Knowledge Compilation with Continuous Random Variables and its Application in Hybrid Probabilistic Logic Programming

Pedro Zuidberg Dos Martires and **Anton Dries** and **Luc De Raedt**
KU Leuven, Belgium

Abstract

In probabilistic reasoning, the traditionally discrete domain has been elevated to the hybrid domain encompassing additionally continuous random variables. Inference in the hybrid domain, however, usually necessitates to condone trade-offs on either the inference on discrete or continuous random variables. We introduce a novel approach based on weighted model integration and algebraic model counting that circumvents these trade-offs. We then show how it supports knowledge compilation and exact probabilistic inference. Moreover, we introduce the hybrid probabilistic logic programming language HAL-ProbLog, an extension of ProbLog, to which we apply our inference approach.

1 Introduction

One of the state-of-the art methods for probabilistic inference in graphical models and probabilistic programming reduces probabilistic inference to weighted model counting (WMC) (Chavira and Darwiche 2008), and then employs WMC solvers based on knowledge compilation (KC) techniques (Darwiche and Marquis 2002). Because weighted model counting applies only to discrete probability distributions, it has recently been extended towards weighted model integration (WMI) (Belle, Passerini, and Van den Broeck 2015; Morettin, Passerini, and Sebastiani 2017) as to support also continuous distributions. However, to the best of the authors' knowledge, knowledge compilation has not yet been used for weighted model integration. Indeed, current approaches to weighted model integration essentially use satisfiability modulo theory (SMT) solvers, usually restricted to linear arithmetic over the reals (SMT($\mathcal{LRA}$)). On the other hand, knowledge compilation has proven to be very effective, especially when many probabilistic queries need to be answered as the theory needs to be compiled only once.

The key contribution of this paper is that we show how standard KC techniques can be applied to solve weighted model integration problems, and that we incorporate such techniques in hybrid probabilistic logic programming languages. This is realized by casting weighted model integration within the framework of algebraic model counting (AMC) (Kimmig, Van den Broeck, and De Raedt 2017). AMC generalizes the standard weighted model counting problem to work with arbitrary semirings. More specifically, we make the following contributions:

1. We introduce the probability density semiring.
2. We show how this allows 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.

Symbo exploits results by (Gehr, Misailovic, and Vechev 2016) to simplify the algebraic expressions.

Algebraic model counting has also been incorporated in logic programming languages such as aProbLog (Kimmig, Van den Broeck, and De Raedt 2011), which is an extension of the probabilistic programming language ProbLog (Fierens et al. 2015) towards semirings and allows to state our next contribution.

4. We use the probability density semiring S within aProbLog to obtain a hybrid probabilistic logic programming language and implementation, that we dub ***h**ybrid **al**gebraic **ProbLog*** or HAL-ProbLog

HAL-ProbLog is related to the Distributional Clauses framework of (Gutmann, Jaeger, and De Raedt 2011; Gutmann et al. 2011; Nitti, De Laet, and De Raedt 2016), differences and similarities are further discussed in subsection 5.2.

2 Preliminaries

2.1 Weighted Model Integration

While the well-known SAT problem is the problem of deciding whether there is a satisfying assignment to a logical formula or not, an SMT problem generalizes SAT and allows in addition to use expressions formulated in a background theory. Consider, for example, the following SMT theory `broken`:

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

where `no_cool` is a Boolean variable, while `t` is a real-valued variable. SMT then answer 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 non-linear real arithmetic SMT formulas.

Definition 1. (SMT($\mathcal{NRA}$) (non-linear real arithmetics)) Let $\mathbb{R}$ denote the set of real values and $\mathbb{B} = \{0, 1\}$ the set of Boolean values. We then define SMT($\mathcal{NRA}$) theories as combinations by means of the standard Boolean operators $\{\neg, \wedge, \vee, \rightarrow, \leftrightarrow\}$ of atomic propositions $a_i \in \mathbb{B}$ and of $\mathcal{NRA}$ atomic formulas in the form $\sum_i c_i \cdot x_i^{p_i} \lesseqqgtr c$. The x_i are variables in $\mathbb{R}$ and $c_i, c, p_i \in \mathbb{Q}$. Allowing only for $p_i = 1$ restricts SMT($\mathcal{NRA}$) theories to SMT($\mathcal{LRA}$ (linear real arithmetics) theories. □

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 now the probability of the formula being true extends the SMT problem to weighted model integration. We introduce WMI following (Morettin, Passerini, and Sebastiani 2017).

Definition 2. (Weighted model integration (WMI)) Given is a set B of M Boolean variables, a set X of N real variables, a weight function $w(X, B) : \mathbb{B}^M \times \mathbb{R}^N \rightarrow \mathbb{R}^+$, and an SMT formula $\phi(X, B) : \mathbb{B}^M \times \mathbb{R}^N \rightarrow \mathbb{B}$, the **weighted model integral (WMI)** is

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

where we use the notation $\mathcal{I}_V(\phi(V))$ to denote the set of assignments to the variables in V that satisfy $\phi(V)$. □

Hence, when computing the weighted model integral (Eq. 2), we first integrate over all X in a formula $\phi(X, \mathbf{b})$ for each possible assignment $\mathbf{b}$ to the Boolean variables B holds and then sum up the values of the integrals. The weight function is used to map a set of variable assignments to their weight. The weight function usually *factorizes* as the product of the weights over the different variables, i.e., $w(\mathbf{x}, \mathbf{b}) = \prod_{x_i \in \mathbf{x}} w(x_i) \prod_{b_j \in \mathbf{b}} w(b_j)$.

