Knowledge Compilation for Incremental and Checkable Stochastic Boolean Satisfiability

Che Cheng¹, Yun-Rong Luo² and Jie-Hong R. Jiang¹,³
¹Graduate Institute of Electronics Engineering, National Taiwan University, Taipei, Taiwan
²University of Michigan, Ann Arbor MI, USA
³Department of Electrical Engineering, National Taiwan University, Taipei, Taiwan
r11943097@ntu.edu.tw, yunrong@umich.edu, jhjiang@ntu.edu.tw

Abstract

Knowledge compilation has proven effective in (weighted) model counting, uniquely supporting incrementality and checkability. For incrementality, compiling an input formula once suffices to answer multiple queries, thus reducing the total solving effort. For checkability, the compiled formula is amenable to producing machine-checkable proofs for verification, thus strengthening the solver’s reliability. In this work, we extend knowledge compilation from model counting to stochastic Boolean satisfiability (SSAT) solving by generalizing the dec-DNNF representation to accommodate the SSAT quantifier structure and integrate it into SharpSSAT, a state-of-the-art SSAT solver. We further study proof generation from the compiled representation and extend CPOG, a certified model-counting toolchain, to generate proofs for certifying the results of SharpSSAT. Experimental results show the benefits of the proposed knowledge compilation approach for SSAT in sharing computation efforts for multiple queries and producing checkable dec-DNNF logs with negligible overhead.

1 Introduction

Stochastic Boolean satisfiability (SSAT) is a formalism that generalizes quantified Boolean formula (QBF) by replacing the universal quantifier with a randomized one. Its ability to capture decision-making under uncertainty makes it suitable for succinct encoding of a variety of problems, e.g., formal verification of probabilistic design [Lee and Jiang, 2018], planning under uncertainty [Salmon and Poupart, 2020], statistical inference [Hsieh and Jiang, 2022], and fairness analysis of machine learning models [Ghosh et al., 2021]. Efficient solving of SSAT has also been an actively studied topic [Majercik and Boots, 2005; Salmon and Poupart, 2020; Chen et al., 2021; Wang et al., 2022; Fan and Jiang, 2023].

Another direction that has been long studied for propositional reasoning tasks is knowledge compilation. Through compiling a propositional theory into a suitable target language, many intractable tasks can be done in polynomial time [Darwiche and Marquis, 2002]. The earliest and most well-known target language is the binary decision diagram (BDD), which is widely applied in circuit synthesis and verification [Clarke et al., 2018].

Despite its support for a wide range of queries and transformations, BDD construction often suffers from memory explosion and fails for large problem instances. Hence, the quest for succinct formats is an important pursuit. Huang and Darwiche [2005] observed that the trace of an exhaustive DPLL-style search with different constraints imposed corresponds to different target languages. Among them, the decision decomposable negation normal form (dec-DNNF) is the most succinct. Recent advances in model counting have led to the development of several dec-DNNF compilers [Muise et al., 2012; Lagniez and Marquis, 2017]. Recent work also studied certified knowledge compilation [Capelli, 2019; Capelli et al., 2021; Bryant et al., 2023], and developed various approaches to verify whether the compiled result is logically equivalent to the compiler’s given input. Certified knowledge compilation is crucial in verifying the results of model counters.

Previous studies concerning knowledge compilation and quantified formulas focus mainly on the theoretical bounds for QBF evaluation. In [Coste-Marquis et al., 2005], the tractability of QBF solving on a variety of target languages, including BDD and dec-DNNF, was studied. In [Fargier and Marquis, 2006; Capelli and Mengel, 2019], new target languages were proposed to achieve quantifier elimination. In [Lai et al., 2017], BDDs were generalized to provide more flexibility with canonicity being maintained. Regarding SSAT, Lee et al. [2018] exploited BDDs for weighted model counting in solving the exist-random quantified subclass of SSAT. Ojala [2022] evaluated SSAT by compiling the matrix formula into a BDD or sentential decision diagram (SDD) with compatible ordering using off-the-shelf compilers. However, its performance is inferior to the state-of-the-art SSAT solvers.

In this paper, we propose a new target language of knowledge compilation, based on the trace of the state-of-the-art SSAT solver SharpSSAT [Fan and Jiang, 2023], named levelized dec-DNNF, that supports incremental and checkable evaluations on an SSAT formula. It allows more relaxed constraints on the ordering compared to similar ones proposed in previous methods [Fargier and Marquis, 2006; Lai et al., 2017], thus allowing more flexible and efficient compilation. In addition, we formalize the pruning techniques implemented in SharpSSAT into our representation to allow partial exploration, resulting in both faster compilation and...
more succinct representation.

In addition, to certify the correctness of SSAT solving, we develop a proof framework cert-SSAT based on CPOG [Bryant et al., 2023], which generates and checks clausal proof for logical equivalence between the input and output of a dec-DNNF compiler. To accommodate partial exploration from the pruning techniques, two dec-DNNF graphs need to be compiled, which are then passed to cert-SSAT for detailed clausal proof generation and validation. Moreover, we relax the logical equivalence checking in CPOG into two logical implication checks to separately certify lower and upper bounds of SSAT satisfying probability.

