exactly. Since all constraints are one-dimensional, distributions can be discretised at all points occurring in inequalities in the proofs of a query, which turns inference into a discrete problem.

A more general related approach to inference with probabilities and constraints is by Dechter [81], which focuses on algorithms for various inference problems. All considered problems share the property that the structure of problem instances can be represented as graphs. Examples of such problems include probabilistic reasoning and constraint satisfiability, which are exactly the problem also unified in our approach. Dechter's work may be the basis of interleaving probabilistic reasoning and constraint checking in a more efficient way in PCLP inference, based on a uniform representation combining both aspect of inference problems. In our current approach, probabilistic inference only uses the information which random variables occur in primitive constraints, but does not make use of the structure of constraints.

Exact inference algorithms have also been developed for the imprecise case. Since these problems are often strictly more complex than the precise case, methods for the precise case are not applicable. For instance, one has to resort to methods as *multi-linear programming* [77]. An exception is *Imprecise Probabilistic Horn Clause Logic* (IPHL) [69] providing an imprecise formalism for which complexity is as hard as for the precise case. However, IPHL restricts the language to binary random variables and Horn clauses, whereas PCLP does not impose those restrictions.

8.2.2. *Approximate inference*

For *Bayesian Networks* one class of approximation algorithms is based on Pearl's *belief propagation* algorithm [10], which computes posterior probabilities by passing messages between nodes. However, the algorithm only terminates and produces correct probabilities for networks which are polytrees, for which inference can be done in polynomial time. For the general case, the algorithm also often converges to a good approximation of the correct probabilities. A lot of work has been done about convergence and quality of approximation [82–85], but hard guarantees about the quality of the result cannot always be provided, in contrast to our inference method.

Similarly to PCLP, there is work on approximation schemes providing hard guarantees for discrete distributions by simplifying the problem [86]. Recently, Renkens et al. have shown the effectiveness of such approach in combination with state-of-the-art knowledge compilation techniques [87]. The main difference between PCLP and this work is that PCLP can be used to model and reason with continuous and imprecise distributions, whereas the former approaches are aimed at approximate inference for discrete and precise problems.

As mentioned, for continuous distributions, another inference method is to use approximate representations of distributions, which are typically mixtures of functions. Examples of such an approach are *mixtures of truncated basis functions* [79] and *piecewise polynomials* [80]. Such approximation functions can be used to compute posterior probabilities, but no hard guarantees about the quality of the result can be given. The accuracy of the approximation is usually measured in terms of the *Kullback–Leibler divergence* [88] between the distributions, which does not directly related to the error of the computed probabilities. This makes it hard to draw conclusions on the quality of probabilities and expectations derived from the results, which is essential for decision making.

Methods which put virtually no restrictions on the distributions used are based on *Markov chain Monte Carlo* (MCMC) sampling. Those methods can be very effective, but often have to be hand-tailored for specific problems. There are however frameworks providing MCMC inference for generic relational probabilistic languages [89,90]. Recently significant progress has been made to improve upon known problems of MCMC methods, such as slow convergence for high-dimensional distributions [91] and distributions with near deterministic probabilities [92]. Despite those improvements, there is a qualitative difference between sampling methods and the method proposed in our work. No hard guarantees can be given about the error of a sampling estimation. Even under perfect circumstances guarantees for MCMC can only be given in terms of for instance the *Monte Carlo standard error*, with gives a probabilistic, but no hard guarantee. Even worse, in realistic cases, nothing is known about the distributions, so no guarantees can be given at all. The approximation might *pseudo converge*, which means that is seems to have converged to a solution, which is actually not the case. Determining whether a Markov chain converged is only possible under complicated theoretical conditions [93]. In general, there is no known way to be sure a Markov chain actually converged (see e.g. [94, p. 21]). In contrast, PCLP pushes expressivity as far as possible, but still allows one to measure the quality, making it possible to decide whether more computation time or effort to acquire evidence should be invested to compute better results.

8.2.3. *Lifted inference*

Another method to deal with intractability of probabilistic problems is lifted inference. The idea is that symmetries in inference problems, which are especially present in first-order models, can be exploited to significantly improve efficiency of inference. A survey of lifted inference approaches is provided by Kersting [95]. Lifted inference can also be applied to continuous distributions, which has been shown by Choi et al. [96]. PCLP inference may also profit from lifted inference, but is beyond the scope of this paper.

9. Conclusions & future work

We introduced a new generalised distribution semantics for probabilistic logic programming making it possible to use random variables with arbitrary ranges. The semantics illustrate that the integration of various known techniques can provide a powerful formalism. In particular, we combine the ability of logic to represent relational knowledge, probability theory to deal with uncertainty and constraints representing conditions on variables with various ranges.

As commonly known, exact probabilistic inference is only possible under conditions which pose strong restrictions on distributions. As an alternative to many other approximation methods, we propose to use a framework based on credal sets where lower and upperbounds on posteriors can be guaranteed. Placing the method in the theory of imprecise probabilities provides a clear view on the approach, allowing one to explore its properties and to compare with other approaches. Also, powerful applications of the approach can straightforwardly be defined, for instance approximating probabilities of events with known and arbitrarily small maximum error in hybrid distributions.

We further show that a concrete language can be based on that concept and that we can perform exact inference with similar complexity as for the precise case, which is a quite unique property for an imprecise formalism. We developed a concrete inference algorithm, based on a generalisation of WMC. The algorithm is able to exploit the same kinds of structure as ordinary WMC, which is the state-of-the-art of precise probabilistic inference. This again shows the benefit of our general semantic foundation integrating existing AI approaches. The language can straightforwardly be extended with arbitrary constraint theories, such as ordinary *constraint logic programming*. Furthermore, for performing inference we can make use of existing algorithms from the field of *logic programming* and *constraint satisfaction*, in particular modern SMT solvers, and generalise probabilistic inference by *weighted model counting*, inheriting the method's strengths.

