How to Exploit Structure while Solving Weighted Model Integration Problems

Samuel Kolb[*] and **Pedro Zuidberg Dos Martires**[*] and **Luc De Raedt**
KU Leuven, Belgium
{samuel.kolb, pedro.zuidbergdosmartires, luc.deraedt}@cs.kuleuven.be

Abstract

Weighted model counting (WMC) is a state-of-the-art technique for probabilistic inference in discrete domains. WMC has recently been extended towards weighted model integration (WMI) in order to handle discrete and continuous distributions alike. While a number of WMI solvers have been introduced, their relationships, strengths and weaknesses are not yet well understood. WMI solving consists of two sub-problems: 1) finding convex polytopes; and 2) integrating over them efficiently. We formalize the first step as λ-SMT and discuss what strategies solvers apply to solve both the λ-SMT and the integration problem. This formalization allows us to compare state-of-the-art solvers and their behaviour across different types of WMI problems. Moreover, we identify factorizability of WMI problems as a key property that emerges in the context of probabilistic programming. Problems that can be factorized can be solved more efficiently. However, current solvers exploiting this property restrict themselves to WMI problems with univariate conditions and fully factorizable weight functions. We introduce a new algorithm, F-XSDD, that lifts these restrictions and can exploit factorizability in WMI problems with multivariate conditions and partially factorizable weight functions. Through an empirical evaluation, we show the effectiveness of our approach.

1 INTRODUCTION

Weighted model integration (WMI) [Belle et al., 2015] is the extension of the better known weighted model counting (WMC) task [Chavira and Darwiche, 2008] from probabilistic inference in the discrete domain to the continuous domain. While WMC relies on the formalism of Boolean satisfiability (SAT), WMI relies on the formalism of satisfiability modulo (Linear) Real Arithmetic (SMT($\mathcal{LRA}$)). Both problems are #P-complete in the general setting. However, following the introduction of WMI, a number of solvers [Morettin et al., 2017, Kolb et al., 2018, Zuidberg Dos Martires et al., 2019] have emerged that are able to exploit structure in WMI problems to solve them more efficiently.

Contrary to solvers for WMC, the relative advantages and drawbacks of the different WMI solvers are not yet well understood. Understanding these solvers and their differences requires a clear separation of the model counting and the integration component. The existing attempts [Belle et al., 2015, Morettin et al., 2017] to separate these two components are all tied to specific solving paradigms. To decouple the formulation in a general way, we, as a **first contribution**, introduce λ-SMT, the problem of rewriting a generic WMI problem into a sum of integrals over convex polytopes. This allows us to formally disentangle the *model counting* (solving λ-SMT) and *integration* steps (computing integrals). As a **second contribution**, we discuss the main paradigms used to solve the λ-SMT step – DPLL search and knowledge compilation – and the integration step – numeric and symbolic integration[1]. This allows us to compare different state-of-the-art solvers and understand how their design choices affect the kind of WMI problems they are able to solve efficiently. Finally, we observe that *fully factorizable* WMI problems have given rise to efficient solvers for subsets of WMI [Belle et al., 2016, Molina et al., 2018]. While *factorizability* naturally emerges in probabilistic

[*] These authors contributed equally to this work.

[1]We understand a numeric integration method as a method that outputs the value of a definite integral (not necessarily obtained through numeric approximations) and symbolic integration as the problem of finding the anti-derivative

programming, the strong conditions imposed by full factorizability – no multivariate conditions and fully factorizable weight functions – fail to cover most applications. Therefore, as a **third contribution**, we present a novel algorithm to solve generic WMI problems that can automatically exploit *factorizability* in the problem structure. We experimentally validate that our solver is able to recover factorizability of WMI problems in and can lead to exponential-to-linear speed-ups.

2 PRELIMINARIES

2.1 Satisfiability Modulo Theories (SMT)

SMT formulas generalize traditional propositional logic formulas by additionally allowing the use of expressions formulated in a background theory. More formally, following [Morettin et al., 2017]:

Definition 1. (SMT($\mathcal{LRA}$) (linear real arithmetics)) Let $\mathbf{b}$ be a set of M Boolean and $\mathbf{x}$ a set of N real variables. An **atomic formula** is an expression of the form $\sum_i c_i \cdot x_i \bowtie c$, where the $x_i \in \mathbf{x}$ and $c_i, c \in \mathbb{Q}$, and $\bowtie \in \{=, \neq, \geq, \leq, >, <\}$. We then define SMT($\mathcal{LRA}$) theories as Boolean combinations (by means of the standard Boolean operators $\{\neg, \wedge, \vee, \rightarrow, \leftrightarrow\}$) of **Boolean variables** $b_i \in \mathbf{b}$ and of **atomic formulas** over $\mathbf{x}$.

Generalizations of this definition to (non-linear) real arithmetics are given in [Zuidberg Dos Martires et al., 2019], where also interpretations of SMT formulas are defined:

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

Following again [Zuidberg Dos Martires et al., 2019], we also introduce the notion of formula abstraction.

Definition 3. (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.

2.2 Model Counting and Knowledge Compilation

State-of-the-art techniques for solving model counting problems, also called #SAT, are based on exhaustive DPLL algorithms [Birnbaum and Lozinskii, 1999], which count the number of satisfying assignments to a formula[2]. These solvers can be divided into two classes: the ones that build up a trace of the DPLL search, and the ones that do not. The latter return immediately the model count. The former build up a diagrammatic representation of the propositional formula over which the model count can be obtained efficiently. By keeping a trace, such #SAT solvers [Huang and Darwiche, 2005, Oztok and Darwiche, 2018] constitute, in fact, top-down knowledge compilation schemes.

KC [Darwiche and Marquis, 2002] has emerged as the go-to technique for dealing with the computational intractability of propositional reasoning (#P-complete [Valiant, 1979]). The key idea is to split up inference on logical formulas into an *off-line* and an *on-line* step. In the off-line step, a propositional formula is compiled from its source representation into a target representation (e.g SDDs [Choi et al., 2013]), in which repeated poly-time inference is available. As such, knowledge compilation has also been shown to be beneficial in probabilistic inference in the discrete domain [Chavira and Darwiche, 2008, Fierens et al., 2015].

#SAT can also be performed by compiling formulas bottom-up [Choi and Darwiche, 2013]. However, it has been shown [Huang and Darwiche, 2005, Oztok and Darwiche, 2018] that top-down compilation, i.e. knowledge compilation through exhaustive DPLL search, outperforms bottom-up compilers.

2.3 Weighted Model Counting and Integration

Weighted model counting generalizes #SAT. Instead of simply counting the number of satisfying assignments, one performs a weighted sum over models.

Definition 4. (WMC) Given a set $\mathbf{b}$ of M Boolean variables, a weight function $w : \mathbb{B}^M \rightarrow \mathbb{R}_{\geq 0}$, and a propositional formula ϕ (called *support*) over $\mathbf{b}$, the **weighted model count** is

$$WMC(\phi, w|\mathbf{b}) = \textstyle\sum_{\mathbf{b}_{\mathcal{I}} \in \mathcal{I}_{\mathbf{b}}(\phi)} w(\mathbf{b}_{\mathcal{I}}) \qquad (1)$$

$\mathcal{I}_{\mathbf{b}}(\phi)$ is the set of interpretations that satisfy ϕ (cf. Definition 2).