With the definition of WMI at hand, we can also produce the weighted model integral by integrating over the continuous random variables, while using the algebraic constraints as boundary conditions and weighting the integrals with the probability of the Boolean variables: $WMI(\texttt{broken}){=}0.01 \int_{20<\texttt{t}\leq 30} \mathcal{N}_{\texttt{t}}(20, 5) d\texttt{t} + \int_{\texttt{t}>30} \mathcal{N}_{\texttt{t}}(20, 5) d\texttt{t}$.

2.2 Algebraic Model Counting

Definition 3. (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: $WMC(\phi, w|B) = \sum_{\mathbf{b} \in \mathcal{I}_B(\phi(B))} \prod_{b_i \in \mathbf{b}} w(b_i)$. Algebraic model counting (Kimmig, Van den Broeck, and De Raedt 2017) generalizes WMC to commutative semirings. More formally,

Definition 4. A **commutative semiring** is an algebraic structure $(\mathcal{A}, \oplus, \otimes, e^{\oplus}, e^{\otimes})$ 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 5. (Algebraic model counting (AMC)) 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:

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

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.

(Kimmig, Van den Broeck, and De Raedt 2017) show also under which conditions the an algebraic model count is a valid computation.

Definition 6. (Neutral-sum property) A semiring addition and labeling function pair $(\oplus, \alpha)$ is neutral *iff.* $\forall b \in B$: $\alpha(b) \oplus \alpha(\neg b) = e^{\otimes}$. □

Theorem 1. (AMC on d-DNNF) 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 of the algebraic model count, cf. (Kimmig, Van den Broeck, and De Raedt 2017). □

2.3 Knowledge Compilation

Knowledge compilation (Darwiche and Marquis 2002) is 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 labels/weights for the literals.

A popular language to compile propositional formulas into are Sentential Decisions Diagrams (SDDs) (Choi, Kisa, and Darwiche 2013). SDDs are s a subset of d-DNNF formulas. We use SDDs to implement our solver, Symbo.

Note that, as SDDs are subset of d-DNNF, Theorem 1 holds also for the them.

3 The probability density semiring and WMI

Now we have all the ingredients to define the *probability density semiring*, which is needed to cast WMI as AMC.

The key difference between WMI and AMC is that in a WMI task there is first a sum, then an integral and then typically a product, while in AMC there is no integral. This intuitively implies that, if we want to cast WMI using AMC, we will have to perform the integration last: WMI = $\int$ AMC. This can only be realized if we keep track of the two elements needed in the integral 1) the formula ϕ defining the values over which to integrate and 2) the weight function w defining the densities according to which the variables in ϕ are distributed. So, the set of elements of semiring that we need to define will consist of tuples, where the first element will denote an algebraic expression and the second the weight function.

Definition 7. (Labeling function α) If the literal l represents either a Boolean variable b or its negation $\neg b$ then the label

$$\alpha(b) := (P(b), \emptyset) \qquad \alpha(\neg b) := (1 - P(b), \emptyset) \qquad (3)$$

where $P(b)$ denotes the probability of the Boolean variable. Otherwise if the literal l corresponds to an algebraic constraint within SMT($\mathcal{NRA}$), depending on the set of real-valued continuous random variables $\{\mathbf{x}\}$, then the label of l is given by:

$$\alpha(l) := \left([l], \mathcal{F}_x^S\right) \qquad \alpha(\neg l) := \left([\neg l], \mathcal{F}_x^S\right) \qquad (4)$$

$\mathcal{F}_x^S$ denotes the set $\{x_i{\sim}f_i\}$, where the x_i are random variables and the f_i the corresponding probability densities. The first definition in Eq. 4 is read as *'l such that any x_i is distributed according to the corresponding f_i'*. □

The brackets around $[l]$ denote the so-called *Iverson brackets*. They evaluate to 1 if their argument l evaluates to true and to 0 otherwise - they are a generalized indicator function.

