Unifying Decision and Function Queries in Stochastic Boolean Satisfiability

Yu-Wei Fan[1], **Jie-Hong R. Jiang**[1,2]

[1]Graduate Institute of Electronics Engineering, National Taiwan University, Taipei, Taiwan
[2]Department of Electrical Engineering, National Taiwan University, Taipei, Taiwan
{r11943096, jhjiang}@ntu.edu.tw

Abstract

Stochastic Boolean satisfiability (SSAT) is a natural formalism for optimization under uncertainty. Its decision version implicitly imposes a final threshold quantification on an SSAT formula. However, the single threshold quantification restricts the expressive power of SSAT. In this work, we enrich SSAT with an additional threshold quantifier, resulting in a new formalism SSAT(Θ). The increased expressiveness allows SSAT(Θ), which remains in the PSPACE complexity class, to subsume and encode the languages in the counting hierarchy. An SSAT(Θ) solver, `ClauSSat(Θ)`, is developed. Experiments show the applicability of the solver in uniquely solving complex SSAT(Θ) instances of parameter synthesis and SSAT extension.

Introduction

Stochastic Boolean satisfiability (SSAT) is a logical formalism, enabling natural characterization for optimizing decisions under uncertainty (Papadimitriou 1985). Due to its powerful expressiveness, recent endeavors have been made in both its efficient solving and potential applications. There are recent developments of specialized solvers (Lee, Wang, and Jiang 2017, 2018) designed for specific fragments of SSAT, and general solvers (Majercik and Boots 2005; Chen, Huang, and Jiang 2021; Wang et al. 2022; Fan and Jiang 2023) that place no restrictions on the SSAT formula. Regarding applications, SSAT has been used in encoding problems, such as contingent planning (Majercik and Littman 2003), partially observable Markov decision processes (POMDPs) (Salmon and Poupart 2020), the fairness analysis of machine learning models (Ghosh, Basu, and Meel 2021), and probabilistic graphical models (Hsieh and Jiang 2022).

Despite its expressiveness and broad applications, certain limitations are inherent to SSAT. For instance, SSAT implicitly imposes a linear ordering upon the dependency sets of existential variables according to the prefix. In (Lee and Jiang 2021), dependency stochastic Boolean satisfiability (DSSAT) (Lee and Jiang 2021) is formulated to allow explicit representation of dependency sets of existential variables. In this work, we tackle the limitations of SSAT from another aspect. Specifically, the decision version of SSAT enforces a single outermost threshold quantification on an SSAT formula, limiting its expressive power. Inspired by the counting quantifier in the Counting Hierarchy (Wagner 1986), we introduce a new formalism SSAT(Θ), which enriches SSAT with the threshold quantifier. We prove that SSAT(Θ) is in the PSPACE complexity class. Remarkably, while remaining in the same complexity class, the new formalism subsumes both SSAT and the languages in the Counting Hierarchy. Therefore, SSAT(Θ) can be powerful in encoding problems not succinctly expressible before.

To demonstrate its practical applications, we develop an SSAT(Θ) solver `ClauSSat(Θ)`, based on the state-of-the-art SSAT solver `ClauSSat` (Chen, Huang, and Jiang 2021). We further provide SSAT(Θ) encodings for probabilistic model checking and its extension specified in bounded probabilistic computation tree logic (BPCTL). To evaluate the SSAT(Θ) solver, experiments were conducted to study the applicability in solving application benchmarks and to investigate the effect of the threshold quantifier on solving efficiency. We note that since SSAT(Θ) subsumes the languages in the Counting Hierarchy, `ClauSSat(Θ)` is the first solver to tackle general counting formulas, while previous work has primarily concentrated on certain restricted fragments, such as counting formulas with only one (Chou et al. 2016) or two (Oztok, Choi, and Darwiche 2016) counting quantifiers.

The rest of this paper is organized as follows. After the preliminary background is first provided, we introduce the new formalism SSAT(Θ) and discuss its properties and complexity results. Then the extension of `ClauSSat` for SSAT(Θ) solving is to be elaborated. The encodings of the model-checking problem specified in BPCTL and the parameter synthesis problems are further detailed. We then investigate the efficiency of the solver on application benchmarks and the effect of threshold quantifiers on general SSAT(Θ) instances. Finally, we conclude this work and outline future work.

Preliminaries

In the sequel, the symbols "$\top$" and "$\bot$" represent Boolean values TRUE and FALSE, respectively. Boolean connectives "$\neg$," "$\vee$," "$\wedge$," "$\rightarrow$," and "$\leftrightarrow$" are associated with their conventional meanings. For simplicity, a conjunction $\wedge$ may be omitted in a Boolean formula. A *literal*, associated with a

variable v, is either the variable itself v or the negation $\neg v$. A *clause* is a conjunction of literals. A Boolean formula is in the *conjunctive normal form* (CNF) if it is a conjunction of clauses.

For a CNF formula ϕ, we use $\text{vars}(\phi)$ to denote the set of variables that appear in ϕ. A Boolean function, represented by a Boolean formula f over variables V, is a mapping $f : \mathbb{B}^{|V|} \to \mathbb{B}$. An *assignment* σ over the variables V, is a mapping $\sigma : V \to \mathbb{B}$. The *induced* formula over an assignment σ, denoted as $f|_\sigma$, is obtained by replacing each variable v with its assigned Boolean value $\sigma(v)$.