In the future the most important aspect to make PCLP practically useful is to implement efficient inference mechanisms. One research direction is to apply knowledge compilation techniques, which are already successfully applied for the standard WMC problem. Also, the naive way to discretise continuous distributions used in this article can probability be improved enormously. Further, computing expectations of random variables values is an important problem to solve for practical applications. We finally plan to apply PCLP for building information fusion models in the context of the maritime safety and security project *METIS* [97,98].

Appendix A. Proofs

Proof of Lemma 1. We show that the equation is equivalent to Definition 4:

$$\begin{aligned} P(q) &= P_{\mathbf{T}}\big(\{(\omega_{\mathbf{V}}, \omega_{\mathbf{L}}) \in \Omega_{\mathbf{T}} \mid \omega_{\mathbf{L}} \models q\}\big) \\ &= P_{\mathbf{V}}\big(\{\omega_{\mathbf{V}} \mid (\omega_{\mathbf{V}}, \omega_{\mathbf{L}}) \in \Omega_{\mathbf{T}},\ M_{\mathbf{L}}(\omega_{\mathbf{V}}) \models \omega_{\mathbf{L}},\ \omega_{\mathbf{L}} \models q\}\big) \quad (\text{Definition 3}) \end{aligned}$$

The condition $M_{\mathbf{L}}(\omega_{\mathbf{V}}) \models \omega_{\mathbf{L}},\ \omega_{\mathbf{L}} \models q$ is equivalent to $M_{\mathbf{L}}(\omega_{\mathbf{V}}) \models q$, since $M_{\mathbf{L}}(\omega_{\mathbf{V}})$ uniquely determines $\omega_{\mathbf{L}}$. Therefore:

$$\begin{aligned} P(q) &= P_{\mathbf{V}}\big(\{\omega_{\mathbf{V}} \mid (\omega_{\mathbf{V}}, \omega_{\mathbf{L}}) \in \Omega_{\mathbf{T}},\ M_{\mathbf{L}}(\omega_{\mathbf{V}}) \models q\}\big) \\ &= P_{\mathbf{V}}\big(\{\omega_{\mathbf{V}} \in \Omega_{\mathbf{V}} \mid M_{\mathbf{L}}(\omega_{\mathbf{V}}) \models q\}\big) \\ &= P_{\mathbf{V}}\big(\mathrm{SE}(q)\big) \quad (\text{Definition 5}) \qquad \square \end{aligned}$$

Proof of Lemma 2. By applying Definitions 1 and 5 we have to prove that:

$$\{\omega_{\mathbf{V}} \in \Omega_{\mathbf{V}} \mid M_{\mathbf{L}}(\omega_{\mathbf{V}}) \models q\} = \Big\{\omega_{\mathbf{V}} \in \Omega_{\mathbf{V}} \mid \bigvee_{\substack{M_{\mathbf{L}}[\Phi] \models q \\ \Phi \subseteq \mathbf{Constr}}} \ \bigwedge_{\varphi \in \Phi} \ \varphi(\omega_{\mathbf{V}})\Big\}$$

($\subseteq$) Assume we have an $\omega_{\mathbf{V}}$, which is element of the left-hand set, i.e. for which $M_{\mathbf{L}}(\omega_{\mathbf{V}}) \models q$. We choose Φ such that $M_{\mathbf{L}}[\Phi] = M_{\mathbf{L}}(\omega_{\mathbf{V}})$. Since for $\omega_{\mathbf{V}}$ all constraints in Φ are satisfiable, then $\bigwedge_{\varphi \in \Phi} \varphi(\omega_{\mathbf{V}})$ holds and therefore $\omega_{\mathbf{V}}$ is element of the right-hand set as well.
($\supseteq$) Assume we have an $\omega_{\mathbf{V}}$, which is element of the right-hand set, i.e. for which there is a Φ, such that $M_{\mathbf{L}}[\Phi] \models q$ and $\bigwedge_{\varphi \in \Phi} \varphi(\omega_{\mathbf{V}})$. Because of the latter, we know that $M_{\mathbf{L}}(\omega_{\mathbf{V}})$ includes $\langle \varphi \rangle$ for all constraints $\varphi \in \Phi$. From $M_{\mathbf{L}}[\Phi] \models q$ we can therefore conclude $M_{\mathbf{L}}(\omega_{\mathbf{V}}) \models q$, due to monotonicity of first-order logic. This means $\omega_{\mathbf{V}}$ is element of the left-hand set, too. $\square$

Proof of Proposition 1. Assuming the three conditions hold, we prove that it is possible to compute the probability of an arbitrary query q as analytic expression. We show that the solution event of q is finite-dimensional, which means it has a finite representation, and can be computed in finite time. The probability of q can then be computed according to Lemma 1 assuming the computable-measure condition.