Experiments on various benchmark instances show the performance advantage of compilation over repeated solving. When certification is of concern, little computational overhead is required for SharpSSAT to generate the additional dec-DNNF logs for further proof checking. SharpSSAT is able to generate dec-DNNF proofs for all solvable instances, and cert-SSAT successfully turns 80.5% of them into clausal proofs and verifies the clausal proofs within a reasonable time. For the remaining 19.5%, cert-SSAT can neither prove or disprove their correctness.

In fact, our proof framework helps reveal and fix previously undetected bugs in SharpSSAT. It is the first work to formally verify the results of an SSAT solver, in contrast to [Fan and Jiang, 2023] with only (lower bound) witnessing ability. Our method enhances the reliability and trustworthiness of SharpSSAT results.

The rest of this paper is organized as follows. Section 2 defines notations and introduces essential backgrounds. Section 3 proposes a new knowledge compilation format for SSAT and discusses its supported queries. Section 4 presents the proof generation and validation framework for SharpSSAT. The experimental evaluation follows in Section 5. Finally, Section 6 concludes this paper.

2 Preliminaries

In this section, we define the notations used throughout the paper, and briefly review the foundations of SSAT and knowledge compilation.

Boolean values TRUE and FALSE are denoted by $\top$ and $\bot$, respectively. Boolean connectives $\neg$, $\land$, $\lor$, and $\rightarrow$ are interpreted in their conventional semantics. A literal $l$ is a Boolean variable $v$ or its negation $\neg v$ (also written as $\overline{v}$ for simplicity), and we write $\text{var}(l) = v$ to denote the corresponding Boolean variable. A clause $C$ is a disjunction of literals. A Boolean formula $\phi$ is in conjunctive normal form (CNF) if it is a conjunction of clauses. We also view a clause $C$ as a set of literals and a CNF formula $\phi$ as a set of clauses whenever appropriate. The substitution of variable $v$ with formula $\psi$ in formula $\phi$ is denoted by $\phi[\psi/v]$. We denote the set of variables occurring in a formula $\phi$ by $\text{vars}(\phi)$.

Let $V$ be a set of Boolean variables. An assignment $\alpha$ on $V$ is a mapping from $V$ to $\mathbb{B} = \{\top, \bot\}$, and we denote the set of all possible assignments on $V$ by $\mathcal{A}(V)$. The projection of formula $\phi$ over variables $V$ onto an assignment $\alpha$ on $V'$ is denoted by $\phi[\alpha]$, which is obtained by substituting each occurrence of each $v \in V'$ with the Boolean value $\alpha(v)$.

2.1 Knowledge Compilation

The idea of knowledge compilation is to first compile a propositional theory (or equivalently, a Boolean function) into a target language in an offline phase. Then, queries on the target language are performed during the online phase. A comparison of supported queries and succinctness of different target languages can be found in [Darwiche and Marquis, 2002].

Among the target languages, we are particularly interested in dec-DNNF, which is a special negation normal form (NNF), to be defined as follows.

Definition 1 (Negation normal form). A Boolean formula $\phi$ is in the negation normal form (NNF) if it consists of only the connectives $\land$, $\lor$, and $\neg$, and the negation connective $\neg$ only occurs immediately before a variable.

We define the notions of decomposability and decision for a formula as follows.

Definition 2 (Decomposability). A conjunction $\bigwedge_i \phi_i$ of subformulas $\phi_i$’s is decomposable if $\text{vars}(\phi_i) \cap \text{vars}(\phi_j) = \emptyset$ for all $i \neq j$. A Boolean formula $\phi$ in NNF is in the decomposable negation normal form (DNNF) if every conjunction in $\phi$ is decomposable.

Definition 3 (Decision). A disjunction $\phi_1 \lor \phi_2$ of two subformulas $\phi_1$ and $\phi_2$ is called a decision with decision variable $v$ if $\phi_1 = v \land \psi_1$ and $\phi_2 = \neg v \land \psi_2$.

Finally, we define dec-DNNF.

Definition 4 (Dec-DNNF). Dec-DNNF is a language consisting of Boolean formulas in NNF, where each conjunction is decomposable, and each disjunction is a decision.

In the sequel, we shall view a dec-DNNF formula $\phi$ as a directed acyclic graph (DAG) with a unique root node labeled with $\phi$. Each leaf node is labeled with either a literal or a Boolean constant, and each internal node is either an AND node, whose $i^{th}$ child is labeled with the conjoined subformula $\phi_i$ as in Definition 2, or a decision node $N$ associated with decision variable $v$, denoted $\text{dvar}(N) = v$, with exactly two children labeled with $\phi_1$ and $\phi_2$ as in Definition 3. Note that each subgraph induced by a node $N$ and its descendants is a dec-DNNF graph representing the subformula $\phi^N$ that $N$ is labeled with. We sometimes use the terms dec-DNNF formula and dec-DNNF graph interchangeably.

2.2 Stochastic Boolean Satisfiability

Stochastic Boolean satisfiability (SSAT) is initially proposed in [Papadimitriou, 1985] as a game against nature.