The interval notation $[l..u]$, for $l, u \in \mathbb{Z}^+ \cup \{0\}$ and $l < u$, represents the set of integers from l to u. We use symbols $\triangleright$ and $\bowtie$ to denote one of the predicates in the set $\{>, \geq\}$ and $\{>, \geq, <, \leq\}$, respectively.

Quantified Boolean and Counting Formulas

A quantified Boolean formula (QBF) in the prenex CNF form is expressed as

$$\mathcal{Q}.\phi\,, \tag{1}$$

where $\mathcal{Q} = Q_1, \ldots, Q_n$, for $Q_i \in \{\exists v_i, \forall v_i\}$, and ϕ is a quantifier-free CNF formula.

The QBF satisfiability well corresponds to the *Polynomial Hierarchy* (PH) (Stockmeyer 1976). Specifically, the language of QBFs with k quantifier alternations, denoted QBF_k, and $Q_1 = \exists$ (resp. $\forall$) is complete in the complexity class Σ^P_{k+1} (resp. Π^P_{k+1}). When k is unbounded, QBF is PSPACE-complete.

The *Counting Hierarchy* (CH) (Wagner 1986) for the counting problems is a complexity analogy to PH for the decision problems. Similar to QBFs, which well characterize PH, there are *counting formulas* (CFs), which well characterize CH. A counting formula in the prenex form can be expressed as

$$\mathsf{C}\, V_1, \ldots, \mathsf{C}\, V_n.\phi\,, \tag{2}$$

where C is the *counting quantifier*, V_i is a set of variables, and ϕ is a quantifier-free CNF formula. The quantification $\mathsf{C}\, V$ asks *whether there exist at least half of the assignments over* V. For singleton $V = \{v\}$, $\mathsf{C}\, V$ is equivalent to $\exists v$. Quantifier $\forall v$ can also be expressed by C by proper formula negation or rewriting. E.g., $\forall v.\phi$ can be rewritten as $\neg\, \mathsf{C}\, v.\neg\phi$, or $\mathsf{C}\, V.\phi \wedge v'$ for $V = \{v, v'\}$, where v' is a fresh new auxiliary variable. Therefore, a counting formula only requires quantifier C, without $\exists$ and $\forall$. The language of counting formulas of k levels of C quantifiers, denoted CF_k, is complete in the complexity class C^P_k. When k is unbounded, CF is PSPACE-complete.

Stochastic Boolean Satisfiability

An SSAT formula in the prenex form can be expressed by Eq. (1), but with the *prefix* $\mathcal{Q} = Q_1, \ldots, Q_n$ for $Q_i \in \{\exists v_i, \mathbb{R}^{p_i}\, v_i\}$ and the matrix ϕ being a quantifier-free CNF formula. In quantifier Q_i, variable v_i is either existential-quantified, i.e., $\exists v_i$, or random-quantified, i.e., $\mathbb{R}^{p_i}\, v_i$, denoting $v_i = \top$ (resp. $\bot$) with probability p_i (resp. $1 - p_i$). The semantics of an SSAT formula $\Phi = Q_1, \ldots, Q_n.\phi = Q_1.\Phi'$ is interpreted as its satisfying probability computed recursively by the following rules:

- $\Pr[\top] = 1$,
- $\Pr[\bot] = 0$,
- $\Pr[\exists v.\Phi'] = \max\{\Pr[\Phi'|_v], \Pr[\Phi'|_{\neg v}]\}$,
- $\Pr[\mathbb{R}^p\, v.\Phi'] = p \cdot \Pr[\Phi'|_v] + (1-p) \cdot \Pr[\Phi'|_{\neg v}]$.

Given an SSAT formula Φ, the *decision version* of SSAT, a PSPACE-complete problem, is to determine if $\Pr[\Phi]$ is greater than a threshold probability, whereas the *optimization (function) version* is to return the probability $\Pr[\Phi]$.

Note that the existential quantifier $\exists$ in SSAT differs from that in QBFs and counting formulas in its function sense, searching for an assignment maximizing the satisfying probability. More precisely, it is a "maximization quantifier." Nevertheless, we abuse the notation as its meaning should be clear from the context.

We remark that SSAT can incorporate the universal quantifier $\forall$ serving as the "minimization quantifier" (Littman, Majercik, and Pitassi 2001). Because $\forall$ quantification can be achieved through $\exists$ quantification and negation, e.g., $\Pr[\neg\exists X.\phi] = \Pr[\forall X.\neg\phi]$, we omit the universal quantifier in our discussion for simplicity.

Threshold Quantifier and SSAT

Before delving into the formal definition of SSAT(Θ), we motivate SSAT(Θ) by showing the limitations of SSAT. Consider the decision problem of the SSAT formula $\mathcal{Q}.\phi$ with a threshold probability p to assert

$$\Pr[\mathcal{Q}.\phi] \geq p, \tag{3}$$

which can be expressed as

$$\Theta^{\geq p}, \mathcal{Q}.\phi \tag{4}$$

by representing $\Pr[\Phi] \geq p$ as a *threshold quantifier* $\Theta^{\geq p}$ over $\Phi = \mathcal{Q}.\phi$. Hence, the SSAT decision problem equivalently imposes a single threshold quantification on the entire formula. It restricts both the number and the position of the threshold quantifier. The new formulation SSAT(Θ) relaxes such restrictions, allowing an arbitrary number of threshold quantifiers to be inserted in arbitrary positions in the prefix.

