Enhancing SMT-based Weighted Model Integration by Structure Awareness

Giuseppe Spallitta[a], Gabriele Masina[a], Paolo Morettin[b], Andrea Passerini[a], Roberto Sebastiani[a]

[a] *University of Trento*
[b] *KU Leuven*

Abstract

The development of efficient exact and approximate algorithms for probabilistic inference is a long-standing goal of artificial intelligence research. Whereas substantial progress has been made in dealing with purely discrete or purely continuous domains, adapting the developed solutions to tackle hybrid domains, characterised by discrete and continuous variables and their relationships, is highly non-trivial. Weighted Model Integration (WMI) recently emerged as a unifying formalism for probabilistic inference in hybrid domains. Despite a considerable amount of recent work, allowing WMI algorithms to scale with the complexity of the hybrid problem is still a challenge. In this paper we highlight some substantial limitations of existing state-of-the-art solutions, and develop an algorithm that combines SMT-based enumeration, an efficient technique in formal verification, with an effective encoding of the problem structure. This allows our algorithm to avoid generating redundant models, resulting in drastic computational savings. Additionally, we show how SMT-based approaches can seamlessly deal with different integration techniques, both exact and approximate, significantly expanding the set of problems that can be tackled by WMI technology. An extensive experimental evaluation on both synthetic and real-world datasets confirms the substantial advantage of the proposed solution over existing alternatives. The application potential of this technology is further showcased on a prototypical task aimed at verifying the fairness of probabilistic programs.

Keywords: Hybrid Probabilistic Inference, Weighted Model Integration,

*A preliminary version of this work was presented at UAI 2022 [1].

1. Introduction

There is a growing interest in the artificial intelligence community in extending probabilistic reasoning approaches to deal with hybrid domains, characterized by both continuous and discrete variables and their relationships. Indeed, hybrid domains are extremely common in real-world scenarios, from transport modelling [2] to probabilistic robotics [3] and cyber-physical systems [4].

Weighted Model Integration (WMI) [5, 6, 7] recently emerged as a unifying formalism for probabilistic inference in hybrid domains. The paradigm extends Weighted Model Counting (WMC) [8], which is the task of computing the weighted sum of the set of satisfying assignments of a propositional formula, to deal with SMT formulas (e.g., [9]) consisting of combinations of Boolean atoms and connectives with symbols from a background theory, like linear arithmetic over rationals ($\mathcal{LRA}$).

Although WMC can be made extremely efficient by leveraging component caching techniques [10, 11], these strategies are difficult to apply for WMI due to the tight coupling induced by arithmetic constraints. Indeed, existing component caching approaches for WMI are restricted to fully factorized densities with few dependencies among continuous variables [12]. Another direction specifically targets acyclic [13, 14] or loopy [15] pairwise models. Approximations with guarantees can be computed for problems in disjunctive normal form [16] or, in restricted cases, conjunctive normal form [17].

Exact solutions for more general classes of densities and constraints mainly take advantage of advances in SMT technology [9] or knowledge compilation (KC) [18]. WMI-PA [6, 19] relies on SMT-based Predicate Abstraction (PA) [20] to both reduce the number of models to be generated and integrated over and efficiently enumerate them, and was shown to achieve substantial improvements over previous solutions. Nevertheless, in this paper we show how WMI-PA has the major drawback of ignoring the conditional structure of the weight function, which prevents from pruning away lots of redundant models. The use of KC for hybrid probabilistic inference was pioneered by [21] and further refined in a series of later works [22, 23, 24, 25]. By compiling a formula into an algebraic circuit, KC techniques can exploit the structure of the problem to reduce the size of the resulting circuit, and are at the core of many state-of-the-art approaches for WMC [8]. Nevertheless, in this paper we show that even the most recent KC solutions for WMI [23, 24] have serious troubles in dealing with densely coupled problems, resulting in exponentially large circuits that are impossible to store or prohibitively expensive to evaluate.

To address these problems, in this paper we introduce SA-WMI-PA-SK, a novel algorithm for WMI that aims to combine the best of both worlds, by introducing weight-structure awareness into PA-based WMI. The main idea is to build a formula, which we call the *conditional skeleton*, which mimics the conditional structure of the weight function, to drive the SMT-based enumeration algorithm preventing it from generating redundant models. An extensive experimental evaluation on synthetic and real-world datasets shows substantial computational advantages of the proposed solution over existing alternatives for the most challenging settings.

Parallel to the introduction of weight-structure awareness, we highlight how PA-based algorithms are *agnostic* to the integration algorithm chosen to compute the volume of each enumerated assignment. To this extent, we extend SA-WMI-PA-SK to support both *exact numerical integration* (the one implicitly embedded in [6, 19]) and *approximate integration*. The advantages of using approximate integration are twofold: (i) it positively affects scalability of SA-WMI-PA-SK with complex instances when integration is the major bottleneck; (ii) it allows applying SA-WMI-PA-SK to problems with non-polynomial weight functions (e.g., Gaussian densities), increasing the applicability of WMI to real-world scenarios.

Our main contributions can be summarized as follows:

- We analyse existing state-of-the-art of WMI and we identify major efficiency issues for both PA and KC based approaches.
- We introduce SA-WMI-PA-SK, a novel WMI algorithm that combines PA with weight-structure awareness by generating explicitly a *conditional skeleton* of the weight function to drive the search of assignments.
- We show how SA-WMI-PA-SK achieves substantial improvements over existing solutions in both synthetic and real-world settings.
- We demonstrate how PA-based approaches can deal with different integration techniques, both exact and approximate, substantially expanding the set of problems that can be tackled by WMI technology.

- We present a prototypical application of SA-WMI-PA-SK to verifying the fairness of probabilistic programs.

The rest of the paper is organized as follows. We start by presenting the related work in §2. In §3 we introduce the background, focusing on the formulation of Weighted Model Integration. In §4 we analyse the WMI approaches based on Knowledge-Compilation and on Predicate Abstraction, identifying some weaknesses for either of them. In §5 we address the gaps mentioned in the previous section, showing theoretical ideas to make WMI-PA structure aware and their implementation into our novel algorithm, SA-WMI-PA-SK. §6 is devoted to an empirical evaluation of SA-WMI-PA-SK with respect to the other existing approaches, considering both synthetic and real-world settings. In §7, we highlight how PA-based approaches are agnostic to the integrator chosen to compute the volume of the polytopes and propose an adaptation of the algorithm to deal with approximated integration. Our conclusion and final considerations are presented in §8.

2. Related work

Traditionally, inference algorithms in hybrid graphical models either disallowed complex algebraic and logical constraints [26, 27, 28], or solved the problem approximately [29, 30]. The use of piecewise polynomial densities when dealing with algebraic constraints was pioneered by Sanner and Abbasnejad [21]. The algorithm, called Symbolic Variable Elimination (SVE), first used a compact representation called eXtended Algebraic Decision Diagrams (XADD) [31] in probabilistic inference tasks. As described in detail in §4, these representations lie at the core of modern compilation-based WMI techniques. Probabilistic inference modulo theories [32] used similar representations together with a modified DPLL procedure. The resulting algorithm, called PRAiSE, showed superior performance with respect to both SVE and the original WMI [5] solver, which didn't exploit any SMT enumeration technique. Follow-up works on SMT-based WMI capitalized on the advanced features of SMT solvers, obtaining state-of-the-art results in many benchmarks [6, 19].

The substantial efficiency gains obtained by caching partial results in WMC [10, 11] motivated their applications in the hybrid case. When the dependencies induced by the algebraic constraints are limited, similar ideas can be applied to WMI [12]. Alternatively to SMT- or compilation-based approaches, WMI can be reduced to Belief Propagation when the algebraic dependencies involve at most two variables. Then, an efficient message passing algorithm on an acyclic pairwise factor graph can solve the problem exactly [14]. This idea was later extended for approximating problems with circular dependencies [15]. In contrast with the work above, this paper focuses on the general WMI problem, where the above assumptions cannot be leveraged.

Approximate WMI algorithms exploit both SMT-based [17] and knowledge compilation-based [23] approaches. While generally applicable, the latter does not provide any guarantee on the approximation error. The former provides practical statistical guarantees in limited scenarios when constraints are expressed in conjunctive form. Instead, if the constraints can be expressed in disjunctive form, WMI admits a fully polynomial randomized approximation scheme (FPRAS) under mild assumptions [16].

3. Background

In this section, we introduce the theoretical background which is needed for a fully-detailed comprehension of this paper. Notice that this section contains several SMT-related technicalities which are needed only to understand some technicalities in the encodings, and as such could be skipped in a first reading.

3.1. SMT, AllSMT and Projected AllSMT

We assume the reader is familiar with the basic syntax, semantics, and results of propositional and first-order logic. We adopt the notation and definitions in [19] —including some terminology and concepts from Satisfiability Modulo Theories (SMT)— which we summarize below.

Notation and terminology. Satisfiability Modulo Theories (SMT) (see [9] for details) consists in deciding the satisfiability of first-order formulas over some given theory. For the context of this paper, we will refer to quantifier-free SMT formulas over linear real arithmetic ($\mathcal{LRA}$), possibly combined with uninterpreted function symbols ($\mathcal{LRA} \cup \mathcal{EUF}$). We use $\mathbb{B} \overset{\text{def}}{=} \{\top, \bot\}$ to indicate the set of Boolean values and $\mathbb{R}$ to indicate the set of real values. We use the sets $\mathbf{A} \overset{\text{def}}{=} \{A_i\}_i$, $\mathbf{B} \overset{\text{def}}{=} \{B_i\}_i$ to denote Boolean atoms and the sets $\mathbf{x} \overset{\text{def}}{=} \{x_i\}_i$, $\mathbf{y} \overset{\text{def}}{=} \{y_i\}_i$ to denote real variables. $\mathit{Atoms}(\varphi)$ denotes the set of atomic formulas occurring in φ, both Boolean and $\mathcal{LRA} \cup \mathcal{EUF}$ ones. SMT($\mathcal{LRA}$) formulas combines Boolean atoms $A_i \in \mathbb{B}$ and $\mathcal{LRA}$ atoms in the form $(\sum_i c_i x_i \bowtie c)$ (where c_i, c are rational values, x_i are real variables in $\mathbb{R}$

and $\bowtie$ is one of the standard arithmetic operators $\{=, \neq, <, >, \leq, \geq\}$) by using standard Boolean operators $\{\neg, \wedge, \vee, \rightarrow, \leftrightarrow\}$. In SMT($\mathcal{LRA} \cup \mathcal{EUF}$), $\mathcal{LRA}$ terms can be interleaved with uninterpreted function symbols. Hereafter, unless otherwise stated, we implicitly refer to a background theory $\mathcal{T} \in \{\mathcal{LRA}, \mathcal{LRA} \cup \mathcal{EUF}\}$.

A *literal* is either an atom (a *positive literal*) or its negation (a *negative literal*). A *clause* $\bigvee_{j=1}^{K} l_j$ is a disjunction of literals. A formula is in *Conjunctive Normal Form (CNF)* if it is written as a conjunction of clauses $\bigwedge_{i=1}^{L} \bigvee_{j_i=1}^{K_i} l_{j_i}$. Some shortcuts of frequently-used expressions (marked as "$[\![\ldots]\!]$") are provided to simplify the reading: the formula $(x_i \geq l) \wedge (x_i \leq u)$ is shortened into $[\![x_i \in [l, u]]\!]$; if $\phi \overset{\text{def}}{=} \bigwedge_i C_i$ is a CNF formula and $C \overset{\text{def}}{=} \bigvee_j l_j$ is a clause, then the formula $\bigwedge_i (C \vee C_i)$ is shortened into $[\![C \vee \phi]\!]$.

Semantics. "$\models_{\mathcal{T}}$" denotes entailment in $\mathcal{T}$ (e.g.,$(x \geq 2) \models_{\mathcal{LRA}} (x \geq 1)$), whereas "$\models_{\mathbb{B}}$" denotes tautological entailment (e.g. $A_1 \wedge (x \geq 2) \models_{\mathbb{B}} ((A_1 \vee (x \leq 1)) \wedge (\neg A_1 \vee (x \geq 2))$.) Note that $\models_{\mathbb{B}}$ is strictly stronger than $\models_{\mathcal{T}}$, that is, if $\varphi_1 \models_{\mathbb{B}} \varphi_2$ then $\varphi_1 \models_{\mathcal{T}} \varphi_2$, but not vice versa. We say that φ_1, φ_2 are $\mathcal{T}$-equivalent, written $\varphi_1 \Leftrightarrow_{\mathcal{T}} \varphi_2$, iff both $\varphi_1 \models_{\mathcal{T}} \varphi_2$ and $\varphi_2 \models_{\mathcal{T}} \varphi_1$, and *tautologically equivalent*, written $\varphi_1 \Leftrightarrow_{\mathbb{B}} \varphi_2$, iff both $\varphi_1 \models_{\mathbb{B}} \varphi_2$ and $\varphi_2 \models_{\mathbb{B}} \varphi_1$. Given a set of $\mathcal{T}$ formulas $\mathbf{\Psi} \overset{\text{def}}{=} \{\psi_1, \ldots, \psi_K\}$, we call a *total [resp. partial] truth assignment* μ for $\mathbf{\Psi}$ any total [resp. partial] map from $\mathbf{\Psi}$ to $\mathbb{B}$. With a little abuse of notation, we represent μ either as a set or a conjunction of literals: $\mu \overset{\text{def}}{=} \{\psi \mid \psi \in \mathbf{\Psi},\ \mu(\psi) = \top\} \cup \{\neg\psi \mid \psi \in \mathbf{\Psi},\ \mu(\psi) = \bot\}$, or $\mu \overset{\text{def}}{=} \bigwedge_{\psi \in \mathbf{\Psi}, \mu(\psi)=\top} \quad \psi \wedge \bigwedge_{\psi \in \mathbf{\Psi}, \mu(\psi)=\bot} \neg\psi$, and we write "$\psi_i \in \mu_1$" and "$\mu_1 \subseteq \mu_2$" as if μ_1, μ_2 were represented as sets (i.e., we write "$\psi_1 \wedge \neg\psi_2 \subseteq \psi_1 \wedge \neg\psi_2 \wedge \psi_3$" meaning "$\{\psi_1, \neg\psi_2\} \subseteq \{\psi_1, \neg\psi_2, \psi_3\}$"). In the latter case, we say that μ_1 is a *sub-assignment* of μ_2, and that μ_2 is a *super-assignment* of μ_1. We denote by $\mathbb{B}^K$ the set of all total truth assignments over $\mathbf{\Psi}$. Given a (partial) truth assignment μ to $\mathit{Atoms}(\varphi)$, we call the *residual* of φ w.r.t. μ, written "$\varphi_{[\mu]}$", the formula obtained from φ by substituting all the atoms assigned in μ with their respective truth value, and by recursively propagating truth values through Boolean operators in the usual way. [1] (Notice that if $\varphi_{[\mu]} = \top$, then $\mu \models_{\mathbb{B}} \varphi$, but not vice versa, see [33, 34] for details.)

Let $\mathbf{x} \overset{\text{def}}{=} \{x_1, \ldots, x_N\} \in \mathbb{R}^N$ and $\mathbf{A} \overset{\text{def}}{=} \{A_1, \ldots, A_M\} \in \mathbb{B}^M$ for some N and M. Consider a generic $\mathcal{T}$ formula φ on (subsets of)$\mathbf{x}$ and $\mathbf{A}$, and let

[1]E.g., $\neg\top \Longrightarrow \bot$, $\varphi \wedge \top \Longrightarrow \varphi$, $\varphi \vee \top \Longrightarrow \top$, $\varphi \leftrightarrow \top \Longrightarrow \varphi$, $\varphi \leftrightarrow \bot \Longrightarrow \neg\varphi$, ...

$\mathbf{\Psi} \stackrel{\text{def}}{=} \mathit{Atoms}(\varphi)$. Given a truth assignment μ for $\mathit{Atoms}(\varphi)$, we denote by $\mu^{\mathbf{A}}$ and $\mu^{\mathcal{T}}$ its two components on the Boolean atoms in $\mathbf{A}$ and on the $\mathcal{T}$ atoms, respectively, so that $\mu \stackrel{\text{def}}{=} \mu^{\mathbf{A}} \wedge \mu^{\mathcal{T}}$. (For example, if $\mu \stackrel{\text{def}}{=} A_1 \wedge \neg A_2 \wedge (x \geq 1) \wedge \neg(x \geq 3)$, then $\mu^{\mathbf{A}} \stackrel{\text{def}}{=} A_1 \wedge \neg A_2$ and $\mu^{\mathcal{LRA}} \stackrel{\text{def}}{=} (x \geq 1) \wedge \neg(x \geq 3)$). Importantly, and unlike pure propositional logic, μ can be $\mathcal{T}$-unsatisfiable due to its $\mu^{\mathcal{T}}$ component (e.g., $\mu \stackrel{\text{def}}{=} \neg A_1 \wedge (x_1 + x_2 = 3) \wedge \neg(x_1 + x_2 \geq 2)$). A (partial) truth assignment μ *propositionally satisfies* φ iff $\mu \models_{\mathbb{B}} \varphi$. Thus, SMT($\mathcal{T}$) reduces to checking the existence of a $\mathcal{T}$-satisfiable assignment μ s.t. $\mu \models_{\mathbb{B}} \varphi$. (Hereafter, we use "$\models$" rather than "$\models_{\mathbb{B}}$" and "$\models_{\mathcal{T}}$" when the distinction is clear from the context.) Given $\varphi(\mathbf{x}, \mathbf{A} \cup \mathbf{B})$, we say that a Boolean (partial) assignment $\mu^{\mathbf{A}}$ satisfies $\exists \mathbf{x}.\exists \mathbf{B}.\varphi$ (written "$\mu^{\mathbf{A}} \models \exists \mathbf{x}.\exists \mathbf{B}.\varphi$") iff $\mu^{\mathbf{A}} \wedge \mu^{\mathbf{B}} \wedge \mu^{\mathcal{T}} \models_{\mathbb{B}} \varphi$ for some assignment $\mu^{\mathbf{B}}$ on $\mathbf{B}$ and some $\mathcal{T}$-satisfiable assignment $\mu^{\mathcal{T}}$ on the $\mathcal{T}$-atoms of φ. Notice that, if $\mu^{\mathbf{A}}$ is partial, then $\mu^{\mathbf{A}} \models \exists \mathbf{x}.\exists \mathbf{B}.\varphi(\mathbf{x}, \mathbf{A} \cup \mathbf{B})$ iff $\eta_{\mathbf{A}} \models \exists \mathbf{x}.\exists \mathbf{B}.\varphi(\mathbf{x}, \mathbf{A} \cup \mathbf{B})$ for every total assignment $\eta_{\mathbf{A}}$ extending $\mu^{\mathbf{A}}$. (The definitions of "$\mu^{\mathbf{A}} \models \exists \mathbf{x}.\varphi(\mathbf{x}, \mathbf{A})$" and of "$\mu^{\mathbf{A}} \models \exists \mathbf{B}.\varphi(\mathbf{A} \cup \mathbf{B})$" follow with $\mathbf{B} = \emptyset, \mu^{\mathbf{B}} = \top$ and $\mathbf{x} = \emptyset, \mu^{\mathcal{T}} = \top$ respectively.)

CNF-ization. $\varphi(\mathbf{x}, \mathbf{A})$ can be converted into CNF as follows: implications and bi-implications are rewritten by applying the rewrite rules $(\alpha \to \beta) \Longrightarrow (\neg\alpha \vee \beta)$ and $(\alpha \leftrightarrow \beta) \Longrightarrow (\neg\alpha \vee \beta) \wedge (\neg\beta \vee \alpha)$; negations are pushed down to the literal level by recursively applying the rewrite rules $\neg(\alpha \wedge \beta) \Longrightarrow (\neg\alpha \vee \neg\beta)$, $\neg(\alpha \vee \beta) \Longrightarrow (\neg\alpha \wedge \neg\beta)$, and $\neg\neg\alpha \Longrightarrow \alpha$. Then we have a few alternatives:
Classic CNF-ization ("$\mathsf{CNF}(\varphi)$") consists in applying recursively the DeMorgan rewrite rule $\alpha \vee (\beta \wedge \gamma) \Longrightarrow (\alpha \vee \gamma) \wedge (\beta \vee \gamma)$ until the result is in CNF. The resulting formula $\phi(\mathbf{x}, \mathbf{A})$ is tautologically equivalent to φ, but its size can be exponential w.r.t. that of φ;
Tseitin CNF-ization [35] ("$\mathsf{CNF}_{\mathsf{ts}}(\varphi)$") consists in applying recursively the "labeling" rewrite rule $\varphi \Longrightarrow \varphi[\psi|B] \wedge (B \leftrightarrow \psi)$ —$\varphi[\psi|B]$ being the results of substituting all occurrences of a subformula ψ with a fresh Boolean atom B— until all conjuncts can be CNF-ized classically without space blow-up. Alternatively, one can apply the rule $\varphi \Longrightarrow \varphi[\psi|B] \wedge (B \to \psi)$ [36] ("$\mathsf{CNF}_{\mathsf{pg}}(\varphi)$"). With both cases, the resulting formula $\phi(\mathbf{x}, \mathbf{A} \cup \mathbf{B})$ is s.t. $\varphi(\mathbf{x}, \mathbf{A})$ is tautologically equivalent to $\exists \mathbf{B}.\phi(\mathbf{x}, \mathbf{A} \cup \mathbf{B})$[2], $\mathbf{B}$ being the set of fresh atoms introduced, and the size of ϕ is linear w.r.t. that of φ.

[2]that is, $\mathcal{M} \models \varphi$ iff there exists a total truth assignment η on $\mathbf{B}$ s.t. $\mathcal{M} \cup \eta \models \phi$.

Assignment Enumeration. Given a theory $\mathcal{T} \in \{\mathcal{LRA}, \mathcal{LRA} \cup \mathcal{EUF}\}$, $\mathcal{TTA}(\varphi) \stackrel{\text{def}}{=} \{\mu_1, \ldots, \mu_j\}$ denotes the set of $\mathcal{T}$-satisfiable *total* assignments over both propositional and $\mathcal{T}$-atoms that propositionally satisfy φ; $\mathcal{TA}(\varphi) \stackrel{\text{def}}{=} \{\mu_1, \ldots, \mu_j\}$ represents one set of $\mathcal{T}$-satisfiable *partial* assignments over both propositional and $\mathcal{T}$ atoms that propositionally satisfy φ, s.t. (i) every total assignment in $\mathcal{TTA}(\varphi)$ is a super-assignment of some partial ones in $\mathcal{TA}(\varphi)$ and (ii) every pair $\mu_i, \mu_j \in \mathcal{TA}(\varphi)$ assigns opposite truth value to at least one atom. We remark that $\mathcal{TTA}(\varphi)$ is unique (modulo reordering), whereas multiple $\mathcal{TA}(\varphi)$s are admissible for the same formula φ (including $\mathcal{TTA}(\varphi)$). The disjunction of the truth assignments in $\mathcal{TTA}(\varphi)$, and that of $\mathcal{TA}(\varphi)$, are $\mathcal{T}$-equivalent to φ. Thus, given $\varphi(\mathbf{x}, \mathbf{A} \cup \mathbf{B})$, $\mathcal{TTA}(\exists\mathbf{x}.\exists\mathbf{B}.\varphi)$ denotes the set of all total truth assignment $\mu^{\mathbf{A}}$ on $\mathbf{A}$ s.t. $\mu^{\mathbf{A}} \models \exists\mathbf{x}.\exists\mathbf{B}.\varphi$, and $\mathcal{TA}(\exists\mathbf{x}.\exists\mathbf{B}.\varphi)$ denotes one set of partial assignments $\mu^{\mathbf{A}}$ on $\mathbf{A}$ s.t. $\mu^{\mathbf{A}} \models \exists\mathbf{x}.\exists\mathbf{B}.\varphi$ complying with conditions (i) and (ii) above where φ is replaced by $\exists\mathbf{x}.\exists\mathbf{B}.\varphi$. The disjunction of the assignments in $\mathcal{TTA}(\exists\mathbf{x}.\exists\mathbf{B}.\varphi)$ and that of $\mathcal{TA}(\exists\mathbf{x}.\exists\mathbf{B}.\varphi)$ are $\mathcal{T}$-equivalent to $\exists\mathbf{x}.\exists\mathbf{B}.\varphi$. (As above, the definitions of $\mathcal{TTA}(\exists\mathbf{x}.\varphi(\mathbf{x}, \mathbf{A}))/\mathcal{TA}(\exists\mathbf{x}.\varphi(\mathbf{x}, \mathbf{A}))$ and of $\mathcal{TTA}(\exists\mathbf{B}.\varphi(\mathbf{A} \cup \mathbf{B}))/\mathcal{TA}(\exists\mathbf{B}.\varphi(\mathbf{A} \cup \mathbf{B}))$ follow with $\mathbf{B} = \emptyset$ and $\mathbf{x} = \emptyset$ respectively.) Notice that, if $\varphi(\mathbf{x}, \mathbf{A} \cup \mathbf{B})$ is the result of applying one of the "labelling" CNF-izations [35, 36] to $\phi(\mathbf{x}, \mathbf{A})$, then $\mathcal{TTA}(\phi)$, $\mathcal{TA}(\phi)$, $\mathcal{TTA}(\exists\mathbf{x}.\phi)$ and $\mathcal{TA}(\exists\mathbf{x}.\phi)$ can be computed as $\mathcal{TTA}(\exists\mathbf{B}.\varphi)$, $\mathcal{TA}(\exists\mathbf{B}.\varphi)$, $\mathcal{TTA}(\exists\mathbf{x}.\exists\mathbf{B}.\varphi)$ and $\mathcal{TA}(\exists\mathbf{x}.\exists\mathbf{B}.\varphi)$ respectively.