Definition 5 (SSAT Syntax). An SSAT formula $\Phi = Q, \phi$ over variables $V$ is of the form

$$Q_1v_1, Q_2v_2, \ldots, Q_nv_n, \phi, \tag{1}$$

where $Q = Q_1v_1, Q_2v_2, \ldots, Q_nv_n$ is the quantifier prefix (prefix for short), and $\phi$ is the matrix. Each $Q_i \in \{\exists, \forall\}$, and $\phi$ is a quantifier-free Boolean formula over $V$ in CNF. The symbol $\mathbb{B}^r$ denotes a randomized quantifier, where $\mathbb{B}^r$ states that variable $r$ evaluates to $\top$ with probability $p$ and $\bot$ with probability $1 - p$. We denote the quantifier of variable $v$ as $Q(v)$, i.e., $Q(v_i) = Q_i$. If $Q(v) = \exists$ (resp. $\forall$), we say that $v$ is an existential (resp. randomized) variable.
The quantification level (to count quantifier alternations) of variable \( v_i \), denoted by \( \text{lvl}(v_i) \), is recursively defined as

- \( \text{lvl}(v_1) = 1 \), and
- \( \text{lvl}(v_i) = \begin{cases} \text{lvl}(v_{i-1}) - 1 & \text{if } Q_i = Q_{i-1} \\ \text{lvl}(v_{i-1}) + 1 & \text{otherwise} \end{cases} \) for \( i > 1 \).

The satisfying probability of an SSAT formula \( \Phi \) is defined recursively by

- \( \text{Pr}[\top] = 1 \), \( \text{Pr}[\bot] = 0 \),
- \( \text{Pr}[\exists v. \Phi'] = \max(\text{Pr}[\Phi'[\bot/v]], \text{Pr}[\Phi'[\top/v]]) \), and
- \( \text{Pr}[\forall v. \Phi'] = (1 - p) \cdot \text{Pr}[\Phi'[\bot/v]] + p \cdot \text{Pr}[\Phi'[\top/v]] \),

where \( \Phi' \) is the SSAT formula obtained by removing the outermost quantifier of \( \Phi \), i.e., \( \Phi = Qv. \Phi' \).

Given an SSAT formula \( \Phi \) and some \( \theta \in [0, 1] \), deciding whether \( \text{Pr}[\Phi] \geq \theta \) is PSPACE-complete [Papadimitriou, 1985]. In addition, Theorem 1 shows that the satisfying probability of an SSAT formula can be computed by the satisfying probability of its disjoint components. A similar property is exhibited in model counting.

**Theorem 1** (Salmon and Poupart, 2020). Given an SSAT formula \( \Phi = Q. \phi \) if \( \phi = A_{i=1}^k \phi_i \) is decomposable (Definition 2), then \( \text{Pr}[\Phi] = \prod_{i=1}^k \text{Pr}[Q. \phi_i] \).

### 2.3 Clausal Proof for Knowledge Compilation

CPOG [Bryant et al., 2023] can be used to certify the results of a knowledge compiler by giving it a CNF formula \( \phi \) and its compiled dec-DNNF formula \( G \) as input. Given \( \phi \) and \( G \), CPOG checks whether \( \phi \) is logically equivalent to the Boolean function represented by \( G \). It is a clausal proof system based on extended resolution [Tseitin, 1983]. A CPOG proof is a sequence of extension, clause addition, and clause deletion steps. An extension step introduces one extension variable \( v \) that either encodes a conjunction or disjunction of literals whose variables, called defining variables of \( v \), can be the previously introduced extension variables or original variables in the input formula. CPOG generates CNF encodings to describe the logical relation between extension variables and their respective defining variables [Tseitin, 1983]. A clause can be added to or deleted from the proof if the clause is an implication redundancy with respect to the set of other clauses in the proof. That is, if \( \psi \rightarrow C \) holds for a clause \( C \) and a CNF formula \( \psi \), then \( C \) can be added to \( \psi \) to form \( \psi \cup \{C\} \) or deleted from \( \psi \cup \{C\} \) to form \( \psi \).

CPOG consists of a proof generator \( \text{cpg-gen} \) and a proof checker \( \text{cpg-check} \), and works as follows. \( \text{cpg-gen} \) first encodes \( G \) into a CNF formula \( \psi_G \) by extension, and then generates a clausal proof for \( \phi \rightarrow \psi_G \) by clause addition and a proof for \( \psi_G \rightarrow \phi \) by clause deletion. Given a CPOG proof log \( P \), \( \text{cpg-check} \) checks whether each extension, clause addition, and deletion step in \( P \) is sound. Upon the completion of proof checking, \( \text{cpg-check} \) claims \( \phi \equiv \psi_G \) if and only if \( 1 \) there is exactly one final unit clause \( \{r\} \) added via implication redundancy, where \( r \) is the extended literal representing the root of \( G \), and \( 2 \) all input clauses of \( \phi \) have been deleted.

### 3 Knowledge Compilation for SSAT

We develop the idea of knowledge compilation for SSAT tightly around the DPLL-style solver SharpSSAT [Fan and Jiang, 2023]. SharpSSAT is a state-of-the-art solver that utilizes component analysis, which is responsible for compiling formulas with the decomposability property introduced in Section 2.1 [Huang and Darwiche, 2005]. Inspired by the design of SharpSSAT, we propose adding an ordering constraint on dec-DNNF and allowing partial exploration to incorporate the prefix structure of SSAT into the representation.

#### 3.1 Levelized Dec-DNNF

As observed in [Huang and Darwiche, 2005], the trace of an exhaustive DPLL-style search with component analysis corresponds to a dec-DNNF formula. Also, in model counting, due to its exhaustive nature, the effort of compilation only has a slight overhead compared to direct solving, which makes it especially compelling. As observed in [Salmon and Poupart, 2020], decomposability holds for SSAT, hence the idea of SSAT evaluation with dec-DNNF seems plausible.

