Exact and Approximate Weighted Model Integration with Probability Density Functions Using Knowledge Compilation

Pedro Zuidberg Dos Martires, Anton Dries, Luc De Raedt
KU Leuven, Belgium
firstname.lastname@cs.kuleuven.be

Abstract

Weighted model counting has recently been extended to weighted model integration, which can be used to solve hybrid probabilistic reasoning problems. Such problems involve both discrete and continuous probability distributions. We show how standard knowledge compilation techniques (to SDDs and d-DNNFs) apply to weighted model integration, and use it in two novel solvers, one exact and one approximate solver. Furthermore, we extend the class of employable weight functions to actual probability density functions instead of mere polynomial weight functions.

1 Introduction

The state-of-the-art method for inference in probabilistic graphical models reduces inference to weighted model counting (WMC) (Chavira and Darwiche 2008), while utilizing knowledge compilation (KC) (Darwiche and Marquis 2002). Knowledge compilation transforms the logical structure underlying a graphical model into an equivalent target representation. Although the knowledge compilation step itself is computationally hard, answering queries in the target representation only requires **polytime**.

Standard weighted model counting only supports discrete probability distributions. To repair this omission, WMC has recently been extended towards weighted model integration (WMI) (Belle, Passerini, and Van den Broeck 2015), supporting additionally continuous variables. However, the weight functions supported within current formulations of WMI (Belle, Passerini, and Van den Broeck 2015; Belle et al. 2016; Morettin, Passerini, and Sebastiani 2017; Kolb et al. 2018) allow only for piecewise polynomial functions. Moreover, none of these prior works has studied the applicability of knowledge compilation to WMI.

The key contribution of this paper is that we show how to handle actual probability density functions instead of piecewise polynomials in the context of WMI by applying standard knowledge compilation techniques. To this end, we cast weighted model integration within the framework of algebraic model counting (AMC) (Kimmig, Van den Broeck, and De Raedt 2017). More specifically, we make the following contributions:

1. We introduce the probability density semiring.
2. We show how this allows us to cast WMI within AMC and thereby to use the general body of literature on knowledge compilation.
3. We introduce *Symbo*, a solver for WMI that realizes knowledge compilation and **exact symbolic inference**.
4. We introduce *Sampo*, a solver for WMI that realizes knowledge compilation and **approximate inference** via sampling.

Symbo exploits the PSI-Solver by (Gehr, Misailovic, and Vechev 2016) to simplify algebraic expressions, while Sampo is based on mapping the arithmetic circuit that results from the KC step onto Edward (Tran et al. 2016), a probabilistic programming language wrapped around TensorFlow (Abadi et al. 2015). The latter transforms approximate inference in probabilistic programs into an embarrassingly parallelizable task.

2 Preliminaries

2.1 Weighted Model Integration

Compared to the well-known SAT problem, where the problem consists of deciding whether there is a satisfying assignment to a logical formula or not, an SMT problem generalizes SAT to additionally allowing the use of expressions formulated in a background theory.

Example 1. Consider the SMT theory `broken`:

$$\texttt{broken} \leftrightarrow (\texttt{no\_cool} \wedge (\texttt{t} > 20)) \vee (\texttt{t} > 30) \qquad (1)$$

where `no_cool` is a Boolean variable and `t` a real-valued variable. SMT then answers the question whether or not there is a satisfying assignment to the formula for the variables `no_cool` and `t`.

In this paper we consider *real arithmetic*, *non-linear real arithmetic* and *linear real arithmetic* SMT formulas.

Definition 1. (SMT($\mathcal{RA}$) (real arithmetics)) Let $\mathbb{R}$ denote the set of reals, $\mathbb{B} = \{\bot, \top\}$ the set of Boolean values, let B be a set of M Boolean and X a set of N real variables. An **atomic formula** is an expression of the form $g(X) \bowtie c$, where $c \in \mathbb{R}$, $\bowtie \in \{=, \neq, \geq, \leq, >, <\}$, and $g : \mathbb{R}^N \to \mathbb{R}$.

We then define SMT($\mathcal{RA}$) theories as Boolean combinations (by means of the standard Boolean operators

$\{\neg, \wedge, \vee, \rightarrow, \leftrightarrow\}$) of **Boolean variables** $b \in B$ and of **atomic formulas** over X.

We distinguish two special cases:

- SMT($\mathcal{NRA}$) (non-linear real arithmetics): atomic formulas take the form $\sum_i c_i \cdot x_i^{p_i} \bowtie c$, where the $x_i \in X$ and $c_i, c, p_i \in \mathbb{Q}$.
- SMT($\mathcal{LRA}$) (linear real arithmetics): $\sum_i c_i \cdot x_i \bowtie c$, where the $x_i \in X$ and $c_i, c \in \mathbb{Q}$.

We have introduced different kinds of SMT formulas as different WMI solvers are only applicable to certain SMT theories. For instance, the solver proposed by (Morettin, Passerini, and Sebastiani 2017) is only applicable to SMT($\mathcal{LRA}$), whereas Symbo can handle SMT($\mathcal{NRA}$) and Sampo even SMT($\mathcal{RA}$).

Definition 2. (Interpretation of SMT formula) Let J and K be two sets of variables and ϕ be an SMT formula over J and K. The set of total interpretations (or total assignments) that satisfy ϕ is the set of assignments to the elements in J and K that satisfy $\exists J, \exists K : \phi(J, K)$. We denote the set of total interpretations by $\mathcal{I}_{J,K}(\phi)$. The set of partial interpretations is denoted by $\mathcal{I}_J(\phi)$, which is the set of assignments to J that satisfy $\exists K : \phi(J, K)$. The set of total assignments to a partially interpreted formula is denoted by $\mathcal{I}_J(\phi^{\mathbf{k}})$, which denotes the set of assignments to the elements in J that satisfy $\phi(J, \mathbf{k})$, with $\mathbf{k} \in \mathcal{I}_K(\phi)$.

Consider again the theory `broken` (cf. Eq. 1). Assume that `t` is distributed according to: $\texttt{t} \sim \mathcal{N}_{\texttt{t}}(20, 5)$ and that the probability for `no_cool` being true is 0.01. Determining the probability of the formula being true extends the SMT problem to weighted model integration.