$\mathcal{TTA}(\ldots)/\mathcal{TA}(\ldots)$ can be computed efficiently by means of *Projected AllSMT*, a technique used in formal verification to compute *Predicate Abstraction* [20]. All these functionalities are provided by the SMT solver MathSAT [37]. In a nutshell, MathSAT works as follows. Given φ and a subset of "relevant" atoms $\mathbf{\Psi} \subseteq \mathit{Atoms}(\varphi)$, $\mathcal{TA}(\ldots)$ generates one-by-one $\mathcal{T}$-satisfiable partial assignments $\mu_i \stackrel{\text{def}}{=} \mu_i^{\overline{\mathbf{\Psi}}} \wedge \mu_i^{\mathbf{\Psi}}$, s.t. $\mu_i^{\overline{\mathbf{\Psi}}}$ is a total assignment on $\mathit{Atoms}(\varphi) \backslash \mathbf{\Psi}$ and $\mu_i^{\mathbf{\Psi}}$ is a *minimal* [3] partial assignment on $\mathbf{\Psi}$ s.t. (i) $\varphi_{[\mu_i]} = \top$, and (ii) for every $j \in \{1, \ldots, i-1\}$, $\mu_j^{\mathbf{\Psi}}, \mu_i^{\mathbf{\Psi}}$ assign opposite truth values to at least one atom in $\mathbf{\Psi}$. Finally, the set $\{\mu_i^{\mathbf{\Psi}}\}_i$ is returned. (We say that the enumeration is *projected over* $\mathbf{\Psi}$.) Thus, $\mathcal{TA}(\varphi(\mathbf{x}, \mathbf{A} \cup \mathbf{B}))$ and $\mathcal{TA}(\exists\mathbf{x}.\exists\mathbf{B}.\varphi(\mathbf{x}, \mathbf{A} \cup \mathbf{B}))$, possibly with $\mathbf{x} = \emptyset$ and/or $\mathbf{B} = \emptyset$, are computed by setting $\mathbf{\Psi} \stackrel{\text{def}}{=} \mathit{Atoms}(\varphi)$ and $\mathbf{\Psi} \stackrel{\text{def}}{=} \mathbf{A}$ respectively. $\mathcal{TTA}(\ldots)$ works in the same way, but forcing the μ_is to be total.

[3]i.e., no literal can be further dropped from $\mu_i^{\mathbf{\Psi}}$ without losing properties (i) and (ii).

3.2. Weighted Model Integration (WMI)

Let $\mathbf{x} \stackrel{\text{def}}{=} \{x_1, \ldots, x_N\} \in \mathbb{R}^N$ and $\mathbf{A} \stackrel{\text{def}}{=} \{A_1, \ldots, A_M\} \in \mathbb{B}^M$ for some integers N and M, and let $w(\mathbf{x}, \mathbf{A})$ denote a non-negative weight function s.t. $w(\mathbf{x}, \mathbf{A}) : \mathbb{R}^N \times \mathbb{B}^M \longmapsto \mathbb{R}^+$. Intuitively, w encodes a (possibly unnormalized) density function over $\mathbf{A} \cup \mathbf{x}$. Given a total assignment $\mu^{\mathbf{A}}$ over $\mathbf{A}$, $w_{[\mu^{\mathbf{A}}]}(\mathbf{x}) \stackrel{\text{def}}{=} w(\mathbf{x}, \mu^{\mathbf{A}})$ is w restricted to the truth values of $\mu^{\mathbf{A}}$.

The **Weighted Model Integral** of $w(\mathbf{x},\mathbf{A})$ over $\varphi(\mathbf{x}, \mathbf{A})$ is defined as [19]:

$$\mathsf{WMI}(\varphi, w|\mathbf{x}, \mathbf{A}) \stackrel{\text{def}}{=} \sum_{\mu^{\mathbf{A}} \in \mathcal{TTA}(\exists \mathbf{x}.\varphi)} \mathsf{WMI}_{\mathsf{nb}}(\varphi_{[\mu^{\mathbf{A}}]}, w_{[\mu^{\mathbf{A}}]}|\mathbf{x}) \qquad (1)$$

$$\mathsf{WMI}_{\mathsf{nb}}(\varphi, w|\mathbf{x}) \stackrel{\text{def}}{=} \int_{\varphi(\mathbf{x})} w(\mathbf{x}) \, \mathrm{d}\mathbf{x}, \qquad (2)$$

$$= \sum_{\mu^{\mathcal{LRA}} \in \mathcal{TA}(\varphi)} \int_{\mu^{\mathcal{LRA}}} w(\mathbf{x}) \, \mathrm{d}\mathbf{x}, \qquad (3)$$

where the $\mu^{\mathbf{A}}$s are total truth assignments on $\mathbf{A}$, $\mathsf{WMI}_{\mathsf{nb}}(\varphi, w|\mathbf{x})$ is the integral of $w(\mathbf{x})$ over the set $\{\mathbf{x} \mid \varphi(\mathbf{x})$ *is true*$\}$ ("$_{\mathsf{nb}}$" means "no-Booleans").

We call a *support* of a weight function $w(\mathbf{x}, \mathbf{A})$ any subset of $\mathbb{R}^N \times \mathbb{B}^M$ out of which $w(\mathbf{x}, \mathbf{A}) = 0$ and we represent it as a $\mathcal{LRA}$-formula $\chi(\mathbf{x}, \mathbf{A})$. We recall that, consequently, $\mathsf{WMI}(\varphi \wedge \chi, w|\mathbf{x}, \mathbf{A}) = \mathsf{WMI}(\varphi, w|\mathbf{x}, \mathbf{A})$.

We consider the class of *feasibly integrable on $\mathcal{LRA}$ ($\mathsf{FI}^{\mathcal{LRA}}$)* functions $w(\mathbf{x})$, which contain no conditional component and for which there exists some procedure able to compute $\mathsf{WMI}_{\mathsf{nb}}(\mu^{\mathcal{LRA}}, w|\mathbf{x})$ for every set of $\mathcal{LRA}$ literals on $\mathbf{x}$. (E.g., polynomials are $\mathsf{FI}^{\mathcal{LRA}}$.) Then we call a weight function $w(\mathbf{x}, \mathbf{A})$, *feasibly integrable under $\mathcal{LRA}$ conditions ($\mathsf{FIUC}^{\mathcal{LRA}}$)* iff it can be described in terms of a support $\mathcal{LRA}$-formula $\chi(\mathbf{x}, \mathbf{A})$ ($\top$ if not provided) and a set $\mathbf{\Psi} \stackrel{\text{def}}{=} \{\psi_i(\mathbf{x}, \mathbf{A})\}_{i=1}^{K}$ of $\mathcal{LRA}$ *conditions*, in such a way that, for every total truth assignment $\mu^{\mathbf{\Psi}}$ to $\mathbf{\Psi}$, $w_{[\mu^{\mathbf{\Psi}}]}(\mathbf{x})$ is total and $\mathsf{FI}^{\mathcal{LRA}}$ in the domain given by the values of $\langle \mathbf{x}, \mathbf{A} \rangle$ which satisfy $(\chi \wedge \mu^{\mathbf{\Psi}})$.[4] Intuitively, each $\mu^{\mathbf{\Psi}}$ describes a portion of the domain of w inside which $w_{[\mu^{\mathbf{\Psi}}]}(\mathbf{x})$ is $\mathsf{FI}^{\mathcal{LRA}}$, and we say that $\mu^{\mathbf{\Psi}}$ *identifies* $w_{[\mu^{\mathbf{\Psi}}]}$ in w. In practice, we assume w.l.o.g. that $\mathsf{FIUC}^{\mathcal{LRA}}$ functions are described as combinations of constants, variables, standard arithmetic operators $+, -, \cdot$, condition-less real-valued functions (e.g., $\exp(.), \ldots$), conditional expressions in the form ($\mathsf{If}\ \psi_i\ \mathsf{Then}\ t_{1i}\ \mathsf{Else}\ t_{2i}$) whose conditions ψ_i are $\mathcal{LRA}$ formulas and terms t_{1i}, t_{2i} are $\mathsf{FIUC}^{\mathcal{LRA}}$.

[4]Notice that the conditions in $\mathbf{\Psi}$ can be non-atomic, and can contain atoms in $\mathbf{A}$.

3.3. WMI via Projected AllSMT

WMI-PA is an efficient WMI algorithm presented in [6, 19] which exploits SMT-based predicate abstraction. Let $w(\mathbf{x}, \mathbf{A})$ be a $\mathsf{FIUC}^{\mathcal{LRA}}$ function as above. WMI-PA is based on the fact that

$$\mathsf{WMI}(\varphi, w|\mathbf{x}, \mathbf{A}) = \sum_{\mu^{\mathbf{A}^*} \in \mathcal{TTA}(\exists \mathbf{x}.\varphi^*)} \mathsf{WMI}_{\mathsf{nb}}(\varphi^*_{[\mu^{\mathbf{A}^*}]}, w^*_{[\mu^{\mathbf{A}^*}]}|\mathbf{x}) \quad (4)$$

$$\varphi^* \stackrel{\text{def}}{=} \varphi \wedge \chi \wedge \bigwedge_{k=1}^{K} (B_k \leftrightarrow \psi_k) \quad (5)$$

s.t. $\mathbf{A}^* \stackrel{\text{def}}{=} \mathbf{A} \cup \mathbf{B}$ s.t. $\mathbf{B} \stackrel{\text{def}}{=} \{B_1, \ldots, B_K\}$ are fresh propositional atoms and $w^*(\mathbf{x}, \mathbf{A} \cup \mathbf{B})$ is the weight function obtained by substituting in $w(\mathbf{x}, \mathbf{A})$ each condition ψ_k in $\mathbf{\Psi}$ with B_k.[5]

The pseudocode of WMI-PA is reported in Algorithm 1. First, the problem is transformed (if needed) by labelling all conditions in $\mathbf{\Psi}$ occurring in $w(\mathbf{x}, \mathbf{A})$ with fresh Boolean atoms $\mathbf{B}$, as above. After this preprocessing stage, the set $\mathcal{M}^{\mathbf{A}^*} \stackrel{\text{def}}{=} \mathcal{TTA}(\exists \mathbf{x}.\varphi^*)$ is computed by projected AllSMT [20]. Then, the algorithm iterates over each Boolean assignment $\mu^{\mathbf{A}^*}$ in $\mathcal{M}^{\mathbf{A}^*}$. $\varphi^*_{[\mu^{\mathbf{A}^*}]}$ is simplified by the $\mathsf{Simplify}$ procedure, which propagates truth values and applies logical and arithmetical simplifications. Then, if $\varphi^*_{[\mu^{\mathbf{A}^*}]}$ is already a conjunction of literals, the algorithm directly computes its contribution to the volume by calling $\mathsf{WMI}_{\mathsf{nb}}(\varphi^*_{[\mu^{\mathbf{A}^*}]}, w^*_{[\mu^{\mathbf{A}^*}]}|\mathbf{x})$. Otherwise, $\mathcal{TA}(\varphi^*_{[\mu^{\mathbf{A}^*}]})$ is computed by AllSMT to produce partial assignments, and the algorithm iteratively computes contributions to the volume for each $\mu^{\mathcal{LRA}}$. See [19] for more details.

Remark 1. Importantly, in the implementation of WMI-PA the potentially large sets $\mathcal{M}^{\mathbf{A}^*}$ and $\mathcal{M}^{\mathcal{LRA}}$ are not generated explicitly; rather, their elements are generated, integrated, and then dropped one-by-one, so as to avoid space blow-up. For example, steps like "$\mathcal{M}^{\mathbf{A}^*} \leftarrow \mathcal{TTA}(\ldots)$; **for** $\mu^{\mathbf{A}^*} \in \mathcal{M}^{\mathbf{A}^*}\ldots$" in Algorithm 1 should be read as "**for** $\mu^{\mathbf{A}^*} \in \mathcal{TTA}(\ldots)\ldots$". (We keep the first representation for readability.) The same applies to all our novel algorithms in this paper.

[5]except for conditions in $\mathbf{\Psi}$ which are already Boolean literals on $\mathbf{A}$.

Algorithm 1 WMI-PA(φ, w , $\mathbf{x}$, $\mathbf{A}$)

```
1:  $\langle \varphi^*, w^*, \mathbf{A}^* \rangle \leftarrow \mathsf{LabelConditions}(\varphi, w, \mathbf{x}, \mathbf{A})$
2:  $\mathcal{M}^{\mathbf{A}^*} \leftarrow \mathcal{TTA}(\exists \mathbf{x}.\varphi^*)$
3:  $vol \leftarrow 0$
4:  for $\mu^{\mathbf{A}^*} \in \mathcal{M}^{\mathbf{A}^*}$ do
5:      $\mathsf{Simplify}(\varphi^*_{[\mu^{\mathbf{A}^*}]})$
6:      if $\mathsf{LiteralConjunction}(\varphi^*_{[\mu^{\mathbf{A}^*}]})$ then
7:          $vol \leftarrow vol + \mathsf{WMI}_{\mathsf{nb}}(\varphi^*_{[\mu^{\mathbf{A}^*}]}, w^*_{[\mu^{\mathbf{A}^*}]} | \mathbf{x})$
8:      else
9:          $\mathcal{M}^{\mathcal{LRA}} \leftarrow \mathcal{TA}(\varphi^*_{[\mu^{\mathbf{A}^*}]})$
10:         for $\mu^{\mathcal{LRA}} \in \mathcal{M}^{\mathcal{LRA}}$ do
11:             $vol \leftarrow vol + \mathsf{WMI}_{\mathsf{nb}}(\mu^{\mathcal{LRA}}, w^*_{[\mu^{\mathbf{A}^*}]} | \mathbf{x})$
12: return $vol$
```

3.4. Hybrid Probabilistic Inference via WMI

The main application of WMI is marginal inference on weighted SMT theories. Similarly to WMC, inference can be reduced to the computation of two weighted model integrals:

$$\Pr(\Delta \mid \chi, w) = \frac{\mathsf{WMI}(\Delta \wedge \chi, w)}{\mathsf{WMI}(\chi, w)} \tag{6}$$

The denominator above is akin to computing the partition function on probabilistic models with unnormalized factors. Crucially, the formulas Δ and χ are arbitrary, possibly encoding complex non-convex regions of a hybrid space. This goes beyond the kind of queries that are supported by more traditional algorithms like Belief Propagation, being particularly beneficial in contexts where it is necessary to compute probabilities of complex properties, like those arising in (probabilistic) formal verification domains. Furthermore, the use of specialized software to deal with constraints yields state-of-the-art results on standard queries when the support of the distribution is highly structured, as shown by the competitive results obtained by reducing inference on discrete models to WMC [38].

4. Analysis of State-Of-The-Art WMI Techniques

We start by presenting an analysis of the KC and WMI-PA approaches, which represent the current state-of-the-art in WMI.

4.1. Knowledge Compilation

We start our analysis by noticing a major problem with existing KC approaches for WMI [23, 24], in that they can easily blow up in space even with simple weight functions. Consider, e.g., the case in which

$$w(\mathbf{x}, \mathbf{A}) \stackrel{\text{def}}{=} \prod_{i=1}^{N} (\mathsf{If}\ \psi_i\ \mathsf{Then}\ w_{i1}(\mathbf{x})\ \mathsf{Else}\ w_{i2}(\mathbf{x})) \tag{7}$$

where the ψ_is are $\mathcal{LRA}$ conditions on $\{\mathbf{x}, \mathbf{A}\}$ and the w_{i1}, w_{i2} are generic functions on $\mathbf{x}$. First, the decision diagrams do not interleave arithmetic and conditional operators; rather, they push all the arithmetic operators below the conditional ones. Thus, with (7) the resulting decision diagrams consist of 2^N branches on the ψ_is, each corresponding to a distinct unconditioned weight function of the form $\prod_{i=1}^{N} w_{ij_i}(\mathbf{x})$ s.t. $j_i \in \{1, 2\}$. See Figure 1 for an example for $N = 3$ and $\psi_i \stackrel{\text{def}}{=} A_i$, $i \in \{1, 2, 3\}$. Second, the decision diagrams are built on the Boolean abstraction of $w(\mathbf{x}, \mathbf{A})$, s.t. they do not eliminate a priori the useless branches consisting of $\mathcal{LRA}$-unsatisfiable combinations of ψ_is, which can be up to exponentially many.

With WMI-PA, instead, the representation of (7) does not grow in size, because $\mathsf{FIUC}^{\mathcal{LRA}}$ functions allow for interleaving arithmetic and conditional operators. Also, the SMT-based enumeration algorithm does not generate $\mathcal{LRA}$-unsatisfiable assignments on the ψ_is.

This fact has been empirically confirmed and is shown in Figure 2, where we plot in logarithmic scale the number of nodes of a KC-based encoding of (7) using XADD for problems of increasing size, compared with the size of the encoding used by WMI-PA. This graph clearly shows the exponential blow-up of XADD size, whereas the size of the encoding used by WMI-PA grows linearly. We stress the fact that (7) is not an artificial scenario: rather, e.g., this is the case of the real-world logistics problems in [19].

4.2. WMI-PA

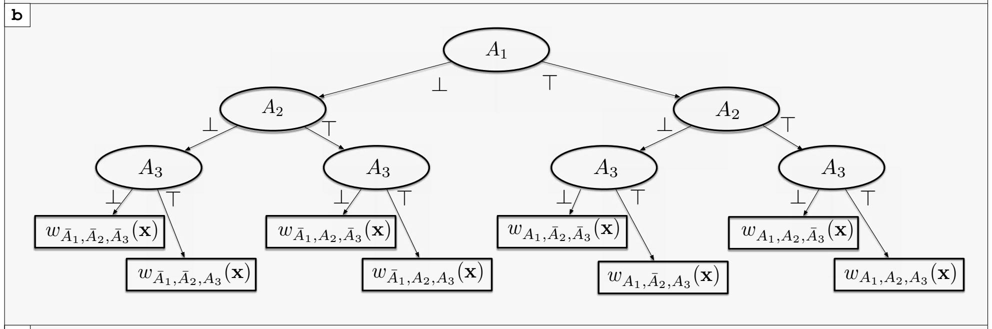
We continue our analysis by noticing a major problem also for the WMI-PA algorithm: unlike with the KC approaches, *it fails to leverage the conditional structure of the weight function to prune the set of models to integrate over.* We illustrate the issue by means of a simple example (see Figure 3).

Example 1. *Let* $\varphi \stackrel{def}{=} \top$, $\chi \stackrel{def}{=} [\![x_1 \in [0, 2]]\!] \wedge [\![x_2 \in [0, 3]]\!]$ *(Figure 3(a)) and let* $w(\mathbf{x}, \mathbf{A})$ *be a tree-structured weight function defined as in Figure 3(b). To*

a

$$w(\mathbf{x}, \mathbf{A}) \overset{\text{def}}{=} \prod_{i=1}^{3} (\text{if } A_i \text{ then } w_{i1}(\mathbf{x}) \text{ else } w_{i2}(\mathbf{x}))$$

b


A_1
$\perp$
$\top$
A_2
A_2
A_3
A_3
A_3
A_3
$w_{\bar{A}_1,\bar{A}_2,\bar{A}_3}(\mathbf{x})$
$w_{\bar{A}_1,\bar{A}_2,A_3}(\mathbf{x})$
$w_{\bar{A}_1,A_2,\bar{A}_3}(\mathbf{x})$
$w_{\bar{A}_1,A_2,A_3}(\mathbf{x})$
$w_{A_1,\bar{A}_2,\bar{A}_3}(\mathbf{x})$
$w_{A_1,\bar{A}_2,A_3}(\mathbf{x})$
$w_{A_1,A_2,\bar{A}_3}(\mathbf{x})$
$w_{A_1,A_2,A_3}(\mathbf{x})$

c

$$w_{\bar{A}_1,\bar{A}_2,\bar{A}_3}(\mathbf{x}) = w_{12}(\mathbf{x})w_{22}(\mathbf{x})w_{32}(\mathbf{x}) \qquad w_{A_1,\bar{A}_2,\bar{A}_3}(\mathbf{x}) = w_{11}(\mathbf{x})w_{22}(\mathbf{x})w_{32}(\mathbf{x})$$
$$w_{\bar{A}_1,\bar{A}_2,A_3}(\mathbf{x}) = w_{12}(\mathbf{x})w_{22}(\mathbf{x})w_{31}(\mathbf{x}) \qquad w_{A_1,\bar{A}_2,A_3}(\mathbf{x}) = w_{11}(\mathbf{x})w_{22}(\mathbf{x})w_{31}(\mathbf{x})$$
$$w_{\bar{A}_1,A_2,\bar{A}_3}(\mathbf{x}) = w_{12}(\mathbf{x})w_{21}(\mathbf{x})w_{32}(\mathbf{x}) \qquad w_{A_1,A_2,\bar{A}_3}(\mathbf{x}) = w_{11}(\mathbf{x})w_{21}(\mathbf{x})w_{32}(\mathbf{x})$$
$$w_{\bar{A}_1,A_2,A_3}(\mathbf{x}) = w_{12}(\mathbf{x})w_{21}(\mathbf{x})w_{31}(\mathbf{x}) \qquad w_{A_1,A_2,A_3}(\mathbf{x}) = w_{11}(\mathbf{x})w_{21}(\mathbf{x})w_{31}(\mathbf{x})$$

Figure 1: Example highlighting the efficiency issues of the knowledge compilation algorithms for WMI. (**a**) definition of a weight function consisting of a product of conditional statements. (**b**) decision diagram generated by knowledge compilation approaches over the weight function. Round nodes indicate if-then-else conditions, with true and false cases on the right and left outgoing edges respectively. Squared nodes indicate $\mathsf{FI}^{\mathcal{LRA}}$ weight functions. Note how the diagram has a number of branches which is exponential in the number of conditions, and no compression is achieved. (**c**) definition of the weight functions at the leaves of the diagram. In naming weight functions, we use $\bar{A}$ for $\neg A$ for the sake of compactness.

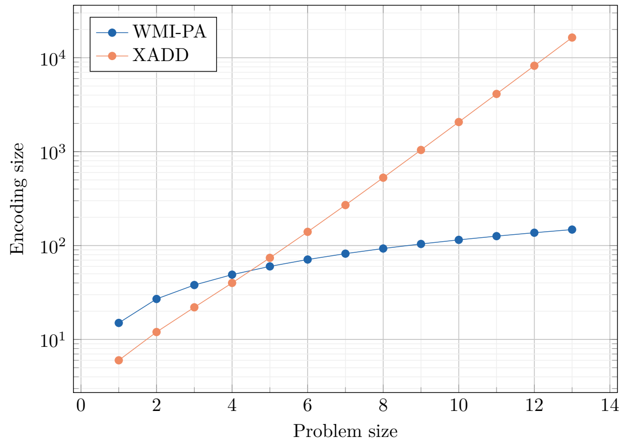


Figure 2: Size of XADD diagram (number of nodes) and WMI-PA formula as the problem size increases, in logarithmic scale. Whereas the size of the diagram grows exponentially, WMI-PA encodes the problem into a formula of linear size.