We can now define the set of elements of the semiring:

Definition 8. (Probability density semiring $\mathcal{S}$) The elements of the semring $\mathcal{S}$ are given by the set

$$\mathcal{A} := \left\{\left(a, \mathcal{F}_x^S\right)\right\} \qquad (5)$$

where a denotes any algebraic expression over $\mathcal{NRA}$, including also Iverson brackets. For instance, $a = 0.01[20 < t \leq 30] + [t > 30]$. $\mathcal{F}_x^S$ is shorthand for the set $\{x_i \sim f_i\}$ with x the set of all real-valued continuous random variables appearing in a.
The neutral elements $\oplus$ and $\otimes$ are defined as:

$$e^{\oplus} := (0, \emptyset) \qquad e^{\otimes} := (1, \emptyset) \qquad (6)$$

For the addition and multiplication we define:

$$\left(a_1, \mathcal{F}_{x_1}^S\right) \oplus \left(a_2, \mathcal{F}_{x_2}^S\right) := \left(a_1 + a_2, \mathcal{F}_{x_1}^S \cup \mathcal{F}_{x_2}^S\right) \qquad (7)$$

$$\left(a_1, \mathcal{F}_{x_1}^S\right) \otimes \left(a_2, \mathcal{F}_{x_2}^S\right) := \left(a_1 \times a_2, \mathcal{F}_{x_1}^S \cup \mathcal{F}_{x_2}^S\right) \qquad (8)$$

□

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

Proof (Sketch). To prove that the structure $\mathcal{S}$ actually constitutes a commutative semiring, we need to show that the properties in Definition 4 hold.

The proof relies on the commutativity and associativity of the Iverson brackets under standard addition and multiplication, and on the commutativity and associativity of the union operator for sets. Similarly for the distributivity of the multiplication over the addition (c.f. 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.

Proof. Let l be a literal with label $\alpha(l) = P(l)$ *iif.* $l \in B$ and $\alpha(l) = [l]$ *iff.* l is an abstraction of an $\mathcal{NRA}$ formula. We then have:

$$\begin{aligned}\alpha(l) \oplus \alpha(\neg l) &= \begin{cases}(P(l), \emptyset) \oplus (1 - P(l), \emptyset) & \text{for } l \in B \\ ([l], \mathcal{F}) \oplus ([\neg l], \mathcal{F}) & \text{else}\end{cases} \\ &= \begin{cases}(1, \emptyset) & \text{for } l \in B \\ ([l] + [\neg l], \mathcal{F}) & \text{else}\end{cases} \\ &= e^{\otimes}\end{aligned}$$

In the last line we used the fact that $(1, \mathcal{F})$ and $(1, \emptyset)$ are equivalent elements within the probability density semiring. □

Lemma 3. (AMC on d-DNNF with $\mathcal{S}$) The algebraic model count is a valid 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. □

An SMT($\mathcal{NRA}$) theory induces an infinity of theories, one for each possible instantiation of continuous random variables. We can utilize the same compiled theory for each of the infinitely many theories. Note that the probability of each instantiation is 0, as the support for a single instantiation is infinitesimally small. In order to retrieve actual probabilities from an SMT, we need to carry out the integration over the continuous random variables.

We show now how computing the algebraic model count in the semiring setting $\mathcal{S}$ yields the probability of a theory being satisfied.

Theorem 2. Let ϕ be an SMT($\mathcal{NRA}$) theory, w a factorized weight function over the Boolean variables B and continuous variables X. Furthermore, assume that $AMC(\phi, w|X \cup B)$ evaluates to $(\Psi, \mathcal{F}_x^S)$ in the semiring $\mathcal{S}$, with $\Psi = \sum_{\mathbf{v} \in \mathcal{I}(\phi(X,B))} \prod_{v_i \in \mathbf{v}} a_{v_i}$, where a_v is an algebraic expression depending on the random variables $\mathbf{v}$. Then

$$WMI(\phi, w|X, B) = \int_{\mathbf{x} \in X} \Psi \prod_{f_i(x_i) \in \mathcal{F}_x^S} f_i(x_i) d\mathbf{x}$$

$\prod_{f_i(x_i) \in \mathcal{F}_x^S} f_i(x_i)$ is the product over the probability density function of the continuous random variables appearing in Ψ.

Proof In the first step we re-write Ψ as the sum-product over the algebraic expression a_v. We note also that the product over the density functions is actually the weight of the continuous random variables in WMI. In the second step (P2 to P3) we split up the sum and the product over the variables $\mathbf{v}$ into sums over the Boolean and continuous random variables - likewise for the product. Next (P3 to P4) we push the product over the Boolean random variables through and note in (P5) that this product corresponds to the weight of the Boolean random variables in WMI.