Definition 3. (Weighted model integration (WMI)) Given a set B of M Boolean variables, X of N real variables, a weight function $w : \mathbb{B}^M \times \mathbb{R}^N \rightarrow \mathbb{R}^+$, and an SMT formula ϕ over $B \cup X$, the **weighted model integral** is

$$WMI(\phi, w \mid X, B) = \sum_{\mathbf{b} \in \mathcal{I}_B(\phi)} \int_{\mathbf{x} \in \mathcal{I}_X(\phi^{\mathbf{b}})} w(\mathbf{x}, \mathbf{b}) d\mathbf{x} \quad (2)$$

For the remainder of the paper we will assume that the weight function factorizes as:

$$w(\mathbf{x}, \mathbf{b}) = w_x(\mathbf{x}) w_b(\mathbf{b}) = w_x(\mathbf{x}) \prod_{b_i \in \mathbf{b}} w_b(b_i) \quad (3)$$

with $w_b : \mathbb{B}^M \rightarrow \mathbb{R}$ and $w_x : \mathbb{R}^N \rightarrow \mathbb{R}$. We can assume this without loss of generality as any weight function that does not follow this factorization can be rewritten as a sum of weight functions over mutually exclusive partial assignments to the Boolean variables, where each individual term of the sum factorizes according to Eq. 3. The weighted model integral is then expressed as a sum over weighted model integrals. We additionally assume that the weight function $w_b(\mathbf{b})$ further factorizes as $\prod_{b_i \in \mathbf{b}} w_b(b_i)$. See also Definition 6.

2.2 Algebraic Model Counting

Definition 4. (Weighted model counting (WMC)). WMC is the special case of weighted model integration where the set of real variables is empty: $X = \emptyset$.

WMC is traditionally used for probabilistic inference in Bayesian networks (Chavira and Darwiche 2008) and probabilistic programming (Fierens et al. 2015) with a factorized weight function: $WMC(\phi, w|B) = \sum_{\mathbf{b} \in \mathcal{I}(\phi(B))} \prod_{b_i \in \mathbf{b}} w(b_i)$. Algebraic model counting (Kimmig, Van den Broeck, and De Raedt 2017) generalizes WMC to commutative semirings. More formally,

Definition 5. A **commutative semiring** is an algebraic structure $(\mathcal{A}, \oplus, \otimes, e^{\oplus}, e^{\otimes})$ equipping a set of elements $\mathcal{A}$ with addition and multiplication such that (1) addition $\oplus$ and multiplication $\otimes$ are binary operations $\mathcal{A} \times \mathcal{A} \rightarrow \mathcal{A}$; (2) addition $\oplus$ and multiplication are associative and commutative binary operations over the set $\mathcal{A}$; (3) $\otimes$ distributes over $\oplus$; (4) $e^{\oplus} \in \mathcal{A}$ is the neutral element of $\oplus$; (5) $e^{\otimes} \in \mathcal{A}$ is the neutral element of $\otimes$; and (6) $e^{\oplus}$ is an annihilator for $\otimes$.

Definition 6. (Algebraic model counting (AMC)) (Kimmig, Van den Broeck, and De Raedt 2017) Given:

- a propositional logic theory ϕ over a set of variables B
- a commutative semiring $(\mathcal{A}, \oplus, \otimes, e^{\oplus}, e^{\otimes})$
- a labeling function $\alpha : \mathcal{L} \rightarrow \mathcal{A}$, mapping literals $\mathcal{L}$ from the variables in B to values from the semiring set $\mathcal{A}$

The algebraic model count of a theory ϕ is then defined as:

$$AMC(\phi, \alpha|B) = \bigoplus_{\mathbf{b} \in \mathcal{I}_B(\phi)} \bigotimes_{b_i \in \mathbf{b}} \alpha(b_i)$$

We use α instead of w and the term label rather than weight to reflect that the elements of the semiring cannot always be interpreted as weights.

2.3 Knowledge Compilation

Knowledge compilation (Darwiche and Marquis 2002) can be regarded as the process of transforming a propositional logic formula into a form that allows for **polytime evaluation** of the formula. Although the knowledge compilation step itself is computationally hard, the overall procedure yields a net benefit when a logical circuit has to be evaluated multiple times, possibly with different weights for the literals.

A popular language to compile propositional formulas into are Sentential Decisions Diagrams (SDDs) (Choi, Kisa, and Darwiche 2013), which we also use to implement our two solvers Symbo and Sampo. SDDs are a subset of d-DNNF formulas (a graphical representation of an example SDD is illustrated in Figure 1). SDDs and d-DNNFs are well-known target langauges for knowledge compilation and they have been used in state-of-the-art inference engines for Bayesian networks. In order to guarantee a correct evaluation of a compiled propositional formula, we require the neutral-sum property to hold:

Definition 7. (Neutral-sum property) A semiring addition and labeling function pair $(\oplus, \alpha)$ is neutral *if and only if* $\forall b \in B : \alpha(b) \oplus \alpha(\neg b) = e^{\otimes}$. (Kimmig, Van den Broeck, and De Raedt 2017)

Theorem 1. (AMC on d-DNNF) ((Kimmig, Van den Broeck, and De Raedt 2017, Theorem 4)) Evaluating a d-DNNF representation of the propositional theory ϕ, using Algorithm 1 in (Kimmig, Van den Broeck, and De Raedt 2017), for a semiring and labeling function with neutral tuple $(\oplus, \alpha)$ is a correct computation (cf. (Kimmig, Van den Broeck, and De Raedt 2017, Definition 10)) of the algebraic model count.

3 The Probability Density Semiring

We are now going to define the probability density semiring and the labeling function, cf. Definition 6. This will allow us to cast WMI as AMC.

Definition 8. (Atomic formula abstraction) Let $c(X)$ be an atomic formula (cf. Definition 1), $abs_{c(X)}$ is then called the *atomic formula abstraction* of c, given that $(abs_{c(X)} \leftrightarrow \exists X.c(X))$ holds.

Definition 9. (Labeling function α) Let l be a literal. Then the label of the literal l is given by:

$$\alpha(l) \coloneqq \begin{cases} (p(l), \emptyset) & \text{if } l \text{ Boolean variable} \\ ([c(X)], X) & \text{if } l \text{ is an atomic formula abstraction} \end{cases}$$

In the former case, $p(l)$ denotes the probability for l being true and in the latter case, $c(X)$ denotes the condition of which l is the abstraction.

The label of a negated literal $\neg l$ is given by:

$$\alpha(\neg l) \coloneqq \begin{cases} (1 - p(l)), \emptyset) & \text{if } l \text{ is a Boolean variable} \\ ([\neg c(X)], X) & \text{if } l \text{ atomic formula abstraction} \end{cases}$$

The brackets [.] around $[c(X)]$ denote the so-called *Iverson brackets* (Knuth 1992). They evaluate to 1 if their argument $c(X)$ evaluates to true and to 0 otherwise.

Example 2. Applying the labeling function α to the literals in our running example yields, for example: $\alpha(\texttt{no\_cool}) = (0.01, \emptyset)$ and $\alpha(abs_{\texttt{t}>20}) = ([\texttt{t} > 20], \{\texttt{t}\})$.