However, we recall the observation that deciding a QBF is PSPACE-hard for arbitrary BDD [Coste-Marquis et al., 2005, Proposition 1] and in \( \mathcal{P} \) for BDD with compatible ordering [Coste-Marquis et al., 2005, Proposition 2]. As QBF evaluation can be reduced to SSAT evaluation and dec-DNNF is a superset of BDD, answering the satisfying probability of an SSAT formula given an arbitrary dec-DNNF formula of its matrix is also PSPACE-hard. Therefore, we know that some ordering constraints must be imposed upon the dec-DNNF formula to achieve SSAT evaluation in polynomial time with respect to the formula size.

We thus propose levelized dec-DNNF to capture such representation. To simplify our discussion, in the sequel we view a leaf node \( N \) associated with literal \( l \) as a decision node with \( \text{dvar}(N) = \text{var}(l) \), and \( \top \) and \( \bot \) as the two children, thus only considering constant nodes as leaf nodes. Note that this modification increases the graph size by a constant factor, for having two additional edges per literal node and two additional constant nodes in total.

**Definition 6** (Levelization). Given an SSAT formula \( \Phi = Q. \phi \) in the form of Eq. (1) and a dec-DNNF graph \( G \),

1. say that \( G \) is levelized with respect to \( \Phi \) if for each pair of decision nodes \( N_1 \) and \( N_2 \), if \( N_2 \) is reachable from \( N_1 \), then \( \text{lvl}(\text{dvar}(N_1)) \leq \text{lvl}(\text{dvar}(N_2)) \).

The condition ensures that for all paths from the root to the leaves in the dec-DNNF formula \( G \), the ordering of the decision nodes is consistent with the prefix of \( \Phi \). In addition, we remark that by the decomposability of the conjunctions in \( N_1 \), \( \text{dvar}(N_1) \) cannot appear in the formulas represented by the descendants of \( N_1 \). In particular, we have \( \text{dvar}(N_1) \neq \text{dvar}(N_2) \). This corresponds to the well-known read-once property [Darwiche and Marquis, 2002].

The idea of levelized dec-DNNF closely resembles that of O-DDG [Fargier and Marquis, 2006] and BDD[\( \wedge \)] [Lai

\^Note that we do not require the function represented by \( G \) to be equivalent to \( \phi \), as this will not be the case when pruning techniques are considered in Section 3.3.
SSAT Evaluation with dec-DNNF

1: procedure EvalSSAT(\(\Phi, G\))
2: \(\triangleright \Phi = \mathbb{Q}, \phi\) is an SSAT formula, and \(G\) is a levelized dec-DNNF graph equivalent to \(\phi\) with \(m\) nodes
3: Let \(P\) be a vector of size \(m\) indexed by the nodes of \(G\)
4: Initialize each cell of \(P\) to null
5: Eval-Node(\(\Phi, G, G.\text{root}\))
6: return \(P[G.\text{root}]\)

7: procedure Eval-Node(\(\Phi, G, N\))
8: \(\triangleright N\) is a node in \(G\)
9: if \(P[N] = \text{null}\) then \(\triangleright\) Each node is computed once
10: \(\triangleright\) Each node is computed only once using memoization
11: \(|P[N]| \leftarrow 0\)
12: else if \(\phi^N = \top\) then
13: \(|P[N]| \leftarrow 1\)
14: else if \(N\) is an AND node then
15: \(\triangleright\) Let \(N_1, \ldots, N_k\) be the children of \(N\)
16: \(p \leftarrow 1\)
17: for each child \(N_i\) of \(N\) do
18: \(\triangleright\) Eval-Node(\(\Phi, G, N_i\))
19: \(p \leftarrow p \cdot |P[N_i]|\)
20: \(P[N] \leftarrow p\)
21: else
22: \(\triangleright N\) is a decision node with decision variable \(v\) and children \(N_1, N_2\)
23: Eval-Node(\(\Phi, G, N_1\))
24: Eval-Node(\(\Phi, G, N_2\))
25: if \(Q(v) = \top\) then
26: \(P[N] \leftarrow p \cdot |P[N_1]| + (1 - p) \cdot |P[N_2]|\)
27: else \(\triangleright Q(v) = \bot\)
28: \(P[N] \leftarrow \max(\{P[N_1], P[N_2]\})\)

Algorithm 1 SSAT Evaluation with dec-DNNF

Theorem 2. Given an SSAT formula \(\Phi = \mathbb{Q}, \phi\) in the form of Eq. (1) and \(\phi\) represented in the levelized dec-DNNF graph \(G\), Algorithm 1 computes the satisfying probability of \(\Phi\) in time \(O(|G|)\).

Proof. We first prove the correctness of the algorithm by induction on the structure of \(G\).

For the base case, let \(G\) consist of a single node \(N\), which must be a constant node. Its satisfying probability is 1 if the node is associated with \(\top\), and 0 if associated with \(\bot\), as shown on Lines 10 to 13.

Suppose the node \(N\) is an AND node with \(k\) children \(N_1, \ldots, N_k\). By Theorem 1, we have \(\Pr[\mathbb{Q}, \phi^{N}] = \prod_{i=1}^{k} \Pr[\mathbb{Q}, \phi^{N_i}]\), as done on Lines 14 to 20.