The language *SSAT(Θ)*, i.e., SSAT augmented with the threshold quantifier, is defined as follows.

Definition 1 (SSAT(Θ) Syntax)**.** The syntax of SSAT(Θ) is the same as that of SSAT except that $Q_i \in \{\exists v_i, \mathbb{R}^{p_i}\, v_i, \Theta^{\triangleright p_i}\}$, with the additional *threshold quantifier* $\Theta^{\triangleright p_i}$. We omit to specify the variables within the scope of the threshold quantifier $Q_i = \Theta^{\triangleright p_i}$ by implicitly assuming its inclusion of all the variables involved in $Q_{i+1}, \ldots, Q_n$.

Definition 2 (SSAT(Θ) Semantics)**.** Given an SSAT(Θ) formula $\Phi = Q_1, \ldots, Q_n.\phi = Q_1.\Phi'$, its *satisfying probability* is defined the same as that of SSAT except for the following additional rule for the Boolean interpretation of threshold quantifier:

- $\Theta^{\triangleright p}.\Phi' = \begin{cases} \top, & \text{if } \Pr[\Phi'] \triangleright p, \\ \bot, & \text{otherwise.} \end{cases}$

Example 1. *Consider the SSAT(Θ) formula*

$$\mathbb{R}^{0.4} x_1, \Theta^{\geq 0.4}, \exists y_1, \mathbb{R}^{0.3} x_2.\ (\bar{y}_1)(x_1 \vee y_1 \vee x_2).$$

We first consider the branch $x_1 = \top$. Since a threshold quantifier $\Theta^{\geq 0.4}$ is encountered, we check if the satisfying probability of the induced formula $\Phi' = \exists y_1, \mathfrak{R}^{0.3} x_2.(\bar{y}_1)$ is greater than or equal to 0.4. It can be checked that $\Pr[\Phi'] = 1$. Therefore, the threshold quantifier returns $\top$ under $x_1 = \top$. Similarly, one can check that under $x_1 = \bot$, the satisfying probability of formula $\Phi'' = \exists y_1, \mathfrak{R}^{0.3} x_2.(\bar{y}_1)(y_1 \vee x_2)$ is 0.3. Therefore, the threshold quantifier returns $\bot$ under $x_1 = \bot$. Finally, the satisfying probability of the entire formula is the weighted sum of the probability of the two branches of x_1, which is $(0 \cdot 0.6 + 1 \cdot 0.4) = 0.4$.

We note that SSAT(Θ) may incorporate the universal quantifier and threshold quantifier of predicates $\{<, \leq\}$. However, as we can rewrite them by negation along with the existential quantifier and threshold quantifier of predicates $\{>, \geq\}$, we omit them in our discussion. E.g., the formula $\Theta^{<p}, \exists x, \mathfrak{R}^{p'} y.\phi$ is equivalent to $\neg\, \Theta^{\geq p}, \exists x, \mathfrak{R}^{p'} y.\phi = \Theta^{\geq 1-p}, \forall x, \mathfrak{R}^{p'} y.\neg\phi$.

An SSAT(Θ) formula is called *closed* if all the variables are quantified, and *opened* otherwise. Also, an SSAT(Θ) formula $\Phi = \mathcal{Q}.\phi$ is called *pure counting* if its prefix $\mathcal{Q}$ consists of only threshold and random quantifiers, and has Q_1 being a threshold quantifier. We also called such prefix $\mathcal{Q}$ *pure counting*. Note that when the random quantifiers in a pure-counting SSAT(Θ) formula are of probability 0.5, Φ is equivalently a counting formula. E.g., the counting formula $\mathsf{C}\, X_1, \ldots, \mathsf{C}\, X_n.\phi$ is equivalent to the SSAT(Θ) formula $\Theta^{\geq 0.5}, \mathfrak{R}^{0.5}\, X_1, \ldots, \Theta^{\geq 0.5}, \mathfrak{R}^{0.5}\, X_n.\phi$. When the random quantifiers in a pure-counting SSAT(Θ) formula are of arbitrary probabilities expressed in binary fractional numbers, we can still derive its equivalent counting formula, exploiting the normalization technique that transforms SSAT formulas to have only probability 0.5 (Wang et al. 2022). We will show that the BPCTL model-checking problem can be naturally encoded with the pure-counting SSAT(Θ) formula.

Note that an opened and pure-counting SSAT(Θ) formula Φ with a set of free variables X represents a Boolean function $f(X)$ such that $f|_{\sigma_X} = \top$ if and only if $\Phi|_{\sigma_X} = \top$, where σ_X is an assignment over X. Unless otherwise stated, we assume that an SSAT(Θ) formula is closed in the sequel.

Note also that, unlike SSAT, SSAT(Θ) requires no distinction between the decision and function versions as the threshold-quantifier extension unifies the decision and function specification.