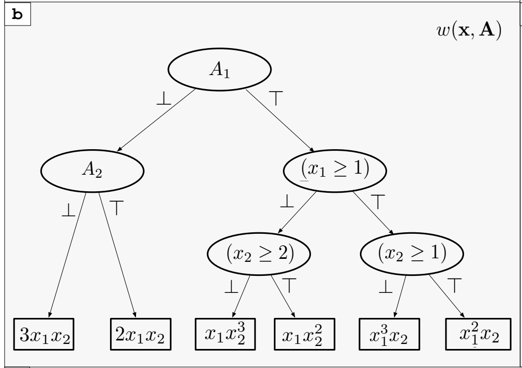
a

$$\varphi \overset{\text{def}}{=} \top,\ \chi \overset{\text{def}}{=} [\![x_1 \in [0,2]]\!] \wedge [\![x_2 \in [0,3]]\!]$$

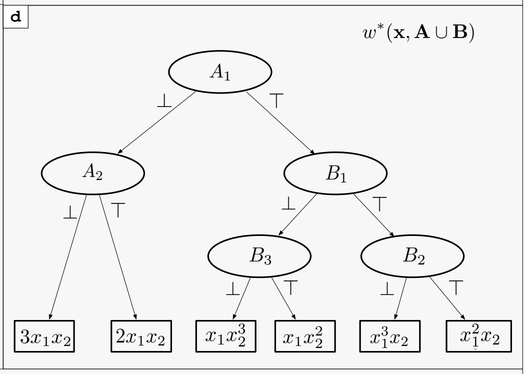
c

$$\varphi^*(\mathbf{x}, \mathbf{A} \cup \mathbf{B}) \overset{\text{def}}{=} \varphi \wedge \chi \wedge B_1 \leftrightarrow (x_1 \geq 1) \wedge B_2 \leftrightarrow (x_2 \geq 1) \wedge B_3 \leftrightarrow (x_2 \geq 2)$$

b



d



e

Assignment	*Range*	w
$\{\ A_1,\ A_2,\ B_1,\ B_2,\ B_3\}$	$[\![x_1 \in [1,2]]\!], [\![x_2 \in [2,3]]\!]$	$x_1^2x_2$
$\{\ A_1,\ A_2,\ B_1,\ B_2, \neg B_3\}$	$[\![x_1 \in [1,2]]\!], [\![x_2 \in [1,2]]\!]$	$x_1^2x_2$
$\{\ A_1,\ A_2,\ B_1, \neg B_2, \neg B_3\}$	$[\![x_1 \in [1,2]]\!], [\![x_2 \in [0,1]]\!]$	$x_1^3x_2$
$\{\ A_1,\ A_2, \neg B_1,\ B_2,\ B_3\}$	$[\![x_1 \in [0,1]]\!], [\![x_2 \in [2,3]]\!]$	$x_1x_2^2$
$\{\ A_1,\ A_2, \neg B_1,\ B_2, \neg B_3\}$	$[\![x_1 \in [0,1]]\!], [\![x_2 \in [1,2]]\!]$	$x_1x_2^3$
$\{\ A_1,\ A_2, \neg B_1, \neg B_2, \neg B_3\}$	$[\![x_1 \in [0,1]]\!], [\![x_2 \in [0,1]]\!]$	$x_1x_2^3$
$\{\ A_1, \neg A_2,\ B_1,\ B_2,\ B_3\}$	$[\![x_1 \in [1,2]]\!], [\![x_2 \in [2,3]]\!]$	$x_1^2x_2$
$\vdots$ [*same as with* $\{A_1, A_2\}$]	$\vdots$	$\vdots$
$\{\ A_1, \neg A_2, \neg B_1, \neg B_2, \neg B_3\}$	$[\![x_1 \in [0,1]]\!], [\![x_2 \in [0,1]]\!]$	$x_1x_2^3$
$\{\neg A_1,\ A_2,\ B_1,\ B_2,\ B_3\}$	$[\![x_1 \in [1,2]]\!], [\![x_2 \in [2,3]]\!]$	$2x_1x_2$
$\{\neg A_1,\ A_2,\ B_1,\ B_2, \neg B_3\}$	$[\![x_1 \in [1,2]]\!], [\![x_2 \in [1,2]]\!]$	$2x_1x_2$
$\{\neg A_1,\ A_2,\ B_1, \neg B_2, \neg B_3\}$	$[\![x_1 \in [1,2]]\!], [\![x_2 \in [0,1]]\!]$	$2x_1x_2$
$\{\neg A_1,\ A_2, \neg B_1,\ B_2,\ B_3\}$	$[\![x_1 \in [0,1]]\!], [\![x_2 \in [2,3]]\!]$	$2x_1x_2$
$\{\neg A_1,\ A_2, \neg B_1,\ B_2, \neg B_3\}$	$[\![x_1 \in [0,1]]\!], [\![x_2 \in [1,2]]\!]$	$2x_1x_2$
$\{\neg A_1,\ A_2, \neg B_1, \neg B_2, \neg B_3\}$	$[\![x_1 \in [0,1]]\!], [\![x_2 \in [0,1]]\!]$	$2x_1x_2$
$\{\neg A_1, \neg A_2,\ B_1,\ B_2,\ B_3\}$	$[\![x_1 \in [1,2]]\!], [\![x_2 \in [2,3]]\!]$	$3x_1x_2$
$\vdots$ [*same as with* $\{\neg A_1, A_2\}$]	$\vdots$	$\vdots$
$\{\neg A_1, \neg A_2, \neg B_1, \neg B_2, \neg B_3\}$	$[\![x_1 \in [0,1]]\!], [\![x_2 \in [0,1]]\!]$	$3x_1x_2$

Figure 3: Example highlighting the efficiency issues of the WMI-PA algorithm. (**a**) definition of formula φ (trivially true) and support χ. (**b**) definition of the weight function $w(\mathbf{x}, \mathbf{A})$. Round nodes indicate if-then-else conditions, with true and false cases on the right and left outgoing edges respectively. Squared nodes indicate $\mathsf{FI}^{\mathcal{LRA}}$ weight functions. (**c**) novel version of the formula $\varphi^*(\mathbf{x}, \mathbf{A} \cup \mathbf{B})$ after the application of the $\mathsf{LabelConditions}(\ldots)$ step of WMI-PA. (**d**) novel version of the weight function $w^*(\mathbf{x}, \mathbf{A} \cup \mathbf{B})$, where all $\mathcal{LRA}$ conditions have been replaced with the fresh Boolean atoms introduced in $\varphi^*(\mathbf{x}, \mathbf{A} \cup \mathbf{B})$. (**e**) List of assignments obtained by WMI-PA on $\mathbf{A} \cup \mathbf{B}$ (split on two columns for the sake of compactness). Notice the amount of assignments sharing the same $\mathsf{FI}^{\mathcal{LRA}}$ weight function.

compute $\mathsf{WMI}(\varphi \wedge \chi, w|\mathbf{x}, \mathbf{A})$*, only six integrals need to be computed:*

$x_1^2x_2$ *on* $[\![x_1 \in [1,2]]\!] \wedge [\![x_2 \in [1,3]]\!]$ *(if* $A_1 = \top$*)*
$x_1^3x_2$ *on* $[\![x_1 \in [1,2]]\!] \wedge [\![x_2 \in [0,1]]\!]$ *(if* $A_1 = \top$*)*
$x_1x_2^2$ *on* $[\![x_1 \in [0,1]]\!] \wedge [\![x_2 \in [2,3]]\!]$ *(if* $A_1 = \top$*)*
$x_1x_2^3$ *on* $[\![x_1 \in [0,1]]\!] \wedge [\![x_2 \in [0,2]]\!]$ *(if* $A_1 = \top$*)*
$2x_1x_2$ *on* $[\![x_1 \in [0,2]]\!] \wedge [\![x_2 \in [0,3]]\!]$ *(if* $A_1 = \perp, A_2 = \top$*)*
$3x_1x_2$ *on* $[\![x_1 \in [0,2]]\!] \wedge [\![x_2 \in [0,3]]\!]$ *(if* $A_1 = \perp, A_2 = \perp$*)*

When WMI-PA is used (Algorithm 1), applying $\mathsf{LabelConditions}(\ldots)$ *we obtain*

$\varphi^*(\mathbf{x}, \mathbf{A} \cup \mathbf{B}) \stackrel{\text{def}}{=} \varphi \wedge \chi \wedge B_1 \leftrightarrow (x_1 \geq 1) \wedge B_2 \leftrightarrow (x_2 \geq 1) \wedge B_3 \leftrightarrow (x_2 \geq 2)$ *(Figure 3(c)), and the weight function* $w^*(\mathbf{x}, \mathbf{A} \cup \mathbf{B})$ *shown in Figure 3(d). Then, by applying* $\mathcal{TTA}(\exists \mathbf{x}.\varphi^*)$ *(line 2) we obtain 24 total assignments* $\mathcal{M}^{\mathbf{A}^*}$ *on* $\mathbf{A} \cup \mathbf{B}$*, as shown in Figure 3(e). (Notice that all assignments containing* $\{\neg B_2, B_3\}$*, and hence* $\{\neg(x_2 \geq 1), (x_2 \geq 2)\}$*, are missing because they are* $\mathcal{LRA}$*-unsatisfiable.) As a result, WMI-PA computes 24 integrals instead of 6. In particular, WMI-PA computes twice each of the first 6 integrals, for* $\{A_1, A_2, \ldots\}$ *and* $\{A_1, \neg A_2, \ldots\}$*; also, it splits into 2 parts each the integrals of* $x_1^2 x_2$ *and* $x_1 x_2^3$ *(on the irrelevant truth values of* B_3 *and* B_2 *respectively) and into 6 parts each the integrals of* $2x_1x_2$ *and* $3x_1x_2$ *(on the 8 irrelevant truth values of* B_1, B_2, B_3 *minus the 2* $\mathcal{LRA}$*-unsatisfiable ones containing* $\{\neg B_2, B_3\}$*).* ◇

The key issue about WMI-PA is that the enumeration of $\mathcal{TTA}(\exists \mathbf{x}.\varphi^*)$ in (4) (line 2 in Algorithm 1) *is not aware of the conditional structure of the weight function* w^*, in particular, it is not aware of the fact that often *partial* assignments to the set of conditions in w^* (both in $\mathbf{A}$ and $\mathbf{B}$, i.e., both original Boolean and $\mathcal{LRA}$ conditions in w) are sufficient to identify the value of w^*, so that it is forced to enumerate all $\mathcal{LRA}$-satisfiable total assignments extending them. (E.g., in Example 1, $\{A_1, B_1, B_2\}$ suffices to identify $x_1^2 x_2$, regardless the values of A_2 and B_3, but WMI-PA enumerates $\{A_1, A_2, B_1, B_2, B_3\}$, $\{A_1, A_2, B_1, B_2, \neg B_3\}$, $\{A_1, \neg A_2, B_1, B_2, B_3\}$, $\{A_1, \neg A_2, B_1, B_2, \neg B_3\}$.) This has two consequences.

First, to make the enumerator split also on the conditions in $\mathbf{\Psi}$, WMI-PA renames them with fresh Boolean atoms $\mathbf{B}$, conjoining $\bigwedge_{k=1}^{K}(B_k \leftrightarrow \psi_k)$ to $\varphi \wedge \chi$ (line 1). Unfortunately, the above equivalences force the enumerator to assign a truth value to *all* the B_is in $\mathbf{B}$ (hence, to all $\mathcal{LRA}$-conditions ψ_is in $\mathbf{\Psi} \setminus \mathbf{A}$) in every assignment, even when the conditional structure of w does not need it. This is what forces the unnecessary partitioning of integrals into subregions. (E.g., in Example 1, the integral of $x_1^2 x_2$ is unnecessarily split into two integrals for $(x_2 \geq 2)$ and $\neg(x_2 \geq 2)$ due to the unnecessary branch on $\{B_3, \neg B_3\}$.)

Second, not knowing w^*, the enumerator is forced to always assign also *all* original Boolean atoms in $\mathbf{A}$, even when the combination $\mu^{\mathbf{A}} \cup \mu^{\mathbf{B}}$ of a *partial* assignment $\mu^{\mathbf{A}}$ on $\mathbf{A}$ and a total assignment $\mu^{\mathbf{B}}$ on $\mathbf{B}$ suffices to satisfy φ^* and to make w^* $\mathsf{FI}^{\mathcal{LRA}}$, that is, $\mathcal{TTA}(\exists \mathbf{x}.\varphi^*)$ is used instead of $\mathcal{TA}(\exists \mathbf{x}.\varphi^*)$ in (4) and in line 2 of Algorithm 1. This is what causes the unnecessary duplication of integrals. (E.g., in Example 1, each of the first 6 integrals is computed twice, for $\{A_1, A_2, \ldots\}$ and $\{A_1, \neg A_2, \ldots\}$, due to the unnecessary branching on $\{A_2, \neg A_2\}$). Although the latter problem could in principle be fixed by caching all the values of the integrals, this could be expensive in terms of both time and memory.

To cope with these issue, we need to modify WMI-PA to make it aware of the conditional structure of w. (One further issue, dealing with the fact that the conditions in $\mathbf{\Psi}$ could not be literals, will be addressed in §5.3.2.)

5. Making WMI-PA Weight-Structure Aware

In order to address the issues described in §4, we aim to combine the best of KC approaches —i.e., weight-structure awareness— with the best of PA-based approaches —i.e., SMT-based enumeration— by introducing weight-structure awareness into PA-based WMI. In §5.1 we present the general idea for making WMI-PA weight-structure aware, by introducing and exploiting the notion of *conditional skeleton* for w. In §5.2, for the sake of compactness, we only summarize the first preliminary approach and algorithm (namely "SA-WMI-PA"), based on an implicit enumeration of a skeleton, which we presented in the conference version of this paper [1] and which is described in full detail in Appendix A. In §5.3 we propose our novel and current best approach and algorithm (namely "SA-WMI-PA-SK"), which is both much simpler and more effective than that from [1], in which the skeleton is generated explicitly.

5.1. General Idea: Exploiting a Conditional Skeleton of w

We start by introducing the following concept.

Definition 1 (Conditional skeleton of w, $\mathsf{sk}(w)$). *Let φ and χ be as above; let $w(\mathbf{x}, \mathbf{A})$ be $\mathsf{FIUC}^{\mathcal{LRA}}$ on the set of conditions $\mathbf{\Psi}$. We call a* **Conditional Skeleton of** *w, written $\mathsf{sk}(w)$, any $\mathcal{LRA}$ formula s.t.:*

(a) its atoms are all and only those occurring in the conditions in $\mathbf{\Psi}$;

(b) $\mathsf{sk}(w)$ is $\mathcal{LRA}$-valid, so that $\varphi \wedge \chi$ is equivalent to $\varphi \wedge \chi \wedge \mathsf{sk}(w)$;

(c) every partial *truth value assignment μ to (the atoms occurring in) the conditions $\mathbf{\Psi}$ which makes $\mathsf{sk}(w)$ true is such that $w_{[\mu]}$ is $\mathsf{FI}^{\mathcal{LRA}}$.*

Example 2. *Consider the problem in Example 1. Then the following formula is a conditional skeleton $\mathsf{sk}(w)$ for $w(\mathbf{x}, \mathbf{A})$:*

$$\begin{array}{l|l}
A_1 \vee \neg A_1 & \textit{split on } A_1 \\
\neg A_1 \vee \ \ (x_1 \geq 1) \vee \neg(x_1 \geq 1) & \textit{if } A_1 \textit{ split on } (x_1 \geq 1) \\
\neg A_1 \vee \neg(x_1 \geq 1) \vee \ \ (x_2 \geq 1) \vee \neg(x_2 \geq 1) & \textit{if } A_1, \ \ (x_1 \geq 1) \textit{ split on } (x_2 \geq 1) \\
\neg A_1 \vee \ \ (x_1 \geq 1) \vee \ \ (x_2 \geq 2) \vee \neg(x_2 \geq 2) & \textit{if } A_1, \neg(x_1 \geq 1) \textit{ split on } (x_2 \geq 2) \\
A_1 \vee \ \ A_2 \vee \neg A_2 & \textit{if } \neg A_1 \textit{ split on } A_2
\end{array}$$

E.g., the partial assignment $\mu \overset{def}{=} \{A_1, (x_1 \geq 1), (x_2 \geq 1)\}$ *satisfies* $\mathsf{sk}(w)$*, and is such that, in Example 1,* $w_{[\mu]} = x_1^2 x_2$*, which is* $\mathsf{FI}^{\mathcal{LRA}}$*.* $\diamond$

Assume we have produced one such formula $\mathsf{sk}(w)$. Unlike with WMI-PA, we do *not* rename with $\mathbf{B}$ the conditions $\mathbf{\Psi}$ in $w(\mathbf{x}, \mathbf{A})$, and we conjoin to $\varphi \wedge \chi$ the skeleton $\mathsf{sk}(w)$ instead of $\bigwedge_{k=1}^{K}(B_k \leftrightarrow \psi_k)$. Unlike with $\bigwedge_{k=1}^{K}(B_k \leftrightarrow \psi_k)$, which needs *total* assignments to $\mathbf{\Psi}$ (i.e., to $\mathbf{B}$) to be satisfied, for $\mathsf{sk}(w)$ it suffices to produce *partial* assignments to $\mathbf{\Psi}$ which verify condition (c), i.e., those assignments μ s.t. $w_{[\mu]}$ is $\mathsf{FI}^{\mathcal{LRA}}$. Thus, we can rewrite (4)–(5) into:

$$\mathsf{WMI}(\varphi, w|\mathbf{x}, \mathbf{A}) = \sum_{\mu \in \mathcal{TA}(\varphi^{***})} 2^{|\mathbf{A} \setminus \mu|} \cdot \mathsf{WMI}_{\mathsf{nb}}(\varphi^{***}_{[\mu]}, w_{[\mu]}|\mathbf{x}) \tag{8}$$

$$\varphi^{***} \overset{\text{def}}{=} \varphi \wedge \chi \wedge \mathsf{sk}(w) \tag{9}$$

(Notice that in (8) μ is in $\mathcal{TA}(\varphi^{***})$, not $\mathcal{TTA}(\varphi^{***})$, that is, we enumerate *partial* assignments μ for φ^{***} and, in particular, *partial* assignments for $\mathbf{\Psi}$.)

Condition (c) in Definition 1 guarantees that $\mathsf{WMI}_{\mathsf{nb}}(\varphi^{***}_{[\mu]}, w_{[\mu]}|\mathbf{x})$ in (8) can be directly computed even though μ is partial, without further partitioning due to irrelevant conditions. The $2^{|\mathbf{A} \setminus \mu|}$ factor in (8), where $|\mathbf{A} \setminus \mu|$ is the number of Boolean atoms in $\mathbf{A}$ that are not assigned by μ, resembles the fact that, if $A_i \in \mathbf{A}$ is not assigned in μ, then $\mathsf{WMI}_{\mathsf{nb}}(\varphi^{***}_{[\mu]}, w_{[\mu]}|\mathbf{x})$ must be counted twice, because μ represents two assignments $\mu \cup \{A_i\}$ and $\mu \cup \{\neg A_i\}$ which would produce two identical integrals, because their $\mathcal{LRA}$ component and the branch of the weight function they identify are identical. When $|\mathbf{A} \setminus \mu| > 0$, this allows to compute only one integral rather than $2^{|\mathbf{A} \setminus \mu|}$ ones (!).

Notice that the latter is only one of the computational advantages of (8)-(9) w.r.t. (4)-(5): even more importantly, (8)-(9) allow for enumerating partial assignments also on the $\mathcal{LRA}$-conditions in $\mathbf{\Psi} \setminus \mathbf{A}$, reducing the number of integrals to compute for each partial assignment μ by a 2^j factor, j being the number of unassigned $\mathcal{LRA}$ atoms in μ. (In this case, no multiplication factor should be considered, because an unassigned $\mathcal{LRA}$ condition ψ_i in μ causes the merging of the two integration domains of $\mu \cup \{\psi_i\}$ and $\mu \cup \{\psi_i\}$.)

Example 3. *Consider the problem in Example 1 and the corresponding* $\mathsf{sk}(w)$ *formula in Example 2. Let* $\varphi^{***} \stackrel{\text{def}}{=} \varphi \wedge \chi \wedge \mathsf{sk}(w)$*, as in* (9)*. We show how* $\mathcal{TA}(\varphi^{***})$ *in* (8) *can be produced by the SMT solver. Assume nondeterministic choices are picked following the order* $A_1, A_2, (x_1 \geq 1), (x_2 \geq 1), (x_2 \geq 2)$*, assigning positive values first. Then in the first branch the following satisfying* total *truth assignment is generated:*

$$\{(x_1 \geq 0), (x_1 \leq 2), (x_2 \geq 0), (x_2 \leq 3)\} \cup \{A_1, A_2, (x_1 \geq 1), (x_2 \geq 1), (x_2 \geq 2)\}$$

(where $\{(x_1 \geq 0), (x_1 \leq 2), (x_2 \geq 0), (x_2 \leq 3)\}$ *is assigned deterministically to satisfy the* $\varphi \wedge \chi$ *part) from which the solver extracts the minimal partial truth assignment which evaluates* $\mathsf{sk}(w)$ *to true:*

$$\mu_1 \stackrel{\text{def}}{=} \{(x_1 \geq 0), (x_1 \leq 2), (x_2 \geq 0), (x_2 \leq 3)\} \cup \{A_1, (x_1 \geq 1), (x_2 \geq 1)\}.$$

In the next branch, the following minimal partial assignment is produced:

$$\mu_2 \stackrel{\text{def}}{=} \{(x_1 \geq 0), (x_1 \leq 2), (x_2 \geq 0), (x_2 \leq 3)\} \cup \{A_1, (x_1 \geq 1), \neg(x_2 \geq 1)\}.$$

s.t. μ_1, μ_2 *assign opposite truth value to (at least) one atom. Overall, the algorithm enumerates the following collection of minimal partial assignments:*

$$\begin{array}{l}
\{(x_1 \geq 0), (x_1 \leq 2), (x_2 \geq 0), (x_2 \leq 3)\} \cup \{\ \ A_1, \ \ (x_1 \geq 1), \ \ (x_2 \geq 1)\},\\
\{(x_1 \geq 0), (x_1 \leq 2), (x_2 \geq 0), (x_2 \leq 3)\} \cup \{\ \ A_1, \ \ (x_1 \geq 1), \neg(x_2 \geq 1)\},\\
\{(x_1 \geq 0), (x_1 \leq 2), (x_2 \geq 0), (x_2 \leq 3)\} \cup \{\ \ A_1, \neg(x_1 \geq 1), \ \ (x_2 \geq 2)\},\\
\{(x_1 \geq 0), (x_1 \leq 2), (x_2 \geq 0), (x_2 \leq 3)\} \cup \{\ \ A_1, \neg(x_1 \geq 1), \neg(x_2 \geq 2)\},\\
\{(x_1 \geq 0), (x_1 \leq 2), (x_2 \geq 0), (x_2 \leq 3)\} \cup \{\neg A_1, \ \ A_2\},\\
\{(x_1 \geq 0), (x_1 \leq 2), (x_2 \geq 0), (x_2 \leq 3)\} \cup \{\neg A_1, \neg A_2\}
\end{array}$$

which correspond to the six integrals of Example 1. Notice that according to (8) *the first four integrals have to be multiplied by 2, because the partial assignment* $\{A_1\}$ *covers two total assignments* $\{A_1, A_2\}$ *and* $\{A_1, \neg A_2\}$. $\diamond$

Notice that logic-wise $\mathsf{sk}(w)$ is non-informative because it is a $\mathcal{LRA}$-valid formula (condition (b) in Definition 1). Nevertheless, the role of $\mathsf{sk}(w)$ is not only to "make the enumerator aware of the presence of the conditions $\mathbf{\Psi}$" —like the $\bigwedge_{k=1}^{K}(B_k \leftrightarrow \psi_k)$ in WMI-PA— but also to mimic the conditional structure of w, forcing every assignment μ to assign truth values to all and only those conditions in $\mathbf{\Psi}$ which are necessary to make $w_{[\mu]}$ $\mathsf{FI}^{\mathcal{LRA}}$, and hence make $\mathsf{WMI}_{\mathsf{nb}}(\varphi^{***}_{[\mu]}, w_{[\mu]}|\mathbf{x})$ directly computable, without further partitioning.