Finally, suppose the root node \(N\) is a decision node with decision variable \(v\) and children \(N_1, N_2\). By the read-once property, we have \(v \notin \text{vars}(\mathbb{Q}^{N_1}) \cup \text{vars}(\mathbb{Q}^{N_2})\), and since \(\phi^N = (v \land \phi^{N_1}) \lor (\neg v \land \phi^{N_2})\), we have \(\phi^{N_1} = \phi^{N}[^v\to v]\) and \(\phi^{N_2} = \phi^{N}[^v\to \neg v]\). Next, by the levelized assumption, \(v\) has the smallest level among all variables in \(\text{vars}(\phi^N)\). The satisfying probability can thus be computed as done on Lines 21 to 28 according to the recursive definition of satisfying probability of SSAT introduced in Section 2.2.

It follows that the satisfying probability of the SSAT formula is correctly computed. In addition, since the probability of each node is computed only once using memoization and each edge is visited once, the procedure runs in time \(O(|G|)\).

3.2 SSAT Evaluation with Levelized Dec-DNNF

3.3 Pruning Techniques

In the previous subsection, we demonstrate how to perform SSAT-related queries with dec-DNNF by imposing an ordering constraint on the representation. However, in addition to the ordering constraint, SSAT also differs from model counting in its optimization nature brought by existential quantification.

SharpSSAT [Fan and Jiang, 2023] utilizes this property and implements some pruning techniques, including pure literal detection and existential early return, to enhance its performance. The techniques exploit the fact that if \(Q(v) = \bot\) and the implication \(\phi[^v\to v] \rightarrow \phi[^v\to \neg v]\) (resp. \(\phi[^v\to v] \rightarrow \phi[^v\to v]\)) holds, then \(\Pr[\mathbb{Q}, \phi]\) equals \(\Pr[\mathbb{Q}, \phi[^v\to v]]\) (resp. \(\Pr[\mathbb{Q}, \phi[^v\to v]]\)). It follows that the solver only needs to explore the branch \(\phi[^v\to v]\) (resp. \(\phi[^v\to v]\)) when \(\phi[^v\to v] \rightarrow \)}
Theorem 4. Given an SSAT formula \( \Phi = Q. \phi \) and a levelized dec-DNNF graph \( G_1 \) compiled from \( \phi \) with pruning enabled, the satisfying probability of \( \Phi \) and a reweighting \( \Phi' \) of \( \Phi \) can be computed correctly by Algorithm 1 using \( G_1 \), while that of a cofactoring \( \Phi'' \) of \( \Phi \) cannot.

Proof. For \( \Phi \) and \( \Phi' \), since the unexplored branch must have a smaller satisfying probability and the decision variable is existential, Algorithm 1 still computes the correct satisfying probability, as the unexplored branch is never chosen on Line 28.

For the cofactoring case, let \( \Phi = \exists v_1. \mathcal{D}^{0.5} v_2. (v_1 \lor v_2) \) and consider the assignment \( \alpha : v_1 \rightarrow \bot \). Since \( Q(v_1) = \exists \) and \( v_1 \) is a pure literal, the pruned representation will essentially represent the formula \( v_1 \). Hence, Algorithm 1 will return \( \Pr[\Phi[\alpha]] = 0 \), while it is in fact 0.5.

We remark that replacing each unexplored node with the node of the explored branch will result in a dec-DNNF graph \( G_u \), which represents a formula \( \phi_u \) such that \( \phi \rightarrow \phi_u \). Notice that replacing \( G_1 \) with \( G_u \) does not affect the correctness of Theorem 4. This observation paves the way for checkable SSAT solving, which will be detailed in Section 4.

By introducing the pruning techniques into the compilation, we effectively utilize the maximizing nature of SSAT to compile a dec-DNNF graph of smaller size that still faithfully captures the essence of the SSAT formula. In addition, we note that the overhead in detecting pure literals is on traversing the clauses, which occurs purely during the compilation process. Hence, this overhead is amortized by the improvement it brings in the multiple queries performed on the compiled graphs.

We conclude this section with an example.

Example 1. Consider the SSAT formula
\[
\Phi = \mathcal{D}^{0.4} x_1. \exists y_1. \exists y_2. \exists y_3. \mathcal{D}^{0.6} x_2.
(\bar{x}_2 \lor \bar{y}_1)(\bar{y}_1 \lor \bar{y}_2 \lor y_3)(\bar{y}_1 \lor \bar{y}_2 \lor \bar{x}_2)
(y_2 \lor y_3 \lor \bar{x}_2)(\bar{y}_1 \lor \bar{x}_2).
\]

The compiled levelized dec-DNNF is shown in Fig. 1a. It is easily verified that the read-once and levelization properties are satisfied.

We now demonstrate how to compute \( \Pr[Q. \phi[y_1]] \) by calculating the probability \( p_i \) of each node \( N_i \), labeled with “(i)” in the figure, of the compiled levelized dec-DNNF graph:

- \( p_{10} = 1 \) and \( p_9 = 0 \) by the base cases,
- \( p_{12} = 0.4 \cdot 1 + 0.6 \cdot 0 = 0.4, p_{11} = p_6 = p_5 = \max(0, 1) = 1, \)
- \( p_8 = 0 \) since \( y_1 = \top \) is assumed,
- \( p_7 = p_{11} \cdot p_{12} = 0.4, \)
- \( p_4 = \max(p_6, p_7) \cdot 1, p_3 = p_5 = 1, p_2 = p_4 \cdot p_8 = 0, \) and \( p_1 = 0.6 \cdot p_2 + 0.4 \cdot p_3 = 0.4 \) following the same set of rules.