Definition 10. (Probability density semiring $\mathcal{S}$) The elements of the semiring $\mathcal{S}$ are given by the set

$$\mathcal{A} \coloneqq \{(a, \mathcal{V}(a))\} \tag{4}$$

where a denotes any algebraic expression over $\mathcal{RA}$ and $\mathcal{V}(a)$ the set of real variables occurring in a. The neutral elements $e^{\oplus}$ and $e^{\otimes}$ are defined as:

$$e^{\oplus} \coloneqq (0, \emptyset) \qquad e^{\otimes} \coloneqq (1, \emptyset) \tag{5}$$

For the addition and multiplication we define:

$$(a_1, \mathcal{V}(a_1)) \oplus (a_2, \mathcal{V}(a_2)) \coloneqq (a_1 + a_2, \mathcal{V}(a_1 + a_2)) \tag{6}$$

$$(a_1, \mathcal{V}(a_1)) \otimes (a_2, \mathcal{V}(a_2)) \coloneqq (a_1 \times a_2, \mathcal{V}(a_1 \times a_2)) \tag{7}$$

Example 3. An example of an algebraic expression over $\mathcal{RA}$ would be $0.01 \times [s + 20 < t] \times [t \leq 30] + [t > 30]$, $s \in \mathbb{R}$, $t \in \mathbb{R}$.

Lemma 1. The structure $\mathcal{S} = (\mathcal{A}, \oplus, \otimes, e^{\oplus}, e^{\otimes})$ is a commutative semiring.

Proof (Sketch). We need to show that the properties in Definition 5 hold. The proof relies on the commutativity and associativity of the Iverson brackets under standard addition and multiplication. Similarly for the distributivity of the multiplication over the addition (cf. Property 3). Lastly, properties 4 to 6 are trivially satisfied. We conclude that the structure $\mathcal{S}$ is indeed a commutative semiring. □

Lemma 2. The pair $(\oplus, \alpha)$ is neutral, i.e. $\alpha(l) \oplus \alpha(\neg l) = e^{\otimes}$, where l is a literal.

Proof. We have two cases:

1. l is a Boolean variable, $\alpha(l) \oplus \alpha(\neg l) = (P(l), \emptyset) \oplus (1 - P(l), \emptyset) = (1, \emptyset)$.
2. l is an atomic formula: $\alpha(l) \oplus \alpha(\neg l) = ([l], \mathcal{V}([l])) \oplus ([\neg l], \mathcal{V}([\neg l])) = ([l] + [\neg l], \mathcal{V}([l] + [\neg l])) = ([\top], \mathcal{V}([\top])) = (1, \emptyset)$ □

Lemma 3. (AMC on d-DNNF with $\mathcal{S}$) The algebraic model count is a correct calculation on a d-DNNF representation of a logic formula given the density semiring $\mathcal{S}$.

Proof. This follows immediately from Lemma 1 and 2, together with Theorem 1. □

4 WMI via AMC

A key difference between WMI and AMC is that in an AMC task there is no integral. This intuitively implies that we need to perform an integration on the algebraic model count if we want to cast WMI using AMC: "WMI = $\int$ AMC". Additionally, WMI is defined on SMT formulas and AMC on propositional logic formulas. We address these differences in theorem 2, which also allows us to show that WMI can be cast as AMC.

Theorem 2. Let ϕ be an SMT($\mathcal{RA}$) formula over the Boolean variables in the set B and continuous variables in the set X. Let ϕ_a be the propositional logic formula over the set of Boolean variables B and B_X, where B_X is the set of abstractions of atomic formulas (cf. Definition 8) in ϕ. Let w be a weight function over the Boolean variables in B and the continuous variables in X. Furthermore, let $AMC(\phi_a, \alpha | B_X \cup B)$ evaluate to $(\Psi, \mathcal{V}(\Psi))$ in the semiring $\mathcal{S}$, with $\Psi = \sum_{\mathbf{v} \in \mathcal{I}_{B,B_X}(\phi_a)} \prod_{v_i \in \mathbf{v}} a_{v_i}$. Then

$$WMI(\phi, w | X, B) \coloneqq \int_{\mathbf{x} \in \mathbf{X}} \Psi w_x(\mathbf{x}) d\mathbf{x} \tag{8}$$

where $\mathbf{X}$ is the set of all possible assignments to the variables in $\mathcal{V}(\Psi)$.

Proof In the first step we rewrite Ψ (an example is given in Example 3) as the sum-product over the algebraic expressions a_v. The a_v are the probability or Iverson labels of the literals from Definition 9. In the second step (P2 to P3) we split up the sum and the product over the variables $\mathbf{v}$ into sums over the abstractions of atomic formulas a_{x_i} and atomic propositions a_{b_i} - likewise for the product. b_i and x_j denote the assignment to a specific variable in the set of assignments $\mathbf{b}$ and $\mathbf{x_a}$ respectively. The superscript $\mathbf{b}$ in $\phi_a^{\mathbf{b}}$ indicates a specific assignment to the Boolean variables corresponding to the atomic propositions. Next (P3 to P4), we push the product over the atomic propositions through and note that this product corresponds to the weight of the Boolean variables $w_b(\mathbf{b})$.

$$\begin{aligned}
&\int_{\mathbf{x} \in \mathbf{X}} \Psi w_x(\mathbf{x}) d\mathbf{x} && \text{P1} \\
&= \int_{\mathbf{x} \in \mathbf{X}} \left(\sum_{\mathbf{v} \in \mathcal{I}_{B,B_X}(\phi_a)} \prod_{v_i \in \mathbf{v}} a_{v_i} \right) w_x(\mathbf{x}) d\mathbf{x} && \text{P2} \\
&= \int_{\mathbf{x} \in \mathbf{X}} \sum_{\mathbf{b} \in \mathcal{I}_B(\phi_a)} \sum_{\mathbf{x_a} \in \mathcal{I}_{B_X}(\phi_a^{\mathbf{b}})} \prod_{b_i \in \mathbf{b}, x_j \in \mathbf{x_a}} a_{b_i} a_{x_j} w_x(\mathbf{x}) d\mathbf{x} && \text{P3} \\
&= \int_{\mathbf{x} \in \mathbf{X}} \sum_{\mathbf{b} \in \mathcal{I}_B(\phi)} \sum_{\mathbf{x_a} \in \mathcal{I}_{B_X}(\phi_a^{\mathbf{b}})} \prod_{x_j \in \mathbf{x_a}} a_{x_j} w_b(\mathbf{b}) w_x(\mathbf{x}) d\mathbf{x} && \text{P4} \\
&= \sum_{\mathbf{b} \in \mathcal{I}_B(\phi)} \int_{\mathbf{x} \in \mathbf{X}} \sum_{\mathbf{x_a} \in \mathcal{I}_{B_X}(\phi_a^{\mathbf{b}})} \prod_{x_j \in \mathbf{x_a}} a_{x_j} w(\mathbf{x}, \mathbf{b}) d\mathbf{x} && \text{P5} \\
&= \sum_{\mathbf{b} \in \mathcal{I}_B(\phi)} \int_{\mathbf{x} \in \mathcal{I}_X(\phi^{\mathbf{b}})} w(\mathbf{x}, \mathbf{b}) d\mathbf{x} && \text{P6}
\end{aligned}$$

In P5 we exchanged the summation and the integration (assuming that Fubini's theorem (Fubini 1907) holds). We also rewrote the product of the weight functions for the Booleans and for the continuous variables as a single weight function, assuming that the weight function factorizes accordingly. The integral over the so-obtained sum-product is the integral over Iverson brackets. In P6 we rewrite the indefinite integral over the Iverson brackets as the definite integral with boundary conditions corresponding to the conditions present in the Iverson brackets. This corresponds to the definition of the weighted model integral. □

We have shown that we can solve a WMI problem by formulating it as an AMC problem, given that the weight function is factorizable. The weighted model integral for a non-factorizable weight function is then obtained by adding up the weighted model integrals for the factorizeable weight functions into which the problem decomposes.