In principle, we could use as $\mathsf{sk}(w)$ (a $\mathcal{LRA}$ formula encoding the conditional structure of) an XADD or (F)XSDD, which do not suffer from the lack of structure awareness of WMI-PA. However, this would cause a blow up in size, as discussed in §4.1. To this extent, avoiding $\mathsf{sk}(w)$ blow up in size is a key issue in our approach, which we will discuss in the following steps.

5.2. Implicit Generation of a Conditional Skeleton

Here we briefly summarize the first preliminary approach we proposed in the conference paper [1]. A fully-detailed description can be found in Appendix A.

Encoding. The conditional skeleton $\mathsf{sk}(w)$ we used in [1] is not generated explicitly. Rather, we have defined it as an existentially-quantified formula $\mathsf{sk}(w) \overset{\text{def}}{=} \exists \mathbf{y}.[\![y = w]\!]_{\mathcal{EUF}}$, where $[\![y = w]\!]_{\mathcal{EUF}}$ is a $\mathcal{LRA} \cup \mathcal{EUF}$-formula on $\mathbf{A}$, $\mathbf{x}$, $\mathbf{y}$ s.t. $\mathbf{y} \overset{\text{def}}{=} \{y, y_1, \ldots, y_k\}$ is a set of fresh $\mathcal{LRA}$ variables. Thus, we can compute $\mathcal{TA}(\varphi \wedge \chi \wedge \exists \mathbf{y}.[\![y = w]\!]_{\mathcal{EUF}})$ as $\mathcal{TA}(\exists \mathbf{y}.(\varphi \wedge \chi \wedge [\![y = w]\!]_{\mathcal{EUF}}))$ because the $\mathbf{y}$ do not occur in $\varphi \wedge \chi$, with no need to generate $\mathsf{sk}(w)$ explicitly.

In a nutshell, $[\![y = w]\!]_{\mathcal{EUF}}$ is obtained by taking $(y = w)$, s.t. y is fresh, and recursively substituting bottom-up every conditional term $(\mathsf{If}\ \psi_i\ \mathsf{Then}\ t_{i1}\ \mathsf{Else}\ t_{i2})$ in it with a fresh variable $y_i \in \mathbf{y}$, adding the definition of $(y_i = (\mathsf{If}\ \psi_i\ \mathsf{Then}\ t_{i1}\ \mathsf{Else}\ t_{i2}))$ as $(\neg\psi_i \vee y_i = t_{i1}) \wedge (\psi_i \vee y_i = t_{i2})$. Terms representing non-linear arithmetic operators (e.g., multiplication) or functions (e.g., $\sin, \exp$) are then substituted with uninterpreted functions, in order to avoid non-linear terms in the formula. Intuitively, each partial assignment satisfying $\exists \mathbf{y}.[\![y = w]\!]_{\mathcal{EUF}}$ represents a branch of the weight function, since it allows to uniquely identify a value for $\mathbf{y}$. (See Appendix A for an example and more details.)

Notice that the idea of renaming if-then-else terms with fresh quantified variables is what avoids the space blow-up of KC approaches, s.t. $[\![y = w]\!]_{\mathcal{EUF}}$ grows linearly in size w.r.t. the size of w. E.g., with (7), $[\![y = w]\!]_{\mathcal{EUF}}$ is $(y = \prod_{i=1}^{N} y_i) \wedge \bigwedge_{i=1}^{N}((\neg\psi_i \vee y_i = w_{i1}(\mathbf{x})) \wedge (\psi_i \vee y_i = w_{i2}(\mathbf{x})))$ —where the products and other non-linear functions are abstracted into uninterpreted functions— whose size grows linearly rather than exponentially w.r.t. N.

The SA-WMI-PA Procedure. We briefly summarize how the enumeration procedure used by SA-WMI-PA works (see Algorithm 5 in Appendix A for details). We first extend $\varphi \wedge \chi$ into $\varphi^{**} \overset{\text{def}}{=} \varphi \wedge \chi \wedge [\![y = w]\!]_{\mathcal{EUF}}$. Then, as with WMI-PA, we enumerate the assignments in two main steps:

1. We first generate a set $\mathcal{M}^{\mathbf{A}^*}$ of *partial* assignments $\mu^{\mathbf{A}^*}$ over the Boolean atoms $\mathbf{A}$, s.t. $\varphi^{**}_{[\mu^{\mathbf{A}^*}]}$ is $\mathcal{LRA}$-satisfiable and does not contain Boolean

atoms anymore. (About assignment enumeration, recall Remark 1.) Unlike with WMI-PA, these assignments are generated in two steps. We first enumerate partial Boolean assignments by invoking $\mathcal{TA}(\exists\mathbf{x}.\exists\mathbf{y}.\varphi^{**})$ and, for each assignment $\mu^{\mathbf{A}}$, we build the (simplified) residual $\varphi^{**}_{[\mu^{\mathbf{A}}]}$. Since $\mu^{\mathbf{A}}$ is partial, however, $\varphi^{**}_{[\mu^{\mathbf{A}}]}$ is not guaranteed to be free of Boolean atoms $\mathbf{A}$. [6] If this is the case, we simply add $\mu^{\mathbf{A}}$ to $\mathcal{M}^{\mathbf{A}^*}$, otherwise we further invoke $\mathcal{TTA}(\exists\mathbf{x}.\exists\mathbf{y}.\varphi^{**}_{[\mu^{\mathbf{A}}]})$ to assign the remaining Boolean atoms, ensuring that the residual now contains only $\mathcal{LRA}$ atoms. Each assignment $\mu^{\mathbf{A}}_{residual}$ to the remaining atoms is then conjoined to $\mu^{\mathbf{A}}$, and $\mu^{\mathbf{A}} \wedge \mu^{\mathbf{A}}_{residual}$ is added to $\mathcal{M}^{\mathbf{A}^*}$.

2. The second step is nearly identical to the "for" loop in WMI-PA. For each $\mu^{\mathbf{A}^*}$ in $\mathcal{M}^{\mathbf{A}^*}$ we enumerate a set $\mathcal{M}^{\mathcal{LRA}} \overset{\text{def}}{=} \mathcal{TA}(\exists\mathbf{y}.\varphi^{**}_{[\mu^{\mathbf{A}^*}]})$ of $\mathcal{LRA}$-satisfiable partial assignments satisfying $\varphi^{**}_{[\mu^{\mathbf{A}^*}]}$. (Like in WMI-PA, if $\varphi^{**}_{[\mu^{\mathbf{A}^*}]}$ is a conjunction of literals, then we have only one $\mu^{\mathcal{LRA}} \overset{\text{def}}{=} \varphi^{**}_{[\mu^{\mathbf{A}^*}]}$.) For each $\mu^{\mathcal{LRA}} \in \mathcal{M}^{\mathcal{LRA}}$ we compute the integral $\mathsf{WMI}_{\mathsf{nb}}(\mu^{\mathcal{LRA}}, w_{[\mu^{\mathbf{A}^*}]}|\mathbf{x})$. Unlike with WMI-PA, since $\mu^{\mathbf{A}^*}$ can be partial and thus represent $2^{|\mathbf{A}\setminus\mu^{\mathbf{A}^*}|}$ total assignments, we multiply the integral by a $2^{|\mathbf{A}\setminus\mu^{\mathbf{A}^*}|}$ factor (as in (8)) and add it to the result.

5.3. Encoding a Conditional Skeleton Explicitly

We propose a novel approach, which is both much simpler and more effective than that from [1], in which the skeleton is generated explicitly.

5.3.1. Encoding

Algorithm 2 shows the procedure for building explicitly a formula $\mathsf{sk}(w)$. Intuitively, the output consists of a conjunction of formulas, each stating "*if all the conditions $\psi_1, .., \psi_i$ of the current sub-branch in w hold, then split on the next condition ψ_{i+1} in the branch*". For example, for the problem in Example 1, Algorithm 2 produces as $\mathsf{sk}(w)$ the formula reported in Example 2, whose satisfying partial assignments can be enumerated as in Example 3.

The recursive procedure $\mathsf{Convert}_{\mathcal{SK}}(w_i, \mathsf{conds})$ takes as input the current subtree w_i of the weight function w and the set of literals conds representing the conditions under whose scope w_i occurs, and returns the CNF representation of $\bigwedge_{\psi_i\in\mathsf{conds}} \psi_i \rightarrow \mathsf{sk}(w_i)$ (i.e., it returns $[\![\bigvee_{\psi_i\in\mathsf{conds}} \neg\psi_i \vee \ \mathsf{sk}(w_i)]\!]$).

[6] see e.g. Example 8 in Appendix A if interested.

Algorithm 2 $\mathsf{Convert}_{\mathcal{SK}}(\mathsf{w}, \mathsf{conds})$ // conds is a set of literal conditions

```
1:  if ({$\mathsf{w}$ is a constant or a polynomial}) then
2:      return $\top$
3:  if ($\mathsf{w} == (\mathsf{w}_1 \bowtie \mathsf{w}_2)$, $\bowtie \in \{+, -, \cdot\}$) then
4:      return $\mathsf{Convert}_{\mathcal{SK}}(\mathsf{w}_1, \mathsf{conds}) \wedge \mathsf{Convert}_{\mathcal{SK}}(\mathsf{w}_2, \mathsf{conds})$
5:  if ($\mathsf{w} == g(\mathsf{w}_1, \dots, \mathsf{w}_k)$, $g$ unconditioned) then
6:      return $\bigwedge_{i=1}^{k} \mathsf{Convert}_{\mathcal{SK}}(\mathsf{w}_i, \mathsf{conds})$
7:  if ($\mathsf{w} == (\mathsf{If}\ \psi\ \mathsf{Then}\ \mathsf{w}_1\ \mathsf{Else}\ \mathsf{w}_2)$) then
8:      if $\psi$ is a literal then
9:          $\mathsf{branch} \leftarrow \bigvee_{\psi_i \in \mathsf{conds}} \neg\psi_i \vee \psi \vee \neg\psi$
10:         $\mathsf{defs}_1 \leftarrow \mathsf{Convert}_{\mathcal{SK}}(\mathsf{w}_1, \mathsf{conds} \cup \{\psi\})$
11:         $\mathsf{defs}_2 \leftarrow \mathsf{Convert}_{\mathcal{SK}}(\mathsf{w}_2, \mathsf{conds} \cup \{\neg\psi\})$
12:         return $\mathsf{branch} \wedge \mathsf{defs}_1 \wedge \mathsf{defs}_2$
13:     else
14:         let $B$ be a fresh Boolean atom
15:         $\mathsf{branch} \leftarrow$ $(\bigvee_{\psi_i \in \mathsf{conds}} \neg\psi_i \vee B \vee \neg B) \wedge$
                $[\![\bigvee_{\psi_i \in \mathsf{conds}} \neg\psi_i \vee \mathsf{CNF}_{\mathsf{pg}}(B \leftrightarrow \psi)]\!]$
16:         $\mathsf{defs}_1 \leftarrow \mathsf{Convert}_{\mathcal{SK}}(\mathsf{w}_1, \mathsf{conds} \cup \{B\})$
17:         $\mathsf{defs}_2 \leftarrow \mathsf{Convert}_{\mathcal{SK}}(\mathsf{w}_2, \mathsf{conds} \cup \{\neg B\})$
18:         return $\mathsf{branch} \wedge \mathsf{defs}_1 \wedge \mathsf{defs}_2$
```

Hence, $\mathsf{Convert}_{\mathcal{SK}}(w, \emptyset)$ returns $\mathsf{sk}(w)$. Notice that the set conds contains the conditions that need to be true in order to activate the current branch.

We first focus on lines 1-12, in which we assume that the weight conditions ψ_i are literals (the behaviour in case of non-literal conditions, lines 14-18, will be explained in §5.3.2). The recursive encoding of $\mathsf{sk}(w)$ works as follows:

(Lines 1-2): A constant or a polynomial does not contain weight conditions. Thus, we can simply encode it with $\top$. Notice that here conds has no role, because $(\bigwedge_{\psi_i \in \mathsf{conds}} \psi_i) \to \top$ reduces to $\top$.

(Lines 3-4): A term representing an arithmetic operator $w_1 \bowtie w_2$, with $\bowtie \in \{+, -, \cdot\}$ must ensure that the conditions in both branches w_1, w_2 are enumerated. Thus, we encode it by conjoining the results of the conversion procedure on w_1 and w_2.

(Lines 5-6): Similarly, a term representing an unconditioned function $g(w_1, \dots, w_k)$, must ensure that the conditions in all the branches

$w_1 \dots w_k$ are enumerated. Thus, we encode it by conjoining the results of the conversion procedure on $w_1, \dots, w_k$.

(Lines 7-12): When encoding a conditional term in the form $(\mathsf{If}\ \psi\ \mathsf{Then}\ w_1\ \mathsf{Else}\ w_2)$, if ψ is a literal, we have the following:

(Line 9): When all conds are true, then ψ must be split upon. This fact is encoded by the valid clause:

$$\bigvee_{\psi_i \in \mathsf{conds}} \neg\psi_i \vee \psi \vee \neg\psi. \tag{10}$$

(Line 10): When (all conds and are true and) ψ is true, all the branches of w_1 must be enumerated. This is encoded by recursively calling the conversion procedure on w_1 adding ψ to the conditions conds that need to be true to activate the branch of w_1, which returns $[\![\bigvee_{\psi_i \in \mathsf{conds}} \neg\psi_i \vee \neg\psi \vee \mathsf{sk}(w_1)]\!]$.

(Line 11): Similarly, when (all conds and are true and) ψ is false, all the branches of w_2 must be enumerated. This is encoded by recursively calling the conversion procedure on w_2 adding $\neg\psi$ to conds, which returns $[\![\bigvee_{\psi_i \in \mathsf{conds}} \neg\psi_i \vee \psi \vee \mathsf{sk}(w_1)]\!]$.

As with $\exists \mathbf{y}.[\![y = w]\!]_{\mathcal{EUF}}$ and unlike with KC approaches, it is easy to see that $\mathsf{sk}(w)$ grows linearly in size w.r.t. w. E.g., with (7), $\mathsf{sk}(w)$ is $\bigwedge_{i=1}^{N}(\psi_i \vee \neg\psi_i)$, whose size grows linearly rather than exponentially w.r.t. N.

5.3.2. Dealing with non-literal conditions

The above description of the skeleton algorithm assumes that the conditions in $\mathbf{\Psi}$ are single literals, either Boolean or $\mathcal{LRA}$ ones. In general, this is not necessarily the case, for instance it is common to have *range conditions* in the form $(x_i \geq l) \wedge (x_i \leq u)$ or even more complex conditions. In these cases, the skeleton formula is not in CNF form. In fact, $\mathsf{CNF}(\bigwedge_i \psi_i \rightarrow \varphi)$ can be straightforwardly computed out of $\mathsf{CNF}(\varphi)$ only if the ψ_is are literals, because this reduces to augment each clause in $\mathsf{CNF}(\varphi)$ with $\bigvee_i \neg\psi_i$.

If this is not the case, then the CNF-ization either may cause a blow-up in size if equivalence-preserving CNF-ization is applied, or it may require some variant of Tseitin CNF-ization [35]. SMT-solvers typically rely on the second option for satisfiability checks. For what enumeration is concerned, however, introducing new atomic labels that define complex sub-formulas can force the enumeration procedure to produce more partial assignments than necessary.

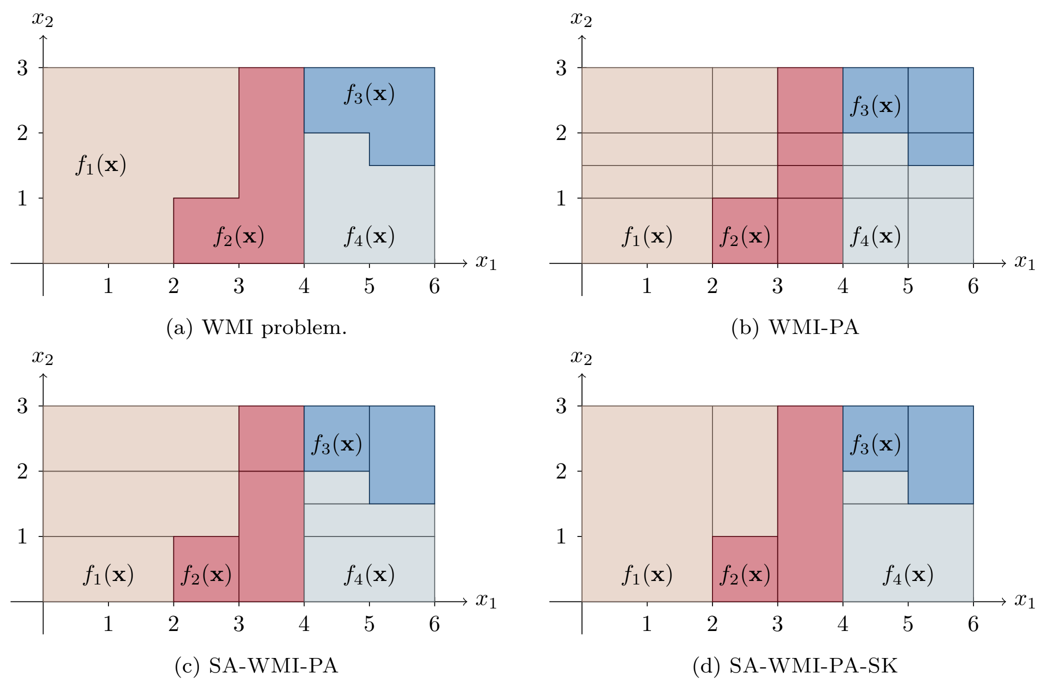


Figure 4: Graphical representation of the effectiveness of SA-WMI-PA-SK as compared to the alternatives: (a) WMI problem from Example 4, with 4 areas of $\mathsf{FI}^{\mathcal{LRA}}$ weights. Notice that the weight function includes non-literal conditions; (b) WMI-PA computes *20 distinct integrals*; (c) SA-WMI-PA, using both the implicit and explicit non-CNF skeleton versions, computes *11 distinct integrals*; (d) SA-WMI-PA-SK, that uses the "local" CNF explicit skeleton, computes *8 distinct integrals*, the minimum number of integrals that is needed to retain convexity of their areas (i.e., $\mathsf{FI}^{\mathcal{LRA}}$ weights).

This is what happens with WMI-PA, and also with SA-WMI-PA, when conditions in $\mathbf{\Psi}$ are non-literals.

We illustrate this problem in Example 4. Our proposed solution will be eventually illustrated in Example 5.

Example 4. *Consider the following WMI problem:*

$$\begin{aligned}
&\varphi = \top \\
&\chi = (x_1 \geq 0) \wedge (x_1 \leq 6) \wedge (x_2 \geq 0) \wedge (x_2 \leq 3) \\
&w = \mathsf{If}\ (x_1 \leq 4)\ \mathsf{Then} \\
&\quad (\mathsf{If}\ (x_1 \leq 2) \vee ((x_1 \leq 3) \wedge (x_2 > 1))\ \mathsf{Then}\ f_1(\mathbf{x})\ \mathsf{Else}\ f_2(\mathbf{x}))\ \mathsf{Else} \\
&\quad (\mathsf{If}\ (x_2 > 2) \vee ((x_1 > 5) \wedge (x_2 > \tfrac{3}{2}))\ \mathsf{Then}\ f_3(\mathbf{x})\ \mathsf{Else}\ f_4(\mathbf{x}))
\end{aligned}$$

s.t. the $f_i(\mathbf{x})$s are condition-less functions, which is graphically represented in Figure 4a. Notice that some conditions of the weight function are not literals. Thus, if we wrote $\varphi^{***} \overset{\text{def}}{=} \varphi \wedge \chi \wedge \mathsf{sk}(w)$ without concerning about this fact, then we would obtain the following non-CNF formula to feed to the SMT solver:

$$\begin{array}{ll}
(x_1 \geq 0) \wedge (x_1 \leq 6) \wedge (x_2 \geq 0) \wedge (x_2 \leq 3) & \wedge \\
((x_1 \leq 4) \vee \neg(x_1 \leq 4)) & \wedge \\
(\neg(x_1 \leq 4) \vee \psi_1 \vee \neg\psi_1) & \wedge \\
(\ \ (x_1 \leq 4) \vee \psi_2 \vee \neg\psi_2) &
\end{array}$$

where

$$\psi_1 \overset{\text{def}}{=} \overbrace{(x_1 \leq 2) \vee \underbrace{((x_1 \leq 3) \wedge (x_2 > 1))}_{B_3}}^{B_1},\ \psi_2 \overset{\text{def}}{=} \overbrace{(x_2 > 2) \vee \underbrace{((x_1 > 5) \wedge (x_2 > \tfrac{3}{2}))}_{B_4}}^{B_2}$$

The solver would apply Tseitin CNF-ization, labelling the subformulas by fresh Boolean atoms $\{B_1, B_2, B_3, B_4\}$ as above, producing the CNF formula ϕ [7]*:*

$$\begin{array}{ll|l}
(x_1 \geq 0) \wedge (x_1 \leq 6) \wedge (x_2 \geq 0) \wedge (x_2 \leq 3) & \wedge & \varphi \wedge \chi \\
\big(\ \ (x_1 \leq 4) \vee \neg(x_1 \leq 4)\big) & \wedge & (x_1 \leq 4) \vee \neg(x_1 \leq 4) \\
\big(\neg(x_1 \leq 4) \vee \ \ B_1 \vee \neg B_1\big) & \wedge & \neg(x_1 \leq 4) \vee B_1 \vee \neg B_1 \\
\big(\neg B_1 \vee \ \ (x_1 \leq 2) \vee \ \ B_3\big) & \wedge & \mathsf{CNF}_{ts}(B_1 \rightarrow \psi_1) \\
\big(\neg B_3 \vee \ \ (x_1 \leq 3)\big) \wedge \big(\neg B_3 \vee \ \ (x_2 > 1)\big) & \wedge & \\
\big(\ \ B_3 \vee \neg(x_1 \leq 3) \vee \neg(x_2 > 1)\big) & \wedge & \\
\big(\ \ B_1 \vee \neg(x_1 \leq 2)\big) \wedge \big(\ \ B_1 \vee \neg B_3\big) & \wedge & \mathsf{CNF}_{ts}(B_1 \leftarrow \psi_1) \\
\big(\ \ (x_1 \leq 4) \vee \ \ B_2 \vee \neg B_2\big) & \wedge & (x_1 \leq 4) \vee \ \ B_2 \vee \neg B_2 \\
\big(\neg B_2 \vee \ \ (x_2 > 2) \vee \ \ B_4\big) & \wedge & \mathsf{CNF}_{ts}(B_2 \rightarrow \psi_2) \\
\big(\neg B_4 \vee \ \ (x_1 > 5)\big) \wedge \big(\neg B_4 \vee \ \ (x_2 > \tfrac{3}{2})\big) & \wedge & \\
\big(\ \ B_4 \vee \neg(x_1 > 5) \vee \neg(x_2 > \tfrac{3}{2})\big) & \wedge & \\
\big(\ \ B_2 \vee \neg(x_2 > 2)\big) \wedge \big(\ \ B_2 \vee \neg B_4\big) & & \mathsf{CNF}_{ts}(B_2 \leftarrow \psi_2)
\end{array} \tag{11}$$

and then compute $\mathcal{TA}(\varphi^{***})$ in (8) as $\mathcal{TA}(\exists \mathbf{B}.\phi)$, s.t. the B_is are not in the relevant set of atoms for the projected enumeration (see §3.1).

[7] Here we colour the labelling clauses introduced by $\mathsf{CNF}_{\mathsf{ts}}(\dots)$ as the corresponding regions in Figure 4. We recall that $\mathsf{CNF}_{\mathsf{ts}}(B_i \leftrightarrow \psi_i)$ is $\mathsf{CNF}_{\mathsf{ts}}(B_i \rightarrow \psi_i) \wedge \mathsf{CNF}_{\mathsf{ts}}(B_i \leftarrow \psi_i)$.