Traditionally, WMC is used when the weight function w factorizes as product of weights of literals: $WMC(\phi, w|\mathbf{b}) = \sum_{\mathbf{b}_{\mathcal{I}} \in \mathcal{I}_{\mathbf{b}}(\phi)} \prod_{b_i \in \mathbf{b}_{\mathcal{I}}} w(b_i)$. When performing probabilistic inference, we take $0 \leq w(b_i) \leq 1$ and $w(b_i) + w(\neg b_i) = 1$. The resulting sum over products is then a computation in the probability semiring [Kimmig et al., 2017].

[2]Note that solvers can use partial assignments to avoid explicitly enumerating all assignments.

Weighted model integration further generalizes WMC from propositional logical formulas to SMT formulas. Following the formulation of [Morettin et al., 2017] we give its definition:

Definition 5. (WMI) Given a set $\mathbf{b}$ of M Boolean variables, $\mathbf{x}$ of N real variables, a weight function $w : \mathbb{B}^M \times \mathbb{R}^N \rightarrow \mathbb{R}_{\geq 0}$, and a support ϕ, in the form of an SMT formula, over $\mathbf{b} \cup \mathbf{x}$, the **weighted model integral** is

$$WMI(\phi, w|\mathbf{x}, \mathbf{b}) = \sum_{\mathbf{b}_{\mathcal{I}} \in \mathcal{I}_{\mathbf{b}}(\phi)} \int_{\mathcal{I}_{\mathbf{x}}(\phi^{\mathbf{b}_{\mathcal{I}}})} w(\mathbf{x}, \mathbf{b}_{\mathcal{I}}) d\mathbf{x} \qquad (2)$$

Weighted model integration problems can be canonically encoded as an abstract syntax tree (AST) in the language used by PySMT [Gario and Micheli, 2015]. These take the form of nested *if-then-else* (ite) statements. A primitive ite expression $\text{ite}(\phi, f, 0)$ encodes the case function $\{\theta : f, \neg\theta : 0\}$, where ϕ is an SMT formula and f a real-valued function. Primitive ite's are combined through addition and multiplication, building up a richer language.

2.4 Decision Diagrams

Decision diagrams (DD) are data structures that compactly represent Boolean formulas. DDs are obtained by compiling Boolean formulas (cf. Subsection 2.2) into directed acyclic graphs. Although traditionally developed for propositional logical formulas, the idea of compiling logical formulas into a more succinct representation has also found its way into the hybrid domain, where a prominent example are XADDs [Sanner et al., 2011, Kolb et al., 2018], a language based on binary decision diagrams [Bryant, 1986]. Recently, the more succinct target representation of sentential decision diagrams (SDD) [Choi et al., 2013] has also been extended towards the hybrid domain [Zuidberg Dos Martires et al., 2019]. We will refer to these *hybrid* analogues of SDDs as *XSDDs*. Like SDDs, XSDDs represent logical formulas as nested conjunctions and disjunctions where conjuncts do not share (Boolean) variables and disjuncts are pairwise logically inconsistent. Contrary to traditional SDDs, XSDDs allow for SMT($\mathcal{LRA}$) literals as leaves. In practice, XSDDs are implemented using traditional SDD software and additional book-keeping for the non-Boolean literals.

Example 1. Consider the following SMT formula:

$$[(x{>}0) \wedge (x{<}1)] \wedge [(y{<}1) \vee ((x{>}y) \wedge (y{>}1/2))] \qquad (3)$$

The weight function is $2xy$. Compiling this formula into an XADD or XSDD, see Figure 1, enables to compute the weighted model integral:

$$\int_{(x>0)\wedge(x<1)\wedge(y<1)\wedge(x\geq y)} 2xydxdy +$$
$$\int_{(x>0)\wedge(x<1)\wedge(y>1/2)\wedge(x>y)} 2xydxdy$$

We see that the atomic formulas in the SMT($\mathcal{LRA}$) formula in Equation 3 become the integration bounds over which to integrate the weight function.

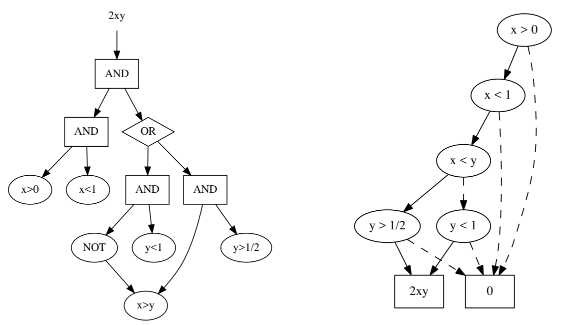

Figure 1: The expression ϕ for weight $2xy$ from Example 1 can be compiled into an equivalent XSDD (left) or XADD (right). The internal nodes in XSDDs correspond to conjunctions or disjunctions, and leaf nodes correspond to literals. In XADDs, the internal nodes are tests on literals (with solid and dashed edges for true and false branches, respectively), while the leaf nodes are polynomials. XADDs with multiple non-zero leaves can be represented by a set of (SDD, polynomial) pairs.

3 λ-SMT

In the weighted model integration literature the weight function w of a WMI problem $\text{WMI}(\phi, w|\mathbf{x}, \mathbf{b})$ over SMT($\mathcal{LRA}$) formulas and polynomial weight functions is expressed as an AST with $\mathcal{LRA}$ atoms and polynomials (cf. Section 2.3). The class of functions expressible by these ASTs is equivalent to the class of piecewise-polynomial case functions, as shown in [Kolb et al., 2018]. A piecewise-polynomial case function $f = \{\phi_1 : \omega_1, \cdots, \phi_n : \omega_n\}$ consists of tuples $\langle \phi_i, \omega_i \rangle$, where ϕ_i is a conjunction of SMT($\mathcal{LRA}$) literals and ω_i is a polynomial over $\mathbf{x}$. The *world-supports* ϕ_i form a partition of the space spanned by the Cartesian product $\mathbf{x} \times \mathbf{b}$, i.e., they are mutually exclusive (disjoint) and exhaustive (covering the whole space). Note that, as we consider only $\mathcal{LRA}$ atomic formulas, every ϕ_i corresponds to a convex polytope in the real space (contrary to the given support ϕ). All assignments to the variables in $\mathbf{x}$, $\mathbf{b}$ that satisfy ϕ_i are weighted with the polynomial *world-weight* ω_i, which no longer relies on $\mathbf{b}$.

Example 2. Consider the weight function $w(\{x, y\}, \{a\}) = \text{ite}(a, 2x+y, x^2y) \times \text{ite}((y < 5), 3, 0)$, where x and y are real variables, and a is a Boolean variable. Moreover, consider the SMT($\mathcal{LRA}$) formula $\phi{=}((a \wedge (x < 5)) \vee (x{>}y)) \wedge \mathit{bounds}$, with $\mathit{bounds} = (x{<}10) \wedge (x{>}2) \wedge (y{<}10) \wedge (y{>}2)$. We can then partition the space with the following case functions:

$$\Big\{(y < 5) \wedge (x < 5) \wedge bounds \wedge a : 6x + 3y,$$
$$(y < 5) \wedge (x \geq 5) \wedge (x > y) \wedge bounds \wedge a : 6x + 3y,$$
$$(y < 5) \wedge (x > y) \wedge bounds \wedge \neg a : 3x^2 y\Big\} \quad (4)$$

we omitted the cases where the polynomial weight is 0.