$$\begin{aligned}
&\int_{\mathbf{x} \in X} \Psi \prod_{f_i(x_i) \in \mathcal{F}_x} f_i(x_i) d\mathbf{x} && \text{P1} \\
&= \int_{\mathbf{x} \in X} \sum_{\mathbf{v} \in \mathcal{I}(\phi(X,B))} \prod_{v_i \in \mathbf{v}} a_{v_i} w(\mathbf{x}) d\mathbf{x} && \text{P2} \\
&= \int_{\mathbf{x} \in X} \sum_{\mathbf{b} \in \mathcal{I}_b(true)} \sum_{\mathbf{x} \in \mathcal{I}_X(\phi(X,\mathbf{b}))} \prod_{b_i \in \mathbf{b}} \prod_{x_i \in \mathbf{x}} a_{b_i} a_{x_i} w(\mathbf{x}) d\mathbf{x} && \text{P3} \\
&= \int_{\mathbf{x} \in X} \sum_{\mathbf{b} \in \mathcal{I}_b(true)} \sum_{\mathbf{x} \in \mathcal{I}_X(\phi(X,\mathbf{b}))} \prod_{x_i \in \mathbf{x}} a_{x_i} \prod_{b_i \in \mathbf{b}} a_{b_i} w(\mathbf{x}) d\mathbf{x} && \text{P4} \\
&= \int_{\mathbf{x} \in X} \sum_{\mathbf{b} \in \mathcal{I}_b(true)} \sum_{\mathbf{x} \in \mathcal{I}_X(\phi(X,b))} \prod_{x_i \in \mathbf{x}} a_{x_i} w(\mathbf{b}) w(\mathbf{x}) d\mathbf{x} && \text{P5} \\
&= \sum_{\mathbf{b} \in \mathcal{I}_b(true)} \int_{\mathbf{x} \in X} \sum_{\mathbf{x} \in \mathcal{I}_X(\phi(X,\mathbf{b}))} \prod_{x_i \in \mathbf{x}} a_{x_i} w(\mathbf{x}, \mathbf{b}) d\mathbf{x} && \text{P6} \\
&= \sum_{\mathbf{b} \in \mathcal{I}_b(true)} \int_{\mathbf{x} \in \mathcal{I}_X(\phi(X,\mathbf{b}))} w(\mathbf{x}, \mathbf{b}) d\mathbf{x} && \text{P7}
\end{aligned}$$

In the last two lines, we exchanged the summation and integral, as Fubini's theorem holds for summations/integrals over probability distributions and densities. The integral over the so-obtained sum-product is the integral over Iverson brackets. We rewrite the indefinite integral over the Iverson brackets as the definite integral having boundary conditions corresponding to the condition present in the Iverson brackets.

The last line (P7) corresponds to the definition of the weighted model integral. We have, hence, shown that WMI can be cast as an AMC task. □

4 Probability of SMT formulas via KC

We describe now **Symbo**, a symbolico-logic algorithm that produces the weighted model integral of an SMT($\mathcal{NRA}$) formula ϕ via knowledge compilation.

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 1, we are hence also capable of obtaining the weighted model integral of the a hybrid propositional formula, given the probability distributions of the random the variables.

At a high level, Symbo takes the following consecutive steps:

1. Abstraction of algebraic constraints in ϕ in order to obtain $\phi^{abstract}$. For instance, a constraint $(t > 20)$ would be abstracted as a Bool $b_{t>20}$.
2. Compilation of $\phi^{abstract}$ into a d-DNNF representation $\phi^{abstract}_{compiled}$.
3. Transforming the logic formula $\phi^{abstract}_{compiled}$ into an arithmetic circuit AC_ϕ.
4. Labeling the literals in AC_ϕ according to the labeling function given in Definition 7.
5. Symbolically evaluating AC_ϕ according to the probability density semiring $\mathcal{S}$.
6. Symbolically multiplying the expression obtained from evaluating AC_ϕ, which is a sum-product of weighted indicator functions (Iverson brackets), by the probability densities according to which the continuous random variables are distributed.
7. Symbolically integrating out the continuous random variables.

Regarding more technical details of the algorithm: Symbo leverages the PSI-Solver (Gehr, Misailovic, and Vechev 2016), a novel approach for exact symbolic analysis of probabilistic programs that carries out inference through symbolic reasoning[1]. When evaluating a compiled hybrid theory, Symbo builds up a symbolic PSI expression for Ψ (cf. Theorem 1). The leaf nodes of the d-DNNF representation are annotated with algebraic expressions. A leaf corresponding to a Boolean literal receives a symbolic value for the probability of being satisfied and a leaf corresponding to an abstraction of an algebraic condition is expressed as a symbolic Iverson bracket. Internal logical nodes of the compiled theory (logical and/or operations) are mapped to the symbolic multiplication/addition of the PSI-Solver. The inference engine of the PSI-Solver tries to symbolically simplify resulting expressions as much as possible. Once the compiled circuit is evaluated, the final expression is multiplied by Symbo with the set of densities corresponding to the continuous random variables in Ψ. The symbolic integration is then again carried out by the PSI-Solver.

[1]This includes, amongst others, algebraic simplifications and guard simplifications. See (Gehr, Misailovic, and Vechev 2016) for a detailed discussion.

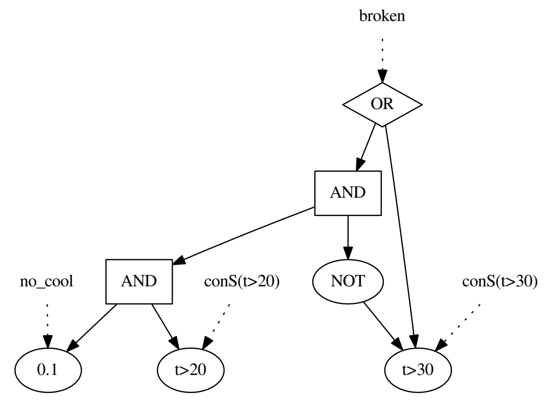

Figure 1: Graphical representation of the propositional formula in Eq. 1.