The solution event lemma (Lemma 2) makes use of subsets of the constraint language and derives unique models from that. The finite-relevant-constraints condition means that, in order to find all Φ for which $M_{\mathbf{L}}[\Phi] \models q$, we only have to consider a finite number of occurrences of $\langle\rangle$, which implies a finite number of constraints. We therefore only have to consider a finite number of subsets of the event space, built from the solution space of the combination of that finite number of constraints. The solution event can consequently be computed in finite time. Moreover, since all constraints in the construction of the solution event are finite-dimensional (finite-dimensional-constraints condition), the solution event is finite as well. $\square$

Proof of Lemma 3. We show there is at least one element $P_{\mathbf{V}}$ in $\mathbf{P_V}$ by constructing it from the finite-dimensional $\mathbf{P}^k_{\mathbf{V}}$. The proof is by induction, assuming the number of random variables is infinite. Obviously the proof also holds for a finite number of random variables. We show that (1) there is at least one $P^1_{\mathbf{V}}$ in $\mathbf{P}^1_{\mathbf{V}}$ given an arbitrary $\mathbf{C}_1$ and (2) that given a $P^k_{\mathbf{V}}$ consistent with $\mathbf{C}_k$ there is always a compatible $P^{k+1}_{\mathbf{V}}$ consistent with $\mathbf{C}_{k+1}$ given that $\mathbf{C}_k$ and $\mathbf{C}_{k+1}$ are compatible. That $P^k_{\mathbf{V}}$ and $P^{k+1}_{\mathbf{V}}$ are compatible means that for any k-dimensional event e: $P^{k+1}_{\mathbf{V}}(\pi_{k+1}(e)) = P^k_{\mathbf{V}}(e)$. We therefore get an infinite sequence of compatible, finite-dimensional probability distributions from which we can construct a probability measure on the entire event space, given the basic assumptions we made in the lemma.

In the following we do not give a complete construction of probability measures, because there are in general multiple distributions with the required probability. Therefore, we only fix the probability of a number of disjoint events in a way that makes it possible to construct a valid probability measure. We assign all probability mass to those disjoint events and no probability mass to the rest of the probability space. All possible measures assigning probabilities to events in this way clearly assign 1 to the entire space, are countably additive and therefore valid probability measures.

(1) The disjoint events we use to construct $P^1_{\mathbf{V}}$ are the intersections and relative complements of the events of $\mathbf{C}_1$ except the empty set. We then make sure all those events are disjoint by restricting them to the minimal ones, i.e. the ones who have no subset in the collection. Intuitively, this can be seen as all areas of a *Venn diagram* which do not include other areas. We denote those events by $f_1, \dots, f_n$ and assign for each $(p_i, e_i) \in \mathbf{C}_1$ probability p_i to an $f_j \subseteq e_i$. There is always at least one such f_j. In case we choose the same f_j for multiple e_i we sum the probabilities.

What remains to be shown is that a measure $P^1_{\mathbf{V}}$ constructed in this way obeys the inequalities of Definition 7, i.e. that for all events e:

$$\sum_{\substack{d \subseteq e \\ (p,d) \in \mathbf{C}_1}} p \le P^1_{\mathbf{V}}(e) \le \sum_{\substack{d \cap e \neq \emptyset \\ (p,d) \in \mathbf{C}_1}} p$$

First we prove that the lower bound holds. For each event e we consider all subsets of e in $\mathbf{C}_1$: $e_1, \dots, e_m$. In the construction of $P^1_{\mathbf{V}}$ all probability mass assigned to all such $e_1, \dots, e_m$ must be assigned to subsets $f_1, \dots, f_{m'}$ of those $e_1, \dots, e_m$, which are in turn subsets of e as well. Therefore:

$$\sum_{\substack{d \subseteq e \\ (p,d) \in \mathbf{C}_1}} p \le P^1_{\mathbf{V}}(e)$$

Similarly, the probability mass of each e_i for which $e_i \cap e = \emptyset$ can only be assigned to $f_j \cap e = \emptyset$, which means:

$$\sum_{\substack{d \cap e = \emptyset \\ (p,d) \in \mathbf{C}_1}} p \le 1 - P^1_{\mathbf{V}}(e) \iff P^1_{\mathbf{V}}(e) \le \sum_{\substack{d \cap e \neq \emptyset \\ (p,d) \in \mathbf{C}_1}} p$$

(2) We assume we have given a $P^k_{\mathbf{V}}$ obeying the specification $\mathbf{C}_k$ and a $\mathbf{C}_{k+1}$ compatible with $\mathbf{C}_k$. From this we construct a probability measure $P^{k+1}_{\mathbf{V}}$ which is compatible with $P^k_{\mathbf{V}}$.

For each (p, e) in $\mathbf{C}_k$ there are $(p_1, e_1), \dots, (p_n, e_n)$ in $\mathbf{C}_{k+1}$ for which each $\pi_k(e_i) = e$, $1 \le j \le n$ and the sum of all p_i equals p, because $\mathbf{C}_k$ and $\mathbf{C}_{k+1}$ are compatible. Further, for each e there is an f to which the probability mass p was assigned in the construction of $P^k_{\mathbf{V}}$. We assign probability masses $p_1, \dots, p_n$ to the intersections and relative complements of $e_1 \cap f, \dots, e_n \cap f$. Such intersections are disjoint and non-empty and we denote them with $f_1, \dots, f_m$.

We have to make sure $P^{k+1}_{\mathbf{V}}$ is compatible with $P^k_{\mathbf{V}}$. For each k-dimensional f there are a number of $k+1$-dimensional $f_1, \dots, f_m$ and for all those events $P^k_{\mathbf{V}}\big(\pi_k(f_j)\big) = P^k_{\mathbf{V}}(f)$. Therefore, it is possible to construct $P^{k+1}_{\mathbf{V}}$ in such a way that it assigns the same probabilities to all g^{k+1} where $g \subseteq f$ as $P^k_{\mathbf{V}}$ assigns to g. This makes both measures compatible, since all f_i cover the entire probability space to which any probability mass is assigned.