5 Probability of SMT Formulas

We describe now **Symbo** and **Sampo**, algorithms that, respectively, produce the exact and the approximate weighted model integral of an SMT formula ϕ and a factorizable weight function w utilizing knowledge compilation. Implementations of both algorithms are available under `https://bitbucket.org/pedrozudo/hal_problog`.

5.1 Symbo

In Lemma 3 we saw that the probability semiring S can be used to calculate the algebraic model count on a d-DNNF representation of a logical formula. Recalling Theorem 2, we are hence also capable of obtaining the weighted model integral for an SMT formula, given the probability distributions of the random variables.

Algorithm 1. (Symbo) Symbo computes the weighted model integral of an SMT($\mathcal{NRA}$) formula ϕ for a factorizable weight function w by executing the following steps:

1. Abstract all atomic formulas in ϕ according to Definition 8 and obtain ϕ_a.
2. Compile ϕ_a into a d-DNNF representation ϕ_{compiled}.
3. Transform ϕ_{compiled} into an arithmetic circuit AC_ϕ by replacing logical and/or operations with symbolic multiplications/additions.
4. Label literals in AC_ϕ according to the labeling function given in Definition 9 with corresponding symbolic values.
5. Symbolically evaluate AC_ϕ and obtain $(\Psi, \mathcal{V}(\Psi))$.
6. Multiply Ψ by the weight of the continuous variables in $\mathcal{V}(\Psi)$.
7. Symbolically integrate over the continuous variables by calling a symbolic inference engine.

We implemented Algorithm 1 using the SDD package [1] for the KC step and the inference engine of the PSI-Solver for symbolic manipulations [2].

Example 4. Consider our initial example in Eq. 1. Executing the first two steps of Symbo yield the compiled logic formula that is shown on the left in Figure 1. Steps number three and four of Symbo produce the arithmetic circuit on the right in Figure 1. The probability for the theory `broken`, which

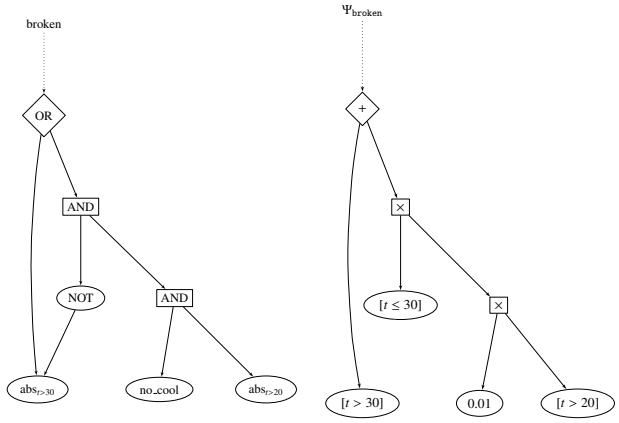


Figure 1: Shown on the left is a graphical representation of the compiled logic formula given in Eq. 1 (in the SDD target language), where the atomic formulas have been abstracted away. On the right we see the corresponding arithmetic circuit where the literals have been replaced by corresponding labels according to the labeling function in 9 and where the logic and/or operation have been replaced by ×/+ respectively. Note, the labels in the arithmetic circuit do not explicitly state the set of continuous variables involved.

coincides with the weighted model integral, is obtained by evaluating the arithmetic circuit (step five), multiplying this expression by the probability density function for `t` (step six) and carrying out the integral (step 7).

$$
\begin{aligned}
& p(\texttt{broken}) \\
&= \int (0.01[\texttt{t>20}][\texttt{t}\leq 30] + [\texttt{t>30}])\, \mathcal{N}_{\texttt{t}}(20,5) d\texttt{t} \\
&= 0.01 \int_{20<\texttt{t}\leq 30} \mathcal{N}_{\texttt{t}}(20,5) d\texttt{t} + \int_{\texttt{t}>30} \mathcal{N}_{\texttt{t}}(20,5) d\texttt{t} \\
&= 1 - 0.01 \int_{-\infty}^{-\frac{5\sqrt{8}}{2}+\frac{20}{\sqrt{8}}} e^{-x^2} dx - 0.99 \int_{-\infty}^{-\frac{5\sqrt{8}}{2}+\frac{30}{\sqrt{8}}} e^{-x^2} dx
\end{aligned}
$$

In Example 4, the weight function on the continuous variables depended only on a single variable. It is, however, easy to see that our formalism does also allow for multivariate distributions that are then used with more intricated integration bounds, such as in Example 3.

5.2 Sampo

In the general case, symbolic inference methods are not able to produce numerical results to a given problem. This is because the resulting integrals are not tractable utilizing symbolic integration. For such cases Monte Carlo (MC) methods are used to compute intractable integrals by approximating the integration by a summation.

[1] `http://reasoning.cs.ucla.edu/sdd/`

[2] For a detailed discussion of allowed symbolic manipulations see (Gehr, Misailovic, and Vechev 2016)

Theorem 3. (MC approximation of WMI) Let ϕ be an SMT($\mathcal{RA}$) theory, w a factorizable weight function over the Boolean variables B and continuous variables X. Furthermore, let $AMC(\phi, w|X \cup B)$ evaluate to $(\Psi, \mathcal{V}(\Psi))$. Then the Monte Carlo approximation of WMI$(\phi, w|X, B)$ is given by:

$$WMI_{\text{MC}}(\phi, w|X, B) \coloneqq \frac{1}{N} \sum_{i=1}^{N} \Psi(\boldsymbol{x}_i) \tag{11}$$

where the $\boldsymbol{x}_i$'s are N independent and identically distributed random variables drawn from the density w.

Proof.

$$\begin{aligned} WMI(\phi, w|X, B) &= \int_{\mathbf{x}\in X} \Psi(\mathbf{x}) w_x(\mathbf{x}) d\mathbf{x} && \text{P1} \\ &= E_{w_x(\mathbf{x})}[\Psi(\mathbf{x})] && \text{P2} \\ &\approx \frac{1}{N} \sum_{i=1}^{N} \Psi(\mathbf{x}_i) && \text{P3} \end{aligned}$$

The expression in P2 denotes the expectation of $\Psi(\mathbf{x})$ with respect to $w(\mathbf{x})$. The approximation in P3 is the mean value of Ψ obtained through MC assignments to the continuous random variables present in Ψ. □

The MC approximation of the weighted model integral of an SMT formula necessitates that we evaluate a compiled SMT problem at N different points, i.e. we need to evaluate a compiled theory N times with different weights. This is exactly where the strength of knowledge compilation lies: expensively compile once and cheaply evaluate often.