Lets look at an example. Consider our initial example in Eq. 1. Compiling it into d-DNNF form yields:

$$(\texttt{no\_cool} \wedge (\texttt{t} > 20) \wedge (\texttt{t} \leq 30)) \vee (\texttt{t} > 30) \tag{10}$$

which is already a propositional formula in a d-DNNF representation. Such a hybrid formula, for which Symbo has kept track of the probabilities and weights involved, can be considered to be the input to the algorithm. We can represent it as a graph where the leaves represent the literals in the formula and internal nodes logical operation, cf. Figure 1. Evaluating this theory using the semantics of the probability semiring $\mathcal{S}$ and the PSI-Solver yields the following result

$$\Psi = 0.01[\texttt{t}{>}20][\texttt{t}{\leq}30] + [\texttt{t}{>}30]$$

Multiplying this expression by the probability density function for `t` and carrying out the integral gives us the weighted model integral for the theory `broken`.

$$\begin{aligned}
p(\texttt{broken}) = &\int (0.01[\texttt{t}{>}20][\texttt{t}{\leq}30] + [\texttt{t}{>}30])\, \mathcal{N}_\texttt{t}(20,5) d\texttt{t} \\
=&0.01 \int_{20<\texttt{t}\leq 30} \mathcal{N}_\texttt{t}(20,5) d\texttt{t} + \int_{\texttt{t}>30} \mathcal{N}_\texttt{t}(20,5) d\texttt{t} \\
=&1 - 0.01 \left(\frac{d}{dx}\right)^{-1} [e^{-x^2}] \left(-\frac{5}{2}\sqrt{8} + \frac{20}{\sqrt{8}}\right) \frac{1}{\sqrt{\pi}} \\
&- 0.9 \left(\frac{d}{dx}\right)^{-1} [e^{-x^2}] \left(-\frac{5\sqrt{8}}{2} + \frac{30}{\sqrt{8}}\right) \frac{1}{\sqrt{\pi}}
\end{aligned}$$

In PSI, terms of the form $(d/dx)^{-1}[e^{-x^2}](a)$ denote the function $\int_{-\infty}^{a} dx e^{-x^2}$, which cannot be simplified any further.

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 knowledge compilation.

5 HAL-ProbLog

Let us now define HAL-ProbLog, a hybrid probabilistic logic programming language based on the distributional clause semantics of (Gutmann et al. 2011; Nitti, De Laet, and De Raedt 2016). By making use of the reduction of WMC to AMC, we can implement HAL-ProbLog as an instance of aProbLog (Kimmig, Van den Broeck, and De Raedt 2011), which itself extends the semantics of ProbLog (Fierens et al. 2015), a probabilistic logic programming language. While ProbLog solves the task of computing the probability of a certain query being true, aProbLog generalizes this to a variety of other tasks by deploying a semiring.

5.1 aProbLog

Definition 9. (aProbLog program) An algebraic ProbLog program consists of: 1) a commutative semiring $\mathcal{S}$, 2) a finite set of ground algebraic facts $F = \{f_i\}$, 3) a finite set BK of background knowledge clauses of the form $h \leftarrow b_1, ..., b_n$ where h and the b_i are logical atoms, and 4) a labeling function $\alpha : \mathcal{L}(F) \rightarrow \mathcal{A}$ where $\mathcal{L}(F)$ contains all facts $f \in F$ and their negation $\neg f$.

Following (Kimmig, Van den Broeck, and De Raedt 2011), we also define an aProbLog query and the label of a resulting theory as follows.

Definition 10. (aProbLog query) An aProbLog query q is a finite set of algebraic literals and atoms from the Herbrand base (HB), i.e. the set of ground atoms that can be constructed from the predicates, functor and constant symbols of the program $q \subseteq \mathcal{L}(F) \cup HB(F \cup BK)$. The set of interpretations $\mathcal{I}(q)$ that makes the query q true is defined as: $\mathcal{I}(q) = \{I \subseteq \mathcal{L}(F) | \forall l \in F : l \in I \leftrightarrow \neg l \notin I \text{ and } I \cup BK \models q\}$.

Definition 11. (Label of aProbLog query) The label of a query q is the label of $\mathcal{I}(q)$: $\boldsymbol{AMC}(q) = \boldsymbol{AMC}(\mathcal{I}(q)) = \bigoplus_{I \in \mathcal{I}(q)} \bigotimes_{l \in I} \alpha(l)$.