Properties and Normal Form

We study some properties of the threshold quantifier and exploit them for a normal form conversion.

Definition 3 (Normal Form of SSAT(Θ))**.** A prenex SSAT(Θ) formula $\Phi = Q_1, \ldots, Q_n.\phi$ is in a normal form if the following two conditions hold:

1. There are no consecutive threshold quantifiers. I.e., Q_i and Q_{i+1}, for $i \in [1..n-1]$, cannot be both threshold quantifiers.
2. A threshold quantifier Q_i, for $i \in [1..n-1]$, cannot be followed by an existential quantifier Q_{i+1}.

The normal form can be enforced for any prenex SSAT(Θ) formula due to Lemmas 1 and 2 stated below.

Given two threshold quantifiers $\Theta^{\triangleright p_1}$ and $\Theta^{\triangleright p_2}$, we say quantifier $\Theta^{\triangleright p_1}$ *dominates* $\Theta^{\triangleright p_2}$ if the implication

$$\Theta^{\triangleright p_2}.\Phi \rightarrow \Theta^{\triangleright p_1}.\Phi \tag{5}$$

holds for any SSAT(Θ) formula Φ. The following lemmas are immediate.

Lemma 1. *Let $\Theta^{\triangleright p_1}$ dominate $\Theta^{\triangleright p_2}$. Then the following equalities hold.*

$$\Theta^{\triangleright p_1}, \Theta^{\triangleright p_2}.\Phi = \Theta^{\triangleright p_2}, \Theta^{\triangleright p_1}.\Phi = \Theta^{\triangleright p_1}.\Phi \tag{6}$$

Lemma 2. *By treating $\top$ (resp. $\bot$) as probability value 1 (resp. 0), and vice versa, the following equality holds.*

$$\Theta^{\triangleright p}, \exists v.\Phi = \exists v, \Theta^{\triangleright p}.\Phi \tag{7}$$

Computation Complexity

Just as each level in PH forms a complete class, each level in CH forms a complete class, which can be characterized by adding the majority quantifier to QBF. E.g., E-MAJSAT (Littman, Goldsmith, and Mundhenk 1998) is $\vee\,\mathsf{C}$-complete (Wagner 1986), the same complexity class as $\mathrm{NP}^{\mathrm{PP}}$.

As the pure-counting SSAT(Θ) formula subsumes the counting formula, SSAT(Θ) can succinctly encode any problem in the counting hierarchy. On the other hand, while CH consists of only decision problems, SSAT(Θ) allows the encoding of function or optimization problems. Hence, SSAT(Θ) is strictly more expressive.

In the following, we give the computation complexity of the decision version of SSAT(Θ).

Theorem 1. *The decision problem of SSAT(Θ) is PSPACE-complete.*

Proof. We shall prove this by showing that the decision problem of SSAT(Θ) is in PSPACE and is PSPACE-hard.

Given an SSAT(Θ) formula Φ and a threshold probability p, the decision problem of SSAT(Θ) is to determine if $\Pr[\Phi]$ is greater than p. To prove it to be in PSPACE, it suffices to show that the algorithm for computing $\Pr[\Phi]$ requires polynomial spaces. Consider the algorithm to evaluate an SSAT(Θ) formula as shown in Algorithm 1. Let $S(i)$ be the space needed for the i^{th} recursive call, which equals the space needed within the i^{th} call plus the space needed for the $(i+1)^{\text{st}}$ recursive calls. Then the total space needed for the whole formula is $S(1)$. Suppose the number of clauses is m and the number of bits for storing a probability is b. For the base case in Line 3, we have to evaluate the truth value of ϕ over an assignment so the space needed $S(n)$ is $O(mn+b)$, where $O(mn)$ is the space needed for the induced formula and $O(b)$ is the space required for the probability. Now consider the i^{th} recursive call. For the case of threshold quantifier in Line 5, the space $S(i) = S(i+1)+O(b)$. For the case of existential and random quantifiers from Line 6 to Line 12, we reuse the space required for computing p_0 when computing p_1. Therefore, the space $S(i) = S(i+1)+O(mn+b)$. It turns out that the total space required $S(1)$ is $O(mn^2+bn)$.

The PSPACE-hardness of SSAT(Θ) is evident by the fact that it subsumes SSAT, which is PSPACE-hard. □

Algorithm 1: SSAT(Θ) Evaluation

```
procedure EVALUATE( Φ = Q_1, ..., Q_n.φ )
    if φ = ⊤ or φ = ⊥ then
        return Pr[φ]
    if Q_1 is Θ^{▷p} then
        return EVALUATE(Q_2, ..., Q_n.φ) ▷p
    v ← the outermost variable
    p_0 ← EVALUATE(Q_2, ..., Q_n.φ|_{¬v})
    p_1 ← EVALUATE(Q_2, ..., Q_n.φ|_v)
    if Q_1 is ∃v then
        return max{p_0, p_1}
    if Q_1 is ⅋^p v then
        return p · p_1 + (1 − p) · p_0
```

An SSAT(Θ) Solver

A naive reference procedure for SSAT(Θ) evaluation is shown in Algorithm 1. Although it requires only polynomial space, it may not be effective in run-time efficiency. Therefore, we need a more advanced algorithm to alleviate the intrinsic hardness of SSAT(Θ).