Numerical computation libraries such as TensorFlow rely heavily on the concept of computation graphs. Realizing that we can translate a d-DNNF formula to a computational graph and express the labels of literals in an SMT formula as tensors, allows us to compute the N evaluations necessary for the MC approximation of the weighted model integral not only cheaply but also in parallel.

Algorithm 2. Sampo computes the weighted model integral of an SMT($\mathcal{RA}$) formula ϕ for a factorizable weight function w by executing the following steps:

1. Abstract all atomic formulas in ϕ according to Definition 8 and obtain ϕ_a.
2. Compile ϕ_a into a d-DNNF representation ϕ_{compiled}.
3. Transform ϕ_{compiled} into an arithmetic circuit AC_ϕ, i.e. replacing logical and/or operations with elementwise tensor multiplications/additions.
4. Label the literals in AC_ϕ according to the labeling function given in Definition 9 with corresponding tensors.
5. Symbolically evaluate AC_ϕ and obtain $(\Psi, \mathcal{V}(\Psi))$ represented by a computation graph CG.
6. Run the CG representing $(\Psi, \mathcal{V}(\Psi))$ N times, where N is the number of samples approximating the probability densities.
7. Take the mean of the values of the N runs of the CG.

We implemented Algorithm 2 using again the SDD package for the KC step and using TensorFlow as the underlying symbolico-numerical computation library. Random variables are sampled using the Edward library.

Example 5. Let us illustrate Sampo on our running example in Eq. 1. Assume therefore that we already have at hand $AC_\phi^{\text{Evaluated}}$. We then need to sample N values for the random variable `t`. Lets suppose we sample 5 values.

$$\texttt{t}_{\text{MC}} \in \{12.8, 35.1, 17.6, 22.2, 21.4\} \tag{13}$$

and plug these samples into Ψ. We map the Boolean random variable `no_cool` to a $1D$ tensor whose entries are 0.01. Consulting the arithmetic circuit in Figure 1, we easily see that we obtain for the MC estimate:

$$\begin{aligned} \Psi_{\text{MC}} &= \begin{bmatrix}0.01\\0.01\\0.01\\0.01\\0.01\end{bmatrix} \circ \begin{bmatrix}[12.8{>}20]\\[35.1{>}20]\\[17.6{>}20]\\[22.2{>}20]\\[21.4{>}20]\end{bmatrix} \circ \begin{bmatrix}[12.8{\leq}30]\\[35.1{\leq}30]\\[17.6{\leq}30]\\[22.2{\leq}30]\\[21.4{\leq}30]\end{bmatrix} + \begin{bmatrix}[12.8{>}30]\\[35.1{>}30]\\[17.6{>}30]\\[22.2{>}30]\\[21.4{>}30]\end{bmatrix} \\ &= \begin{bmatrix}0.01\\0.01\\0.01\\0.01\\0.01\end{bmatrix} \circ \begin{bmatrix}0\\1\\0\\1\\1\end{bmatrix} \circ \begin{bmatrix}1\\0\\1\\1\\1\end{bmatrix} + \begin{bmatrix}0\\1\\0\\0\\0\end{bmatrix} = \begin{bmatrix}0\\1\\0\\0.01\\0.01\end{bmatrix} \end{aligned} \tag{14}$$

where $\circ$ denotes the elementwise multiplication of tensors. With the Monte Carlo estimate of Ψ we obtain the MC estimate for the weighted model integral by simply averaging:

$$WMI_{\text{MC}} = \frac{1}{5} \sum_{i=1}^{5} \Psi_{\text{MC},i} = 1.02/5 = 0.204$$

Compiling an SMT formula and transforming the resulting arithmetic circuit into a computation graph has the advantage that sampling becomes embarrassingly parallelizable. To the best of our knowledge, Sampo is the first probabilistic inference algorithm for the hybrid domain that is able to harness parallelization on a GPU.

5.3 Discussion on Complexity

The complexity of Symbo and Sampo is mainly determined by the complexity of their subcomponents. The knowledge compilation step is #P-complete. The evaluation of the resulting arithmetic circuit is done in polytime. Symbo, however, suffers from the problem that the search for simplifications in symbolic expressions is a hard problem. One such simplification is the symbolic integration step itself. For example, integrating convex polytopes is #P-complete. These complexity concerns do not hold for Sampo, as we are dealing with mere additions and multiplications on the GPU.

In the next section the computational complexity of symbolic simplifications becomes experimentally apparent in the ClickGraph benchmark for Symbo (cf. Table 1). We are currently already investigating how to practically circumvent the computational hardness in such cases. For example, by subquerying or by static program analysis and detecting where to intermediately integrate variables.

6 Experimental Evaluation

In the previous section we have developed two algorithms that perform weighted model integration for weight functions in the form of probability density functions. Because general probability density functions are common in probabilistic programs, but have only been approximated in existing weighted model integration algorithms (using piecewise polynomial

weight functions), we compare Symbo and Sampo with state-of-the-art inference algorithms in probabilistic programming.

To this end, we extended the syntax of the probabilistic programming system ProbLog2 (Dries et al. 2015), so that it allows for the use of abstractions of atomic formulas and for the declaring how continuous random variables are distributed. ProbLog2 implements inference for the probabilistic programming language aProbLog (Kimmig, Van den Broeck, and De Raedt 2011), where inference is done through algebriac model conting.

We are interested in two main questions during the experimental evaluation of Symbo and Sampo. **Q1:** How does Symbo, a logico-symbolic solver, compare to a pure, state-of-the-art, symbolic solver for the hybrid domain? **Q2:** How does Sampo compare to related state-of-the-art probabilistic inference algorithms? **Q3:** In the interest of completeness we also adopted Symbo to solve traditional weighted model integration problems, where the weight function is expressed as a polynomial function.

We answer **Q1** by comparing Symbo, which uses the PSI-Solver and combines it with KC, to pure symbolic inference with the PSI-Solver.

For **Q2**, we compare Sampo to the inference algorithms of Distributional Clauses (DC) (Nitti, De Laet, and De Raedt 2016)[3], BLOG (Milch et al. 2007)[4] and to Hybrid Probabilistic Model Counting (IHPMC) (Michels, Hommersom, and Lucas 2016)[5]. These are state-of-the-art probabilistic programming systems that all support first order logic as well as hybrid representations.

In **Q3** we compare Symbo to the existing WMI solver of (Morettin, Passerini, and Sebastiani 2017), which uses predicate abstraction, SMT solving and numerical integration, and to the solver of (Kolb et al. 2018), which uses XADDs (Sanner and Abbasnejad 2012) and hence symbolic integration.

Experiments were performed on a laptop Intel(R) i7 CPU 2.60GHz with 16 Gb memory. Sampo took additionally advantage of an NVIDIA Quadro M1000M.