By using the notion of case functions, the weighted model integral (Eq. 2) can be rewritten as a sum of integrations of polynomials over convex polytopes. In order to show this, we first write the definite sum and integral as an indefinite sum and integral of indicator functions multiplied by the weight function w. For the indicator function we use the commonly used Iverson bracket notation $[\![\cdot]\!]$.

$$\begin{aligned}\mathrm{WMI}(\phi, w|\mathbf{x}, \mathbf{b}) &= \textstyle\sum_{\mathbf{b}_\mathcal{I} \in \mathcal{I}_\mathbf{b}(\phi)} \int_{\mathcal{I}_\mathbf{x}(\phi^{\mathbf{b}_\mathcal{I}})} w(\mathbf{x}, \mathbf{b}_\mathcal{I}) d\mathbf{x} \\ &= \textstyle\sum_\mathbf{b} \int [\![\phi(\mathbf{x}, \mathbf{b})]\!] w(\mathbf{x}, \mathbf{b}) d\mathbf{x} \qquad (5)\end{aligned}$$

We continue manipulating this expression by 1) rewriting $[\![\phi(\mathbf{x}, \mathbf{b})]\!] w(\mathbf{x}, \mathbf{b})$ as a case-function $f(\mathbf{x}, \mathbf{b})$ with tuples $\mathcal{W}_f = \{\langle \phi_i, \omega_i \rangle\}$, 2) exploiting the mutual exclusivity of the world-supports to rewrite $f(\mathbf{x}, \mathbf{b}) = \sum_{\langle \phi_i, \omega_i \rangle \in \mathcal{W}_f} [\![\phi_i(\mathbf{x}, \mathbf{b})]\!] \omega_i(\mathbf{x})$, and 3) reordering the summations:

$$\begin{aligned}&\textstyle\sum_\mathbf{b} \int f(\mathbf{x}, \mathbf{b}) d\mathbf{x} && (6)\\ &= \textstyle\sum_\mathbf{b} \int \sum_{\langle \phi_i, \omega_i \rangle \in \mathcal{W}_f} [\![\phi_i(\mathbf{x}, \mathbf{b})]\!] \omega_i(\mathbf{x}) d\mathbf{x} && (7)\\ &= \textstyle\sum_{\langle \phi_i, \omega_i \rangle \in \mathcal{W}_f} \sum_\mathbf{b} \int [\![\phi_i(\mathbf{x}, \mathbf{b})]\!] \omega_i(\mathbf{x}) d\mathbf{x} && (8)\end{aligned}$$

Finally, we push the Iverson brackets back into the index and the bound of the summation and integral, respectively.

$$\textstyle\sum_{\langle \phi_i, \omega_i \rangle \in \mathcal{W}_f} \sum_{\mathbf{b}_\mathcal{I} \in \mathcal{I}_\mathbf{b}(\phi_i)} \int_{\mathcal{I}_X(\phi_i^{\mathbf{b}_\mathcal{I}})} \omega_i(\mathbf{x}) d\mathbf{x} \qquad (9)$$

In accordance to the terms world-support and world-weight, we define the *world-volume* vol w.r.t. to a tuple $\langle \phi_i, \omega_i \rangle$ as:

$$\mathrm{vol}(\phi_i, \omega_i | \mathbf{x}, \mathbf{b}) = \textstyle\sum_{\mathbf{b}_\mathcal{I} \in \mathcal{I}_\mathbf{b}(\phi_i)} \int_{\mathcal{I}_\mathbf{x}(\phi_i^{\mathbf{b}_\mathcal{I}})} \omega_i(\mathbf{x}) d\mathbf{x} \qquad (10)$$

Solving any WMI problem can thus be reduced to a two step procedure: 1) rewriting the problem into a sum over tuples $\langle \phi_i, \omega_i \rangle$ of disjoint world-supports and world-weights (i.e., convex polytopes and polynomials), and 2) integrating every world-weight ω_i over the corresponding world-support ϕ_i. We formally define the first step of this procedure as *λ-SMT*:

Definition 6. (λ-SMT) **Given** a WMI problem $\mathrm{WMI}(\phi, w|\mathbf{x}, \mathbf{b})$, **find** a set $\mathcal{W}$ of pairwise logically inconsistent world-supports ϕ_i (i.e., their conjunction is unsatisfiable) and world-weights ω_i such that the sum over their world-volumes is equal to the weighted model integral:

$$\textstyle\sum_{\langle \phi_i, \omega_i \rangle \in \mathcal{W}} \mathrm{vol}(\phi_i, \omega_i | \mathbf{x}, \mathbf{b}) = \mathrm{WMI}(\phi, w|\mathbf{x}, \mathbf{b}) \qquad (11)$$

We call a tuple $\langle \phi_i, \omega_i \rangle \in \mathcal{W}$ *redundant* if ϕ_i is not satisfiable (i.e., logically inconsistent) or $\omega_i = 0$, since integrating over them yields 0. Further, we call a λ-SMT solution $\mathcal{W}^*$ *reduced*, if no element in $\mathcal{W}^*$ is redundant.

The λ-SMT problem lies at the heart of all WMI solvers, and it is easy to see that such sets $\mathcal{W}$ always exist, as any WMI problem can be written as a case-function (see above and [Kolb et al., 2018]) and enumerating all cases yields a solution to the λ-SMT problem. In order to obtain a reduced solution, we can discard all redundant cases – doing this efficiently is a key part of WMI-solving.

Conceptually similar 2-step decompositions of WMI solving can be found in prior works. In [Morettin et al., 2017, Definition 5], the authors separate the steps of finding truth assignments to the Boolean variables and solving non-Boolean WMI problems (WMI_{nb}). In [Kolb et al., 2018, Section 3], a case function representation is built by compiling the WMI problem to an XADD. We continue this discussion in Section 5.

4 ANATOMY OF A SOLVER

With the formal definition of λ-SMT we can now study how different solvers handle this problem and how the representation of the solution to the λ-SMT problem influences strategies to solve the subsequent integration step. Even though the λ-SMT and the integration steps can be intertwined, there are broadly two ways to tackle the λ-SMT step, and two ways to tackle integration.

4.1 λ-SMT: Search vs Compilation

λ-SMT concerns the analysis of the structure of the WMI problem, finding a way to rewrite the problem into disjoint, convex polytopes with purely polynomial weight functions. The techniques used to solve this part of the problem relate closely to techniques used for WMC and #SAT. On the one hand, solvers such as PA [Morettin et al., 2017] and PRAiSE [De Salvo Braz et al., 2016] use variants of DPLL search to enumerate the conditions of the polytopes over which to integrate. On the other hand, XADD-based solvers [Sanner et al., 2011, Kolb et al., 2018] and SDD-based Symbo [Zuidberg Dos Martires et al., 2019] use knowledge compilation to compile the problem structure into a compilation language in which the solutions to the λ-SMT problem are efficiently represented.

A key aspect to solving λ-SMT efficiently is detecting redundant tuples $\langle \phi, \omega \rangle$ early in the solving process. For many solvers, the support ϕ of a WMI problem plays a crucial role in avoiding the enumeration of redundant tuples by restricting their search to the feasible space described by the support.

Solvers relying on compilation aim to find small representations for λ-SMT solutions. By checking for redundant tuples during or after the compilation, they try to find reduced λ-SMT solutions. Additionally, they often *compress* the representation of an λ-SMT solution by grouping tuples $\langle \phi_1, \omega \rangle, ..., \langle \phi_n, \omega \rangle$ that have the same world-weight ω as $\langle \bigvee_i \phi_i, \omega \rangle$ (e.g., by using DAGs [Kolb et al., 2018]).