5.2 Syntax and semantics of HAL-ProbLog

We now apply aProbLog to obtain HAL-Problog, which we first illustrate on a simple example modeling the behavior of a machine under different temperature conditions. This examples is an extension of the SMT formula in Eq. 1.

$$
\begin{aligned}
&0.2 :: \mathtt{h}. && \%0.2 \text{ chance of being a hot day}\\
&0.01 :: \mathtt{no\_cool}. && \%0.01 \text{ chance of cooling not working}\\
&\mathtt{normal}(20,5) :: \mathtt{t} \leftarrow \neg\mathtt{h}. && \%\text{temperature distribution}\\
&\mathtt{normal}(27,5) :: \mathtt{t} \leftarrow \mathtt{h}. &&\\
&\mathtt{broken} \leftarrow \mathtt{valS}(\mathtt{t},T), \mathtt{conS}(T > 30). &&\\
&\mathtt{broken} \leftarrow \mathtt{no\_cool}, \mathtt{valS}(\mathtt{t},T), \mathtt{conS}(T > 20). && (11)
\end{aligned}
$$

Looking at the program in Eq. 11, we observe two differences in comparison to orthodox ProbLog syntax. Firstly, we can describe not only Boolean random variables but also continuous random variables, and specify how the random variables are distributed. This is realized by statements of the form $D :: t \leftarrow b_1, ..., b_n$, which denotes that $t\theta$ is a continuous random variable distributed according to $D\theta$ whenever $b_1\theta, ..., b_n\theta$ are true for a substitution θ that grounds the rule. We will use the shorthand $t\theta | b_1\theta, ..., b_n\theta \sim D\theta$ (we read this *t given* b_1 *and ... and* b_n*'*). In our example we have $\mathtt{t}|\mathtt{not}(\mathtt{w}) \sim \mathcal{N}(20,5)$. The temperature random variable $\mathtt{t}$ is distributed according to a specific normal distribution given it being a hot day or not.

A second difference to ordinary ProbLog lies in allowing HAL-Problog to also encompasses conditional statements - allowing to define binary random variables that depend on continuous ones. Therefore, we utilize the two built-in predicates $\mathtt{valS}/2$ and $\mathtt{conS}/1$. $\mathtt{valS}/2$ takes as first argument a variable and the second argument unifies with a symbol representing the value of the variable. The $\mathtt{conS}/1$ predicate denotes an Iverson bracket involving symbolic values. Note that HAL-ProbLog allows to deploy conditions such as the following: $(\mathtt{t} > \mathtt{r} + 10)$ and $(\mathtt{t}^3 > e^{\mathtt{r}})$. Whether programs involving such expression can be solved or not relies on the solver. Using Symbo as a solver only programs reducible to $\mathcal{NRA}$ formulas are guaranteed to be solvable.

In order to obtain meaningful HAL-ProbLog programs, each possible world allows for only one possible definition of one and the same continuous random variable, this construct is similar to that of the Distributional Clauses (Nitti, De Laet, and De Raedt 2016; Gutmann, Jaeger, and De Raedt 2011). This effectively means that, exactly as Distributional Clauses, we only allow for mixtures of continuous random variables and is guaranteed by requiring that rules with identical heads have mutually exclusive bodies, as in distributional clauses (see (Gutmann, Jaeger, and De Raedt 2011) for formal details). Lifting this restrictions would necessitate to capture interactions between different worlds as convolutions (Lucas and Hommersom 2015).

Contrary to Distributional Clauses, however, HAL-ProbLog allows for defining one and the same discrete random variable multiple times, which results in effectively encoding a noisy-or gate. On this stance, HAL-ProbLog inherits its semantics from (a)ProbLog.

We are now able to interpret the HAL-ProbLog example program: a situation is modeled where the machine breaks down given that the temperature rises above 30 degrees or given that there is no cooling and the temperature rises above 20 degrees. The probability density modeling the temperature depends on whether it is hoy or not.

Let's move on by defining the semantics of the $\mathtt{valS}/2$ predicate. In the example in Eq. 11 we saw that one and the same random variable can be distributed according to different distributions f_i, given mutually exclusive bodies $\mathtt{b}_i$. This entails that the value of the random variable depends on the world one is in. If we have now a predicate $\mathtt{valS}(\mathtt{t}, T)$, then we accommodate for this fact by allowing the logic variable $\mathtt{T}$ to unify with all symbolic values for $\mathtt{t}$: $\mathtt{T}/\mathtt{t}_i|\mathtt{b}_i$. In other words: $\mathtt{T}$ unifies with the values of any $\mathtt{t}_i$. The different $\mathtt{t}_i$'s are distinguishable by their mutually exclusive bodies $\mathtt{b}_i$. In our example, T unifies with the value of the temperature given that it is a hot day and given that it is not a hot day.

Now we define the semantics of the $\mathtt{conS}/1$ predicate. Suppose that we have in a body of a clause an Iverson predicate $\mathtt{conS}(C)$, where the algebraic condition depends on a set of logic variables $\{\mathtt{V}_i\}$. Then we add, for each ordered set of symbolic values $\{\mathtt{v}_i\}$ that unifies with the ordered set of logic

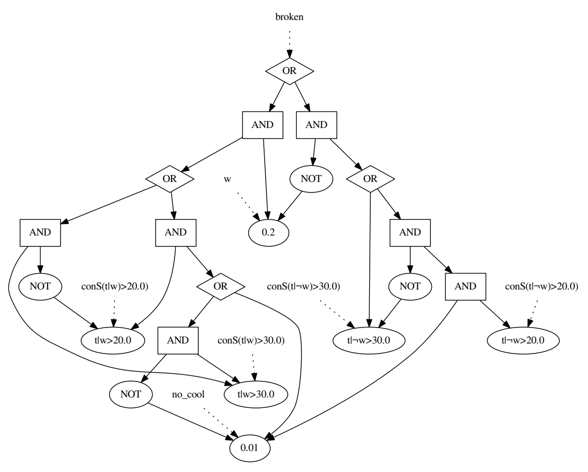

Figure 2: Graphical representation of the HAL-ProbLog program in Eq. 11 compiled into an SDD.