Q1 (Symbo): We compared Symbo and the PSI-Solver on the set of benchmark experiments given in (Gehr, Misailovic, and Vechev 2016, section F of Appendix)[6].

In Table 1, we observe that Symbo outperforms the PSI-Solver for $9/10$ benchmarks, for $7/10$ even when including the time spent on the knowledge compilation step. Only for the ClickGraph benchmark PSI performs better than Symbo, which timed-out after 15s during circuit evaluation. This is because PSI integrates out variables after loop iterations. This is not yet supported in the ProbLog implementation and Symbo ends up with a large symbolic expression that is hard to integrate over. This could be solved, for example, by using sub-queries, as can be done in ProbLog2.

We note that the symbolic inference engine underlying the PSI-Solver has until now only been used for imperative programing. The implementation of Symbo shows that the powerful symbolic inference engine can also be adopted for logic programming when making use of KC.

To conclude, it is generally beneficial to perform logical inference on top of symbolic inference in the hybrid domain.

Benchmark	KC	Evaluation	PSI	Domain
BurglarAlarm	31.4	0.8	190.1	D
CoinBias	41.9	7.9	12.9	H
Grass	31.2	1.2	228.0	D
NoisyOR	35.8	11.2	12.7	D
TwoCoins	27.0	2.1	57.8	D
ClickGraph	4300	–	10500	H
ClinicalTrial	54.6	25.7	3400	H
AddFun/max	25.2	4.4	53.1	H
AddFun/sum	27.1	2.1	84.9	H
MurderMystery	27.6	0.3	65.4	D

Table 1: Knowledge compilation and arithmetic circuit evaluation times for Symbo, and problem solving time for PSI. Times are given in ms. Run times were averaged over 50 runs. The domain column indicates whether the problem is **D**iscrete or **H**ybrid.

Q2 (Sampo): In order to evaluate Sampo, we chose benchmarks from (Nitti, De Laet, and De Raedt 2016) and (Michels, Hommersom, and Lucas 2016), which were stated to be hardest in terms of query complexity. We show our results in Figures 2 and 3. In Figure 2, we compare Sampo to DC and BLOG. A comparison with IHPMC for this first problem is not possible as IHPMC does not allow for expressing hierarchical models. DC an BLOG are, just like Sampo, sampling based methods, which use both importance sampling and likelihood weighting[7]. This is why we plot the evaluation time and the standard deviation in function of the number of samples. IHPMC is not a sampling based method but iteratively splits up the space into mutually exclusive pieces and calculates bounds for each piece, which translates to iteratively tighter and tighter error bounds. For this reason we investigate in the plots in Figure 3 the standard deviation of the four methods scrutinized in function of the run time.

All four plots clearly indicate that once Sampo has transformed a probabilistic program into an arithmetic circuit, the run time is not only lower but also that Sampo is more accurate than the competing algorithms. This is especially true for two distinct cases. Firstly, when there are binary random variables present. Contrary to DC and BLOG, Sampo does not sample these random variables but includes their probability as weight in the circuit evaluation. This can be seen Figure 2a and 3a. The reason why the STD is not zero for Sampo in 2a is due to floating point rounding errors. The second case where Sampo clearly outperforms the other methods is when we condition on low probability events, cf. Figure 3b. Here we condition on an event that has probability 0.0001 to occur. The logic structure of the problem implies that the query given the observation must be satisfied. In Figure 3b we see that Sampo is the only algorithm that picks up this structure. As the inference reduces to inference on exclusviely Boolean

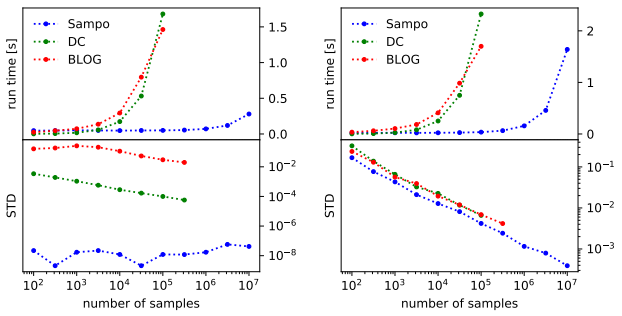
[3] `https://bitbucket.org/problog/dc_problog`

[4] `https://bayesianlogic.github.io`

[5] `https://github.com/SteffenMichels/IHPMC`

[6] cf.: Fun (Minka et al. 2014) and R2 (Nori et al. 2014)

[7] BLOG also provides rejection sampling and MCMC.



(a) (b)

Figure 2: The example used is *drawing balls (denoted by b_i and having different size, color and material) with replacement from an urn* (Nitti, De Laet, and De Raedt 2016). The queries used are (a) $p(b_1{=}b_2 \wedge col(b_1){=}black)$ and (b) $p(b_1{=}1 | 0.39{<}size(b_1){<}0.41)$. The upper panel shows the run time (circuit evaluation for Sampo). Evaluation runs are averaged over 50 runs and the knowledge compilation step is averaged over 50 compilations. Time-out was set at 2.5s. The linear behavior of Sampo towards higher sample numbers is due to the GPU starting to run out of memory. Sampo spent 1.62s for (a) and 0.11s for (b) on the knowledge compilation step, averaged over 50 runs.

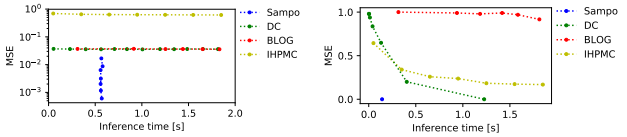
random variables, Sampo immediately finds the correct solution without drawing any samples for continuous random variables, in contrast to the other algorithms.

We also observe that there is practically no time penalty for the number of samples for Sampo, contrary to DC and BLOG. This behavior manifests itself most prominently in the upper panel of Figure 2a and in Figure 3a. For the latter, we see that higher sample numbers, which correspond to lower STDs, take up just as much time as lower sample numbers. This produces the quasi-vertical line Figure 3a. This behavior is due to delegating the N evaluations of the arithmetic circuits, which correspond to N times sampling the continuous random variables, to the GPU and executing the evaluation in parallel. Only in Figure 2a we observe a linear dependency of the run time in function of the number of samples towards high sample numbers. This is caused by the GPU running out of memory.

Q3 (WMI): By allowing Symbo to handle also bounded polynomial weights, instead of probability density distributions, we can compare Symbo to the existing exact WMI solvers of WMI-PA (Morettin, Passerini, and Sebastiani 2017) and WMI-XADD (Kolb et al. 2018). This extension of Symbo is necessary as these solvers are limited to polynomial weights and cannot handle proper probability densities.