The set $\mathcal{W}$ can be further compressed when a solver supports the concept of a *labeling function*. The labeling function originates from the WMC literature, where a labeling function α maps literals to real-valued weights. This allows them to factorize the weight function over the Boolean variables: $w = \prod_{b \in \mathbf{b}} \alpha_b(b)$. The Symbo solver [Zuidberg Dos Martires et al., 2019] reuses the notion of a labeling functions to *partially* factorize the weight function w as $w(\mathbf{x}, \mathbf{b}) = w'(\mathbf{x}, \mathbf{b}) \cdot \prod_{b \in \mathbf{b}} \alpha_b(b)$ or $w'(\mathbf{x}, \mathbf{b}) \cdot \prod_{b \in \mathbf{b}} \alpha_b(\mathbf{x}, b)$. Consider two tuples $\langle \phi_1, \omega_1 \rangle$ and $\langle \phi_2, \omega_2 \rangle$ of a λ-SMT solution, where $\phi_1 = \phi_s \wedge b$ and $\phi_2 = \phi_s \wedge \neg b$ and $b \in \mathbf{b}$. If, then, $\omega_1 = \omega_s \cdot \alpha_b(\texttt{true})$ and $\omega_2 = \omega_s \cdot \alpha_b(\texttt{false})$, we can group these cases as $\langle \phi_s, \omega_s \cdot \alpha_b(b) \rangle$. Labeling functions over multiple literals can thus allow a number of tuples exponential in the number of literals to be grouped together (e.g., within a single SDD [Zuidberg Dos Martires et al., 2019]). Note, that we now have to relax the fact that the world weight does not depend on $\mathbf{b}$. We can reintroduce this dependency on $\mathbf{b}$ as long as for any Boolean instantiation, the world-weight will be polynomial.

4.2 Numeric vs Symbolic Integration

With respect to integration, WMI solvers fall into two categories: those using *numeric* integration, and those using *symbolic* integration. Given the function w and the set of free variables $\mathbf{x}$ to integrate, numeric integration approaches directly compute the result as a real number. Symbolic integration approaches will instead integrate out the variables one-by-one, obtaining symbolic intermediate results (i.e., repeated variable elimination). Integrating out a variable that has multiple symbolic lower- or upper-bounds causes these expressions to grow quickly. Therefore, numeric integration procedures are usually more efficient at performing individual integrations.

There are, however, two key advantages of symbolic integration, as demonstrated in [Kolb et al., 2018]. First, symbolic integration can be used to solve structured problems by reusing intermediate results across different vol(ϕ_i, ω_i) computations. In doing so, it aims to avoid having to compute a potentially exponential number of individual integrations. This reuse resembles caching in traditional DPLL search and exploits the compression of the λ-SMT models discussed in the previous section. Secondly, when computing the probabilities of multiple queries Q over a small subset of variables $\mathbf{x}_Q$, the result r of integrating out all variables except those in $\mathbf{x}_Q$ can be computed symbolically and then reused to quickly compute query probabilities, integrating out $\mathbf{x}_Q$ from $r \cdot q, \forall q \in Q$ [Kolb et al., 2018]. Note that this inference scheme is similar to the *compile once, query multiple times* knowledge compilation approach.

5 EXISTING SOLVERS

In this section we categorize current solvers by analyzing how they handle both the λ-SMT problem and integration.

Predicate Abstraction [Morettin et al., 2017] (PA)
The PA solver uses the MathSAT [Cimatti et al., 2013] SMT solver to solve WMI problems. First, it solves the λ-SMT problem by introducing fresh Boolean variables for the SMT conditions in the weight function and performing a two-step DPLL search. In the first step, it finds truth assignments to the original and fresh Boolean variables. Such an assignment defines a set $\mathcal{W}_i$ of world-supports (defined by the original variables) with a common world-weight (defined by the fresh variables). If, after using the SMT solver to remove redundant tuples from $\mathcal{W}_i$, $\mathcal{W}_i$ contains more than one world-support, the second step enumerates them by finding truth assignments to the $\mathcal{LRA}$ atoms in the support. The PA solver avoids enumerating sets of equivalent world-supports by using *partial assignments*. Second, for every world-support ϕ_i, it uses numeric integration to integrate the polynomial over the convex polytope specified by ϕ_i. For the integration it relies on the Latte integration software [De Loera et al., 2013].

Bound Resolution [Kolb et al., 2018] (BR)
The BR solver tackles a WMI problem by first compiling it to an equivalent XADD, which represents a solution to the λ-SMT problem. Compilation is performed through a bottom-up compilation scheme[3], using recursive *apply* operations. Pruning unsatisfiable paths within XADDs can be done using an LP or SMT oracle. While it is an expensive operation, it is crucial to keep the XADDs small and efficient. Paths in the XADD correspond to tuples in $\mathcal{W}$ and the DAG structure of the XADD compresses

[3]The authors refer to their compilation as top-down, however, within the knowledge compilation literature, this type of recursive compilation is commonly referred to as bottom-up.

paths with the same world-weight. Instead of performing integration separately for every path and summing the results (which could be done using symbolic or total integration), the BR algorithm exploits overlapping paths using symbolic integration. The algorithm recursively integrates every variable, tracking only one pair of upper- and lower-bound at a time, and building the result of the integration dynamically as a new XADD (integration is a closed operation for polynomial case functions with linear conditions).

Symbo [Zuidberg Dos Martires et al., 2019]

The Symbo solver first computes a compressed representation of an λ-SMT solution $\mathcal{W}$ in the form of pairs of XSDDs (representing $\bigvee_i \phi_i$) and distinct world-weights ω. XSDD compilation is performed bottom-up and, as current SDD software is strictly discrete, fresh Boolean variables are introduced as abstractions for atomic SMT formulas. Currently, pruning redundant tuples within the circuit is not supported, instead, unfeasible world-supports are detected only later, at the integration stage. Symbo supports labeling functions (see Section 4.1), which enables a more compressed representation of $\mathcal{W}$. Solving and integrating requires a bottom-up pass over the circuit that reassembles the tuples $\mathcal{W}$, checking for redundancy during the assembly and, finally, integrating the different (weighted) tuples using a symbolic integration engine. Both inconsistency checking and symbolic integrations are performed using the computer algebra system PSI [Gehr et al., 2016].

PRAiSE [De Salvo Braz et al., 2016]

Another DPLL based solver is PRAiSE, which, contrary to PA, performs symbolic integration, representing intermediate results as symbolic expression trees. These expression trees can be seen as tree-based alternatives to XADDs. They represent a compiled version of the problem *after* (symbolic) integration. However, as PRAiSE does not keep a trace of its DPLL search, it is foremost a search based approach.

Other solvers

As mentioned in the section on PA, other DPLL based methods exist, however we consider them superseded by PA. A functionally different solver is **CC** solver [Belle et al., 2016], which also performs DPLL search over abstractions of SMT formulas but introduces component caching to compute solutions more efficiently. However, the solver is restricted to a subset of WMI, requiring axis-aligned (univariate) $\mathcal{LRA}$ atoms in the real space and weight functions that factorize over all Boolean and real variables [Belle et al., 2016, Proof Theorem 6].

We also introduce a solver denoted as *XSDD(Latte)*. XSDD(Latte) follows the same strategy as Symbo to solve the λ-SMT problem using SDD compilation, but it then builds a list of all convex polytopes by doing a bottom-up evaluation of the circuit and evaluates them using numeric integration with Latte.

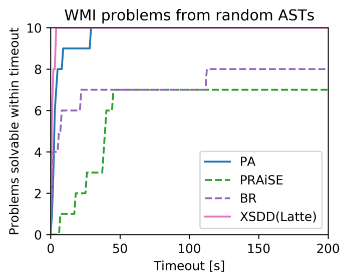

Figure 2: We test the performance of several solvers on a set of random WMI problems generated using the PA benchmark generator. For problems that have little structure and dense inequalities, solvers using numeric integration (full lines) perform better than solvers using symbolic integration (dashed lines). Since the problems are relatively shallow, the differences in solving the λ-SMT problem are secondary (runtimes include compilation).