variables $\{\mathtt{V}_i\}$ a clause of the following form to the program

$$\left(C\theta_h, \mathcal{F}^S_{\mathtt{v}_i|\mathtt{b}_i}\right) :: \mathtt{conS}(C\theta_{iv}) \leftarrow \bigwedge_i \mathtt{b}_i. \qquad (12)$$

Where we have the substitutions $\theta_h = \{\mathtt{V}_i/\mathtt{v}_i|\mathtt{b}_i\}$ and $\theta_{iv} = \{\mathtt{V}_i/\mathtt{v}_i\}$. The $\mathtt{b}_i$'s are the bodies of clauses that have as head the random variables $\mathtt{v}_i$, respectively. $\mathcal{F}^S_{v_i|b_i}$ is short-hand for the set of distributions $\{\mathtt{v}_i|\mathtt{b}_i \sim f_i\}$.

The following example shows the effect of this transformation on the program in Eq. 11.[2]

$$\begin{aligned}
&0.2{::}\mathtt{h}. \quad 0.01{::}\mathtt{no\_cool}. && (13)\\
&(\mathtt{t}|\neg\mathtt{h} > 20, \mathtt{normal}_{\mathtt{t}|\neg\mathtt{h}}(20,5)){::}\mathtt{conS}(\mathtt{t} > 20) \leftarrow \neg\mathtt{h}.\\
&(\mathtt{t}|\mathtt{w} > 20, \mathtt{normal}_{\mathtt{t}|\mathtt{w}}(27,5)){::}\mathtt{conS}(\mathtt{t} > 20) \leftarrow \mathtt{h}.\\
&(\mathtt{t}|\neg\mathtt{h} > 30, \mathtt{normal}_{\mathtt{t}|\neg\mathtt{h}}(20,5)){::}\mathtt{conS}(\mathtt{t} > 30) \leftarrow \neg\mathtt{h}.\\
&(\mathtt{t}|\mathtt{w} > 30, \mathtt{normal}_{\mathtt{t}|\mathtt{w}}(27,5)){::}\mathtt{conS}(\mathtt{t} > 30) \leftarrow \mathtt{h}.\\
&\mathtt{broken} \leftarrow \mathtt{conS}(\mathtt{t} > 30).\ \mathtt{broken} \leftarrow \mathtt{no\_cool}, \mathtt{conS}(\mathtt{t} > 20).
\end{aligned}$$

We see that the Iverson predicate now functions as a literal. The transformation is finalized by removing the clauses whose head is a probability density from the program as they are no longer needed. This transformation is integrated within the aProbLog grounder (cf. section 5.3). Compiling the program in Eq. 13 into an SDD and calculating the probability of `broken` yields the following expression:

$$\begin{aligned}
&(1-0.2)\left[0.01 \int_{20\leq 30} \mathcal{N}_{\mathtt{t}}(20,5)d\mathtt{t} + \int_{30>\mathtt{t}} \mathcal{N}_{\mathtt{t}}(20,5)d\mathtt{t}\right]\\
&+ 0.02\left[0.01 \int_{20<\mathtt{t}\leq 30} \mathcal{N}_{\mathtt{t}}(30,5)d\mathtt{t} + \int_{30>\mathtt{t}} \mathcal{N}_{\mathtt{t}}(30,5)d\mathtt{t}\right]
\end{aligned}$$

Grounding the Iverson predicates in a HAL-ProbLog program actually results in an aProbLog program. Therefore, we say that a HAL-ProbLog program P is valid if grounding all Iverson predicates results in a valid aProbLog program, given the probability density semiring $\mathcal{S}$.

5.3 Implementation

We implemented HAL-ProbLog as an extension of the publicly available ProbLog2 system (Dries et al. 2015)[3]. This system provides a state-of-the-art implementation of the ProbLog language which is accessible as a module in Python. The ProbLog system evaluates ProbLog models through the following pipeline: (1) reads the ProbLog model and transforms it into a clausal database representation, (2) generate a propositional and-or-graph representing the queries and evidence in the model using a built-in Prolog-based grounding engine, (3) transform the and-or-graph by removing cyclic dependencies (optional), (4) compile the model into an evaluatable form using knowledge compilation. Therefor we used SDDs, which are a subset of d-DNNF, as a target representation[4] (Choi and Darwiche 2013). (5) perform weighted model counting on this formula to obtain the final probability of interest. The ProbLog2 system also provides an implementation of aProbLog by allowing custom semirings to be defined in the final step of the process.