As SSAT(Θ) generalizes SSAT, existing SSAT solvers could be extended for SSAT(Θ). In this work, we consider the state-of-the-art general SSAT solvers `SharpSSAT` (Fan and Jiang 2023) and `ClauSSat` (Chen, Huang, and Jiang 2021) for such extension.

For the case of `SharpSSAT`, a direct extension is not possible unfortunately because the component decomposition property of SSAT (Salmon and Poupart 2020) does not hold for SSAT(Θ).

Lemma 3. *Given an SSAT(Θ) formula $\Phi = \mathcal{Q}.\phi$ with matrix $\phi = \phi_1 \wedge \ldots \wedge \phi_k$, where $\mathrm{vars}(\phi_i) \cap \mathrm{vars}(\phi_j) = \emptyset$ for $i \neq j$, let $\Phi_i = \mathcal{Q}.\phi_i$. Then the equality $\Pr[\Phi] = \prod_{i=1}^{k} \Pr[\Phi_i]$ does not hold in general.*

Proof. As a counterexample, consider the SSAT(Θ) formula

$$\Phi = \text{⅋}^{0.5}\, v_1, v_2, \Theta^{\geq 0.5}, \text{⅋}^{0.5}\, v_3, v_4.\ (v_1 \leftrightarrow v_3) \wedge (v_2 \leftrightarrow v_4)\,.$$

Its matrix can be decomposed into $\phi_1 \wedge \phi_2$, where $\phi_1 = (v_1 \leftrightarrow v_3)$ and $\phi_2 = (v_2 \leftrightarrow v_4)$ have disjoint support variables. However, $\Pr[\Phi] = 0 \neq \Pr[\Phi_1] \cdot \Pr[\Phi_2] = 0.5 \cdot 0.5 = 0.25$. □

For the case of `ClauSSat`, its extension to SSAT(Θ) solving is possible, as discussed below.

`ClauSSat` solves an SSAT formula by partitioning the literals in a clause in the matrix into several groups with respect to the quantification levels according to the prefix.

For the extension to SSAT(Θ), given an SSAT(Θ) formula Φ, if the outermost quantifier is a threshold quantifier $\Theta^{\triangleright p}$, we omit it and solve the remaining formula Φ'. (Otherwise, we solve Φ directly.) Once $\Pr[\Phi']$ is computed, we simply check if $\Pr[\Phi'] \triangleright p$ and return the corresponding truth or falsity. Moreover, we modify the definition of quantification level as follows: Consider the formula $Q_1, \ldots, Q_n.\phi$. For a random- or existential- quantified variable at quantification Q_i, let k be the number of the encountered alternations in one of the forms ∃-⅋, ⅋-∃, and Θ-⅋, when traversing from Q_1 to Q_i. Then the quantification level of that variable is defined as $k + 1$. E.g., for the formula $\text{⅋}\, x_1, \exists x_2, \text{⅋}\, x_3, \Theta, \text{⅋}\, x_4.\phi$, the quantification levels of x_1, x_2, x_3, and x_4 are 1, 2, 3, and 4, respectively.

The literals in a clause in the matrix are then partitioned into groups with respect to the newly defined quantification levels. The formulas are then solved recursively on the quantification levels in the same way as SSAT, except that for the random quantifier Q_i with an outer threshold quantifier $Q_{i-1} = \Theta^{\triangleright p}$, the probability should be mapped to a Boolean value according to $\Theta^{\triangleright p}$.

To effectively prune the search space, `ClauSSat` incorporates several pruning techniques for the alternations ∃-⅋ and ⅋-∃. To handle these techniques, we follow the same implementation as `ClauSSat` when encountering the ∃-⅋ and ⅋-∃ alternations. For the ∃-Θ-⅋ alternation, as stated in Lemma 2, since the probability is preserved under the reordering between the threshold and existential quantifiers, the techniques for ∃-⅋ are applicable in ∃-Θ-⅋. For the ⅋-Θ-⅋ alternation, we disable all the pruning techniques proposed in `ClauSSat`.

Encoding Application Problems

This section first gives some background on discrete-time Markov Chains (DTMCs) and bounded probabilistic computation tree logic (BPCTL). We then show how to encode the model-checking problem with pure-counting formulas. Further, we introduce the parameter synthesis problem and demonstrate its SSAT(Θ) encoding.

DTMCs and BPCTL

Discrete-Time Markov Chains A *discrete-time Markov chain* can be viewed as a state transition system where each transition takes place with a transition probability. It is a common model used to describe the behavior of a probabilistic system.

Definition 4 (Discrete-Time Markov Chain.)**.** A discrete-time Markov Chain is a tuple $\mathcal{M} = (\mathcal{S}, s_0, \mathbf{P}, \mathcal{AP}, \mathcal{L})$, where

- $\mathcal{S}$ is a finite set of states, and $s_0 \in \mathcal{S}$ is an initial state,
- $\mathbf{P}$ is a probabilistic matrix characterizes the transition probabilities, where $\mathbf{P} : \mathcal{S} \times \mathcal{S} \to [0, 1]$ and $\Sigma_{s' \in \mathcal{S}} \mathbf{P}(s, s') = 1$ for each $s \in \mathcal{S}$,
- $\mathcal{AP}$ is a set of atomic propositions, and
- $\mathcal{L} : \mathcal{S} \to 2^{\mathcal{AP}}$ is a labelling function for states.