Consider, e.g., the partial truth assignment:

$$\mu \overset{def}{=} \{(x_1 \geq 0), \quad (x_1 \leq 6), \quad (x_2 \geq 0), \quad (x_2 \leq 3)\} \quad \cup \qquad (12)$$
$$\{(x_1 \leq 4), \neg(x_1 \leq 2), \neg(x_1 \leq 3), \neg B_1, \neg B_3\}$$

which would be sufficient to make $w_{[\mu]}$ $\mathsf{FI}^{\mathcal{LRA}}$ *because it suffices to identify* $f_2(\mathbf{x})$*. Unfortunately,* μ *does not suffice to evaluate* (11) *to true because of the last six clauses, which represent* $B_2 \leftrightarrow ((x_2 > 2) \vee ((x_1 > 5) \wedge (x_2 > \frac{3}{2})))$*. Thus, when computing* $\mathcal{TA}(\exists \mathbf{B}.\phi)$*, the solver is forced to assign truth values also to other atoms, in particular to* B_2 *and* B_4*. Whereas* B_4 *can be consistently assigned only to false due to the clause* $(\neg B_4 \vee (x_1 > 5))$ *and the fact that* $(x_1 \leq 4)$ *is true,* B_2 *can assume consistently both truth values, so that the solver is forced to branch on* B_2 *and* $\neg B_2$*, which respectively force* $(x_2 > 2)$ *and* $\neg(x_2 > 2)$*, causing the unnecessary generation of two distinct integrals on* $f_2(\mathbf{x})$*, for* $x_1 \in [3,4], x_2 \in [0,2]$ *and* $x_1 \in [3,4], x_2 \in]2,3]$ *respectively, instead of only one on* $x_1 \in [3,4], x_2 \in [0,3]$*.(Notice that the atom* $(x_2 > 2)$ *belongs to a condition in the "*$(x_1 \leq 4) = \perp$*" branch of* w*, whereas with* μ *we are actually considering the "*$(x_1 \leq 4) = \top$*" branch.)*

This issue is graphically represented in Figure 4c, where we show the regions enumerated by the SA-WMI-PA procedure on the above problem, by using either the implicit version of $\mathsf{sk}(w)$ *as in the original algorithm or the non-CNF explicit version of* $\mathsf{sk}(w)$*: 11 convex regions are enumerated. Notice that the brown, red and grey areas are each split into tree regions, whereas it is possible to split each into two convex regions only.* ◇

The source of the problem is that, with Tseitin CNF-ization, each "labelling definition" $\mathsf{CNF_{ts}}(B_i \leftrightarrow \psi_i)$ is conjoined to the rest of the formula, so that the SMT solver is forced to enumerate the different ways to satisfy it *even when the rest of the assignment selects a branch which does not involve the condition* ψ_i. Notice that the fact that B_i is implicitly treated as existentially-quantified, so that it is not in the relevant-atom set for the projected enumerator (see §3.1) does not help, because ψ_i and the atoms in it are in the relevant-atom set, s.t. the enumerator is forced to split on their truth values anyway. (E.g., in Example 4, although B_2 is implicitly existentially quantified, the enumerator is forced to split on $(x_2 > 2)$ anyway.)

To cope with this problem, we propose a variant of the Tseitin CNF-ization for the skeleton where each "labelling definition" $\mathsf{CNF_{ts}}(B \leftrightarrow \psi)$ and all occurrences of B are "local", that is, they occur in the formula only implied by the conditions in conds, so that they are simplified to true unless the assignment selects a branch which satisfies all conditions in conds. Thus, in Algorithm 2, lines 14-18, we substitute each non-literal condition ψ in branch and defs_i from lines 9-12 with a fresh Boolean atom B and we add $[\![\bigvee_{\psi_i \in \mathsf{conds}} \neg\psi_i \vee \mathsf{CNF}_{\mathsf{pg}}(B \leftrightarrow \psi)]\!]$ [8] instead of $\mathsf{CNF}_{\mathsf{ts}}(B \leftrightarrow \psi)$ to the main formula, where $\mathsf{CNF}_{\mathsf{pg}}(\dots)$ is the CNF-ization in [36] [9], so that branch is tautologically equivalent to $(\bigwedge_{\psi_i \in \mathsf{conds}} \psi_i) \rightarrow ((B \vee \neg B) \wedge \mathsf{CNF}_{\mathsf{pg}}(B \leftrightarrow \psi))$. (We recall that all ψ_is in conds are literals by construction, see Algorithm 2.)

Example 5. *With the problem of Example 4, $\varphi^{***} \overset{\text{def}}{=} \varphi \wedge \chi \wedge \mathsf{sk}(w)$ is* [10]*:*

$$
\begin{array}{lc|l}
(x_1 \ge 0) \wedge (x_1 \le 6) \wedge (x_2 \ge 0) \wedge (x_2 \le 3) & \wedge & \varphi \wedge \chi \\
\Big(\quad (x_1 \le 4) \vee \neg(x_1 \le 4)\Big) & \wedge & (x_1 \le 4) \vee \neg(x_1 \le 4) \\
\Big(\neg(x_1 \le 4) \vee \quad B_1 \vee \neg B_1\Big) & \wedge & \neg(x_1 \le 4) \vee B_1 \vee \neg B_1 \\
\Big(\neg(x_1 \le 4) \vee \neg B_1 \vee \quad (x_1 \le 2) \vee \quad B_3\Big) & \wedge & [\![\neg(x_1 \le 4) \vee \mathsf{CNF}_{pg}(B_1 \rightarrow \psi_1)]\!] \\
\Big(\neg(x_1 \le 4) \vee \neg B_1 \vee \neg B_3 \vee \quad (x_1 \le 3)\Big) & \wedge & \\
\Big(\neg(x_1 \le 4) \vee \neg B_1 \vee \neg B_3 \vee \quad (x_2 > 1)\Big) & \wedge & \\
\Big(\neg(x_1 \le 4) \vee \quad B_1 \vee \neg(x_1 \le 2)\Big) & \wedge & [\![\neg(x_1 \le 4) \vee \mathsf{CNF}_{pg}(B_1 \leftarrow \psi_1)]\!] \\
\Big(\neg(x_1 \le 4) \vee \quad B_1 \vee \neg(x_1 \le 3) \vee \neg(x_2 > 1)\Big) & \wedge & \\
\Big(\quad (x_1 \le 4) \vee \quad B_2 \vee \neg B_2\Big) & \wedge & (x_1 \le 4) \vee B_2 \vee \neg B_2 \\
\Big(\quad (x_1 \le 4) \vee \neg B_2 \vee \quad (x_2 > 2) \vee \quad B_4\Big) & \wedge & [\![\quad (x_1 \le 4) \vee \mathsf{CNF}_{pg}(B_2 \rightarrow \psi_2)]\!] \\
\Big(\quad (x_1 \le 4) \vee \neg B_2 \vee \neg B_4 \vee \quad (x_1 > 5)\Big) & \wedge & \\
\Big(\quad (x_1 \le 4) \vee \neg B_2 \vee \neg B_4 \vee \quad (x_2 > \tfrac{3}{2})\Big) & \wedge & \\
\Big(\quad (x_1 \le 4) \vee \quad B_2 \vee \neg(x_2 > 2)\Big) & \wedge & [\![\quad (x_1 \le 4) \vee \mathsf{CNF}_{pg}(B_2 \leftarrow \psi_2)]\!] \\
\Big(\quad (x_1 \le 4) \vee \quad B_2 \vee \neg(x_1 > 5) \vee \neg(x_2 > \tfrac{3}{2})\Big) & &
\end{array}
$$

(Here B_1, B_2 are labels for conditions $\psi_1 \overset{\text{def}}{=} (x_1 \le 2) \vee ((x_1 \le 3) \wedge (x_2 > 1))$, $\psi_2 \overset{\text{def}}{=} (x_2 > 2) \vee ((x_1 > 5) \wedge (x_2 > \frac{3}{2}))$ respectively, as in Example 4, whereas

[8] recall from §3.1 that $[\![C \vee \bigwedge_i C_i]\!] \overset{\text{def}}{=} \bigwedge_i (C \vee C_i)$.

[9] Here we use $\mathsf{CNF}_{\mathsf{pg}}$ instead of $\mathsf{CNF}_{\mathsf{ts}}$ because the former generates fewer clauses and behaves generally better, although we could use $\mathsf{CNF}_{\mathsf{ts}}$ without affecting correctness.

[10] Here we colour the literals deriving from the labelling clauses introduced by $\mathsf{CNF}_{\mathsf{pg}}(\dots)$ as the corresponding regions in Figure 4. We recall that $\mathsf{CNF}_{\mathsf{pg}}(B_i \leftrightarrow \psi_i)$ is $\mathsf{CNF}_{\mathsf{pg}}(B_i \rightarrow \psi_i) \wedge \mathsf{CNF}_{\mathsf{pg}}(B_i \leftarrow \psi_i)$.

Algorithm 3 SA-WMI-PA-SK($\varphi, w, \mathbf{x}, \mathbf{A}$)

```
1:  $\mathsf{sk}(w) \leftarrow \mathsf{Convert}_{\mathcal{SK}}(w, \emptyset)$
2:  $\varphi^{***} \leftarrow \varphi \wedge \chi \wedge \mathsf{sk}(w)$
3:  $\mathcal{M}^{\mathbf{A}} \leftarrow \mathcal{TA}(\exists \mathbf{x}.\varphi^{***})$
4:  $vol \leftarrow 0$
5:  for $\mu^{\mathbf{A}} \in \mathcal{M}^{\mathbf{A}}$ do
6:      $k \leftarrow |\mathbf{A} \setminus \mu^{\mathbf{A}}|$
7:      $\mathsf{Simplify}(\varphi^{***}_{[\mu^{\mathbf{A}}]})$
8:      if $\mathsf{LiteralConjuction}(\varphi^{***}_{[\mu^{\mathbf{A}}]})$ then
9:          $vol \leftarrow vol + 2^k \cdot \mathsf{WMI}_{\mathsf{nb}}(\varphi^{***}_{[\mu^{\mathbf{A}}]}, w_{[\mu^{\mathbf{A}}]}|\mathbf{x})$
10:     else
11:         $\mathcal{M} \leftarrow \mathcal{TA}(\varphi^{***}_{[\mu^{\mathbf{A}}]})$
12:         for $(\mu \overset{\text{def}}{=} \mu^{\mathbf{A}\prime} \wedge \mu^{\mathcal{LRA}}) \in \mathcal{M}$ do
13:             $k' \leftarrow k - |\mu^{\mathbf{A}\prime}|$
14:             $vol \leftarrow vol + 2^{k'} \cdot \mathsf{WMI}_{\mathsf{nb}}(\mu^{\mathcal{LRA}}, w_{[\mu^{\mathbf{A}} \wedge \mu]}|\mathbf{x})$
15: return $vol$
```

B_3, B_4 are labels introduced by $\mathsf{CNF}_{pg}(\ldots)$, which do not produce valid clauses in the form $(\ldots \vee B_i \vee \neg B_i)$.) For instance, the assignment μ in (12) *evaluates to true all the clauses of φ^{***}, so that the solver considers it as satisfying assignment for the formula.*

In Figure 4d we show the regions enumerated by the SA-WMI-PA-SK procedure (§5.3.3), using as skeleton the "local" CNF version of $\mathsf{sk}(w)$*. We see that the regions enumerated are 8, instead of the 11 of Figure 4c. In this case 8 is also the minimum number of convex areas into which we can split the problem, because the 4 distinct coloured areas in Figure 4a are non-convex, and as such their integral cannot be computed with single integrations.* $\diamond$

We stress the importance of the locality of labelling clauses in the enumeration of truth assignments: the SMT enumerator needs branching on B (hence, on ψ) *only in those branches in which all* conds *are true*, that is, only in those branches where a truth value for condition ψ is needed to define a branch of w. Notice that, if $\mathsf{CNF}_{\mathsf{pg}}$ introduces some labelling definitions, then also the latter are active only when all conds are true.

5.3.3. The SA-PA-WMI-SK Procedure

We devised a new algorithm, namely SA-WMI-PA-SK, that *explicitly* encodes the weight function as $\mathsf{sk}(w)$ (§5.3.1), handles non-literal conditions effectively (§5.3.2), and introduces some further improvements in the enumeration, which we describe below. The outline of the procedure is shown in Algorithm 3. (To simplify the narration, here we assume that the encoding of $\mathsf{sk}(w)$ did not cause the introduction of fresh Boolean variables $\mathbf{B}$, see §5.3.2; if not so, then $\mathcal{TA}(\exists \mathbf{x}.\varphi^{***})$ and $\mathcal{TA}(\varphi^{***}_{[\mu^{\mathbf{A}}]})$ in Algorithm 3 should be replaced with $\mathcal{TA}(\exists \mathbf{x}.\exists \mathbf{B}.\varphi^{***})$ and $\mathcal{TA}(\exists \mathbf{B}.\varphi^{***}_{[\mu^{\mathbf{A}}]})$ respectively.)

- (Lines 1-3) After producing $\mathsf{sk}(w)$ by Algorithm 2, we first enumerate by projected AllSMT a set $\mathcal{M}^{\mathbf{A}}$ of *partial* truth assignments $\mu^{\mathbf{A}}$ over $\mathbf{A}$, s.t. $\mathcal{M}^{\mathbf{A}} \stackrel{\text{def}}{=} \mathcal{TA}(\exists \mathbf{x}.(\varphi \wedge \chi \wedge \mathsf{sk}(w)))$.
- (Lines 5-14) For each $\mu^{\mathbf{A}} \in \mathcal{M}^{\mathbf{A}}$ we compute and simplify the residual formula $\varphi^{***}_{[\mu^{\mathbf{A}}]}$. As with WMI-PA and SA-WMI-PA, if $\varphi^{***}_{[\mu^{\mathbf{A}}]}$ is already a conjunction of literals, then we can simply compute and return its integral, which is multiplied by a 2^k factor, k being the number of unassigned Boolean atoms in $\mu^{\mathbf{A}}$. [11] If this is not the case, we proceed by enumerating a set $\mathcal{M}$ of *partial* truth assignments $\mu = \mu^{\mathbf{A}'} \wedge \mu^{\mathcal{LRA}}$ over the remaining atoms of the formula, s.t. $\mathcal{M} \stackrel{\text{def}}{=} \mathcal{TA}(\exists \mathbf{x}.\varphi^{***}_{[\mu^{\mathbf{A}}]})$. Notice that, since $\mu^{\mathbf{A}}$ is partial, $\varphi^{***}_{[\mu^{\mathbf{A}}]}$ could still contain Boolean atoms (see Examples 6 and 8), so that μ could contain some non-empty Boolean component $\mu^{\mathbf{A}'}$. We then compute the integral of the $\mathsf{FI}^{\mathcal{LRA}}$ function $w_{[\mu^{\mathbf{A}} \wedge \mu]}$ over the convex area $\mu^{\mathcal{LRA}}$. As above, we need to multiply this integral by a $2^{k'}$ factor, $k' \stackrel{\text{def}}{=} k - |\mu^{\mathbf{A}'}|$ being the number of unassigned Boolean atoms in $\mu^{\mathbf{A}} \wedge \mu$.

As with WMI-PA and SA-WMI-PA, in the implementation (Lines 3-5 and 10-12) the sets $\mathcal{M}^{\mathbf{A}}$ and $\mathcal{M}$ are not generated explicitly; rather, their elements are generated, integrated and then dropped one-by-one, so as to avoid space blow-up (see Remark 1).

We remark that producing smaller partial assignments over the Boolean atoms, as done with SA-WMI-PA-SK, has a positive effect regardless of

[11]Notice that in this case the conjunction of literals $\varphi^{***}_{[\mu^{\mathbf{A}}]}$ does not contain Boolean literals, because they would have already been assigned by $\mu^{\mathbf{A}}$.

the structure of the weight function. In Example 6 we give an intuition of the benefits introduced by the improvements of the enumeration phase in SA-WMI-PA-SK.

$\mu^{\mathbf{A}}$			$\mu^{\mathbf{A}}_{residual}$		$\mu^{\mathcal{LRA}}$	
$\neg A_1$	$\neg A_2$	A_3			$(x \geq 0)$	$(x \leq 3)$
A_1	$\neg A_2$			A_3	$(x \geq 1)$	$(x \leq 3)$
A_1	$\neg A_2$			$\neg A_3$	$(x \geq 1)$	$(x \leq 4)$
	A_2		A_1	A_3	$(x \geq 2)$	$(x \leq 3)$
	A_2		A_1	$\neg A_3$	$(x \geq 2)$	$(x \leq 4)$
	A_2		$\neg A_1$	A_3	$(x \geq 2)$	$(x \leq 3)$
	A_2		$\neg A_1$	$\neg A_3$	$(x \geq 2)$	$(x \leq 4)$

(a) Assignments enumerated by WMI-PA and SA-WMI-PA

$\mu^{\mathbf{A}}$			$\mu \overset{\text{def}}{=} \mu^{\mathbf{A}\prime} \wedge \mu^{\mathcal{LRA}}$		
$\neg A_1$	$\neg A_2$	A_3		$(x \geq 0)$	$(x \leq 3)$
A_1	$\neg A_2$		A_3	$(x \geq 1)$	$(x \leq 3)$
A_1	$\neg A_2$		$\neg A_3$	$(x \geq 1)$	$(x \leq 4)$
	A_2		A_3	$(x \geq 2)$	$(x \leq 3)$
	A_2		$\neg A_3$	$(x \geq 2)$	$(x \leq 4)$

(b) Assignments enumerated by SA-WMI-PA-SK.

Table 1: Assignments enumerated by WMI-PA and SA-WMI-PA (a) and by SA-WMI-PA-SK (b) for the problem in Example 6. (We omit the $\mathcal{LRA}$-literals which are implied by the others and do not contribute to the integral —e.g., $(x \geq 0)$ when $(x \geq 2)$ is true.)

Example 6. *Consider the following WMI problem:*

$$\begin{aligned}
&\varphi = (A_1 \vee A_2 \vee A_3) \wedge (\neg A_1 \vee (x \geq 1)) \wedge (\neg A_2 \vee (x \geq 2)) \wedge (\neg A_3 \vee (x \leq 3)) \\
&\chi = (x \geq 0) \wedge (x \leq 4) \\
&w = 1.0
\end{aligned}$$

(w is constant and $\mathbf{\Psi} = \emptyset$, thus $\bigwedge_i B_i \leftrightarrow \psi_i$, $\exists \mathbf{y}.[\![y = w]\!]_{\mathcal{EUF}}$, $\mathsf{sk}(w)$ reduce to $\top$ and $\varphi^* = \varphi^{**} = \varphi^{***} \overset{def}{=} \varphi \wedge \chi$, so that the kind of conditional skeleton used is irrelevant.) Both WMI-PA and SA-WMI-PA enumerate the 7 assignments listed in Table 1a, whereas with SA-WMI-PA-SK the number of assignments enumerated shrinks to 5, as shown in Table 1b.

With WMI-PA, $\mathcal{TTA}(\exists \mathbf{x}.\varphi^*)$ directly generates in one step the 7 total truth assignments $\mu^{\mathbf{A}} \wedge \mu^{\mathbf{A}}_{residual}$ in Table 1a.

With both SA-WMI-PA and SA-WMI-PA-SK, $\mathcal{TA}(\exists \mathbf{x}.\varphi^{**})$ and $\mathcal{TA}(\exists \mathbf{x}.\varphi^{***})$ produce the partial assignment $\mu^{\mathbf{A}} = \{A_2\}$, so that $\varphi^{**}_{[\mu^{\mathbf{A}}]}$ and $\varphi^{***}_{[\mu^{\mathbf{A}}]}$ reduce to $(x \geq 0) \wedge (x \leq 4) \wedge (\neg A_1 \vee (x \geq 1)) \wedge (x \geq 2) \wedge (\neg A_3 \vee (x \leq 3))$. Then:

with SA-WMI-PA, $\mathcal{TTA}(\exists \mathbf{x}.\varphi^{**}_{[\mu^{\mathbf{A}}]})$ enumerates all 4 total residual assignments on A_1, A_3,[12] *duplicating the two integrals on $x \in [2,3]$ and $x \in [2,4]$ by uselessly case-splitting on A_1 and $\neg A_1$, as in the last four rows in Table 1a;*

with SA-WMI-PA-SK, $\mathcal{TA}(\varphi^{***}_{[\mu^{\mathbf{A}}]})$ enumerates only the two partial assignments:[13] $\{\ \ A_3, (x \geq 0), (x \leq 4), (x \geq 1), (x \geq 2), (x \leq 3)\}$,
$\{\neg A_3, (x \geq 0), (x \leq 4), (x \geq 1), (x \geq 2)\}$,
as reported in the last two rows in Table 1b. The two corresponding integrals are multiplied by $2^{(3-|\{A_2\}|-|\{A_3\}|)} = 2$ and $2^{(3-|\{A_2\}|-|\{\neg A_3\}|)} = 2$ respectively. $\diamond$

The benefit becomes more evident as the number of Boolean atoms increases, since if we avoid computing the $\mathcal{TTA}(\exists \mathbf{x}.\varphi^{**}_{[\mu^{\mathbf{A}}]})$ over the residual Boolean atoms, then we are potentially largely cutting the number of integrals.

6. Experimental Evaluation

In the following experiments, we evaluate the performance gain of SA-WMI-PA-SK over the current state of the art in WMI. Specifically, we explore the effect of the novel structure encoding using the same setting of our recent conference paper [1]. To this extent, we compare the proposed algorithms, SA-WMI-PA and SA-WMI-PA-SK, with the previous SMT-based approach WMI-PA [19], and with the state-of-the-art KC-based approaches: XADD [22], XSDD and FXSDD [24]. For WMI-PA, SA-WMI-PA and SA-WMI-PA-SK, we use‘a slightly-modified version of MathSAT [37] [14] for SMT enumeration.[15].

[12]Here $\exists \mathbf{x}.\varphi^{**}_{[\mu^{\mathbf{A}}]}$ corresponds to valid Boolean formula on $\{A_1, A_3\}$.

[13]Here the truth value of A_1 is irrelevant because $(x \geq 1)$ is forced to be true by $(x \geq 2)$.

[14]We have added one option to MathSAT allowing to disable the simplification of valid clauses, which would otherwise eliminate all the clauses "$(... \vee \psi_i \vee \neg\psi_i)$" from $\mathsf{sk}(w)$. The binary file integrating these modifications is attached to the code of SA-WMI-PA-SK.

[15]To compute $\mathcal{TA}(\ldots)$, MathSAT is invoked with the options: `-dpll.allsat_allow_duplicates=false` (all assignments differ for at least one truth value), `-dpll.allsat_minimize_model=true` (assignments are partial and are minimized), `-preprocessor.simplification=0` and `-preprocessor.toplevel_propagation=false` (disable several non-validity-preserving preprocessing steps). To compute $\mathcal{TTA}(\ldots)$, the second option is set to `false`.

Convex integration is handled by LattE Integrale [39] in the SMT-based approaches, whereas KC-based solvers use PSI Algebra for both integration and inconsistency checking [40]. This difference is not penalizing the KC-based approaches, which were shown to achieve worse results when using LattE [24].

All experiments are performed on an Intel Xeon Gold 6238R @ 2.20GHz 28 Core machine with 128 GB of RAM and running Ubuntu Linux 20.04. The code of both SA-WMI-PA and SA-WMI-PA-SK is freely available at `https://github.com/unitn-sml/wmi-pa`, and the benchmarks are available at `https://github.com/weighted-model-integration/wmi-benchmarks` in pywmi [41] format.

We report our findings using cactus plots. Each tick on the x-axis corresponds to a single instance, and results on the y-axis are sorted independently for each method. Steeper slopes mean lower efficiency.

6.1. Synthetic experiments

Dataset description. As in our previous paper, random formulas are obtained using the generator introduced by Morettin et al. [19]. In contrast with other recent works [22, 24], these synthetic benchmarks do not contain strong structural regularities, offering a more neutral perspective on how the different techniques are expected to perform on average. We fix to 3 the number of Boolean and real variables, while varying the depth of the weights in the range $[4, 7]$. Timeout is set to 3600 seconds.

Results. Consistently with previous papers, in this setting SMT-based solvers obtain superior performance with respect to KC-based ones (Figure 5 left), which suffer from the complex algebraic dependencies between continuous variables. Our structure-aware algorithms, SA-WMI-PA and SA-WMI-PA-SK, further drastically improve over WMI-PA in terms of time and number of computed integrals (Figure 5). These results are aligned with our expectations. Indeed, the advantage of structure-awareness given by the introduction of either form of skeleton is eye-catching.