We made the experimental comparison of the three methods on a set of synthetic problems given in (Morettin, Passerini, and Sebastiani 2017)[8]. The benchmarks consist of WMI problems that have from five to seven Boolean variables

[8] https://github.com/unitn-sml/wmi-pa



(a) query: $p(f_{99})$ (b) query: $p(f_{99}|f_0)$

Figure 3: We show the dependencies of the mean squared error on time for two queries of the theory: $f_i \leftrightarrow d_i \vee c > l_i \vee f_{i-1}$, cf. (Michels, Hommersom, and Lucas 2016). f_i and d_i are Bools. The probability of d_i being true is 0.0001. c and l_i are normally distributed variables with mean 20 and 30 respectively and standard deviation 5. Note the two different scales for the plots on the y-axis. The mean squared errors are averaged over 50 runs. The average KC time (over 50 iterations) is 4.34s for (a) and 2.66s for (b). In the left plot the mean squared error was calculated with respect to the mean of 50 runs using Sampo with 10^5 samples. In the right plot, we stopped when all the runs for a given number of samples for an algorithm reached the correct solution (which is 1.0) or the algorithm timed-out after 2s.

and where the weight functions are multivariate polynomials of dimensions two to three.

We compared the three methods on the benchmarks with two dimensional polynomial weights. We observe in Figure 4 that Symbo solves the majority of the problems with bivariate polynomials faster than the other two methods. We omit the comparison plot for the benchmarks with three dimensional polynomials, as here the other methods, which are specialized algorithms for polynomial weight functions, beat Symbo at large. Symbo spends most of the time on the final integration step (cf. point 7 in Algorithm 1). In fact, Symbo spent at most 0.32s on the KC step, at most 0.34s on the circuit evaluation and any remaining time on the symbolic integration - for any of the presented benchmarks. Using a dedicated integrator for bounded polynomials instead of the generic PSI integrator could mitigate this problem. Integrating out variables during the evaluation of the arithmetic circuit could also be beneficial, as this leads to smaller symbolic expressions.

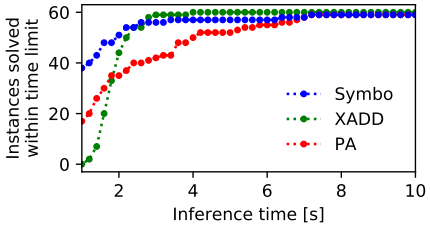


Figure 4: We show the number of problem instances solved below the time limit for problems with bivariate polynomial weights.

7 Related Work

In the initial work on weighted model integration (Belle, Passerini, and Van den Broeck 2015) the authors perform weighted model integration on piecewise polynomials by iteratively generating models by adding the negation of the model from the previous iteration to the formula. In subsequent work by (Morettin, Passerini, and Sebastiani 2017) the number of generated models is substantially reduced by deploying SMT-based predicate abstraction (Graf and Saïdi 1997). In this line of work (Belle et al. 2016) also investigated component caching while performing a DPLL search when calculating a weighted model integral. Their approach is indeed related to knowledge compilation. However, it is not applicable in cases when algebraic constraints exist between variables and couple these. The methods proposed on WMI are strictly limited to piecewise polynomials. We, completely lift this restrictions and are able to perform WMI via knowledge compilation on SMT($\mathcal{RA}$) and SMT($\mathcal{NRA}$) formulas using probability density functions instead of piecewise polynomials on SMT($\mathcal{LRA}$).

As seen in section 6, WMI has also been studied in the context of XADDs (Kolb et al. 2018) and the approach is closely related to Symbo. Here again, the weight functions considered are only of polynomial form. Another drawback of this approach is that, unlike for the SDDs and d-DNNFs used in our approach, there are not yet any efficient compilers available for converting WMI to XADDs. For SDDs and d-DNNFs, one can employ standard state-of-the-art KC technology. SDDs are more succinct than BDDs, of which XADDs are an extension. This entails, in turn, that SDDs are more succinct than XADDs. Standard knowledge compilation techniques are not readily available for XADDs as Kolb et al.'s compilation algorithm interleaves symbolic and logic inference.

Somewhat related to WMI with piecewise polynomials is the work of (Gutmann, Jaeger, and De Raedt 2011), who restricted distributions to Gaussians, which are chopped up into easily integrable and axis-aligned pieces.

Contrary to these works, we generalize WMI by providing a much larger class of weight functions and constraints.

With respect to inference for probabilistic programming in the hybrid domain, two classes of algorithms exist: approximate and exact. Firstly, for what concerns one of the few exact inference systems, there is the already mentioned work for imperative probabilistic programming (Gehr, Misailovic, and Vechev 2016), which has contributed the PSI-Solver that we use in Symbo. The PSI-Solver beats other recent approaches in exact probabilistic inference (Narayanan et al. 2016). We show that knowledge compilation speeds up pure symbolic inference and that Symbo outperforms the PSI-Solver.

Another approach, related to exact inference in probabilistic logic programming, is that of (Islam, Ramakrishnan, and Ramakrishnan 2012). Similarly to Symbo, they symbolically evaluate a theory in order to obtain an expression for a probability density. However, their approach is restricted to Gaussians (although gamma distributions are in theory also implementable), and more importantly it is built on top of Prism (Sato 1995), which assumes that proofs are mutually exclusive, and which avoids the disjoint sum problem. As a consequence they do not support WMI in its full generality. Supporting WMI requires the KC step, which they do not address.

Secondly, for what concerns approximate inference, we have the sampling approaches in Distributional Clauses by (Gutmann et al. 2011; Nitti, De Laet, and De Raedt 2016) and BLOG by (Milch et al. 2007), which we have already discussed in section 6 and which both deploy importance sampling in order to sample from probability distributions and densities alike, combined with likelihood weighting.

Approximate inference is also performed in (Michels, Hommersom, and Lucas 2016). In their work, a hybrid probabilistic problem is represented by so called *hybrid probability trees* (discussed in Section 6). Our experiments show that Sampo outperforms DC, BLOG and IHPMC. Moreover, Sampo has the advantage that when conditioning on rare events in the discrete domain, we still obtain reliable estimates of the weighted model integral. Using pure sampling based methods such as in Distributional Clauses and BLOG leads to poor results. This is known to be problematic when using importance sampling based methods.

Note that when conditioning on rare events in continuous domains, Sampo performs as poorly as other sampling techniques as it performs essentially rejection sampling with almost all samples being rejected.

8 Conclusion and Future Work

We have shown how knowledge compilation can be applied to the task of weighting model integration by leveraging algebraic model counting and thereby presenting a unified formalism for weighted model integration and knowledge compilation. We have also introduced an exact and an approximate solver based on this idea and demonstrated their effectiveness. Sampo is to the best of our knowledge the first sampling based algorithm deployable in the WMI setting.

In future work, we would like to investigate in more detail the relationship between Kolb et al.'s work and the work presented here, theoretically as well as experimentally. Moreover, we would like to integrate Symbo and Sampo into a full-fledged probabilistic programming language and investigate thoroughly how it compares to existing languages, especially DC, BLOG, Anglican, Church.