The constructed probability measure obeys the bounds defined in Definition 7 with similar arguments as for $P^1_{\mathbf{V}}$. For each event e_i in $\mathbf{C}_{k+1}$ there are disjoint events $f_1, \dots, f_m$ to which some probability mass is assigned. All event in $\mathbf{C}_{k+1}$ which are subsets of any $f_1, \dots, f_m$ certainly contribute to the probability of e while disjoint events do certainly not contribute to it. □

Proof of Theorem 1. The theorem follows directly from Lemma 3, Definition 3 and Definition 4. □

Proof of Proposition 2. Bounds for the probabilities of any event are defined by Definition 7. The theorem above puts those bounds on the query's solution event SE(q), whose probability is equal to the query's probability (Lemma 1). To show that the bounds are actually the minimum and maximum of the set of all probabilities, and therefore the most tight bounds, we show for each query q there is a probability distribution in $\mathbf{P}$ such that the probability of q equals the lower (1) and a distribution such that the probability equals the upper bound (2).

We proof this by a construction similar to the proof of Lemma 3. The construction below applies to the construction of $P^1_{\mathbf{V}}$ as well as to the constructions of $P^{k+1}_{\mathbf{V}}$. We assign probability masses to disjoint events $f_1, \dots, f_n$ for each e in $\mathbf{C}_i$, but we do not assign the probability mass to arbitrary events as before. The events $f_1, \dots, f_n$ are as before the minimal intersections and relative complements of all events in the credal set specification. Additionally, we split events f_i which are not subset of or disjoint with the solution event into two: $f_i \cap \mathrm{SE}(q)$ and $f_i \setminus \mathrm{SE}(q)$.

(1) For each e which is not a subset of the solution event we assign the probability mass to an event from $f_1, \dots, f_n$ disjoint with the solution event. Such event always exists, since all $f_1, \dots, f_n$ which are subset of e, but not of $\mathrm{SE}(q)$, are split into a part which is disjoint with the solution event and a part which is not. This means only e which are subsets of the solution event contribute to the probability, which consequently equals the lower bound as defined by the proposition.

(2) We do the same kind of construction, but this time assign always to events which are subsets of the solution event if possible. This leaves out e which are disjoint with the solution event and the probability equals the upper bound as defined by the proposition. □

Proof of Theorem 2. Using Proposition 2 we get the bounds given a finite-dimensional $\mathbf{C}_k$. If we increase k, some of the events e in $\mathbf{C}_k$ may split into subsets of e.

Events e of which the probability contributes to the lower bound in k-dimensions will do so as well in higher dimensions, since all events, e is possibly split into, are subsets of e. Some subsets of events not contributing to the lower bound in k dimensions may do so in higher dimensions. This means the lower bound is never overestimated for finite k and can only come closer to the actual lower bound for higher dimensions. Similarly, the upper bound is never underestimated for finite k and with increasing k the probability comes closer to the actual upper bound.

In case the bounds are underestimated or overestimated, this is always caused by the fact some event will be split for higher k, so for each event this can be compensated by increasing to that higher k. This means one can get arbitrarily close to the actual bounds. □

Proof of Corollary 1. This follows directly from Theorem 2, the sum rule of limits, the fact that probabilities of credal set specifications sum up to 1.0 (Definition 6) and the fact that $\mathrm{SE}(q)$ is disjoint with $\mathrm{SE}(\neg q)$ and $\mathrm{SE}(q) \cup \mathrm{SE}(\neg q) = \Omega_{\mathbf{V}}$. □

Proof of Proposition 3. We prove the equation for the upper bound. The proof for the lower bound is similar. We know that $P(q \mid e) = P(q \wedge e)/P(e)$, therefore $P_{max}(q \mid e) = P_{max}(q \wedge e)/Z$ where Z is the partition function which maximises $P_{max}(q \mid e)$. The same partition function minimises $P_{min}(\neg q \mid e)$, therefore $P_{min}(\neg q \mid e) = P_{min}(\neg q \wedge e)/Z$. By equivalence transformations and substitution of Z we get $P_{max}(q \wedge e)P_{min}(\neg q \mid e) = P_{min}(\neg q \wedge e)P_{max}(q \mid e)$. By using Corollary 1 ($P_{max}(q \mid e) + P_{min}(\neg q \mid e) = 1$) we can substitute $P_{min}(\neg q \mid e)$ and get $P_{max}(q \wedge e)(1 - P_{max}(q \mid e)) = P_{min}(\neg q \wedge e)P_{max}(q \mid e)$ which is equivalent to the equation above. □

Proof of Theorem 3. In the proof of Proposition 1 we show that we can determine a finite-dimensional solution event in case the finite-relevant-constraints condition and finite-dimensional-constraints condition holds.

The bounds can be computed using Proposition 2. The equations in Proposition 2 require summing over all elements of a $\mathbf{C}_k$. This k has to be chosen such that all variables restricted by the solution event are included. A finite k can be found, because the finite-dimensional-constraints condition is fulfilled.

We finally can decide in finite time whether the event contained in each element of the finite-dimensional credal set specification is disjoint with or a subset of the solution event (disjoint-events-decidability condition). □

Proof of Lemma 4. Follows directly from the fact that the finite credal set specifications are compatible and Lemma 3. □

Proof of Proposition 4. The proof is a variation of the proof of Proposition 2. Disjointness of events corresponds to unsatisfiability of the corresponding constraints' conjunction and that an event is a subset of another corresponds to satisfiability of the implication ($\varphi_1 \rightarrow \varphi_2$), which is equivalent to unsatisfiability of $\varphi_1 \wedge \neg\varphi_2$. □

Proof of Corollary 2. This follows directly from Definition 12 and Proposition 4. □

Proof of Corollary 3. This follows directly from Theorem 3, since the disjoint-events-decidability condition corresponds to decidable satisfiability of constraints, as shown in Proposition 4. □