Additionally, SA-WMI-PA-SK is able to gain some advantage over its previous version SA-WMI-PA, although the order of magnitude remains quite the same. Notice that in this setting, the number of Boolean atoms is limited to 3, so that the revised enumeration technique of §5.3.3 for the Boolean component of the formula does not show its full potential. The benefits of the novel algorithm SA-WMI-PA-SK with respect to SA-WMI-PA (and

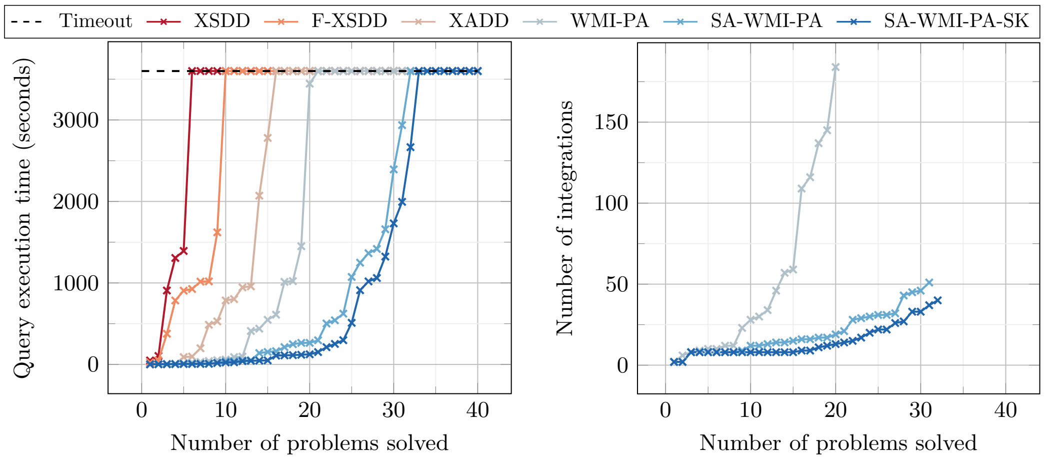


Figure 5: Cactus plots of the synthetic experiments reporting execution time (left) for all the methods; number of integrals (right) for WMI-PA, SA-WMI-PA and SA-WMI-PA-SK.

WMI-PA) will be evident in the next experiment, where more-complex logical structures are involved.

6.2. Inference on Density Estimation Trees

Dataset description. We then consider the problem of computing arbitrary algebraic queries over *Density Estimation Trees* (DETs) [42]. DETs are the density estimation equivalent of decision or regression trees, encoding piecewise constant distributions over mixed continuous/discrete domains. Having only univariate conditions in the internal nodes, DETs support efficient density estimation and marginal inference over single variables. Answering algebraic queries over multiple variables, like $\Pr(X \leq Y)$, requires instead marginalizing over an oblique constraint, which is reduced to:

$$\Pr(X \leq Y) = \frac{\mathsf{WMI}((X \leq Y) \wedge \chi_{DET}, w_{DET})}{\mathsf{WMI}(\chi_{DET}, w_{DET})}. \tag{13}$$

This type of query commonly arises when applying WMI-based inference to probabilistic formal verification tasks, when properties involve multiple continuous variables. The support χ_{DET} is obtained by conjoining the lower and upper bounds of each continuous variable, while w_{DET} encodes the density estimate by means of nested if-then-elses and non-negative constant leaves. Consistently with our previous conference paper [1], we train DETs on a selection of hybrid datasets from the UCI repository [43]. The datasets and resulting models are summarized in §Appendix B. Following the approach of [44], discrete numerical features are relaxed into continuous variables, while n-ary categorical features are one-hot encoded with n binary variables.

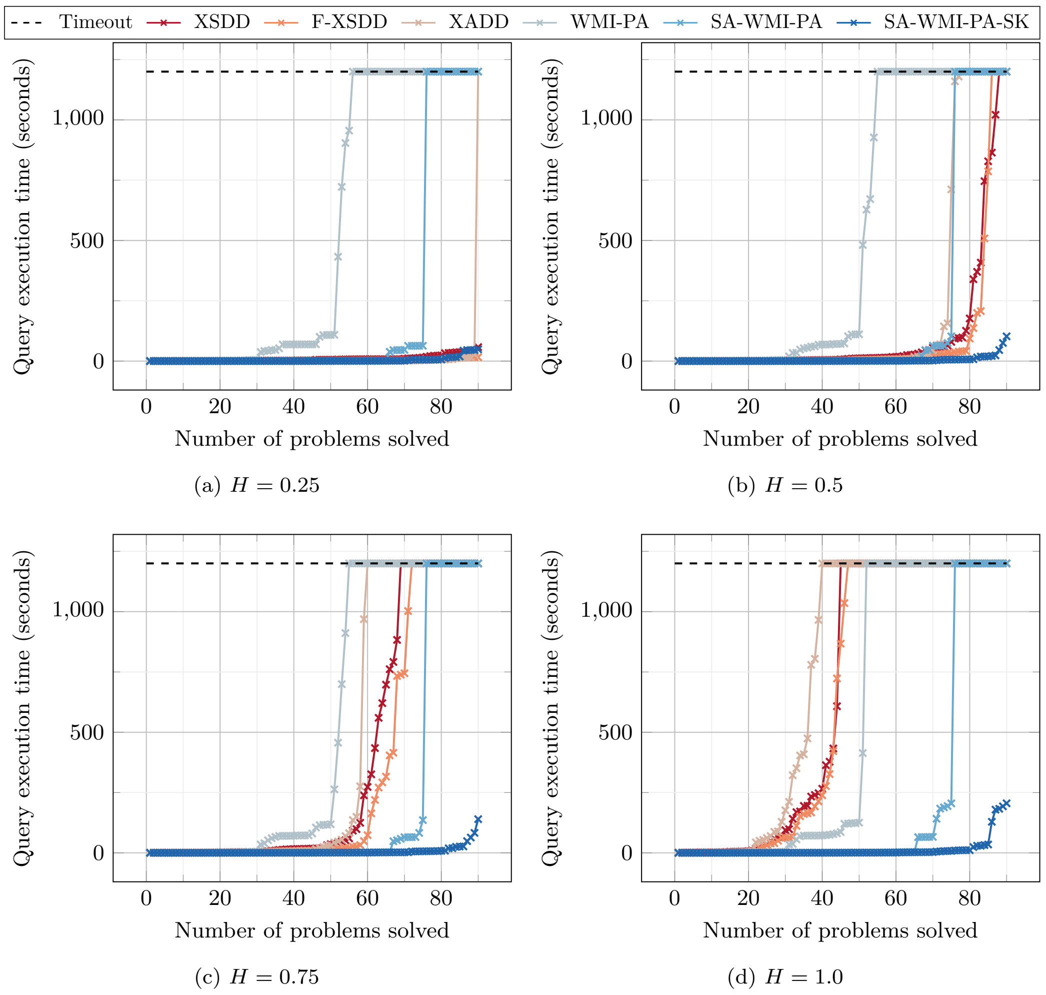
DETs are trained on each dataset using the standard greedy procedure [42] with bounds on the number of instances for each leaf set to $[100, 200]$. We investigate the effect of algebraic dependencies by generating increasingly complex queries involving a number of variables $k = \max(1, \lfloor H \cdot |\mathbf{x}| \rfloor)$, where $H \in [0, 1]$ is the ratio of continuous variables appearing in the inequality. For each value of H, 5 random inequalities involving k variables are generated, for a total of 25 problem instances for each dataset. Given the larger number of problems in this setting, the timeout has been decreased to 1200 seconds.

Results. Figure 6 depicts the runtime of the algorithms for $H \in \{0.25, 0.5, 0.75, 1\}$. Thanks to the absence of arithmetic operations in the internal nodes, DETs are more suited for KC compared to the weight functions used in 6.1. While in the simplest inference cases ($H \leq 0.5$) substantial factorization of the integrals is possible, when the coupling between variables increases this advantage is overweighted by the combinatorial reasoning capabilities of SMT-based approaches.

The efficacy of SA-WMI-PA-SK with respect to both SA-WMI-PA and WMI-PA is particularly evident on this batch of problems. Analyzing the number of integrals computed by PA-based algorithms (Figure 7) we notice a drastic reduction in the number of integrals generated by SA-WMI-PA-SK. This behaviour is expected: the number of Boolean atoms is typically very high, so that we are able to drastically reduce the number of partial assignment over the Boolean component with respect to SA-WMI-PA thanks to the revised enumeration strategy.
Crucially, in contrast with the recent SA-WMI-PA, the novel algorithm is able to compete or even outperform KC-based solvers even when the oblique queries involve a low number of variables.

7. Extending WMI-PA with different integration approaches

A key feature of SMT-based solvers like WMI-PA, which has been inherited by our structure-aware procedures, is that of *being agnostic of the integration procedure used as backend.* Besides developing a novel structure encoding, we implemented a modular approach to PA-based solvers, allowing the use of different integration strategies, including approximate ones. This



(a) $H = 0.25$

(b) $H = 0.5$

(c) $H = 0.75$

(d) $H = 1.0$

Figure 6: Cactus plots representing average query execution times and standard deviation in seconds on a set of DET problems with $H \in \{0.25, 0.5, 0.75, 1\}$.

provides multiple advantages. First, the general integration procedure can be substituted with a specialized one, targeting specific weight families such as piecewise-constants or leveraging structural properties like full factorizability. Second, approximate integration procedures can be applied when the number of continuous variables makes exact integration of $\mathsf{FIUC}^{\mathcal{LRA}}$ functions infeasible. Noticeably, the enumeration of convex polytopes is still exact, with the advantage that the approximation error does not grow multiplicatively. Third, it broadens the use of these solvers beyond the $\mathsf{FIUC}^{\mathcal{LRA}}$ family, to

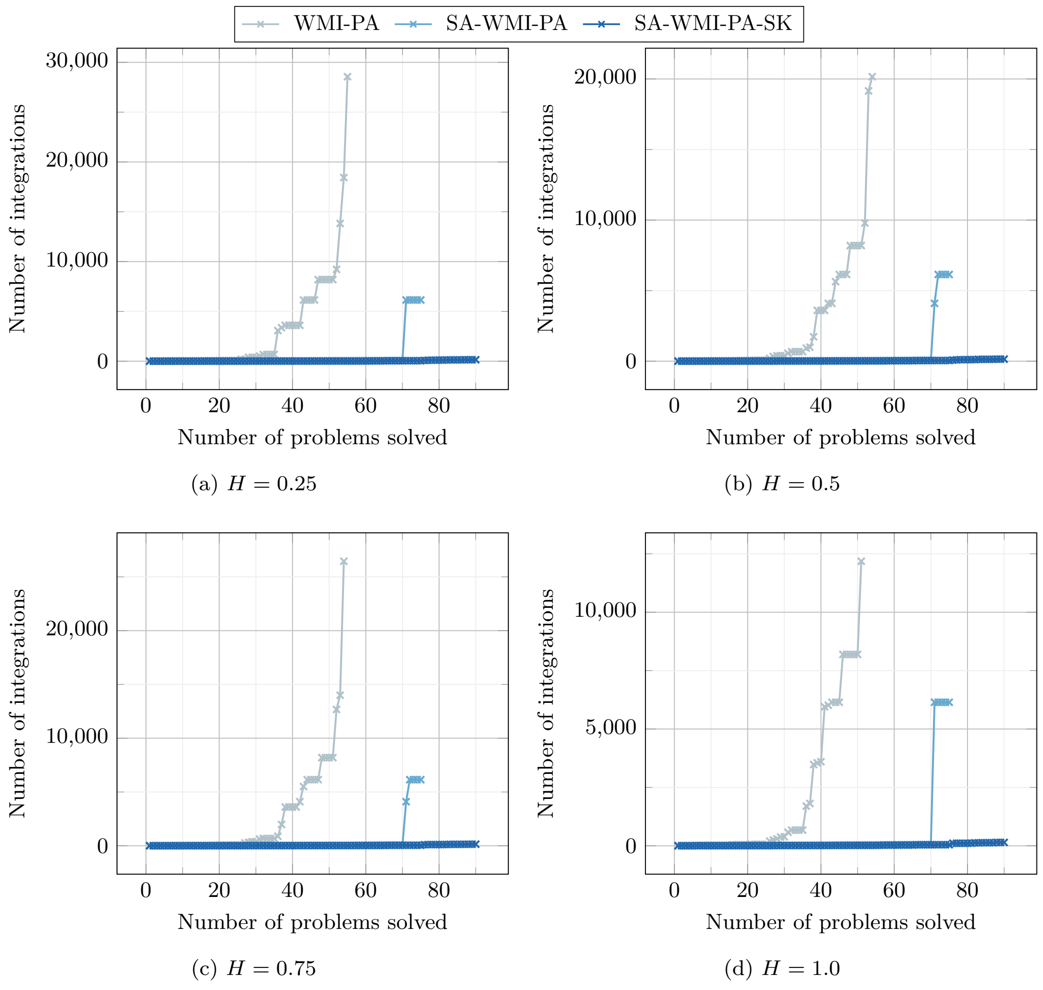


Figure 7: Cactus plots representing the number of computed integrals on a set of DET problems with $H \in \{0.25, 0.5, 0.75, 1\}$.

functions whose integral over convex polytopes can be approximated with a finite sample, like multivariate Gaussian distributions or even the output of a simulation or any other black-box function.

We explored the practical benefits of this approach with a variant of SA-WMI-PA-SK using Monte Carlo (MC) approximate integration [45]:

$$\int_{\mu^{\mathcal{LRA}}} w(\mathbf{x})\ d\mathbf{x} \approx \overbrace{\int_{\mu^{\mathcal{LRA}}} 1\ d\mathbf{x}}^{\mathsf{Vol}(\mu^{\mathcal{LRA}})} \cdot \mathbb{E}_{x \sim \mathcal{U}(\mu^{\mathcal{LRA}})}[w(x)] \tag{14}$$

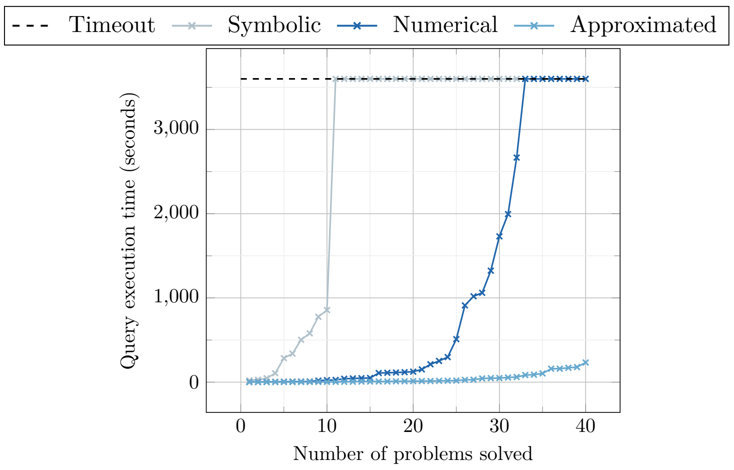


Figure 8: Cactus plots comparing the execution time of SA-WMI-PA-SK using different integration strategies in the synthetic datasets.

In practice, our MC integration procedure is implemented via VolEsti [46], a library for volume approximation and sampling in convex polytopes. VolEsti is used for both approximating $\mathsf{Vol}(\mu^{\mathcal{LRA}})$ and for sampling w. We showcase the resulting solver, dubbed SA-WMI-PA-SK (VolEsti), in two scenarios: (i) scaling up weighted model integration with piecewise polynomial weights and (ii) solving problems with Gaussian weights.

For the first scenario, we consider the problems described in §6.1. Figure 8 displays the computational gain obtained with the above *approximate* MC integrator with respect to LattE's exact *numerical* approach (the one we used in §6). In all problems where an exact solution could be computed, the approximated version SA-WMI-PA-SK(VolEsti) consistently returned an approximation with less than 10% relative error at a fraction of the query execution time. We additionally equipped SA-WMI-PA-SK with an exact *symbolic* integrator. Noticeably, whereas symbolic approaches were shown to be preferable in conjunction with KC-based algorithms (see Figure 3 in [24]) and do not assume convex integration bounds, numerical integration performs substantially better with our approach.

For the second scenario, we consider the task of verifying fairness properties of probabilistic programs, borrowing the setting and examples from FairSquare [47]. In their setting, a (deterministic) *decision program* dec has to decide whether to hire a candidate or not, according to some features. The input of dec is distributed according to a Gaussian probabilistic program, called the *population model* pop, which additionally models sensitive conditions that are not accessed by dec, such as whether an individual belongs to a minority group. The goal is verifying if dec satisfies a *group fairness* criterion under the distribution pop, that is, if the probability of a decision being based on the sensitive condition is low enough:

$$\frac{\Pr(\mathsf{hire} \mid \mathsf{minority})}{\Pr(\mathsf{hire} \mid \neg\mathsf{minority})} > 1 - \varepsilon \tag{15}$$

Similarly to our approach, FairSquare encodes the verification problem in SMT($\mathcal{LRA}$) and reduces probabilistic inference to (unweighted) volume computations. In contrast with our approach, which directly approximates the ratio in (15) by sampling from Gaussians, FairSquare iteratively computes tighter lower and upper bounds by computing increasingly accurate axis-aligned approximations of the true distribution until convergence. We report our results on the same introductory examples used in the Fairsquare paper, motivating future research efforts in the WMI-based verification of probabilistic programs. The detailed description of the programs considered here is reported in Appendix C.

With SA-WMI-PA-SK (VolEsti), we could indeed show that the initial dec does not respect the fairness criterion in (15) with $\varepsilon = 0.1$. This problem was addressed by making dec less sensible to a feature that was directly influenced by the sensitive condition. This second version of the program is fair with high probability, as reported in Figure 9. Computing the ratios with SA-WMI-PA-SK (VolEsti) took negligible time ($< 3s$).

An advantage of the WMI-based approach with respect to FairSquare is that it is not limited to Gaussian priors on pop. Moreover, SMT-based approaches are a natural fit for highly combinatorial problems, such as those arising when dec is a complex classifier learned from data. A current limitation of this approach is that VolEsti does not directly offer statistical guarantees on the relative error. In problems like volume approximation [48] and sampling [49], practical statistical bounds on the error were obtained using diagnostics like the empirical sample size (ESS) or the potential scale reduction factor (PSRF) [50]. The adoption of similar mechanisms in our MC integrator, a fundamental step towards the application of WMI in formal verification, and the evaluation of WMI-based verifiers on more realistic programs, is left for future work.

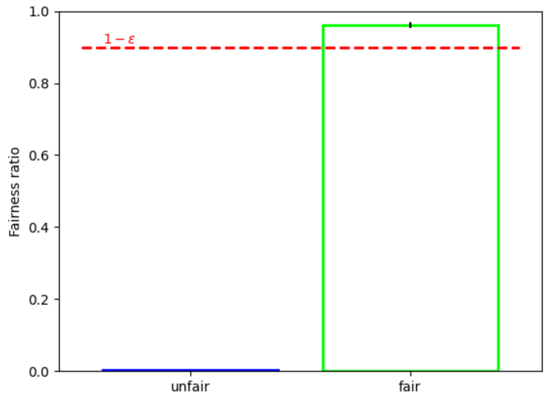


Figure 9: The ratio in Eq. 15 computed for the initial (unfair) `dec` and the modified one. Average and standard deviation over 5 runs are reported, confirming the results reported in the FairSquare paper with high confidence.

8. Conclusion

The difficulty of dealing with densely coupled problems has been a major obstacle to a wider adoption of hybrid probabilistic inference technology beyond academia. In this paper, we made a significant step towards removing this obstacle. Starting from the identification of the limitations of existing solutions for WMI, we proposed a novel algorithm combining predicate abstraction with weight-structure awareness, thus unleashing the full potential of SMT-based technology. An extensive experimental evaluation showed the advantage of the proposed algorithm over current state-of-the-art solutions on both synthetic and real-world problems. These include inference tasks over constrained density functions learned from data, which highlight the potential of the approach as a flexible solution for probabilistic inference in hybrid domains.

Whereas the improvements described in this work drastically reduce the number of integrals required to perform WMI, the integration itself can be a bottleneck for scalability. To deal with this issue, we showed how SMT-based approaches can seamlessly incorporate multiple integration strategies, including symbolic, numeric and approximate ones, allowing to choose the most appropriate strategy for the problem at hand. In particular, e.g., using approximate MC integration techniques could drastically improve performances at the cost of limited losses in terms of precision.

We further showcased the potential utility of our approach on a prototypical application aimed at verifying the fairness of probabilistic programs, as a first step towards the application of WMI technology to the probabilistic verification of hybrid systems and machine learning models.

References

[1] G. Spallitta, G. Masina, P. Morettin, A. Passerini, R. Sebastiani, SMT-based Weighted Model Integration with Structure Awareness, in: Proceedings of the 38th Conference on Uncertainty in Artificial Intelligence, volume 180, 2022, pp. 1876–1885.

[2] D. A. Hensher, K. J. Button, Handbook of Transport Modelling, 2 ed., Emerald Group Publishing Limited, 2007.

[3] S. Thrun, W. Burgard, D. Fox, Probabilistic Robotics, Intelligent Robotics and Autonomous Agents, MIT Press, 2005.

[4] E. A. Lee, Cyber Physical Systems: Design Challenges, in: The 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing, 2008, pp. 363–369.

[5] V. Belle, A. Passerini, G. Van Den Broeck, Probabilistic Inference in Hybrid Domains by Weighted Model Integration, in: Proceedings of the 24th International Joint Conference on Artificial Intelligence, 2015, pp. 2770–2776.

[6] P. Morettin, A. Passerini, R. Sebastiani, Efficient Weighted Model Integration via SMT-Based Predicate Abstraction, in: Proceedings of the 26th International Joint Conference on Artificial Intelligence, 2017, pp. 720–728.

[7] P. Morettin, P. Zuidberg Dos Martires, S. Kolb, A. Passerini, Hybrid Probabilistic Inference with Logical and Algebraic Constraints: A Survey, in: Proceedings of the 30th International Joint Conference on Artificial Intelligence, 2021, pp. 4533–4542.

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

[9] C. W. Barrett, R. Sebastiani, S. A. Seshia, C. Tinelli, Satisfiability Modulo Theories, in: Handbook of Satisfiability, volume 336, 2 ed., 2021, pp. 1267–1329.

[10] T. Sang, F. Bacchus, P. Beame, H. A. Kautz, T. Pitassi, Combining Component Caching and Clause Learning for Effective Model Counting, in: International Conference on Theory and Applications of Satisfiability Testing, 2004, pp. 20–28.

[11] F. Bacchus, S. Dalmao, T. Pitassi, Solving #SAT and Bayesian Inference with Backtracking Search, Journal of Artificial Intelligence Research 34 (2009) 391–442.

[12] V. Belle, G. V. den Broeck, A. Passerini, Component Caching in Hybrid Domains with Piecewise Polynomial Densities, Proceedings of the AAAI Conference on Artificial Intelligence 30 (2016).

[13] Z. Zeng, G. V. den Broeck, Efficient Search-Based Weighted Model Integration, in: Proceedings of the 35th Conference on Uncertainty in Artificial Intelligence, 2020, pp. 175–185.

[14] Z. Zeng, P. Morettin, F. Yan, A. Vergari, G. V. D. Broeck, Scaling up Hybrid Probabilistic Inference with Logical and Arithmetic Constraints via Message Passing, in: Proceedings of the 37th International Conference on Machine Learning, 2020, pp. 10990–11000.

[15] Z. Zeng, P. Morettin, F. Yan, A. Vergari, G. Van den Broeck, Probabilistic Inference with Algebraic Constraints: Theoretical Limits and Practical Approximations, in: Advances in Neural Information Processing Systems, volume 33, 2020, pp. 11564–11575.

[16] R. Abboud, İ. İ. Ceylan, R. Dimitrov, On the Approximability of Weighted Model Integration on DNF Structures, Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning 17 (2020) 828–837.

[17] V. Belle, G. Van den Broeck, A. Passerini, M. Meila, T. Heskes, Hashing-based approximate probabilistic inference in hybrid domains, in: Proceedings of the 31st Conference on Uncertainty in Artificial Intelligence, 2015, pp. 141–150.

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

[19] P. Morettin, A. Passerini, R. Sebastiani, Advanced SMT Techniques for Weighted Model Integration, Artificial Intelligence 275 (2019) 1–27.

[20] S. K. Lahiri, R. Nieuwenhuis, A. Oliveras, SMT Techniques for Fast Predicate Abstraction, in: Computer Aided Verification, 2006, pp. 424–437.