9 Acknowledgments

This work has been supported by the Research Foundation - Flanders (FWO) project G0D7215N, the European Research Council Advanced Grant project SYNTH[9] (ERC-AdG-694980), and the H2020 CHIST-ERA and FWO project ReGround[10] (G0D7215N). The authors would like to thank Sebastijan Dumančić, Angelika Kimmig, Ondřej Kuželka and Robin Manhaeve for reading and commenting on first drafts of this paper, and Samuel Kolb for inspiring discussions on the topic of WMI.

[9] https://synth.cs.kuleuven.be/

[10] http://reground.cs.kuleuven.be

References

Abadi, M.; Agarwal, A.; Barham, P.; Brevdo, E.; Chen, Z.; Citro, C.; Corrado, G. S.; Davis, A.; Dean, J.; Devin, M.; Ghemawat, S.; Goodfellow, I.; Harp, A.; Irving, G.; Isard, M.; Jia, Y.; Jozefowicz, R.; Kaiser, L.; Kudlur, M.; Levenberg, J.; Mané, D.; Monga, R.; Moore, S.; Murray, D.; Olah, C.; Schuster, M.; Shlens, J.; Steiner, B.; Sutskever, I.; Talwar, K.; Tucker, P.; Vanhoucke, V.; Vasudevan, V.; Viégas, F.; Vinyals, O.; Warden, P.; Wattenberg, M.; Wicke, M.; Yu, Y.; and Zheng, X. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. Software available from tensorflow.org.

Belle, V.; Van den Broeck, G.; Passerini, A.; et al. 2016. Component Caching in Hybrid Domains with Piecewise Polynomial Densities. In *AAAI*, 3369–3375.

Belle, V.; Passerini, A.; and Van den Broeck, G. 2015. Probabilistic Inference in Hybrid Domains by Weighted Model Integration. In *Proceedings of 24th International Joint Conference on Artificial Intelligence (IJCAI)*, 2770–2776.

Chavira, M., and Darwiche, A. 2008. On Probabilistic Inference by Weighted Model Counting. *Artificial Intelligence* 172(6):772 – 799.

Choi, A.; Kisa, D.; and Darwiche, A. 2013. *Compiling Probabilistic Graphical Models Using Sentential Decision Diagrams*. Berlin, Heidelberg: Springer Berlin Heidelberg. 121–132.

Darwiche, A., and Marquis, P. 2002. A Knowledge Compilation Map. *J. Artif. Int. Res.* 17(1):229–264.

Dries, A.; Kimmig, A.; Meert, W.; Renkens, J.; Van den Broeck, G.; Vlasselaer, J.; and De Raedt, L. 2015. *ProbLog2: Probabilistic Logic Programming*. Cham: Springer International Publishing. 312–315.

Fierens, D.; Van den Broeck, G.; Renkens, J.; Shterionov, D.; Gutmann, B.; Thon, I.; Janssens, G.; and De Raedt, L. 2015. Inference and Learning in Probabilistic Logic Programs Using Weighted Boolean Formulas. *Theory and Practice of Logic Programming* 15(3):358–401.

Fubini, G. 1907. Sugli Integrali Multipli. *Rom. Acc. L. Rend. (5)* 16(1):608–614.

Gehr, T.; Misailovic, S.; and Vechev, M. 2016. PSI: Exact Symbolic Inference for Probabilistic Programs. In *International Conference on Computer Aided Verification*, 62–83. Springer.

Graf, S., and Saïdi, H. 1997. Construction of Abstract State Graphs with PVS. In *International Conference on Computer Aided Verification*, 72–83. Springer.

Gutmann, B.; Thon, I.; Kimmig, A.; Bruynooghe, M.; and De Raedt, L. 2011. The Magic of Logical Inference in Probabilistic Programming. *Theory and Practice of Logic Programming* 11(4-5):663–680.

Gutmann, B.; Jaeger, M.; and De Raedt, L. 2011. *Extending ProbLog with Continuous Distributions*. Berlin, Heidelberg: Springer Berlin Heidelberg. 76–91.

Islam, M. A.; Ramakrishnan, C.; and Ramakrishnan, I. 2012. Inference in Probabilistic Logic Programs with Continuous Random Variables. *Theory and Practice of Logic Programming* 12(4-5):505–523.

Kimmig, A.; Van den Broeck, G.; and De Raedt, L. 2011. An Algebraic Prolog for Reasoning about Possible Worlds. In *AAAI*.

Kimmig, A.; Van den Broeck, G.; and De Raedt, L. 2017. Algebraic Model Counting. *Journal of Applied Logic* 22:46–62.

Knuth, D. E. 1992. Two Notes on Notation. *Am. Math. Monthly* 99(5):403–422.

Kolb, S.; Mladenov, M.; Sanner, S.; Belle, V.; and Kersting, K. 2018. Efficient Symbolic Integration for Probabilistic Inference. In *IJCAI*, 5031–5037.

Michels, S.; Hommersom, A.; and Lucas, P. J. F. 2016. Approximate Probabilistic Inference with Bounded Error for Hybrid Probabilistic Logic Programming. In *IJCAI 2016*.

Milch, B.; Marthi, B.; Russell, S.; Sontag, D.; Ong, D. L.; and Kolobov, A. 2007. BLOG: Probabilistic Models with Unknown Objects. In Getoor, L., and Taskar, B., eds., *Statistical Relational Learning*. MIT Press.

Minka, T.; Winn, J.; Guiver, J.; Webster, S.; Zaykov, Y.; Yangel, B.; Spengler, A.; and Bronskill, J. 2014. Infer.NET 2.6. Microsoft Research Cambridge. http://research.microsoft.com/infernet.

Morettin, P.; Passerini, A.; and Sebastiani, R. 2017. Efficient Weighted Model Integration via SMT-Based Predicate Abstraction. *def* 1(x1):x2.

Narayanan, P.; Carette, J.; Romano, W.; Shan, C.; and Zinkov, R. 2016. Probabilistic Inference by Program Transformation in Hakaru (System Description). In *International Symposium on Functional and Logic Programming - 13th International Symposium, FLOPS 2016, Kochi, Japan, March 4-6, 2016, Proceedings*, 62–79. Springer.

Nitti, D.; De Laet, T.; and De Raedt, L. 2016. Probabilistic Logic Programming for Hybrid Relational Domains. *Machine Learning* 103(3):407–449.

Nori, A. V.; Hur, C.-K.; Rajamani, S. K.; and Samuel, S. 2014. R2: An Efficient MCMC Sampler for Probabilistic Programs. In *AAAI*, 2476–2482.

Sanner, S., and Abbasnejad, E. 2012. Symbolic Variable Elimination for Discrete and Continuous Graphical Models. In *AAAI*.

Sato, T. 1995. A Statistical Learning Method for Logic Programs with Distribution Semantics. In *In proceedings of the 12TH international conference on logic programming (ICLP'95*. Citeseer.

Tran, D.; Kucukelbir, A.; Dieng, A. B.; Rudolph, M.; Liang, D.; and Blei, D. M. 2016. Edward: A Library for Probabilistic Modeling, Inference, and Criticism. *arXiv preprint arXiv:1610.09787*.