Proof of Theorem 4. We focus on the problem of deciding whether $\underline{P}(\varphi) > p$, with probability $0 \le p \le 1$. To prove membership in PP^{C}, we show that this can be decided by a Probabilistic Turing Machine $\mathcal{M}$ in polynomial time, given an oracle for checking satisfiability of constraints. $\mathcal{M}$ computes the joint probability over choices and constraints in φ. First, $\mathcal{M}$ iterates over each variable $\mathbf{V}_i$ and choosing a particular constraint for that variable conform the probability mass associated to that constraint. Each computation path then corresponds to a specific choice Ψ for $\mathbf{P}$. Then, $\mathcal{M}$ computes $\mathit{check}(\Psi \wedge \neg\varphi) = \mathit{unsat}$. If true, the state is accepted with probability $\frac{1}{2} + (1 - p) \cdot \epsilon$, and with probability $\frac{1}{2} - p \cdot \epsilon$ otherwise. The probability of entering an accepting state is therefore $\underline{P}(\varphi) \cdot (\frac{1}{2} + (1 - p)\epsilon) + (1 - \underline{P}(\varphi)) \cdot (\frac{1}{2} - p \cdot \epsilon) = \frac{1}{2} + \underline{P}(\varphi) \cdot \epsilon - p \cdot \epsilon$. Now, the probability of arriving in an accepting state is strictly larger than $\frac{1}{2}$ if and only if $\underline{P}(\varphi) > p$. □

Proof of Theorem 5. We assume we have given a tree decomposition with treewidth t and use it to determine a variable order. Suppose we start at an arbitrary node s of the tree and make case distinctions for all variables in that node. The number of resulting CNFs is bounded by d^t, as the size of the node is bounded by t and for each random variables at most d choices can be made. Furthermore, the number of choices which are incorporated into each of those CNFs is bounded by t. The complexity of the computation is therefore $O(t\,d^t)$, if we do not consider the recursive calls to OGWMC for the decomposed sub-CNFs. The reason for this is that not all sub-CNFs are different and we can exploit that using caching (Line 3 of Algorithm 2) and the number of distinct sub-CNFs is bounded.

All sub-CNFs passed to further recursive calls do not include any of the variables of s. Either they are eliminated by simplification or they were included in an imprecise constraint. In the latter case, for each disjunction in which such imprecise constraints are included, choices for all random variables included are made and the optimisation in Lines 6–8 of Algorithm 2 can be applied. For each sub-CNF passed as recursive call a new decomposition tree can be assigned, after decomposition is applied (Line 13 of the algorithm). We eliminate s and all variables in s from the tree and get a number of disconnected subtrees. To each sub-CNF exactly one such subtree can be assigned, such that the same random variables are included in the sub-CNF as well as in the subtree. The subtrees do not share variables (Property 3 of Definition 15) and each random variable still remaining in the passed sub-CNFs is present in one of such subtree (Property 1 of Definition 15). This means each such random variable is present in **exactly** one subtree. Furthermore, for each sub-CNF there is exactly one tree decomposition including all variables in that sub-CNF, because of Property 2 of Definition 15 and the definition of the constraint primal graph (Definition 14).

Because we apply caching the number of consecutive computations is bounded by the number of different inputs of the algorithm. We only have to consider the number of distinct CNFs, since all variables occurring in choices have been removed, as discussed before. For each of such subtrees the number of distinct corresponding CNFs is bounded by d^m, where m the number of variables the subtree and the node n chosen in the previous step have in common. If we now for each such CNFs use the node of the subtree, originally connected to n, we have at most $t-m$ variables remaining. Therefore, the complexity for each single CNF is $O(t\,d^{t-m})$ and we have at most d^m different CNFs, which means the complexity for all CNFs associated to that subtree is $O(t\,d^t)$.

This can be repeated, until all m tree nodes are eliminated. At least then computation terminates (Lines 4, 5, or 16 or the algorithm). We can conclude that the total complexity is $O(m\,t\,d^t)$. □

References

[1] L. Getoor, B. Taskar (Eds.), An Introduction to Statistical Relational Learning, MIT Press, 2007.

[2] A. Kimmig, F. Costa, Link and node prediction in metabolic networks with probabilistic logic, in: Bisociative Knowledge Discovery, Springer, 2012, pp. 407–426.

[3] B. Moldovan, P. Moreno, M. van Otterlo, J. Santos-Victor, L. De Raedt, Learning relational affordance models for robots in multi-object manipulation tasks, in: 2012 IEEE International Conference on Robotics and Automation, ICRA, IEEE, 2012, pp. 4373–4378.

[4] S. Michels, M. Velikova, A. Hommersom, P.J. Lucas, A decision support model for uncertainty reasoning in safety and security tasks, in: 2013 IEEE International Conference on Systems, Man, and Cybernetics, SMC, IEEE, 2013, pp. 663–668.

[5] T. Sato, A statistical learning method for logic programs with distribution semantics, in: ICLP, 1995, pp. 715–729.

[6] L.D. Raedt, A. Kimmig, H. Toivonen, ProbLog: a probabilistic prolog and its application in link discovery, in: Proc. of the 20th International Joint Conference on Artificial Intelligence, AAAI Press, 2007, pp. 2468–2473.

[7] T. Sato, Y. Kameya, PRISM: a language for symbolic-statistical modeling, in: Proc. of the 15th International Joint Conference on Artificial Intelligence, 1997, pp. 1330–1335.

[8] D. Poole, The independent choice logic and beyond, in: L. De Raedt, P. Frasconi, K. Kersting, S. Muggleton (Eds.), Probabilistic Inductive Logic Programming, Springer-Verlag, Berlin, Heidelberg, 2008, pp. 222–243, http://dl.acm.org/citation.cfm?id=1793956.1793966.

[9] 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.

[10] J. Pearl, Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference, Morgan Kaufmann, 1988.