[21] S. Sanner, E. Abbasnejad, Symbolic Variable Elimination for Discrete and Continuous Graphical Models, in: 26th AAAI Conference on Artificial Intelligence, 2012, pp. 1954–1960.

[22] S. Kolb, M. Mladenov, S. Sanner, V. Belle, K. Kersting, Efficient Symbolic Integration for Probabilistic Inference, in: Proceedings of the 27th International Joint Conference on Artificial Intelligence, 2018, pp. 5031–5037.

[23] P. Z. Dos Martires, A. Dries, L. D. Raedt, Exact and Approximate Weighted Model Integration with Probability Density Functions Using Knowledge Compilation, Proceedings of the AAAI Conference on Artificial Intelligence 33 (2019) 7825–7833.

[24] S. Kolb, P. Z. D. Martires, L. D. Raedt, How to Exploit Structure while Solving Weighted Model Integration Problems, in: Proceedings of The 35th Conference on Uncertainty in Artificial Intelligence, 2020, pp. 744–754.

[25] J. Feldstein, V. Belle, Lifted Reasoning Meets Weighted Model Integration, in: Proceedings of the 37th Conference on Uncertainty in Artificial Intelligence, 2021, pp. 322–332.

[26] S. L. Lauritzen, F. Jensen, Stable Local Computation with Conditional Gaussian Distributions, Statistics and Computing 11 (2001) 191–203.

[27] E. Yang, Y. Baker, P. Ravikumar, G. Allen, Z. Liu, Mixed Graphical Models via Exponential Families, in: Proceedings of the 17th International Conference on Artificial Intelligence and Statistics, 2014, pp. 1042–1050.

[28] A. Molina, A. Vergari, N. Di Mauro, S. Natarajan, F. Esposito, K. Kersting, Mixed Sum-Product Networks: A Deep Architecture for Hybrid Domains, Proceedings of the AAAI Conference on Artificial Intelligence 32 (2018) 3828–3835.

[29] H. M. Afshar, S. Sanner, C. Webers, Closed-Form Gibbs Sampling for Graphical Models with Algebraic Constraints, in: 30th AAAI Conference on Artificial Intelligence, 2016, pp. 3287–3293.

[30] V. Gogate, R. Dechter, Approximate inference algorithms for Hybrid Bayesian Networks with discrete constraints, in: Proceedings of the 21st Conference on Uncertainty in Artificial Intelligence, 2005, pp. 209–216.

[31] S. Sanner, K. V. Delgado, L. N. de Barros, Symbolic Dynamic Programming for Discrete and Continuous State MDPs, in: Proceedings of the 17th Conference on Uncertainty in Artificial Intelligence, 2011, pp. 643–652.

[32] R. De Salvo Braz, C. O'Reilly, V. Gogate, R. Dechter, Probabilistic Inference Modulo Theories, in: Proceedings of the 25th International Joint Conference on Artificial Intelligence, 2016, pp. 3591–3599.

[33] R. Sebastiani, Are You Satisfied by This Partial Assignment?, CoRR abs/2003.04225 (2020). `arXiv:2003.04225`.

[34] S. Möhle, R. Sebastiani, A. Biere, Four Flavors of Entailment, in: Theory and Applications of Satisfiability Testing, volume 12178, 2020, pp. 62–71.

[35] G. S. Tseitin, On the Complexity of Derivation in Propositional Calculus, in: Automation of Reasoning, 1983, pp. 466–483.

[36] D. A. Plaisted, S. Greenbaum, A Structure-preserving Clause Form Translation, Journal of Symbolic Computation 2 (1986) 293–304.

[37] A. Cimatti, A. Griggio, B. J. Schaafsma, R. Sebastiani, The MathSAT5 SMT Solver, in: Tools and Algorithms for the Construction and Analysis of Systems, 2013, pp. 93–107.

[38] T. Sang, P. Bearne, H. Kautz, Performing Bayesian Inference by Weighted Model Counting, in: Proceedings of the 20th National Conference on Artificial Intelligence, 2005, pp. 475–481.

[39] J. A. De Loera, R. Hemmecke, J. Tauzer, R. Yoshida, Effective Lattice Point Counting in Rational Convex Polytopes, Journal of Symbolic Computation 38 (2004) 1273–1302.

[40] T. Gehr, S. Misailovic, M. Vechev, PSI: Exact Symbolic Inference for Probabilistic Programs, in: Computer Aided Verification, Lecture Notes in Computer Science, 2016, pp. 62–83.

[41] S. Kolb, P. Morettin, P. Zuidberg Dos Martires, F. Sommavilla, A. Passerini, R. Sebastiani, L. De Raedt, S. Kraus, The Pywmi Framework and Toolbox for Probabilistic Inference using Weighted Model Integration, in: Proceedings of the 28th International Joint Conference on Artificial Intelligence, 2019, pp. 6530–6532.

[42] P. Ram, A. G. Gray, Density Estimation Trees, in: Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2011, pp. 627–635.

[43] D. Dua, C. Graff, UCI Machine Learning Repository, 2017. URL: `http://archive.ics.uci.edu/ml`.

[44] P. Morettin, S. Kolb, S. Teso, A. Passerini, Learning Weighted Model Integration Distributions, Proceedings of the AAAI Conference on Artificial Intelligence 34 (2020) 5224–5231.

[45] M. E. J. Newman, G. T. Barkema, Monte Carlo Methods in Statistical Physics, Clarendon Press, 1999.

[46] A. Chalkis, V. Fisikopoulos, Volesti: Volume Approximation and Sampling for Convex Polytopes in R, The R Journal 13 (2021) 561.

[47] A. Albarghouthi, L. D'Antoni, S. Drews, A. V. Nori, FairSquare: Probabilistic Verification of Program Fairness, Proceedings of the ACM on Programming Languages 1 (2017) 1–30.

[48] A. Chalkis, I. Z. Emiris, V. Fisikopoulos, Practical Volume Estimation by a New Annealing Schedule for Cooling Convex Bodies, CoRR abs/1905.05494 (2019). `arXiv:1905.05494`.

[49] A. Chalkis, V. Fisikopoulos, E. Tsigaridas, H. Zafeiropoulos, Geometric Algorithms for Sampling the Flux Space of Metabolic Networks, arXiv (2021). `arXiv:2012.05503`.

[50] A. Gelman, D. B. Rubin, Inference from Iterative Simulation Using Multiple Sequences, Statistical Science 7 (1992) 457–472.

Appendix A. Implicit Enumeration of a Conditional Skeleton

Here we report in detail the first preliminary approach we proposed in [1].

Appendix A.1. Encoding

The key idea here is to build $\mathsf{sk}(w)$ *implicitly* as a disjunction of partial assignments over $\mathbf{\Psi}$, which we enumerate progressively. To this extent, here we first define $\mathsf{sk}(w) \stackrel{\text{def}}{=} \exists \mathbf{y}.[\![y = w]\!]$ where $[\![y = w]\!]$ is a formula on $\mathbf{A}, \mathbf{x}, \mathbf{y}$ s.t. $\mathbf{y} \stackrel{\text{def}}{=} \{y, y_1, \ldots, y_k\}$ is a set of fresh $\mathcal{LRA}$ variables. Thus, $\mathcal{TA}(\exists \mathbf{x}.(\varphi \wedge \chi \wedge \exists \mathbf{y}.[\![y = w]\!]))$ can be computed as $\mathcal{TA}(\exists \mathbf{x}.\exists \mathbf{y}.(\varphi \wedge \chi \wedge [\![y = w]\!]))$ because the $\mathbf{y}$s do not occur in $\varphi \wedge \chi$, with no need to generate $\mathsf{sk}(w)$ explicitly. The enumeration of $\mathcal{TA}(\exists \mathbf{x}.\exists \mathbf{y}.(\varphi \wedge \chi \wedge [\![y = w]\!]))$ is performed by the very same SMT-based procedure used in [19].

$[\![y = w]\!]$ is obtained by taking $(y = w)$, s.t. y is fresh, and recursively substituting bottom-up every conditional term $(\mathsf{If}\ \psi_i\ \mathsf{Then}\ t_{i1}\ \mathsf{Else}\ t_{i2})$ in it with a fresh variable $y_i \in \mathbf{y}$, adding the definition of $(y_i = (\mathsf{If}\ \psi_i\ \mathsf{Then}\ t_{i1}\ \mathsf{Else}\ t_{i2}))$:

$$(\neg\psi_i \vee y_i = t_{i1}) \wedge (\psi_i \vee y_i = t_{i2}). \tag{A.1}$$

This labelling&rewriting process, which is inspired to Tseitin's labelling CNF-ization [35], guarantees that the size of $[\![y = w]\!]$ is linear w.r.t. that of w. E.g., with (7), $[\![y = w]\!]$ is $(y = \prod_{i=1}^{N} y_i) \wedge \bigwedge_{i=1}^{N}((\neg\psi_i \vee y_i = w_{i1}(\mathbf{x})) \wedge (\psi_i \vee y_i = w_{i2}(\mathbf{x})))$.

One problem with the above definition of $[\![y = w]\!]$ is that it is not a $\mathcal{LRA}$-formula, because w may include multiplications or even transcendental functions out of the conditions $\mathbf{\Psi}$[16], which makes SMT reasoning over it dramatically hard or even undecidable. We notice, however, that when computing $\mathcal{TA}(\exists \mathbf{x}.\exists \mathbf{y}.(\varphi \wedge \chi \wedge [\![y = w]\!]))$ the arithmetic functions and operators occurring in w out of the conditions $\mathbf{\Psi}$ have no role, since the only aspect we need to guarantee for the validity of $\mathsf{sk}(w)$ is that they are indeed functions, so that $\exists y.(y = f(\ldots))$ is always valid. [17] (In substance, during the enumeration

[16]The conditions in $\mathbf{\Psi}$ contain only Boolean atoms or linear terms by definition of $\mathsf{FIUC}^{\mathcal{LRA}}$.

[17]This propagates down to the recursive structure of $[\![y = w]\!]$ because φ is equivalent to $\exists y.(\varphi[t|y] \wedge (y = t))$, and, if y does not occur in ψ, then $\exists y.(y = (\mathsf{If}\ \psi\ \mathsf{Then}\ t_1\ \mathsf{Else}\ t_2))$ is equivalent to $((\mathsf{If}\ \psi\ \mathsf{Then}\ \exists y.(y = t_1)\ \mathsf{Else}\ \exists y.(y = t_2)))$.

Algorithm 4 $\mathsf{Convert}(\mathsf{w}, \mathsf{conds})$
returns $\langle w', \mathsf{defs}, \mathbf{y}\rangle$
w': the term w is rewritten into
conds: the current partial assignment to conditions $\mathbf{\Psi}$,
representing the set of conditions which w depends on
defs: a conjunction of definitions in the form $y_i = w_i$ needed to rewrite w into w'
$\mathbf{y}$: newly-introduced variables labelling if-then-else terms
$f^g, f^{\bowtie}$: uninterpreted function naming the function (operator) g ($\bowtie$)
called as $\langle w', [\![y = w]\!]_{\mathcal{EUF}}, \mathbf{y}\rangle \leftarrow \mathsf{Convert}(w, \emptyset)$

```
1:  if ({w constant or variable}) then
2:      return ⟨w, ⊤, ∅⟩
3:  if (w == (w_1 ⋈ w_2), ⋈ ∈ {+, −, ·}) then
4:      $\langle \mathsf{w}'_i, \mathsf{defs}_i, \mathbf{y}_i\rangle = \mathsf{Convert}(\mathsf{w}_i, \mathsf{conds}),\ i \in \{1, 2\}$
5:      return $\langle f^{\bowtie}(\mathsf{w}'_1, \mathsf{w}'_2), \mathsf{defs}_1 \wedge \mathsf{defs}_2, \mathbf{y}_1 \cup \mathbf{y}_2\rangle$
6:  if $(\mathsf{w} == g(\mathsf{w}_1, \dots, \mathsf{w}_k)$, $g$ unconditioned) then
7:      $\langle \mathsf{w}'_i, \mathsf{defs}_i, \mathbf{y}_i\rangle = \mathsf{Convert}(\mathsf{w}_i, \mathsf{conds}),\ i \in 1, \dots, k$
8:      return $\langle f^g(\mathsf{w}'_1, \dots, \mathsf{w}'_k), \bigwedge_{i=1}^k \mathsf{defs}_i, \cup_{i=1}^k \mathbf{y}_i\rangle$
9:  if $(\mathsf{w} == (\mathsf{If}\ \psi\ \mathsf{Then}\ \mathsf{w}_1\ \mathsf{Else}\ \mathsf{w}_2))$ then
10:     $\langle \mathsf{w}'_1, \mathsf{defs}_1, \mathbf{y}_1\rangle = \mathsf{Convert}(\mathsf{w}_1, \mathsf{conds} \cup \{\ \ \psi\})$
11:     $\langle \mathsf{w}'_2, \mathsf{defs}_2, \mathbf{y}_2\rangle = \mathsf{Convert}(\mathsf{w}_2, \mathsf{conds} \cup \{\neg\psi\})$
12:     let $y$ be a fresh variable
13:     $\mathsf{defs} = \mathsf{defs}_1 \wedge \mathsf{defs}_2\ \wedge$
14:         $(\bigvee_{\psi_i \in \mathsf{conds}} \neg\psi_i \vee \neg\psi \vee (y = \mathsf{w}'_1))\ \wedge$
15:         $(\bigvee_{\psi_i \in \mathsf{conds}} \neg\psi_i \vee \ \ \psi \vee (y = \mathsf{w}'_2))\ \wedge$
16:         $(\bigvee_{\psi_i \in \mathsf{conds}} \neg\psi_i \vee \neg(y = \mathsf{w}'_1) \vee \neg(y = \mathsf{w}'_2))$
17:     $\mathbf{y} = \mathbf{y}_1 \cup \mathbf{y}_2 \cup \{y\}$
18:     return $\langle y, \mathsf{defs}, \mathbf{y}\rangle$
```

we are interested only in the truth values of the conditions $\mathbf{\Psi}$ in μ which make $w_{[\mu]}$ $\mathsf{FI}^{\mathcal{LRA}}$, regardless of the actual values of $w_{[\mu]}$). Therefore, we can safely substitute condition-less arithmetic functions and operators with fresh *uninterpreted function symbols*, obtaining a $\mathcal{LRA} \cup \mathcal{EUF}$-formula $[\![y = w]\!]_{\mathcal{EUF}}$, which is relatively easy to solve by standard SMT solvers [9]. It is easy to see that a partial assignment μ evaluating $[\![y = w]\!]$ to true is satisfiable in the theory iff its corresponding assignment $\mu_{\mathcal{EUF}}$ is $\mathcal{LRA} \cup \mathcal{EUF}$-satisfiable.[18] Thus, we can modify the enumeration procedure into $\mathcal{TA}(\exists \mathbf{x}.\exists \mathbf{y}.(\varphi \wedge \chi \wedge [\![y = w]\!]_{\mathcal{EUF}}))$.

Finally, we enforce the fact that the two branches of an if-then-else are alternative by adding a mutual-exclusion constraint $\neg(y_i = t_{i1}) \vee \neg(y_i = t_{i2})$ to (A.1), so that the choice of the function is univocally associated to the set of decisions on the ψ_is. The procedure generating $[\![y = w]\!]_{\mathcal{EUF}}$ is described in detail in Algorithm 4.

Example 7. *Consider the problem in Example 1. Figure A.10 shows the relabelling process applied to the weight function w. The resulting $[\![y = w]\!]_{\mathcal{EUF}}$ formula is:*

$$\begin{array}{lll}
& [\![y = w]\!](\mathbf{x} \cup \mathbf{y}, \mathbf{A}) \overset{def}{=} & \\
& (\neg A_1 \vee \neg(x_1 \geq 1) \vee \neg(x_2 \geq 1) & \vee\ (y_1 = f_{11}(\mathbf{x}))) \\
\wedge & (\neg A_1 \vee \neg(x_1 \geq 1) \vee \ (x_2 \geq 1) & \vee\ (y_1 = f_{12}(\mathbf{x}))) \\
\wedge & (\neg A_1 \vee \neg(x_1 \geq 1) \vee \neg(y_1 = f_{11}(\mathbf{x})) & \vee \neg(y_1 = f_{12}(\mathbf{x}))) \\
\wedge & (\neg A_1 \vee \ (x_1 \geq 1) \vee \neg(x_2 \geq 2) & \vee\ (y_2 = f_{21}(\mathbf{x}))) \\
\wedge & (\neg A_1 \vee \ (x_1 \geq 1) \vee \ (x_2 \geq 2) & \vee\ (y_2 = f_{22}(\mathbf{x}))) \\
\wedge & (\neg A_1 \vee \ (x_1 \geq 1) \vee \neg(y_2 = f_{21}(\mathbf{x})) & \vee \neg(y_2 = f_{22}(\mathbf{x}))) \\
\wedge & (\neg A_1 \vee \neg(x_1 \geq 1) & \vee\ (y_3 = y_1)) \\
\wedge & (\neg A_1 \vee \ (x_1 \geq 1) & \vee\ (y_3 = y_2)) \\
\wedge & (\neg A_1 \vee \neg(y_3 = y_1) \vee \neg(y_3 = y_2)) & \\
\wedge & (\ A_1 \vee \neg A_2 & \vee\ (y_4 = f_3(\mathbf{x}))) \\
\wedge & (\ A_1 \vee \ A_2 & \vee\ (y_4 = f_4(\mathbf{x}))) \\
\wedge & (\ A_1 \vee \neg(y_4 = f_3(\mathbf{x})) \vee \neg(y_4 = f_4(\mathbf{x}))) & \\
\wedge & (\neg A_1 & \vee\ (y_5 = y_3)) \\
\wedge & (\ A_1 & \vee\ (y_5 = y_4)) \\
\wedge & (\neg(y_5 = y_3) \vee \neg(y_5 = y_4)) & \\
\wedge & (\ (y = y_5)) &
\end{array}$$

Figure A.11 illustrates a possible enumeration process. The algorithm enumerates partial assignments satisfying $\varphi \wedge \chi \wedge [\![y = w]\!]_{\mathcal{EUF}}$, restricted on the conditions $\Psi \overset{def}{=} \{A_1, A_2, (x_1 \geq 1), (x_2 \geq 1), (x_2 \geq 2)\}$, which is equivalent to enumerate $\mathcal{TA}(\exists \mathbf{y}.(\varphi \wedge \chi \wedge [\![y = w]\!]_{\mathcal{EUF}}))$. Assuming that the enumeration

[18]This boils down to the fact that y occurs only in the top equation and as such it is free to assume arbitrary values, and that all arithmetic functions are total in the domain so that, for every possible values of $\mathbf{x}$ a value for $\mathbf{y}$ always exists iff there exists in the $\mathcal{EUF}$ version.

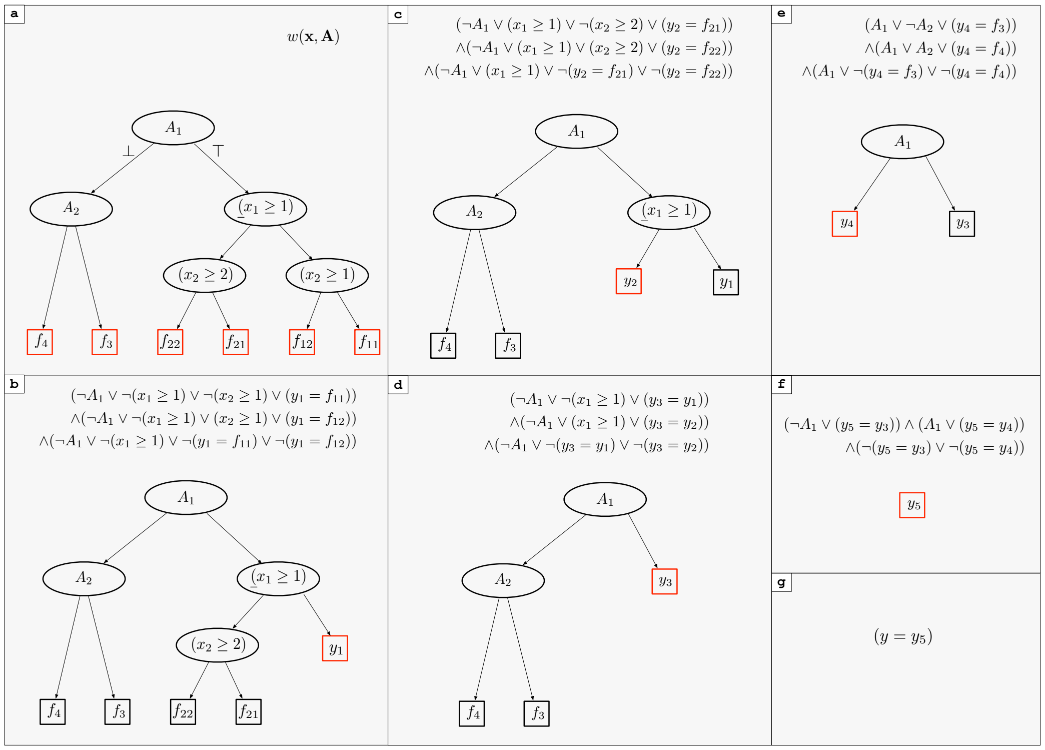
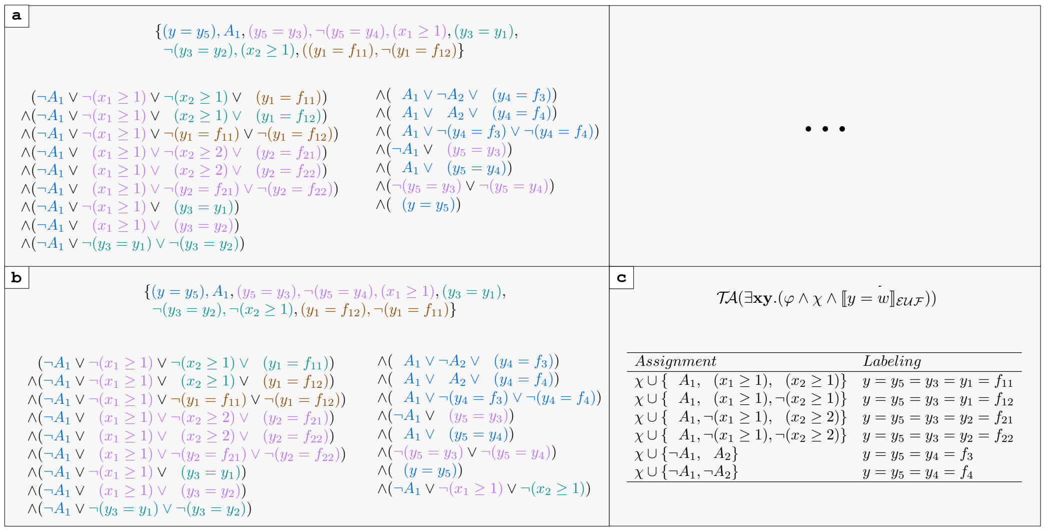

a
$w(\mathbf{x}, \mathbf{A})$
A_1
$\perp$
$\top$
A_2
$(x_1 \geq 1)$
$(x_2 \geq 2)$
$(x_2 \geq 1)$
f_4 f_3 f_{22} f_{21} f_{12} f_{11}
b
$(\neg A_1 \vee \neg(x_1 \geq 1) \vee \neg(x_2 \geq 1) \vee (y_1 = f_{11}))$
$\wedge(\neg A_1 \vee \neg(x_1 \geq 1) \vee (x_2 \geq 1) \vee (y_1 = f_{12}))$
$\wedge(\neg A_1 \vee \neg(x_1 \geq 1) \vee \neg(y_1 = f_{11}) \vee \neg(y_1 = f_{12}))$
A_1 A_2 $(x_1 \geq 1)$ $(x_2 \geq 2)$ y_1
f_4 f_3 f_{22} f_{21}
c
$(\neg A_1 \vee (x_1 \geq 1) \vee \neg(x_2 \geq 2) \vee (y_2 = f_{21}))$
$\wedge(\neg A_1 \vee (x_1 \geq 1) \vee (x_2 \geq 2) \vee (y_2 = f_{22}))$
$\wedge(\neg A_1 \vee (x_1 \geq 1) \vee \neg(y_2 = f_{21}) \vee \neg(y_2 = f_{22}))$
A_1 A_2 $(x_1 \geq 1)$ y_2 y_1
f_4 f_3
d
$(\neg A_1 \vee \neg(x_1 \geq 1) \vee (y_3 = y_1))$
$\wedge(\neg A_1 \vee (x_1 \geq 1) \vee (y_3 = y_2))$
$\wedge(\neg A_1 \vee \neg(y_3 = y_1) \vee \neg(y_3 = y_2))$
A_1 A_2 y_3
f_4 f_3
e
$(A_1 \vee \neg A_2 \vee (y_4 = f_3))$
$\wedge(A_1 \vee A_2 \vee (y_4 = f_4))$
$\wedge(A_1 \vee \neg(y_4 = f_3) \vee \neg(y_4 = f_4))$
A_1 y_4 y_3
f
$(\neg A_1 \vee (y_5 = y_3)) \wedge (A_1 \vee (y_5 = y_4))$
$\wedge(\neg(y_5 = y_3) \vee \neg(y_5 = y_4))$
y_5
g
$(y = y_5)$

Figure A.10: Example of bottom-up procedure for computing the relabelling function $[\![y = w]\!]_{\mathcal{EUF}}$. (**a**) Replacement of $\mathsf{FI}^{\mathcal{LRA}}$ weight functions (the leaves of the tree, highlighted in red) with $\mathcal{EUF}$ function symbols (we dropped the dependency on x for compactness). (**b-g**) Sequence of relabelling steps. At each step, a conditional term is replaced by a fresh $\mathcal{LRA}$ variable y_i. The encoding of the variable in shown in the upper part, while the lower part shows the weight function with the branch of the conditional term replaced with y_i (highlighted in red). The last step consists in renaming the top variable as y, so that $y = w(\mathbf{x}, \mathbf{A})$. The relabelling function $[\![y = w]\!]_{\mathcal{EUF}}$ is simply the conjunction of the encodings in the different steps.

procedure picks nondeterministic choices following the order of the above set [19] *and assigning positive values first, then in the first branch the following satisfying partial assignment is generated, in order:* [20]

[19]Like in Algorithm 5, we pick Boolean conditions first.