We hence conclude that \( \Pr[Q. \phi[y_1]] = 0.4. \)

Since \( y_3 \) is a pure literal in \( \Phi \), we can simplify the compilation by exploring the branch \( y_3 = \top \) only. The resulting dec-DNNF graph is shown in Fig. 1b. It can be seen that the technique simplifies the representation when compared to that shown in Fig. 1a.

4 SSAT Validation with Levelized Dec-DNNF

In order to extend CPOG to cert-SSAT for SharpSSAT certification, two main modifications to CPOG are needed.
First, cert-SSAT has to additionally validate levelization of dec-DNNF. Second, more importantly, cert-SSAT has to support checking the partial exploration introduced by the pruning techniques. Checking the former is relatively straightforward. To achieve the latter, instead of simply checking logical equivalence between the input CNF and compiled dec-DNNF formulas as mentioned in Section 2.3, cert-SSAT requires SharpSSAT to compile two dec-DNNF graphs $G_t$ and $G_u$ and check Eq. (2) and Eq. (3) by Theorem 5 as follows. 

**Theorem 5.** Given an SSAT formula $\Phi = Q, \phi$ and two levelized dec-DNNF graphs $G_t$ and $G_u$ respectively respecting the quantifier level of $\Phi$, then $\Pr[\Phi] = p$ if the following holds:

\[
(G_t \rightarrow \phi) \land (\phi \rightarrow G_u), \quad \text{and} \quad E_{\text{AL}}(\Phi, G_t) = E_{\text{AL}}(\Phi, G_u) = p. \tag{3}
\]

**Proof.** The proof follows from the same reasoning for pruning techniques in Section 3.3. \hfill \square

SharpSSAT requires a minor modification to compile $G_t$ and $G_u$, in addition to computing satisfying probability. Whenever pruning occurs, SharpSSAT marks the pruned branch so that the unexplored branch of the decision node can later be connected to the constant node $\bot$ or the explored branch to form the nodes $N_t \in G_t$ and $N_u \in G_u$, as explained in Section 3.3.

The computation flow of cert-SSAT is shown in Fig. 2. Given an input SSAT instance $\Phi$ in the sdimacs format, the modified SharpSSAT under the certificate generation option first generates two dec-DNNF graphs $G_t$ and $G_u$ in the nnf format in addition to computing $\Pr[\Phi]$. Next, EvalSSAT is applied to check whether Eq. (3) and the levelization constraints hold for $G_t$ and $G_u$. Then, the proof generator cpgogen takes $\Phi, G_t$, and $G_u$ as input, generates CNF formulas $\psi_{G_t}$ of $G_t$ and $\psi_{G_u}$ of $G_u$, and emits a clausal proof that establishes Eq. (2) in the cpg format [Bryant et al., 2023]. Finally, the proof checker cpg-check checks whether $\psi_{G_t}$ (resp. $\psi_{G_u}$) is generated from $G_t$ (resp. $G_u$) by the extension process and verifies the correctness of the clausal proof for Eq. (2) by the standard reverse unit propagation (RUP) checking [Goldberg and Novikov, 2003; Gelder, 2008].

Let the outputs of SharpSSAT and cpg-gen be referred to as nnf-proof and cpg-proof, respectively. Suppose SharpSSAT reports $\Pr[\Phi] = p$ for an SSAT instance $\Phi = Q.\phi$. The cpg-proof for $(\psi_{G_t} \rightarrow \phi)$ and that for $(\phi \rightarrow \psi_{G_u})$ can certify the lower and upper bounds of $\Pr[\Phi]$, respectively. Since it is computationally easier to verify the first implication, even if cert-SSAT fails to verify the second implication and hence fails to conclusively prove that $\Pr[\Phi] = p$, chances are that cert-SSAT could at least provide a lower-bound proof. In particular, when $p = 1$, it suffices to prove that $\Pr[\Phi] \geq 1$, and cert-SSAT only need to prove the lower-bound case.

## 5 Experimental Results

We equipped the state-of-the-art SSAT solver SharpSSAT [Fan and Jiang, 2023] with a knowledge compilation capability, implemented EvalSSAT of Algorithm 1 in the C++ language, and modified CPOG for the proposed SSAT validation toolchain cert-SSAT. For incremental SSAT evaluation using EvalSSAT, an option (−d) was implemented in SharpSSAT to compile the levelized dec-DNNF graph $G_t$. For proof checking using the cert-SSAT flow, another option (−l) was implemented in SharpSSAT to compile both $G_t$ and $G_u$, and EvalSSAT was also modified to check the ordering of the levelized dec-DNNF graph.

We only enable the pruning techniques of Section 3.3 when the original option for pure literal detection (−p) is turned on. To study the effectiveness of incremental SSAT evaluation, we conducted experiments on reweighting and cofactoring queries. To study the effectiveness of SSAT proof checking, we applied the cert-SSAT computation on all the instances. The benchmark formulas are taken from [Chen et al., 2021].