[11] M. Richardson, P. Domingos, Markov logic networks, Mach. Learn. 62 (1–2) (2006) 107–136.

[12] J. Wang, P. Domingos, Hybrid Markov logic networks, in: Proceedings of the 23rd National Conference on Artificial Intelligence, vol. 2, AAAI'08, AAAI Press, 2008, pp. 1106–1111, http://dl.acm.org/citation.cfm?id=1620163.1620244.

[13] B. Gutmann, M. Jaeger, L. De Raedt, Extending ProbLog with continuous distributions, in: P. Frasconi, F.A. Lisi (Eds.), Proc. of the 20th International Conference on Inductive Logic Programming, ILP-10, Firenze, Italy, 2010.

[14] 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.

[15] J. Jaffar, J. Lassez, Constraint logic programming, in: Proceedings of the 14th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages, ACM, 1987, pp. 111–119.

[16] N. Nilsson, Probabilistic logic, Artif. Intell. 28 (1) (1986) 71–87.

[17] S. Michels, A. Hommersom, P.J.F. Lucas, M. Velikova, P.W.M. Koopman, Inference for a new probabilistic constraint logic, in: F. Rossi (Ed.), IJCAI, IJCAI/AAAI, 2013.

[18] M. Chavira, A. Darwiche, On probabilistic inference by weighted model counting, Artif. Intell. 172 (6–7) (2008) 772–799.

[19] D. Fierens, G. Van den Broeck, I. Thon, B. Gutmann, L. De Raedt, Inference in probabilistic logic programs using weighted CNF's, in: F. Gagliardi Cozman, A. Pfeffer (Eds.), Proceedings of the 27th Conference on Uncertainty in Artificial Intelligence, UAI, 14–17 July 2011, Barcelona, Spain, 2011, pp. 211–220.

[20] J. Binder, D. Koller, S. Russell, K. Kanazawa, P. Smyth, Adaptive probabilistic networks with hidden variables, Mach. Learn. (1997) 213–244.

[21] R. Valdez, P. Yoon, T. Liu, M. Khoury, Family history and prevalence of diabetes in the us population the 6-year results from the national health and nutrition examination survey (1999–2004), Diabetes Care 30 (10) (2007) 2517–2522.

[22] J. Lloyd, Foundations of Logic Programming, 2nd edition, Springer, 1987.

[23] E. Grädel, On transitive closure logic, in: Computer Science Logic: 5th Workshop, CSL'91, vol. 626, Springer, 1992, pp. 149–163.

[24] K.R. Apt, H.A. Blair, A. Walker, Towards a theory of declarative knowledge, IBM TJ Watson Research Center, 1986.