A *k-path* π in a DTMC is a finite sequence of states $(s^1, \cdots, s^k)$ with length k. We use $\pi(i)$ to denote the state s^i. The set of all the k-paths starting with a state s is denoted as $Path_k(s)$. For simplicity, we use *path* to refer to a k-path in the sequel. The probability of a path π, denoted as $\Pr[\pi]$, is $\Pi_{i=1}^{k-1} \mathbf{P}(\pi(i), \pi(i+1))$.

Bounded Probabilistic Computation Tree Logic Given a probabilistic system and a temporal property, the problem of probabilistic model checking is to check if the system satisfies the given property. In this work, the considered system description is the DTMC, and the concerned properties are specified in the *bounded probabilistic computation tree logic*

(BPCTL), which is a bounded-step fragment of the probabilistic computation tree logic. The syntax of BPCTL is defined as follows.

Definition 5 (BPCTL Syntax)**.** The *state formula*, denoted as φ, and *path formula*, denoted as ψ, are the two main components of a BPCTL formula with the following definition:

$$\begin{aligned}\varphi &:= \top \mid a \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \mathbb{P}^{\bowtie p}[\psi] \\ \psi &:= \mathbb{F}^{\leq k}\varphi \mid \varphi_1 \mathbb{U}^{\leq k}\varphi_2 \mid \mathbb{X}\varphi\end{aligned}$$

where a is an atomic proposition, and $\mathbb{F}$, $\mathbb{U}$, $\mathbb{X}$, and $\mathbb{P}$ are the *future*, *until*, *next*, and *probability* operators, respectively (Ciesinski and Größer 2004). A BPCTL formula is a state formula.

The semantics of BPCTL can then be defined with respect to a DTMC as follows.

Definition 6 (BPCTL Semantics)**.** Let the DTMC $\mathcal{M} = (\mathcal{S}, s_0, \mathbf{P}, \mathcal{AP}, \mathcal{L})$. The semantics of the satisfaction relation are defined by:

$\mathcal{M}, s \vDash \top$	for all $s \in \mathcal{S}$,
$\mathcal{M}, s \vDash a$	if $a \in \mathcal{L}(s)$,
$\mathcal{M}, s \vDash \neg\varphi$	if $\mathcal{M}, s \nvDash \varphi$,
$\mathcal{M}, s \vDash \varphi_1 \wedge \varphi_2$	if $\mathcal{M}, s \vDash \varphi_1$ and $\mathcal{M}, s \vDash \varphi_2$,
$\mathcal{M}, s \vDash \mathbb{P}^{\bowtie p}[\psi]$	if $\Sigma_{\pi \in Path(s), \mathcal{M}, \pi \vDash \psi} \Pr[\pi] \bowtie p$,
$\mathcal{M}, \pi \vDash \mathbb{F}^{\leq k}\varphi$	if $\exists i \in [1..k] : \mathcal{M}, \pi(i) \vDash \varphi$,
$\mathcal{M}, \pi \vDash \varphi_1 \mathbb{U}^{\leq k}\varphi_2$	if $\exists i \in [1..k] : \mathcal{M}, \pi(i) \vDash \varphi_2$ and $\mathcal{M}, \pi(j) \vDash \varphi_1 \ \forall j \in [1..i-1]$, and
$\mathcal{M}, \pi \vDash \mathbb{X}\varphi$	if $\mathcal{M}, \pi(2) \vDash \varphi$.

With Definition 6, we say the DTMC $\mathcal{M}$ satisfies φ for $\mathcal{M}, s_0 \vDash \varphi$, and say state s satisfies φ for $\mathcal{M}, s \vDash \varphi$. In the sequel, we refer to the model-checking problem of a DTMC specified in BPCTL as model checking. We note that when the outermost operator in φ is the $\mathbb{P}$-operator, the state-of-the-art probabilistic model checkers, e.g., `Prism` (Kwiatkowska, Norman, and Parker 2002) and `Storm` (Dehnert et al. 2017), provide the option $\mathbb{P}^{=?}$ to return a probability value rather than a Boolean value. We allow such an extension in the following SSAT(Θ) encoding of model checking.

BPCTL Model Checking for DTMC

The main idea is to somehow map the $\mathbb{P}$-operator in BPCTL to the threshold quantifier. In consequence, a BPCTL formula nested with the $\mathbb{P}$-operator would correspond to a pure-counting SSAT(Θ) formula. Before delving into the details, we shall first transform φ into the form where only $\mathbb{P}^{>p}$ and $\mathbb{P}^{\geq p}$ are allowed for the $\mathbb{P}$-operator. This can be done by replacing each $\mathbb{P}^{<p}$ and $\mathbb{P}^{\leq p}$ with $\neg\mathbb{P}^{\geq p}$ and $\neg\mathbb{P}^{>p}$, respectively. In the following, we assume that a DTMC $\mathcal{M} = (\mathcal{S}, s_0, \mathbf{P}, \mathcal{AP}, \mathcal{L})$ and a BPCTL formula φ of the mentioned form are given. Also, given an atomic proposition a and any state s, let F_a be the Boolean function such that $F_a(s) = \top$ if and only if $a \in \mathcal{L}(s)$.