Table 1: Overview of the solvers discussed and their properties.

	PA	BR	Symbo	PRAiSE
λ-SMT				
DPLL	✓			✓
Compilation		XADD	XSDD	
Integration				
Numeric	Latte			
Symbolic		XADD	PSI (Tree)	Exp. Tree

Discussion

The solver design choices (see Table 1 for an overview), search vs compilation, and numeric vs symbolic integration, as well as how to implement them have a big influence on what problems the solver will be able to solve efficiently. On the one hand, solving the λ-SMT problem through DPLL avoids a potentially expensive compilation step and is less sensitive to the problem formulation. This allows the PA solver to efficiently tackle problems like their road-network problem [Morettin et al., 2017]. On the other hand, compiling a circuit representation of λ-SMT can avoid recomputing λ-SMT solutions as the circuit can be efficiently combined with other circuits (representing, e.g., queries for conditional inference). Moreover, solving structured problems in which many world-supports share a common (base) world-weight, for example, problems using mutual exclusivity constraints over terms, and computing multiple query probabilities benefit from knowledge compilation and symbolic integration [Kolb et al., 2018]. Caching in DPLL-based approaches is used in a similar spirit, however, no current DPLL-based WMI solver exploits caching for WMI problems with non-axis aligned $\mathcal{LRA}$ atoms. When integration steps are not reusable, numeric integration approaches will fare better than symbolic integration for single WMI queries (see Fig. 2).

6 FACTORIZABLE WMI PROBLEMS

There is an additional type of structure that has been exploited in the WMI literature to speed up inference: *factorizable* WMI problems, which is a subset of WMI [Belle et al., 2016, Molina et al., 2018]. However, it is usually subjected to the strong constraints of using only axis-aligned $\mathcal{LRA}$ atoms and *fully* factorizable weight functions. In this subset of WMI problems, the computation of $\mathrm{vol}(\phi, \omega)$ for any tuple $\langle \phi_i, \omega_i \rangle = \langle \bigwedge_{x \in \mathbf{x}} \phi_x(x), \prod_{x \in \mathbf{x}} \omega_x(x) \rangle$ can be rewritten as $\mathrm{vol}(\phi, \omega) = \int \prod_{x \in \mathbf{x}} [\![\phi_x(x)]\!] \omega_x(x) d\mathbf{x}$. This formulation allows the integrations to be *pushed inwards* for any partition of the variables $\mathbf{x}$ into disjoint sets $\mathbf{x}_1$ and $\mathbf{x}_2$: $\int \prod_{x \in \mathbf{x}_1} [\![\phi_x(x)]\!] \omega_x(x) \left[\int \prod_{y \in \mathbf{x}_2} [\![\phi_y(y)]\!] \omega_y(y) d\mathbf{x}_2 \right] d\mathbf{x}_1$.

Factorization is less straight-forward in the case of general WMI (which includes non-axis aligned $\mathcal{LRA}$ atoms). However, many problems, especially those coming from a probabilistic programming context, are partially factorizable [Gehr et al., 2016]. Therefore, we introduce *F-XSDD*, a new solver that exploits factorized solving for WMI problems with multivariate inequalities and weight functions that might not factorize completely. Our method F-XSDD compiles a WMI problem into a set of XSDDs, performs a static circuit analysis and structures WMI as factorized, symbolic integration over the XSDDs.

6.1 Factorized Solving

After compiling a λ-SMT solution $\mathcal{W}$ for a WMI problem $\mathrm{WMI}(\phi, w | \mathbf{x}, \mathbf{b})$, every set of tuples $\langle \phi_i, \omega \rangle$ sharing the same world-weight ω is grouped as $\langle \bigvee_i \phi_i, \omega \rangle$. Each disjunction of world-supports $(\bigvee_i \phi_i)$ is represented as an XSDD D[4]. Computing $\mathrm{WMI}(\phi, w | \mathbf{x}, \mathbf{b})$ now consists of summing over all $\langle D, \omega \rangle$ pairs and computing $\sum_{\phi_i} \mathrm{vol}(\phi_i, \omega)$ for every pair. Instead of integrating every $\langle \phi_i, \omega \rangle$ tuple separately, however, we first reconsider jointly integrating ω over $D = \bigvee_i \phi_i$. Using indefinite sums and integrals, and Iverson brackets (like in Eq. 5), we can rewrite:

$$\sum_{\phi_i} \mathrm{vol}(\phi_i, \omega) = \sum_{\phi_i} \sum_{\mathbf{b}} \int [\![\phi_i(\mathbf{x}, \mathbf{b})]\!] \omega(\mathbf{x}) d\mathbf{x} \quad (12)$$

$$= \sum_{\mathbf{b}} \int [\![\bigvee_i \phi_i(\mathbf{x}, \mathbf{b})]\!] \omega(\mathbf{x}) d\mathbf{x} \quad (\phi_i \text{ are disjoint}) \quad (13)$$

$$= \sum_{\mathbf{b}} \int [\![D(\mathbf{x}, \mathbf{b})]\!] \omega(\mathbf{x}) d\mathbf{x} \quad (\bigvee_i \phi_i \text{ as XSDD } D) \quad (14)$$

By representing the disjunction as an XSDD, we can use its nested representation to push integrations *inside*, i.e., towards the leaves of the SDD. The intuition is that if the expression D contains disjoint sub-expressions over independent sets of variables, those sub-expressions can be integrated separately and the results can be reused if those sub-expressions occur multiple times (we give an example in the supplementary material).

[4]XSDD compilation follows the XADD compilation scheme described in [Kolb et al., 2018] using tuples $\langle D, \omega \rangle$ of SDDs and polynomials to represent case-functions.

Algorithm 1 Factorized Integration

```
1:  world-weight $\omega$
2:  procedure vol(XSDD $D$, vars $\mathbf{x}$)
3:      if $\mathbf{x} = \emptyset$ then
4:          return $[\![D]\!]$
5:      else if $D$ is terminal then
6:          return $\int [\![D]\!] \prod_{x \in \mathbf{x}} \omega_x(x) d\mathbf{x}$
7:      else if $D = \bigvee_c D_c$ then
8:          return $\sum_c \mathrm{vol}(D_c, \mathbf{x})$
9:      else if $D = D_1 \wedge D_2$ then
10:         $\mathbf{x}_s = \mathbf{x} \cap \mathrm{vars}(D_1) \cap \mathrm{vars}(D_2)$
11:         $\mathbf{x}_1^*, \mathbf{x}_2^* = \mathrm{vars}(D_1) \setminus \mathbf{x}_s, \mathrm{vars}(D_2) \setminus \mathbf{x}_s$
12:         $r_1 = \mathrm{vol}(D_1, \mathbf{x}_1^* \cap \mathbf{x})$
13:         $r_2 = \mathrm{vol}(D_2, \mathbf{x}_2^* \cap \mathbf{x})$
14:         return $\int r_1 \cdot r_2 \cdot \prod_{x \in \mathbf{x}_s} \omega_x(x) d\mathbf{x}_s$
```

Consider, first, the case of a fully factorizable world-weight $\omega(\mathbf{x}) = \prod_{x \in \mathbf{x}} \omega_x(x)$, an assumption we will revisit later. Using the corresponding XSDD D, we can decompose the integral recursively into smaller sub-problems.