[25] A. Van Gelder, K.A. Ross, J.S. Schlipf, The well-founded semantics for general logic programs, J. ACM 38 (3) (1991) 619–649.
[26] M. Gelfond, V. Lifschitz, The stable model semantics for logic programming, in: ICLP/SLP, vol. 88, 1988, pp. 1070–1080.
[27] A.N. Kolmogorov, Foundations of the Theory of Probability, 1960.
[28] P. Walley, Towards a unified theory of imprecise probability, Int. J. Approx. Reason. 24 (2) (2000) 125–148.
[29] I. Levi, The Enterprise of Knowledge, 1980.
[30] J. Neveu, Mathematical Foundations of the Calculus of Probability, Holden-Day, San Francisco, 1965.
[31] T. Sato, Y. Kameya, New advances in logic-based probabilistic modeling by prism, in: Probabilistic Inductive Logic Programming, Springer, 2008, pp. 118–155.
[32] K. Weichselberger, T. Augustin, On the symbiosis of two concepts of conditional interval probability, in: ISIPTA, vol. 3, 2003, pp. 608–629.
[33] C.H. Papadimitriou, On the complexity of integer programming, J. ACM 28 (4) (1981) 765–768.
[34] J.H. Davenport, J. Heintz, Real quantifier elimination is doubly exponential, J. Symb. Comput. 5 (1) (1988) 29–35.
[35] P.V. Hentenryck, Constraint Satisfaction in Logic Programming, The MIT Press, 1989.
[36] J. Jaffar, S. Michaylov, P.J. Stuckey, R.H.C. Yap, The CLP(R) language and system, ACM Trans. Program. Lang. Syst. 14 (3) (1992) 339–395, http://dx.doi.org/10.1145/129393.129398.
[37] J. Jaffar, M.J. Maher, K. Marriott, P.J. Stuckey, The semantics of constraint logic programs, J. Funct. Logic Program. 37 (1–3) (1998) 1–46, http://dx.doi.org/10.1016/S0743-1066(98)10002-X.
[38] T. Janhunen, Representing normal programs with clauses, in: ECAI, vol. 16, 2004, p. 358.
[39] T. Mantadelis, G. Janssens, Dedicated tabling for a probabilistic setting, in: ICLP (Technical Communications), 2010, pp. 124–133.
[40] D. Poole, N.L. Zhang, Exploiting contextual independence in probabilistic inference, J. Artif. Intell. Res. 18 (2003) 263–313.
[41] A. Darwiche, Recursive conditioning, Artif. Intell. 126 (1) (2001) 5–41.
[42] F. Jensen, S. Anderson, Approximations in Bayesian belief universe for knowledge based systems, in: Proceedings of the Sixth Conference Annual Conference on Uncertainty in Artificial Intelligence, UAI-90, AUAI Press, Corvallis, Oregon, 1990, pp. 162–169.
[43] C. Boutilier, N. Friedman, M. Goldszmidt, D. Koller, Context-specific independence in Bayesian networks, in: Proceedings of the Twelfth International Conference on Uncertainty in Artificial Intelligence, Morgan Kaufmann Publishers Inc., 1996, pp. 115–123.
[44] R.J. Bayardo, J.D. Pehoushek, Counting models using connected components, in: AAAI/IAAI, 2000, pp. 157–162.
[45] T. Sang, P. Beame, H. Kautz, Heuristics for fast exact model counting, in: Theory and Applications of Satisfiability Testing, Springer, 2005, pp. 226–240.
[46] L. De Moura, N. Bjørner, Z3: an efficient SMT solver, in: Proceedings of the Theory and Practice of Software, 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS'08/ETAPS'08, Springer-Verlag, Berlin, Heidelberg, 2008, pp. 337–340, http://dl.acm.org/citation.cfm?id=1792734.1792766.
[47] B. Dutertre, L.D. Moura, The Yices SMT solver, Tech. rep., Computer Science Laboratory, SRI International, 2006.
[48] C.P. De Campos, F.G. Cozman, The inferential complexity of Bayesian and credal networks, in: IJCAI, vol. 5, 2005, pp. 1313–1318.
[49] P.J. Downey, R. Sethi, R.E. Tarjan, Variations on the common subexpression problem, J. ACM 27 (4) (1980) 758–771.
[50] N. Karmarkar, A new polynomial-time algorithm for linear programming, in: Proceedings of the Sixteenth Annual ACM Symposium on Theory of Computing, ACM, 1984, pp. 302–311.
[51] Y.V. Matiyasevich, Diophantine representation of recursively enumerable predicates, in: J. Fenstad (Ed.), Second Scandinavian Logic Symposium, North-Holland Publishing Company, 1971, pp. 171–177.
[52] H.L. Bodlaender, A tourist guide through treewidth, Technical report RUU-CS 92, 1992.
[53] M. Samer, S. Szeider, Algorithms for propositional model counting, J. Discrete Algorithms 8 (1) (2010) 50–64.
[54] R. Diestel, Graph Theory, 3rd edition, Grad. Texts Math., vol. 173, Springer-Verlag, Berlin, 2005.
[55] S. Arnborg, D.G. Corneil, A. Proskurowski, Complexity of finding embeddings in ak-tree, SIAM J. Algebr. Discrete Methods 8 (2) (1987) 277–284.
[56] H.L. Bodlaender, A linear time algorithm for finding tree-decompositions of small treewidth, in: Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing, ACM, 1993, pp. 226–234.
[57] T. Sang, P. Beame, H. Kautz, Solving Bayesian networks by weighted model counting, in: Proceedings of the Twentieth National Conference on Artificial Intelligence, AAAI-05, vol. 1, 2005, pp. 475–482.
[58] D. Fierens, G. Van den Broeck, J. Renkens, D. Shterionov, B. Gutmann, I. Thon, G. Janssens, L. De Raedt, Inference and learning in probabilistic logic programs using weighted Boolean formulas, Theory Pract. Log. Program. (2014) 1–44, First view.
[59] A. Darwiche, New advances in compiling CNF to decomposable negation normal form, in: Proc. of ECAI, Citeseer, 2004, pp. 328–332.
[60] C. Muise, S.A. Mcilraith, J.C. Beck, E. Hsu, DSHARP: fast d-DNNF compilation with sharpSAT, in: Canadian Conference on Artificial Intelligence, 2012.
[61] K. Laskey, MEBN: a language for first-order Bayesian knowledge bases, Artif. Intell. 172 (2–3) (2008) 140–178.
[62] A. McCallum, K. Schultz, S. Singh, Factorie: probabilistic programming via imperatively defined factor graphs, in: Advances in Neural Information Processing Systems, 2009, pp. 1249–1257.
[63] A. Pfeffer, Figaro: an object-oriented probabilistic programming language, Technical report, Charles River Analytics, 2009, p. 137.
[64] N. Goodman, V. Mansinghka, D. Roy, K. Bonawitz, J. Tenenbaum, Church: a language for generative models, in: Proceedings of the 24th Conference on Uncertainty in Artificial Intelligence, UAI 2008, 2008, pp. 220–229.
[65] B. Milch, B. Marthi, S.J. Russell, D. Sontag, D.L. Ong, A. Kolobov, BLOG: probabilistic models with unknown objects, in: Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, IJCAI-05, July 30–August 5, 2005, Edinburgh, Scotland, UK, 2005, pp. 1352–1359, http://www.ijcai.org/papers/1546.pdf.
[66] S.L. Lauritzen, Propagation of probabilities, means, and variances in mixed graphical association models, J. Am. Stat. Assoc. 87 (420) (1992) 1098–1108.
[67] F.G. Cozman, Credal networks, Artif. Intell. 120 (2) (2000) 199–233.
[68] R. Ng, V.S. Subrahmanian, Probabilistic logic programming, Inf. Comput. 101 (2) (1992) 150–201.
[69] S. Michels, A. Hommersom, P.J.F. Lucas, M. Velikova, Imprecise probabilistic horn clause logic, in: ECAI 2014: 21st European Conference on Artificial Intelligence, Including Prestigious Applications of Intelligent Systems, PAIS 2014, 18–22 August 2014, Prague, Czech Republic, 2014, pp. 621–626.
[70] P. Domingos, S. Kok, D. Lowd, H. Poon, M. Richardson, P. Singla, Markov logic, in: Probabilistic Inductive Logic Programming, 2008, pp. 92–117.
[71] V.S. Costa, D. Page, M. Qazi, J. Cussens, CLP (BN): constraint logic programming for probabilistic knowledge, in: Proceedings of the Nineteenth Conference on Uncertainty in Artificial Intelligence, Morgan Kaufmann Publishers Inc., 2002, pp. 517–524.
[72] V.S. Costa, A. Paes, On the relationship between PRISM and CLP($\mathcal{BN}$), in: Proc. of the Int. Workshop on Statistical Relational Learning, SRL-2009, 2009.
[73] N. Angelopoulos, clp(pdf(y)): constraints for probabilistic reasoning in logic programming, in: F. Rossi (Ed.), Principles and Practice of Constraint Programming 2003, in: Lect. Notes Comput. Sci., vol. 2833, Springer, Berlin, Heidelberg, 2003, pp. 784–788.
[74] S. Reizler, Probabilistic constraint logic programming, Ph.D. thesis, University of Tubingen, Tubingen, Germany, 1998.
[75] T. Sato, M. Ishihata, K. Inoue, Constraint-based probabilistic modeling for statistical abduction, Mach. Learn. 83 (2) (2011) 241–264.
[76] L. van der Gaag, On probability intervals and their updating, Technical report, Department of Computer Science, University of Utrecht, 1990, http://books.google.nl/books?id=nH9JGwAACAAJ.
[77] F.G. Cozman, C.P. De Campos, J.S. Ide, J.C.F. da Rocha, Propositional and relational Bayesian networks associated with imprecise and qualitative probabilistic assessments, in: Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence, AUAI Press, 2004, pp. 104–111.