BPCTL Encoding For readability, we assume the state space $\mathcal{S} = [0..|\mathcal{S}| - 1]$ and let $n = \lfloor \log_2 |S| \rfloor + 1$. We refer the *state variables* to a vector of n-bits Boolean variables $X = (x_1, \cdots, x_n)$, who takes integer values in $\mathcal{S}$. We introduce one new random-quantified variable for each transition (s, s'), denoted as $x_{r(s,s')}$, where $x_{r(s,s')} = \top$ with probability $\mathbf{P}(s, s')$. We let the set of the allocated random-quantified variables be $X_r = \{x_{r(s,s')} \mid s, s' \in \mathcal{S},\ \mathbf{P}(s, s') > 0\}$.

To derive an equivalent SSAT(Θ) formula, we recursively define the opened pure-counting formulas for both the state formula and path formula. In order for that, we need a *probabilistic transition relation* $T(X_s, X_{s'})$ over the current state variables X_s and next state variables $X_{s'}$. Intuitively, given a current state s and a next state s', we want the satisfying probability $\Pr[T(s, s')] = \mathbf{P}(s, s')$. The following is the considered probabilistic transition relation $T(X_s, X_{s'})$:

$$\begin{aligned}\bar{\exists}\, X_r.\quad & \textstyle\bigwedge_{s,s' \in \mathcal{S}}((X_s = s \wedge X_{s'} = s') \rightarrow x_{r(s,s')}) \wedge \\ & \textstyle\bigvee_{s,s' \in \mathcal{S}, \mathbf{P}(s,s') > 0}(X_s = s \wedge X_{s'} = s')\end{aligned} \tag{8}$$

With Eq. (8), we can expand time-frame for a k-path through proper variable-renaming as follows:

$$T^{k-1}(X_{s^1}, \cdots, X_{s^k}) = \bigwedge_{i=1}^{k-1} T(X_{s^i}, X_{s^{i+1}}), \tag{9}$$

where the state variables X_{s^i} corresponds to the i^{th} state in the path. Given a path $(s^1, \cdots, s^k)$, $\Pr\big[T^{k-1}(s^1, \cdots, s^k)\big]$ equals $\Pi_{i=1}^{k-1}\mathbf{P}(s^i, s^{i+1})$.

We use C_Ψ to represent the opened pure-counting formula of Ψ, either a state formula or a path formula. Recall that we already transformed the given BPCTL formula with only $\mathbb{P}^{\triangleright p}$ for the $\mathbb{P}$-operator. For the rules in Definition 5, we derive their corresponding opened pure-counting formulas:

$$\begin{aligned}C_\top(X_s) &= \top, \\ C_a(X_s) &= F_a(X_s), \\ C_{\neg\varphi}(X_s) &= \neg C_\varphi(X_s), \\ C_{\varphi_1 \wedge \varphi_2}(X_s) &= C_{\varphi_1}(X_s) \wedge C_{\varphi_2}(X_s), \\ C_{\mathbb{P}^{\triangleright p}[\psi]}(X_s) &= \Theta^{\triangleright p/2^t}\, C_\psi(X_s), \\ C_{\mathbb{F}^{\leq k}\varphi}(X_s) &= \bar{\exists}^{0.5}\, X_{s^2}, \ldots, \bar{\exists}^{0.5}\, X_{s^{k+1}}. \\ &\quad\ T^k(X_s, X_{s^2}, \cdots, X_{s^{k+1}}) \wedge \\ &\quad\ (C_\varphi(X_s) \vee \textstyle\bigvee_{i=2}^{k+1} C_\varphi(X_{s^i})), \\ C_{\varphi_1 \mathbb{U}^{\leq k}\varphi_2}(X_s) &= \bar{\exists}^{0.5}\, X_{s^2}, \ldots, \bar{\exists}^{0.5}\, X_{s^{k+1}}. \\ &\quad\ T^k(X_s, X_{s^2}, \cdots, X_{s^{k+1}}) \wedge \\ &\quad\ (C_{\varphi_2}(X_s) \vee (C_{\varphi_1}(X_s) \wedge \\ &\quad\ \textstyle\bigvee_{i=2}^{k+1}(C_{\varphi_2}(X_{s^i}) \wedge \bigwedge_{j=2}^{i-1} C_{\varphi_1}(X_{s^j})))), \\ C_{\mathbb{X}\varphi}(X_s) &= \bar{\exists}^{0.5}\, X_{s'}.T(X_s, X_{s'}) \wedge C_\varphi(X_{s'}).\end{aligned} \tag{10}$$

Intuitively, for a state formula φ and a state s, $C_\varphi(s) = \top$ if and only if s satisfies φ. For a path formula ψ, $\Pr[C_\psi(s)]$ gives the sum of the probabilities of the paths in $Path_k(s)$ satisfying ψ. In the construction for the $\mathbb{P}$-operator, the factor $1/2^t$, with t being the number of the extra quantified state variables in the construction of the outermost opened pure-counting formula, is a scaling factor for the original threshold p since we use probability 0.5 for each state variable in the construction. On the other hand, in the constructions

involving multiple opened pure-counting formulas, namely, $C_{\varphi_1 \wedge \varphi_2}$, $C_{\mathbb{F}^{\leq k}\varphi}$, and $C_{\varphi_1 \mathbb{U}^{\leq k}\varphi_2}$, the quantified variables for transition relations have to be duplicated among the formulas so that their quantified variables are disjoint.