[20]Here nondeterministic choices are underlined. The atoms in χ are assigned deterministically.



a

$$\{(y = y_5), A_1, (y_5 = y_3), \neg(y_5 = y_4), (x_1 \geq 1), (y_3 = y_1), \neg(y_3 = y_2), (x_2 \geq 1), ((y_1 = f_{11}), \neg(y_1 = f_{12})\}$$

$$\begin{array}{ll}
(\neg A_1 \vee \neg(x_1 \geq 1) \vee \neg(x_2 \geq 1) \vee (y_1 = f_{11})) & \wedge(A_1 \vee \neg A_2 \vee (y_4 = f_3)) \\
\wedge(\neg A_1 \vee \neg(x_1 \geq 1) \vee (x_2 \geq 1) \vee (y_1 = f_{12})) & \wedge(A_1 \vee A_2 \vee (y_4 = f_4)) \\
\wedge(\neg A_1 \vee \neg(x_1 \geq 1) \vee \neg(y_1 = f_{11}) \vee \neg(y_1 = f_{12})) & \wedge(A_1 \vee \neg(y_4 = f_3) \vee \neg(y_4 = f_4)) \\
\wedge(\neg A_1 \vee (x_1 \geq 1) \vee \neg(x_2 \geq 2) \vee (y_2 = f_{21})) & \wedge(\neg A_1 \vee (y_5 = y_3)) \\
\wedge(\neg A_1 \vee (x_1 \geq 1) \vee (x_2 \geq 2) \vee (y_2 = f_{22})) & \wedge(A_1 \vee (y_5 = y_4)) \\
\wedge(\neg A_1 \vee (x_1 \geq 1) \vee \neg(y_2 = f_{21}) \vee \neg(y_2 = f_{22})) & \wedge(\neg(y_5 = y_3) \vee \neg(y_5 = y_4)) \\
\wedge(\neg A_1 \vee \neg(x_1 \geq 1) \vee (y_3 = y_1)) & \wedge((y = y_5)) \\
\wedge(\neg A_1 \vee (x_1 \geq 1) \vee (y_3 = y_2)) & \\
\wedge(\neg A_1 \vee \neg(y_3 = y_1) \vee \neg(y_3 = y_2)) &
\end{array}$$

$$\cdots$$

b

$$\{(y = y_5), A_1, (y_5 = y_3), \neg(y_5 = y_4), (x_1 \geq 1), (y_3 = y_1), \neg(y_3 = y_2), \neg(x_2 \geq 1), (y_1 = f_{12}), \neg(y_1 = f_{11})\}$$

$$\begin{array}{ll}
(\neg A_1 \vee \neg(x_1 \geq 1) \vee \neg(x_2 \geq 1) \vee (y_1 = f_{11})) & \wedge(A_1 \vee \neg A_2 \vee (y_4 = f_3)) \\
\wedge(\neg A_1 \vee \neg(x_1 \geq 1) \vee (x_2 \geq 1) \vee (y_1 = f_{12})) & \wedge(A_1 \vee A_2 \vee (y_4 = f_4)) \\
\wedge(\neg A_1 \vee \neg(x_1 \geq 1) \vee \neg(y_1 = f_{11}) \vee \neg(y_1 = f_{12})) & \wedge(A_1 \vee \neg(y_4 = f_3) \vee \neg(y_4 = f_4)) \\
\wedge(\neg A_1 \vee (x_1 \geq 1) \vee \neg(x_2 \geq 2) \vee (y_2 = f_{21})) & \wedge(\neg A_1 \vee (y_5 = y_3)) \\
\wedge(\neg A_1 \vee (x_1 \geq 1) \vee (x_2 \geq 2) \vee (y_2 = f_{22})) & \wedge(A_1 \vee (y_5 = y_4)) \\
\wedge(\neg A_1 \vee (x_1 \geq 1) \vee \neg(y_2 = f_{21}) \vee \neg(y_2 = f_{22})) & \wedge(\neg(y_5 = y_3) \vee \neg(y_5 = y_4)) \\
\wedge(\neg A_1 \vee \neg(x_1 \geq 1) \vee (y_3 = y_1)) & \wedge((y = y_5)) \\
\wedge(\neg A_1 \vee (x_1 \geq 1) \vee (y_3 = y_2)) & \wedge(\neg A_1 \vee \neg(x_1 \geq 1) \vee \neg(x_2 \geq 1)) \\
\wedge(\neg A_1 \vee \neg(y_3 = y_1) \vee \neg(y_3 = y_2)) &
\end{array}$$

c

$$\mathcal{TA}(\exists \mathbf{xy}.(\varphi \wedge \chi \wedge [\![y = \dot{w}]\!]_{\mathcal{EUF}}))$$

Assignment	*Labeling*
$\chi \cup \{ A_1, (x_1 \geq 1), (x_2 \geq 1)\}$	$y = y_5 = y_3 = y_1 = f_{11}$
$\chi \cup \{ A_1, (x_1 \geq 1), \neg(x_2 \geq 1)\}$	$y = y_5 = y_3 = y_1 = f_{12}$
$\chi \cup \{ A_1, \neg(x_1 \geq 1), (x_2 \geq 2)\}$	$y = y_5 = y_3 = y_2 = f_{21}$
$\chi \cup \{ A_1, \neg(x_1 \geq 1), \neg(x_2 \geq 2)\}$	$y = y_5 = y_3 = y_2 = f_{22}$
$\chi \cup \{\neg A_1, A_2\}$	$y = y_5 = y_4 = f_3$
$\chi \cup \{\neg A_1, \neg A_2\}$	$y = y_5 = y_4 = f_4$

Figure A.11: Example of structure-aware enumeration performed by SA-WMI-PA on the problem in Example 1. (**a**) Generation of the first assignment. The assignment is on top, while the bottom part shows the $[\![y = w]\!]_{\mathcal{EUF}}$ formula. Colors indicate the progression of the generation, in terms of atoms added (top) and parts of the formula to be removed as a consequence (bottom). For the sake of simplicity, all atoms until the next atom in $\Psi \overset{\text{def}}{=} \{A_1, A_2, (x_1 \geq 1), (x_2 \geq 1), (x_2 \geq 2)\}$ (if any) are given the same colour. (**b**) Generation of the second assignment. Note how the $[\![y = w]\!]_{\mathcal{EUF}}$ formula is enriched with the blocking clause $\neg A_1 \vee \neg(x_1 \geq 1) \vee \neg(x_2 \geq 1)$ preventing the first assignment to be generated again. (**c**) Final result of the enumeration (which contains six assignments in total). The partial assignments are obtained by restricting the generated assignments on the conditions in Ψ (and combining them with the atoms of χ, which here are assigned deterministically.). For each assignment, the corresponding chain of equivalences of the **y**s with the identified leaf $\mathsf{FI}^{\mathcal{LRA}}$ function is displayed.

$$\begin{aligned}
&\chi \cup \{(y = y_5), \underline{A_1}, (y_5 = y_3), \neg(y_5 = y_4), \underline{(x_1 \geq 1)}, \\
&(y_3 = y_1), \neg(y_3 = y_2), \underline{(x_2 \geq 1)}, (y_1 = f_{11}(\mathbf{x})), \\
&\neg(y_1 = f_{12}(\mathbf{x}))\}.
\end{aligned}$$

(Notice that, following the chains of true equalities, we have $y = y_5 = y_3 = y_1 = f_{11}(\mathbf{x})$.) Then the SMT solver extracts from it the subset $\{A_1, (x_1 \geq 1), (x_2 \geq 1)\}$ restricted to the conditions in Ψ. Then the blocking clause $\neg A_1 \vee \neg(x_1 \geq 1) \vee \neg(x_2 \geq 1)$ is added to the formula, which prevents to enumerate the same subset again. This forces the algorithm to backtrack and generate:

$$\begin{aligned}&\chi \cup \{(y=y_5), \underline{A_1}, (y_5=y_3), \neg(y_5=y_4), \underline{(x_1 \geq 1)},\\ &(y_3=y_1), \neg(y_3=y_2), \neg(x_2 \geq 1), (y_1=f_{12}(\mathbf{x})),\\ &\neg(y_1=f_{11}(\mathbf{x}))\}.\end{aligned}$$

producing the assignment: $\{A_1, (x_1 \geq 1), \neg(x_2 \geq 1)\}$. [21]

Overall, the algorithm enumerates the following ordered collection of partial assignments restricted to $Atoms(\varphi \wedge \chi) \cup \mathbf{\Psi}$*:*

$$\begin{array}{ll}\chi \cup \{\ \ A_1,\ \ (x_1 \geq 1),\ \ (x_2 \geq 1)\}, & //y = ... = f_{11}(\mathbf{x})\\ \chi \cup \{\ \ A_1,\ \ (x_1 \geq 1), \neg(x_2 \geq 1)\}, & //y = ... = f_{12}(\mathbf{x})\\ \chi \cup \{\ \ A_1, \neg(x_1 \geq 1),\ \ (x_2 \geq 2)\}, & //y = ... = f_{21}(\mathbf{x})\\ \chi \cup \{\ \ A_1, \neg(x_1 \geq 1), \neg(x_2 \geq 2)\}, & //y = ... = f_{22}(\mathbf{x})\\ \chi \cup \{\neg A_1,\ \ A_2\}, & //y = ... = f_3(\mathbf{x})\\ \chi \cup \{\neg A_1, \neg A_2\} & //y = ... = f_4(\mathbf{x})\end{array}$$

which correspond to the six integrals of Example 1. Note that according to (1) *the first four integrals have to be multiplied by 2, because the partial assignment* $\{A_1\}$ *covers two total assignments* $\{A_1, A_2\}$ *and* $\{A_1, \neg A_2\}$*. Notice also that the disjunction of the six partial assignments:*

$$(A_1 \wedge (x_1 \geq 1) \wedge (x_2 \geq 1)) \vee ... \vee (\neg A_1 \wedge \neg A_2),$$

matches the definition of $\mathsf{sk}(w)$*, which we have computed by progressive enumeration rather than encoded a priori.* $\diamond$

Based on previous ideas, we develop SA-WMI-PA, a novel "weight-structure aware" variant of WMI-PA. The pseudocode of SA-WMI-PA is reported in Algorithm 5.
As with WMI-PA, we enumerate the assignments in two main steps: in the first loop (lines 4-8) we generate a set $\mathcal{M}^{\mathbf{A}^*}$ of partial assignments $\mu^{\mathbf{A}^*}$ over the Boolean atoms $\mathbf{A}$, s.t. $\varphi^{**}_{[\mu^{\mathbf{A}^*}]}$ is $\mathcal{LRA}$-satisfiable and does not anymore contain Boolean atoms. In Example 7 $\mathcal{M}^{\mathbf{A}^*} \overset{\text{def}}{=} \{\{A_1\}, \{\neg A_1, A_2\}, \{\neg A_1, \neg A_2\}\}$. In the second loop (lines 12-20), for each $\mu^{\mathbf{A}^*}$ in $\mathcal{M}^{\mathbf{A}^*}$ we enumerate the set $\mathcal{M}^{\mathcal{LRA}}$ of $\mathcal{LRA}$-satisfiable partial assignments satisfying $\varphi^{**}_{[\mu^{\mathbf{A}^*}]}$ (that is, on $\mathcal{LRA}$ atoms in $Atoms(\varphi \wedge \chi) \cup \mathbf{\Psi}$), we compute the integral $\mathsf{WMI}_{\mathsf{nb}}(\mu^{\mathcal{LRA}}, w_{[\mu^{\mathbf{A}^*}]}|\mathbf{x})$, multiply it by the $2^{|\mathbf{A} \setminus \mu^{\mathbf{A}^*}|}$ factor and add it to the result. In Example 7, if e.g. $\mu^{\mathbf{A}^*} = \{A_1\}$, $\mathcal{TA}(\exists \mathbf{x}.\exists \mathbf{y}.\varphi^{**}_{[\mu^{\mathbf{A}^*}]})$ computes the four partial assignments $\{\chi \cup \{(x_1 \geq 1), (x_2 \geq 1)\}, \ldots, \chi \cup \{\neg(x_1 \geq 1), \neg(x_2 \geq 2)\}\}$.

[21]We refer the reader to [20] for more details on the SMT-based enumeration algorithm.

Algorithm 5 SA-WMI-PA$(\varphi, w, \mathbf{x}, \mathbf{A})$

```
1: $\mathcal{M}^{\mathbf{A}^*} \leftarrow \emptyset$; $vol \leftarrow 0$
2: $\langle w', [\![y = w]\!]_{\mathcal{EUF}}, \mathbf{y}\rangle \leftarrow \mathsf{Convert}(w, \emptyset)$
3: $\varphi^{**} \leftarrow \varphi \wedge \chi \wedge [\![y = w]\!]_{\mathcal{EUF}}$
4: $\mathcal{M}^{\mathbf{A}} \leftarrow \mathcal{TA}(\exists \mathbf{x}.\exists \mathbf{y}.\varphi^{**})$
5: for $\mu^{\mathbf{A}} \in \mathcal{M}^{\mathbf{A}}$ do
6:     $\mathsf{Simplify}(\varphi^{**}_{[\mu^{\mathbf{A}}]})$
7:     if $\varphi^{**}_{[\mu^{\mathbf{A}}]}$ does not contain Boolean atoms then
8:         $\mathcal{M}^{\mathbf{A}^*} \leftarrow \mathcal{M}^{\mathbf{A}^*} \cup \{\mu^{\mathbf{A}}\}$
9:     else
10:        for $\mu^{\mathbf{A}}_{residual} \in \mathcal{TTA}(\exists \mathbf{x}.\exists \mathbf{y}.\varphi^{**}_{[\mu^{\mathbf{A}}]})$ do
11:            $\mathcal{M}^{\mathbf{A}^*} \leftarrow \mathcal{M}^{\mathbf{A}^*} \cup \{\mu^{\mathbf{A}} \wedge \mu^{\mathbf{A}}_{residual}\}$
12: for $\mu^{\mathbf{A}^*} \in \mathcal{M}^{\mathbf{A}^*}$ do
13:     $k \leftarrow |\mathbf{A} \setminus \mu^{\mathbf{A}^*}|$
14:     $\mathsf{Simplify}(\varphi^{**}_{[\mu^{\mathbf{A}^*}]})$
15:     if $\mathsf{LiteralConjunction}(\varphi^{**}_{[\mu^{\mathbf{A}^*}]})$ then
16:         $vol \leftarrow vol + 2^k \cdot \mathsf{WMI}_{\mathsf{nb}}(\varphi^{**}_{[\mu^{\mathbf{A}^*}]}, w_{[\mu^{\mathbf{A}^*}]}|\mathbf{x})$
17:     else
18:         $\mathcal{M}^{\mathcal{LRA}} \leftarrow \mathcal{TA}(\exists \mathbf{y}.\varphi^{**}_{[\mu^{\mathbf{A}^*}]})$
19:         for $\mu^{\mathcal{LRA}} \in \mathcal{M}^{\mathcal{LRA}}$ do
20:             $vol \leftarrow vol + 2^k \cdot \mathsf{WMI}_{\mathsf{nb}}(\mu^{\mathcal{LRA}}, w_{[\mu^{\mathbf{A}^*}]}|\mathbf{x})$
21: return $vol$
```

In detail, in line 3 we extend $\varphi \wedge \chi$ with $[\![y = w]\!]_{\mathcal{EUF}}$ to provide structural awareness. (We recall that, unlike with WMI-PA, we do not label $\mathcal{LRA}$ conditions with fresh Boolean atoms $\mathbf{B}$.) Next, in line 4 we perform $\mathcal{TA}(\exists \mathbf{x}.\exists \mathbf{y}.\varphi^{**})$ to obtain a set $\mathcal{M}^{\mathbf{A}}$ of partial assignments restricted to Boolean atoms $\mathbf{A}$. Then, for each assignment $\mu^{\mathbf{A}} \in \mathcal{M}^{\mathbf{A}}$ we build the (simplified) residual $\varphi^{**}_{[\mu^{\mathbf{A}}]}$. Since $\mu^{\mathbf{A}}$ is partial, $\varphi^{**}_{[\mu^{\mathbf{A}}]}$ is not guaranteed to be free of Boolean atoms $\mathbf{A}$, as shown in Example 8. If this is the case, we simply add $\mu^{\mathbf{A}}$ to $\mathcal{M}^{\mathbf{A}^*}$, otherwise, we invoke $\mathcal{TTA}(\exists \mathbf{x}.\exists \mathbf{y}.\varphi^{**}_{[\mu^{\mathbf{A}}]})$ to assign the remaining atoms and conjoin each assignment $\mu^{\mathbf{A}}_{residual}$ to $\mu^{\mathbf{A}}$, ensuring that the residual now contains only $\mathcal{LRA}$ atoms (lines 5- 11). The second loop (lines 12-20) resembles the main loop in WMI-PA, with the only relevant difference that, since $\mu^{\mathbf{A}^*}$ is partial, the integral is multiplied by a $2^{|\mathbf{A} \setminus \mu^{\mathbf{A}^*}|}$ factor, as in (8).

Notice that in general the assignments $\mu^{\mathbf{A}^*}$ are partial even if the steps in lines 10-11 are executed; the set of residual Boolean atoms in $\varphi^{**}_{[\mu^{\mathbf{A}}]}$ are a (possibly much smaller) subset of $\mathbf{A} \setminus \mu^{\mathbf{A}}$ because some of them do not occur anymore in $\varphi^{**}_{[\mu^{\mathbf{A}}]}$ after the simplification, as shown in the following example.

Example 8. *Let* $\varphi \overset{def}{=} (A_1 \vee A_2 \vee A_3) \wedge (\neg A_1 \vee A_2 \vee (x \geq 1)) \wedge (\neg A_2 \vee (x \geq 2)) \wedge (\neg A_3 \vee (x \leq 3))$, $\chi \overset{def}{=} (x_1 \geq 0) \wedge (x_1 \leq 4)$ *and* $w(\mathbf{x}, \mathbf{A}) \overset{def}{=} 1.0$.
Since w is constant, we have that $\varphi^{**} \overset{def}{=} \varphi \wedge \chi$. *Suppose* $\mathcal{TA}(\exists \mathbf{x}.\varphi^{**})$ *finds the partial assignment* $\{(x \geq 0), (x \leq 4), A_2, (x \geq 1), (x \geq 2), (x \leq 3)\}$, *whose projected version is* $\mu^{\mathbf{A}} \overset{def}{=} \{A_2\}$ *(line 4). Then* $\varphi^{**}_{[\mu^{\mathbf{A}}]}$ *reduces to* $(\neg A_3 \vee (x \leq 3))$, *so that* $\mathcal{M}^{\mathbf{A}^*}$ *is* $\{\{A_2, A_3\}, \{A_2, \neg A_3\}\}$, *avoiding branching on* A_1. $\diamond$

We stress the fact that in our actual implementation, like with that of WMI-PA, the potentially-large sets $\mathcal{M}^{\mathbf{A}^*}$ and $\mathcal{M}^{\mathcal{LRA}}$ are not generated explicitly. Rather, their elements are generated, integrated and then dropped one-by-one, so that to avoid space blow-up.
We highlight two main differences w.r.t. WMI-PA. First, unlike with WMI-PA, the generated assignments $\mu^{\mathbf{A}}$ on $\mathbf{A}$ are partial, each representing $2^{|\mathbf{A} \setminus \mu^{\mathbf{A}}|}$ total ones. Second, the assignments on (non-Boolean) conditions $\mathbf{\Psi}$ inside the $\mu^{\mathcal{LRA}}$s are also partial, whereas with WMI-PA the assignments to the $\mathbf{B}$s are total. This may drastically reduce the number of integrals to compute, as empirically demonstrated in the next section.

Appendix B. Description of the datasets for the DET experiments

Table B.2 describes the datasets used to train the DETs considered in 6.2. For each dataset, we report the number of Boolean and continuous variables, the size of the training/validation splits and the size of the resulting DET in terms of number of internal nodes (conditions).

Appendix C. Details on the program fairness verification task

The population model pop (Algorithm 6) is a probabilistic program with Gaussian priors, modelling a distribution over real-valued variables: (i) *ethnicity*; *colRank* (rank of the college attended by the person, the lower, the better); (iii) *yExp* (years of work experience). In this population model, *colRank* is influenced by *ethnicity*. The decision program dec (Algorithm 7) takes as input *colRank* and *yExp* and outputs hire. Notice that, while dec

Dataset	$\lvert\mathbf{A}\rvert$	$\lvert\mathbf{x}\rvert$	# Train	# Valid	Size
balance-scale	3	4	1875	205	5
iris	3	4	450	50	3
cars	33	7	2115	234	14
diabetes	1	8	4149	459	28
breast-cancer	12	4	1650	180	13
glass2	1	9	970	100	7
glass	7	9	1280	140	7
breast	1	10	4521	495	31
solar	25	3	2522	273	13
cleve	17	6	2492	266	18
hepatitis	14	6	940	100	5
heart	3	11	2268	252	16
australian	34	6	6210	690	46
crx	38	6	6688	736	46
german	41	10	12600	1386	93
german-org	13	12	15000	1650	108
auto	56	16	2522	260	18
anneal-U	74	9	21021	2301	144

Table B.2: UCI datasets considered in our experiments. For each dataset, we report the number of Boolean and continuous variables, number of training/validation instances and the size of the resulting Density Estimation Tree in terms of number of conditions.

doesn't have access to the sensible condition minority $= \mathit{ethnicity} > 10$, the program is still unfair (Figure 9) due to the high importance attributed to *colRank*. This can be amended by modifying the second condition to $(5 \cdot \mathit{yExp} - \mathit{colRank} > -5)$, putting more emphasis on the years of work experience. This modification indeed makes the program pass the fairness criterion with $\varepsilon = 0.1$ given pop.

Algorithm 6 pop

```
$\mathit{ethnicity} \leftarrow \mathcal{N}(0, 10)$
$\mathit{colRank} \leftarrow \mathcal{N}(25, 10)$
$\mathit{yExp} \leftarrow \mathcal{N}(10, 5)$
**if** $\mathit{ethnicity} > 10$ **then**
    $\mathit{colRank} \leftarrow \mathit{colRank} + 5$
```

Algorithm 7 dec

```
if ($colRank \leq 5$) then
    hire $\leftarrow$ $True$
else if ($yExp - colRank > -5$) then
    hire $\leftarrow$ $True$
else
    hire $\leftarrow$ $False$
```