[78] M.A. Islam, C.R. Ramakrishnan, I.V. Ramakrishnan, Inference in probabilistic logic programs with continuous random variables, Theory Pract. Log. Program. 12 (4–5) (2012) 505–523.

[79] H. Langseth, T.D. Nielsen, A. Salmerón, et al., Mixtures of truncated basis functions, Int. J. Approx. Reason. 53 (2) (2012) 212–227.

[80] S. Sanner, E. Abbasnejad, Symbolic variable elimination for discrete and continuous graphical models, in: AAAI, 2012.

[81] R. Dechter, Reasoning with probabilistic and deterministic graphical models: exact algorithms, Synth. Lect. Artif. Intell. Mach. Learn. 7 (3) (2013) 1–191.

[82] K.P. Murphy, Y. Weiss, M.I. Jordan, Loopy belief propagation for approximate inference: an empirical study, in: Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence, Morgan Kaufmann Publishers Inc., 1999, pp. 467–475.

[83] J.S. Yedidia, W.T. Freeman, Y. Weiss, Constructing free-energy approximations and generalized belief propagation algorithms, IEEE Trans. Inf. Theory 51 (7) (2005) 2282–2312.

[84] S.C. Tatikonda, M.I. Jordan, Loopy belief propagation and Gibbs measures, in: Proceedings of the Eighteenth Conference on Uncertainty in Artificial Intelligence, Morgan Kaufmann Publishers Inc., 2002, pp. 493–500.

[85] J.M. Mooij, H.J. Kappen, Sufficient conditions for convergence of loopy belief propagation, in: Proceedings of the 21st Conference in Uncertainty in Artificial Intelligence, UAI '05, July 26–29, 2005, Edinburgh, Scotland, 2005, pp. 396–403.

[86] D. Poole, Logic programming, abduction and probability, New Gener. Comput. 11 (3–4) (1993) 377–400.

[87] J. Renkens, A. Kimmig, G. Van den Broeck, L. De Raedt, Explanation-based approximate weighted model counting for probabilistic logics, in: Proceedings of the 28th AAAI Conference on Artificial Intelligence, 2014.

[88] S. Kullback, R.A. Leibler, On information and sufficiency, Ann. Math. Stat. 22 (1) (1951) 79–86.

[89] N.S. Arora, R. de Salvo Braz, E.B. Sudderth, S.J. Russell, Gibbs sampling in open-universe stochastic languages, in: Proceedings of the Twenty-Sixth Conference on Uncertainty in Artificial Intelligence, UAI 2010, July 8–11, 2010, Catalina Island, CA, USA, 2010, pp. 30–39.

[90] D. Nitti, T.D. Laet, L.D. Raedt, A particle filter for hybrid relational domains, in: 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems, November 3–7, 2013, Tokyo, Japan, 2013, pp. 2764–2771.

[91] M.D. Hoffman, A. Gelman, The no-u-turn sampler: adaptively setting path lengths in Hamiltonian Monte Carlo, J. Mach. Learn. Res. 15 (1) (2014) 1593–1623, http://dl.acm.org/citation.cfm?id=2638586.

[92] L. Li, B. Ramsundar, S. Russell, Dynamic scaled sampling for deterministic constraints, in: Proceedings of the Sixteenth International Conference on Artificial Intelligence and Statistics, 2013, pp. 397–405.

[93] J.G. Propp, D.B. Wilson, Exact sampling with coupled Markov chains and applications to statistical mechanics, Random Struct. Algorithms 9 (1–2) (1996) 223–252.

[94] S. Brooks, A. Gelman, G. Jones, X.-L. Meng, Handbook of Markov Chain Monte Carlo, CRC Press, 2011.

[95] K. Kersting, Lifted probabilistic inference, in: ECAI, 2012, pp. 33–38.

[96] J. Choi, E. Amir, D.J. Hill, Lifted inference for relational continuous models, in: UAI, vol. 10, 2010, pp. 126–134.

[97] T. Hendriks, P. van de Laar, Metis: dependable cooperative systems for public safety, Proc. Comput. Sci. 16 (2013) 542–551.

[98] M. Velikova, P. Novák, B. Huijbrechts, J. Laarhuis, J. Hoeksma, S. Michels, An integrated reconfigurable system for maritime situational awareness, in: ECAI 2014: 21st European Conference on Artificial Intelligence, Including Prestigious Applications of Intelligent Systems, PAIS 2014, 18–22 August 2014, Prague, Czech Republic, 2014, pp. 1197–1202.