### 5.1 Incremental SSAT Evaluation

The experiments were conducted on a Linux machine with an Intel Xeon Silver 4210 CPU processor at 2.2 GHz. For each SSAT formula, we randomly generated 10 instances for both query types and compared the performance of solving incrementally with knowledge compilation to that of solving from scratch for each instance. We refer to the three solving methods as KC-p, KC-np, and baseline, which corresponds to knowledge compilation with and without pruning and solving from scratch each time, respectively. Because SharpSSAT is a state-of-the-art solver, and, to the best of our knowledge, no SSAT solvers support incremental solving, we did not compare the performance with other SSAT solvers, such as ClausSSAT [Chen et al., 2021] and ElimSSAT [Wang et al., 2022], whose ability on SSAT solving has already been compared against SharpSSAT in [Fan and Jiang, 2023].

To fairly judge the performance of the two approaches, we set a time limit of 2000 sec for each SSAT formula as follows: For the settings KC-p and KC-np, we set a time limit of 1000 sec for compilation and 100 sec for each query. If the compilation failed within the time limit, we view the queries as failed as well. For the setting baseline, we set a time limit of 200 sec for each query. The overall performance is then

\[\text{first implication,}\] even if cert-SSAT fails to verify the second implication and hence fails to conclusively prove that $\Pr[\Phi] = p$, chances are that cert-SSAT could at least provide a lower-bound proof. In particular, when $p = 1$, it suffices to prove that $\Pr[\Phi] \geq 1$, and cert-SSAT only need to prove the lower-bound case.

### 5.1 Incremental SSAT Evaluation

The experiments were conducted on a Linux machine with an Intel Xeon Silver 4210 CPU processor at 2.2 GHz. For each SSAT formula, we randomly generated 10 instances for both query types and compared the performance of solving incrementally with knowledge compilation to that of solving from scratch for each instance. We refer to the three solving methods as KC-p, KC-np, and baseline, which corresponds to knowledge compilation with and without pruning and solving from scratch each time, respectively. Because SharpSSAT is a state-of-the-art solver, and, to the best of our knowledge, no SSAT solvers support incremental solving, we did not compare the performance with other SSAT solvers, such as ClausSSAT [Chen et al., 2021] and ElimSSAT [Wang et al., 2022], whose ability on SSAT solving has already been compared against SharpSSAT in [Fan and Jiang, 2023].

To fairly judge the performance of the two approaches, we set a time limit of 2000 sec for each SSAT formula as follows: For the settings KC-p and KC-np, we set a time limit of 1000 sec for compilation and 100 sec for each query. If the compilation failed within the time limit, we view the queries as failed as well. For the setting baseline, we set a time limit of 200 sec for each query. The overall performance is then

\[\text{first implication,}\] even if cert-SSAT fails to verify the second implication and hence fails to conclusively prove that $\Pr[\Phi] = p$, chances are that cert-SSAT could at least provide a lower-bound proof. In particular, when $p = 1$, it suffices to prove that $\Pr[\Phi] \geq 1$, and cert-SSAT only need to prove the lower-bound case.

### 5.1 Incremental SSAT Evaluation

The experiments were conducted on a Linux machine with an Intel Xeon Silver 4210 CPU processor at 2.2 GHz. For each SSAT formula, we randomly generated 10 instances for both query types and compared the performance of solving incrementally with knowledge compilation to that of solving from scratch for each instance. We refer to the three solving methods as KC-p, KC-np, and baseline, which corresponds to knowledge compilation with and without pruning and solving from scratch each time, respectively. Because SharpSSAT is a state-of-the-art solver, and, to the best of our knowledge, no SSAT solvers support incremental solving, we did not compare the performance with other SSAT solvers, such as ClausSSAT [Chen et al., 2021] and ElimSSAT [Wang et al., 2022], whose ability on SSAT solving has already been compared against SharpSSAT in [Fan and Jiang, 2023].

To fairly judge the performance of the two approaches, we set a time limit of 2000 sec for each SSAT formula as follows: For the settings KC-p and KC-np, we set a time limit of 1000 sec for compilation and 100 sec for each query. If the compilation failed within the time limit, we view the queries as failed as well. For the setting baseline, we set a time limit of 200 sec for each query. The overall performance is then
when the compilation failed, the benefit of incremental solving is overshadowed by the timeout penalty.

Using the proposed compilation-based incremental solving, the pruning techniques are indeed useful and that the task setting is overshadowed by the timeout penalty.

To truly understand the power of incremental solving, we further break down the compilation and query time for the successfully compiled formulas and compare them against the baseline setting. The results are shown in Table 2.7 We can see that each query only takes around 9.3% of the compilation time for the reweighting task and around 5.8% for the cofactoring task. On average, it takes 1.53 and 3.62 queries for the compilation-based method to outperform the baseline for the reweighting and cofactoring task, respectively. This result shows that whenever compilation is successful, it only takes very few queries to compensate for the additional effort.

The reason that the cofactoring task requires more queries is two-fold: First, as discussed in Theorem 3, pruning techniques are not applicable to the task. Therefore, baseline has the advantage of utilizing pruning techniques to speed up. Second, the cofactoring query introduces additional unit clauses in the SSAT formula in the baseline setting, and may thus be easier to solve.

To summarize, the reweighting task can be readily sped up by the KC-p setting, and while the cofactoring task may seem less favorable from Table 1, the breakdown analysis in Table 2 demonstrates its potential.