In order to implement our system we extended the grounder (ProbLog's step (2)) with support for the `conS/1` built-in that allows us to create dependencies of discrete random variables on continuous random variables. It adds a `conS` node to the ground formula for all possible combination of ground literals that the algebraic condition involved depends on, cf. Eq. 12. Moreover, our system deploys Symbo in ProbLog's step (5) for evaluating compiled SMT($\mathcal{NRA}$) theories originating from hybrid HAL-ProbLog programs.

6 Experimental Evaluation

The question we would like to answer during the experimental evaluation is the following: How does solving hybrid probabilistic programs using Symbo, a logico-symbolic solver, compare to a pure, state-of-the-art, symbolic solver?

We answer this question by comparing HAL-ProbLog which uses Symbo to pure symbolic inference with the PSI-Solver in its native language. We compared Symbo and the PSI-Solver on the set of benchmark experiments given in section F of the Appendix in (Gehr, Misailovic, and Vechev 2016).[5]

Experiments were performed on a laptop Intel(R) i7 CPU 2.60GHz with 16 Gb memory.

In Table 1, we observe that Symbo outperforms the PSI-Solver on 9/10 benchmarks, on 7/10 even when including the time spend on the knowledge compilation step. Only for the ClickGraph benchmark PSI performs better than Symbo, which timed-out after 15s during the integration step. This is because PSI integrates out variables after loop iterations. This is not yet supported in HAL-ProbLog and Symbo ends up with a large symbolic expression that is hard to integrate

[2]The program in Eq. 13 is also labeling atoms in the heads of clauses, which is supported by the aProbLog implementation and which can easily be eliminated. (just introduce a new predicate $q(X)$ that returns $X = n$ if the n-th condition is true, and then replace the occurrences of $conS(Y)$ by $conS(Y, q(N))$).

[3]`https://dtai.cs.kuleuven.be/problog/`

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

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

over. This could be solved by, for example, using sub-queries in HAL-ProbLog, as can be done in ProbLog2.

It is generally beneficial to perform logical inference on top of symbolic inference in the hybrid and thereby also in the discrete 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.

7 Related Work

While knowledge compilation with SDDs and other representations has been used for (WMC) in probabilistic graphical models (Choi, Kisa, and Darwiche 2013) and probabilistic logic programming (Vlasselaer et al. 2016), it has to the best of our knowledge, not yet been applied to support hybrid exact inference.

W.r.t. hybrid inference in probabilistic programming, there are basically two classes of approaches: approximate and exact. Firstly, for what concerns exact inference, 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. Furthermore, our work shows that knowledge compilation can speed up the inference in PSI and that the resulting framework applies hybrid probabilistic logics, too. 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 Gaussian densities 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 is not addressed in their work.

Secondly, for what concerns approximate inference, we have the sampling approaches in distibutional clauses by (Gutmann et al. 2011; Nitti, De Laet, and De Raedt 2016) and BLOG (Milch et al. 2007). For Distributional Clauses, one uses importance sampling 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*. A node in the tree can then split up a continuous variable on an arbitrary value and for each child of the node an upper and lower probability bound can be calculated, which then gives upper and lower probability bounds at the splitting node. Going deeper in the tree yields tighter and tighter bounds.

Finally, there is the work on inference in weighted model integration (Belle, Passerini, and Van den Broeck 2015; Morettin, Passerini, and Sebastiani 2017), which handles probability densities by splitting them up (and thereby approximating) into piecewise polynomials and then carrying out exact inference. This is somewhat related also to (Gutmann, Jaeger, and De Raedt 2011), who pursued a similar procedure by restricting distributions to Gaussians which can be chopped up into easily integrable pieces. In contrast to these works, we provide a much larger class of densities and constraints.

In this line of work (Belle et al. 2016) also investigated component caching while performing a DPLL search when calculating a WMI and DPLL search is indeed related to knowledge compilation. However, the method proposed in their work is strictly limited to piecewise polynomials. We, again, completely lift this restrictions and are able to perform WMI via knowledge compilation on SMT($\mathcal{NRA}$) formulas using probability density functions instead of piecewise polynomials on SMT($\mathcal{LRA}$).

8 Conclusion

We have shown how knowledge compilation can be applied to the task of weighting model integration by leveraging algebraic model counting. We have also introduced an effective logico-symbolic solver based on this idea. Finally, we presented HAL-ProbLog, a probabilistic logic programming language that is capable of fully harnessing the logical structure underlying a hybrid probabilistic program through KC in the hybrid domain.

In future work we would like explore non-factorized weight functions in the context of knowledge compilation and weighted model integration. Especially, as these non-factorized weight functions are presently predominantly used, cf. (Morettin, Passerini, and Sebastiani 2017).

References

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., and Darwiche, A. 2013. Dynamic minimization of sentential decision diagrams. In *AAAI*.

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.

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.

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.

Lucas, P. J. F., and Hommersom, A. 2015. Modeling the interactions between discrete and continuous causal factors in bayesian networks. *International Journal of Intelligent Systems* 30(3):209–235.

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.

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.

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.

Vlasselaer, J.; den Broeck, G. V.; Kimmig, A.; Meert, W.; and Raedt, L. D. 2016. Tp-compilation for inference in probabilistic logic programs. *International Journal of Approximate Reasoning* 78:15 – 32.