OR Node
If D corresponds to an OR node, i.e., $D = \bigvee_{D_c} D_c$ where all D_c (again SDDs) are mutually exclusive, we obtain:

$$\sum_{\mathbf{b}} \int [\![\bigvee_{D_c} D_c(\mathbf{x}, \mathbf{b})]\!] \omega(\mathbf{x}) d\mathbf{x} \quad (15)$$

$$= \sum_{\mathbf{b}} \int (\sum_{D_c} [\![D_c(\mathbf{x}, \mathbf{b})]\!]) \omega(\mathbf{x}) d\mathbf{x} \quad (16)$$

$$= \sum_{D_c} \sum_{\mathbf{b}} \int [\![D_c(\mathbf{x}, \mathbf{b})]\!] \omega(\mathbf{x}) d\mathbf{x} \quad (17)$$

AND node
If $\phi(\mathbf{x})$ corresponds to an AND node, i.e., $D(\mathbf{x}, \mathbf{b}) = D_1(\mathbf{x}_1, \mathbf{b}) \wedge D_2(\mathbf{x}_2, \mathbf{b})$, and we denote $\mathbf{x}_s = \mathbf{x}_1 \cap \mathbf{x}_2$, $\mathbf{x}_1^* = \mathbf{x}_1 \setminus \mathbf{x}_s$, $\mathbf{x}_2^* = \mathbf{x}_2 \setminus \mathbf{x}_s$, $\omega_{\mathbf{x}} = \prod_{x \in \mathbf{x}} \omega_x(x)$, $D_1 = D_1(\mathbf{x}_1, \mathbf{b})$, $D_2 = D_2(\mathbf{x}_2, \mathbf{b})$, we obtain:

$$\sum_{\mathbf{b}} \int [\![D_1 \wedge D_2]\!] \omega_{\mathbf{x}} d\mathbf{x} = \quad (18)$$

$$\sum_{\mathbf{b}} \int [\![D_1]\!] [\![D_2]\!] \omega_{\mathbf{x}_1^*} \omega_{\mathbf{x}_2^*} \omega_{\mathbf{x}_s} d\mathbf{x} = \quad (19)$$

$$\sum_{\mathbf{b}} \int \Big[\int [\![D_1]\!] \omega_{\mathbf{x}_1^*} d\mathbf{x}_1^* \Big] \Big[\int [\![D_2]\!] \omega_{\mathbf{x}_2^*} d\mathbf{x}_2^* \Big] \omega_{\mathbf{x}_s} d\mathbf{x}_s = \quad (20)$$

$$\int \Big[\sum_{\mathbf{b}_1} \int [\![D_1]\!] \omega_{\mathbf{x}_1^*} d\mathbf{x}_1^* \Big] \Big[\sum_{\mathbf{b}_2} \int [\![D_2]\!] \omega_{\mathbf{x}_2^*} d\mathbf{x}_2^* \Big] \omega_{\mathbf{x}_s} d\mathbf{x}_s \quad (21)$$

This decomposition allows us to compute weighted model integrals using a recursive symbolic integration algorithm (Alg. 1). Given the world-weight ω, XSDD D and variables to integrate $\mathbf{x}$, the algorithm computes $\sum_{\mathbf{b}} \int [\![D(\mathbf{x}, \mathbf{b})]\!] \omega(\mathbf{x}) d\mathbf{x}$. If the set of variables to integrate over is empty, the algorithm returns $[\![D]\!]$ (the integrations will occur higher in the circuit). If D is a literal (leaf of the XSDD), the integral over $[\![D]\!]$ is computed for the variables $\mathbf{x}$. If D is an OR node, the variables $\mathbf{x}$ are recursively integrated from the child nodes and the results are summed (line 8). Finally, if D is an AND node, the subsets of $\mathbf{x}$ that only occur in one of the child nodes are recursively integrated out and multiplied ($r_p \cdot r_s$), and the remaining subset $\mathbf{x}_s \subseteq \mathbf{x}$ that occur in both children are integrated out from the resulting expression (line 14). The vars values (lines 10 and 11) are precomputed using static circuit analysis.

Let us briefly explain why the outer sum over the Boolean variables $\mathbf{b}$ does not occur in the algorithm. Conjuncts of conjunctions in SDDs do not share Boolean variables and disjuncts of disjunctions in SDDs are pairwise logically inconsistent. We further assume that the algorithm is applied to a *smoothed* circuit, i.e., logically irrelevant Boolean variables are not dropped from the circuit. The sum over the truth values of a Boolean can be pushed into the integration (cf. Equations 21) until it reaches a disjunction (cf. Equations 17) for which every disjunct is logically consistent only for one of the truth values, which eliminates the sum over that Boolean. For non-smoothed circuits such a disjunction does not necessarily exist.

In order to avoid traversing the SDD multiple times unnecessarily, intermediate results are cached using the tuple $\langle$SDD D, variables $\mathbf{x}\rangle$ as key. Nodes will only be revisited if different variables need to be integrated out from them. If a node is visited multiple times with different sets of integration variables, the common subset of variables could be detected using circuit analysis and integrated out first. The result can then be cached and reused.

Generic weight functions We describe three mechanisms to relax the condition that world-weight functions have to factorize as expressions over single variables. First, any polynomial can be rewritten into sums of products over powers of single variables. Integration can then be performed separately – each product fully factorizes into expressions over single variables – and the results can be summed to obtain the final result. Second, if the world-weight function factorizes into several products of expressions over non-overlapping sets of variables, we can group these sets of variables together, treating them as inseparable units (and substituting each inseparable unit by a single "set-variable" in the algorithm). If this factorization is not readily apparent, i.e., it is not a product of expressions that do not share real variables, the world-weight could be analyzed upfront to find a suitable decomposition. Third, F-XSDDs, like Symbo, support labeling functions (see Section 4.1) which occur frequently in the context of probabilistic programming [Kimmig et al., 2011]. To use a labeling function α, the factorized integration algorithm requires two changes: 1) for any Boolean literal l over a Boolean variable b, the evaluation of $[\![l]\!]$ has to be replaced by the label assigned to that literal $\alpha_b(l)$; and 2) if labels are polynomials, vars(D) has to include the real variables occurring in the labels of Boolean literals in D (to prevent integrating out those real variables too early).

Circuit ordering and redundancy To what extent a WMI problem can be exploited by factorized solving depends on the problem itself, the structure of the XSDD and the ordering of the literals in the XSDD. Currently, compilation procedures for XSDDs do not take into account that some of the literals are abstractions of inequalities, and, therefore, they do not use information about the real variables and their occurrences in inequalities and labels.

6.2 Experimental Evaluation

In this section we want to answer three research questions. **Q1** Can the F-XSDD solver exploit factorizability in WMI problems? **Q2** What is the influence of the symbolic integration back-end on F-XSDD solver (F-XSDD(PSI) vs F-XSDD(BR)? **Q3** How does F-XSDD perform compared to state-of-the-art solvers?