### 5.2 SSAT Proof Generation and Validation

We ran cert-SSAT on a Linux machine with a 12th Gen Intel Core i9-12900 processor, and set a time limit of 1000 sec

---

7We skip the setting KC-np for the reweighting task, as it is inferior to KC-p.
for SharpSSAT and EvalSSAT and a time limit of 2500 sec for cpog-gen and cpog-check. We excluded the unsolvable instances by SharpSSAT reported in previous work [Fan and Jiang, 2023], yielding 18 families of benchmark with 236 instances in total for the experiment.

The experimental results are summarized as follows. SharpSSAT successfully generated nnf-proofs for all 236 instances, and all of them passed the check of EvalSSAT. For 205 instances (86.9%), cert-SSAT successfully proved the easy implication $\psi_{Gi} \rightarrow \phi$ and hence verified their lower bounds. Among these instances, the other implication $\phi \rightarrow \psi_{Gi}$ was proved for 190 instances (80.5%); hence their satisfying probabilities were proved to be tight. For the 190 proven instances, the ratios between the combined time for running EvalSSAT, cpog-gen, and cpog-check versus the time for SharpSSAT had a harmonic mean of 6.6. As for proof sizes, the ratios between the number of clauses in cpog-proofs and that in the original sdimacs files had a harmonic mean of 3.21.

We analyzed the 46 (236 - 190) instances that cert-SSAT cannot fully prove or disprove the solving results. Among them, 17 instances suffered from time out; cpog-gen unexpectedly aborted on 13 cases. cpog-gen failed to generate cpog-proofs for all 13 instances in the tlc family because the matrix of these instances is unsatisfiable, which cpog-gen cannot handle. As an alternative, we can pass their matrices directly into SAT solvers and apply standard proof logging and checking tools [Heule et al., 2013; Wetzler et al., 2014] to verify the results. cpog-check detected three cases, Tiger-25, c432_re, and cnt03e violating decomposability. We conjectured that this was caused by integrating both component caching and clause learning in SharpSSAT. After we disabled clause learning in SharpSSAT, Tiger-25 passed the overall check of cert-SSAT, while SharpSSAT timed out on solving the other two instances. We leave fixing these issues for future work.

We provide detailed runtime and proof size analyses of cert-SSAT in Fig. 3 and Fig. 4, respectively. Fig. 3a shows the relative runtime between a cert-SSAT run (i.e., the combined runtime for EvalSSAT, cpog-gen, and cpog-check) and SharpSSAT. We report both the runtime of complete (proving both implications) and partial (proving only the lower bound) runs. The complete (resp. partial) runs are marked in orange (resp. blue) dots (resp. triangles). Fig. 3b shows relative runtime between each cert-SSAT subprocess and SharpSSAT. Dark-blue squares, sky-blue triangles, and cyan dots represent the results of cpog-gen, cpog-check, and EvalSSAT, respectively. We can see that generating and verifying certificates tend to take much longer when the formula is harder to compile. In addition, the partial verification is often substantially faster than the complete verification. This fact is helpful when certifying the lower bound suffices.

Fig. 4 compares the relative proof size between cpog-proofs and nnf-proofs. The proof sizes are measured in terms of the number of clauses. We calculate the clause number of nnf-proofs by the number of clauses in their corresponding

Table 2: Breakdown results of compilation and query time.
CNF encodings. It can be observed that cpog-proofs for the easy implication (marked in blue triangles) are almost linear in size with respect to their corresponding nnf-proofs, while cpog-proofs for the difficult implication (marked in orange dots) are typically larger than their corresponding nnf-proofs.

With cert-SSAT, we were able to discover a tricky bug of SharpSSAT in pure literal detection, which mistakenly negated the phase of pure literals. The bug was hard to find because it did not affect the reported satisfying probability but resulted in wrongly extracted strategies. This use case of cert-SSAT suggests its usefulness in validation solving results and enhancing solver reliability.

6 Conclusions and Future Work

We proposed a dec-DNNF-based knowledge compilation technique for SSAT. The technique was implemented based on the state-of-the-art SSAT solver SharpSSAT [Fan and Jiang, 2023], and the toolchain CPOG was extended to generate checkable proofs from the compiled results. Empirical results demonstrated its effectiveness for incremental solving of a wide range of benchmarks, and that checkable dec-DNNF logs can be generated with little computational overhead. Essentially, the cert-SSAT enhanced the trustworthiness of SharpSSAT results.

For future work, we remark that our implementation may serve as a general-purpose level-ordered dec-DNNF compiler, analogous to the dec-DNNF compilers based on #SAT solvers. It would be interesting to explore more applications of the compiler. In addition, the current supported incremental solving is restricted to reweighting and cofactoring queries. It might be worthwhile to explore other powerful techniques, such as clause-learning-based incremental QBF solving [Lonsing and Egly, 2014] and dynamic model counting [Li et al., 2006], to allow addition and deletion of literals and clauses. For cert-SSAT, we plan to make the entire toolchain formally verified with theorem provers. We also plan to extend it to support proof generation and validation for SSAT preprocessor and DSSAT solver [Cheng and Jiang, 2023].

Acknowledgements

This work was supported in part by the National Science and Technology Council of Taiwan under Grant NSTC 111-2923-E-002-013-MY3, 112-2119-M-002-017, and 113-2119-M-002-024.

Contribution Statement

C.C. and Y.R.L. contributed equally.

References

[Bryant et al., 2023] Randal E. Bryant, Wojciech Nawrocki, Jeremy Avigad, and Marijn J. H. Heule. Certified knowl-