In our experimental evaluation we compare the state-of-the-art solvers described in Section 5 to a variety of XSDD based solvers. F-XSDD(PSI) and F-XSDD(BR) are both implementations of our factorized integration algorithm. F-XSDD(PSI) uses PSI as its symbolic integration back-end, while F-XSDD(BR) relies on XADDs to represent symbolic intermediate results and uses the BR (bound-resolution) algorithm to perform symbolic integration (on the XADDs). Using the BR algorithm within the F-XSDD solving scheme allows for having different variable integration orderings in different subdiagrams and to use labeling functions in combinataion with XADDs. When a problem does not factorize, F-XSDD(BR) reduces to BR with the overhead of performing a static circuit analysis. XSDD(PSI) and XSDD(BR) are implementations of F-XSDD(PSI) and F-XSDD(BR) where we turned off the factorization. XSDD(PSI) is a reimplementation of Symbo [Zuidberg Dos Martires et al., 2019]. XSDD(Sampling) is functionally equivalent to XSDD(Latte) (cf. Section 5, Other Solvers), using a simple rejection-sampling based backend for Monte Carlo integration (10^5 samples per integration).[5]

We test the solvers on four WMI problem-templates, whose size can be controlled by a parameter (n). The **click-graph** problem is a probabilistic program and part

[5]All experimentally evaluated solvers are part of the pywmi library [Kolb et al., 2019] (http://pywmi.org).

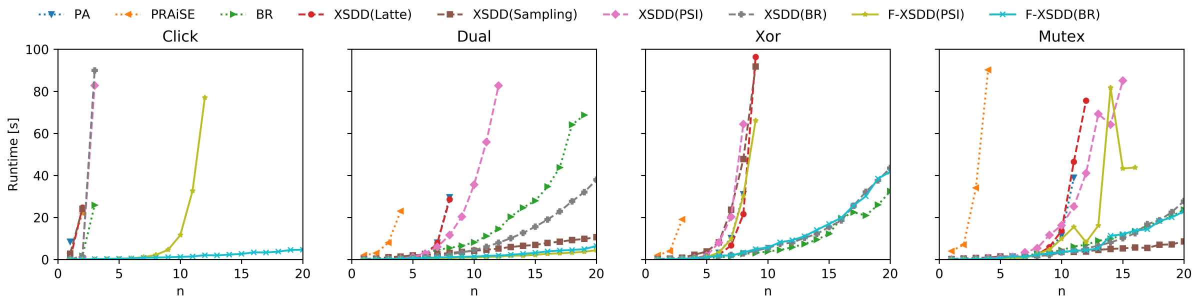

Figure 3: On the *click-graph* (far-left) and pairwise-factorizable *dual-mutex* (middle-left) problems, F-XSDD outperforms all exact state-of-the-art solvers, while on the highly structured *xor* (middle-right) and *mutually-exclusive* (far right) problems, it achieves performance on par with BR when using XADDs and bound-resolution as integration back-end (runtimes include compilation).

of the benchmark used to compare Symbo and PSI [Zuidberg Dos Martires et al., 2019], and is the problem both solvers struggled with most. We encoded the problem as a WMI problem template. We introduce a synthetic problem, **dual-mutex**, that is both structured and factorizable with $\phi = (\bigvee_i (x_{i0} \leq x_{i1})) \wedge \bigwedge_{i,j\neq i} \neg(x_{i0} \leq x_{i1}) \vee \neg(x_{j0} \leq x_{j1})$ and $w = 1$. Additionally, we encoded the highly-structured **mutually-exclusive** and **xor** problems from [Kolb et al., 2018]. Our experimental results are shown in Figure 3.

We can answer **Q1** affirmatively, as the performance of both F-XSDD solvers on click-graph and dual-mutex demonstrates that it can succesfully exploit the high degree of factorizability of those problems.

For **Q2**, we can observe that the BR solver and the corresponding XADD structure is required to solve the highly structured mutually-exclusive and xor problems. While both back-ends perform similarly on dual, only F-XSDD(BR) can achieve an exponential-to-linear in reduction time for the click-graph problem by combining factorizable solving with XADDs with BR. We can see the effect of using DAGs to compactly represent intermediate symbolic results and exploiting the overlapping paths using bound resolution. However, the results also clearly show that the BR solver is unable to efficiently solve these problems *without* factorized solving.

With respect to **Q3** we can clearly see that F-XSDD(BR) consistently delivers best-in-class results across these benchmark problems. The F-XSDD solvers outperform the numeric solvers PA and XSDD(Latte) on these structured problems. On the click-graph, mutually-exclusive and xor problems we see that F-XSDD(PSI), like PRAiSE, suffers from its tree-based representation for intermediate symbolic results. The performance of XSDD(Sampling) indicates that the number of tuples in the solutions of the λ-SMT problem of dual-mutex and mutually-exclusive grows polynomially, while for the click-graph and xor problems, they exhibit exponential growth. This demonstrates the need for symbolic integration and the ability to reuse symbolic integration steps in these cases.

6.3 Beyond Piecewise-Polynomial WMI

Our implementation of factorized solving focuses on WMI problems with SMT($\mathcal{LRA}$) atoms and piecewise polynomial densities. This puts us on even footing with most of the WMI literature [Belle et al., 2015, Morettin et al., 2017, Kolb et al., 2018]. However, factorized solving does not require piecewise *polynomial* densities, the expressiveness depends on the back-end that is used for symbolic computations. As a matter of fact, by using PSI as a back-end for symbolic expression manipulation, our factorized solving implementation can already deal with common probability distributions such as Gaussians out of the box.

7 CONCLUSIONS

We introduced the problem of λ-SMT, which allowed us to dissect in detail different state-of-the-art solvers. Moreover, we introduced F-XSDD, a novel solver that exploits factorizable weight functions through static circuit analysis and that outperforms or is on par with the state-of-the-art.

A promising road for future research would be to realise an XSDD implementation that treats $\mathcal{LRA}$ literals as first-class citizens, e.g., through a top-down knowledge compiler for SMT formulas, combining the strengths of DPLL search and knowledge compilation.

Acknowledgements

This work has received funding from ERC AdG SYNTH(694980) and from CHIST-ERA and Research Foundation-Flanders (FWO) under the H2020 project ReGround (G0D7215N). Samuel Kolb is supported by the FWO. We thank Stefano Teso for his helpful comments.

References

[Belle et al., 2015] Belle, V., Passerini, A., and Van den Broeck, G. (2015). Probabilistic Inference in Hybrid Domains by Weighted Model Integration. In *IJCAI*, pages 2770–2776.

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

[Birnbaum and Lozinskii, 1999] Birnbaum, E. and Lozinskii, E. L. (1999). The Good Old Davis-Putnam Procedure Helps Counting Models. *Journal of Artificial Intelligence Research*, 10:457–477.

[Bryant, 1986] Bryant, R. E. (1986). Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers*, 35(8).

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

[Choi and Darwiche, 2013] Choi, A. and Darwiche, A. (2013). Dynamic Minimization of Sentential Decision Diagrams. In *AAAI*.

[Choi et al., 2013] Choi, A., Kisa, D., and Darwiche, A. (2013). Compiling Probabilistic Graphical Models Using Sentential Decision Diagrams. In *European Conference on Symbolic and Quantitative Approaches to Reasoning and Uncertainty*.

[Cimatti et al., 2013] Cimatti, A., Griggio, A., Schaafsma, B. J., and Sebastiani, R. (2013). The mathsat5 SMT Solver. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 93–107.

[Darwiche and Marquis, 2002] Darwiche, A. and Marquis, P. (2002). A Knowledge Compilation Map. *Journal of Artificial Intelligence Research*, 17(1):229–264.

[De Loera et al., 2013] De Loera, J., Dutra, B., Koeppe, M., Moreinis, S., Pinto, G., and Wu, J. (2013). Software for Exact Integration of Polynomials over Polyhedra. *Comput. Geom.*, 46(3):232–252.

[De Salvo Braz et al., 2016] De Salvo Braz, R., O'Reilly, C., Gogate, V., and Dechter, R. (2016). Probabilistic Inference Modulo Theories. In *IJCAI*, pages 3591–3599.

[Fierens et al., 2015] 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.

[Gario and Micheli, 2015] Gario, M. and Micheli, A. (2015). PySMT: a solver-agnostic library for fast prototyping of SMT-based algorithms. In *Proceedings of the 13th International Workshop on Satisfiability Modulo Theories (SMT)*, pages 373–384.

[Gehr et al., 2016] Gehr, T., Misailovic, S., and Vechev, M. (2016). PSI: Exact Symbolic Inference for Probabilistic Programs. In *CAV*, pages 62–83. Springer.

[Huang and Darwiche, 2005] Huang, J. and Darwiche, A. (2005). DPLL with a Trace: from SAT to Knowledge Compilation. In *IJCAI*, volume 5, pages 156–162.

[Kimmig et al., 2011] Kimmig, A., den Broeck, G. V., and Raedt, L. D. (2011). An Algebraic Prolog for Reasoning about Possible Worlds. In *AAAI*.

[Kimmig et al., 2017] Kimmig, A., Van den Broeck, G., and De Raedt, L. (2017). Algebraic Model Counting. *Journal of Applied Logic*, 22:46–62.

[Kolb et al., 2018] Kolb, S., Mladenov, M., Sanner, S., Belle, V., and Kersting, K. (2018). Efficient Symbolic Integration for Probabilistic Inference. In *IJCAI*, pages 5031–5037.

[Kolb et al., 2019] Kolb, S., Morettin, P., Zuidberg Dos Martires, P., Sommavilla, F., Passerini, A., Sebastiani, R., and De Raedt, L. (2019). The pywmi Framework and Toolbox for Probabilistic Inference using Weighted Model Integration. In *IJCAI*.

[Molina et al., 2018] Molina, A., Vergari, A., Di Mauro, N., Natarajan, S., Esposito, F., and Kersting, K. (2018). Mixed Sum-Product Networks: a Deep Architecture for Hybrid Domains. In *AAAI*.

[Morettin et al., 2017] Morettin, P., Passerini, A., and Sebastiani, R. (2017). Efficient Weighted Model Integration via SMT-Based Predicate Abstraction. In *IJCAI*, pages 720–728.

[Oztok and Darwiche, 2018] Oztok, U. and Darwiche, A. (2018). An Exhaustive DPLL Algorithm for Model Counting. *Journal of Artificial Intelligence Research*, 62:1–32.

[Sanner et al., 2011] Sanner, S., Delgado, K. V., and De Barros, L. N. (2011). Symbolic Dynamic Programming for Discrete and Continuous State MDPs. In *UAI*.

[Valiant, 1979] Valiant, L. G. (1979). The complexity of computing the permanent. *Theoretical Computer Science*, 8(2):189–201.

[Zuidberg Dos Martires et al., 2019] Zuidberg Dos Martires, P., Dries, A., and De Raedt, L. (2019). Exact and Approximate Weighted Model Integration with Probability Density Functions Using Knowledge Compilation. In *AAAI*.

A Solvers Used in the Experiments

In the table below we give an overview of WMI solvers used in our empirical analysis. For the PA [Morettin et al., 2017] and the PRAiSE [De Salvo Braz et al., 2016] solvers we used the original implementation, and we reimplemented the BR [Kolb et al., 2018] and the Symbo (listed as XSDD(PSI)) [Zuidberg Dos Martires et al., 2019] algorithms. We indicate whether the solvers are based on knowledge compilation and whether the integration is performed through symbolic integration. All listed solvers perform exact WMI inference with the exception of XSDD(Sampling), which approximates the integration step with Monte Carlo integration through rejection sampling.

Name	KC	Symbolic	Implementation
PA			Original
PRAiSE		✓	Original
BR	✓	✓	Reimplemented
XSDD(Latte)	✓		New
XSDD(Sampling)	✓		New
XSDD(PSI)	✓	✓	Reimplemented
XSDD(BR)	✓	✓	New
F-XSDD(PSI)	✓	✓	New
F-XSDD(BR)	✓	✓	New

B Extended Running Example

In this section we extend Example 1 in the paper in order to describe in more detail our factorized solving approach F-XSDD. Therefore, recall the SMT formula given from Example 1:

$$\begin{aligned} x>0 \wedge \\ x<1 \wedge \\ (y<1) \vee ((x>y) \wedge \\ y>{}^1\!/\!_2 \end{aligned} \qquad (22)$$

Abstracting the atomic SMT formulas we can compile this formula into the XSDD seen in Figure 4.

We are now able to compute the weighted model integral of the problem given in Example 1 by evaluating the XSDD and integration out the resulting symbolic expression:

$$\int \Big([\![x>0]\!][\![x<1]\!][\![y<1]\!][\![x \geq y]\!] + \\ [\![x>0]\!][\![x<1]\!][\![y>{}^1\!/\!_2]\!][\![x>y]\!] \Big) 2xydxdy$$

To solve this integral efficiently we would like to push the integration inside the evaluation of the XSDD and reuse intermediate integration steps. This process of pushing-in the integration over variables is visualized in Figure 5.

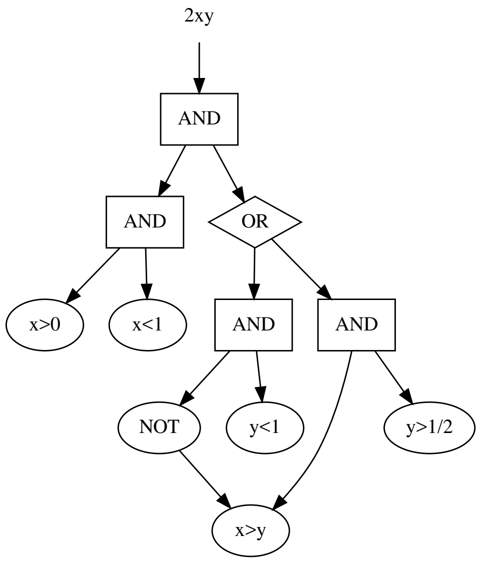

Figure 4: The SMT formula in Equation 22 for weight $2xy$ from Example 1 compiled into an equivalent XSDD.

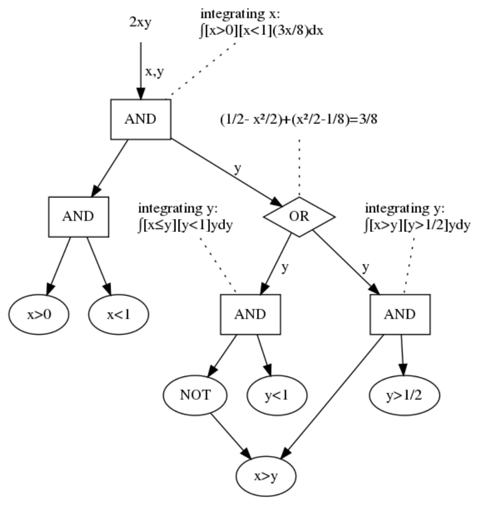

Figure 5: We show how integration variables can be pushed inside the XSDD, to integrate subexpressions separately.

Evaluating the XSDD depicted in Figure 5 results in computing the following integral:

$$2 \int_{(x>0)\wedge(x<1)} \left(\int_{(y<1)\wedge(x\geq y)} ydy + \int_{(y>{}^1\!/\!_2)\wedge(x>y)} ydy \right) xdx$$

We see that the atomic formulas in the SMT($\mathcal{LRA}$) formula in Equation 3 become the integration bounds over which to integrate the weight function.