The entire working flow is as follows: We first derive the opened pure-counting formula $C_\varphi(X_s)$ of the BPCTL formula φ and then transform it into an equivalent SSAT(Θ) in the prenex form $\Theta, \mathbb{Q}, \ldots, \Theta, \mathbb{Q}\,.\phi$ with the rules stated in the following three propositions.

Proposition 1 (Conjunction). *Let $\Phi_1 = \Theta^{\triangleright p_1}, \mathbb{Q}\, Y, \mathcal{Q}_1.\phi_1$ and $\Phi_2 = \Theta^{\triangleright p_2}, \mathcal{Q}_2.\phi_2$ be two pure-counting formulas, possibly with a common set of free variables X. Also, let $\mathcal{Q}_1$ be pure counting. If all the variables in Y are not quantified in Φ_2, then*

$$\Phi_1 \wedge \Phi_2 \leftrightarrow \Theta^{\triangleright p_1}, \mathbb{Q}\, Y.((\mathcal{Q}_1.\phi_1) \wedge (\Theta^{\triangleright p_2}, \mathcal{Q}_2.\phi_2)).$$

Proof. We show that under any assignment over X, the left-hand side (LHS) $\Phi_1 \wedge \Phi_2$ is equivalent to the right-hand side (RHS) $\Theta^{\triangleright p_1}, \mathbb{Q}\, Y.((\mathcal{Q}_1.\phi_1) \wedge (\Theta^{\triangleright p_2}, \mathcal{Q}_2.\phi_2))$. Let Φ' and Φ'' be the LHS formula $\Phi_1|_{\sigma_X} \wedge \Phi_2|_{\sigma_X}$ and the RHS formula $\Theta^{\triangleright p_1}, \mathbb{Q}\, Y.((\mathcal{Q}_1.\phi_1|_{\sigma_X}) \wedge (\Theta^{\triangleright p_2}, \mathcal{Q}_2.\phi_2|_{\sigma_X}))$, respectively, induced under assignment σ_X.

To show $\Phi' \rightarrow \Phi''$, we have $\Phi' = \top$ if and only if

$$\Theta^{\triangleright p_1}, \mathbb{Q}\, Y, \mathcal{Q}_1.\phi_1|_{\sigma_X} = \top \tag{11}$$

and

$$\Theta^{\triangleright p_2}, \mathcal{Q}_2.\phi_2|_{\sigma_X} = \top. \tag{12}$$

We have $\Phi'' = \Theta^{\triangleright p_1}, \mathbb{Q}\, Y.\mathcal{Q}_1.\phi_1|_{\sigma_X}$ by Eq. (12), and thus $\Phi'' = \top$ by Eq. (11).

To show $\Phi' \leftarrow \Phi''$, we have $\Phi'' = \top$ only if Eq. (12) holds. Otherwise, $\Phi'' = \Theta^{\triangleright p_1}, \mathbb{Q}\, Y.((\mathcal{Q}_1.\phi_1|_{\sigma_X}) \wedge \bot) = \bot$, leading to a contradiction. It follows that $\Phi'' = \Theta^{\triangleright p_1}, \mathbb{Q}\, Y.\mathcal{Q}_1.\phi_1|_{\sigma_X}$. Under the assumption $\Phi'' = \top$, Eq. (11) must hold. □

Proposition 2 (Disjunction). *Let $\Phi_1 = \Theta^{\triangleright p_1}, \mathbb{Q}\, Y, \mathcal{Q}_1.\phi_1$ and $\Phi_2 = \Theta^{\triangleright p_2}, \mathcal{Q}_2.\phi_2$ be two pure-counting formulas, possibly with a common set of free variables X. Also, let $\mathcal{Q}_1$ be pure counting. If all the variables in Y are not quantified in Φ_2, then*

$$\Phi_1 \vee \Phi_2 \leftrightarrow \Theta^{\triangleright p_1}, \mathbb{Q}\, Y.((\mathcal{Q}_1.\phi_1) \vee (\Theta^{\triangleright p_2}, \mathcal{Q}_2.\phi_2)).$$

Proof. The proof is similar to that of Proposition 1, with "$\wedge$" being replaced by "$\vee$". □

Proposition 3 (Negation). *Let $\Phi = \Theta^{\triangleright p}, \mathbb{Q}\, Y, \mathcal{Q}.\phi$ be a pure-counting formula, possibly with free variables X. Also, let $\mathcal{Q}$ be pure counting. Then*

$$\neg\Phi \leftrightarrow \Theta^{\triangleright(1-p)}, \mathbb{Q}\, Y, \neg\mathcal{Q}.\phi.$$

Proof. We prove the case when there is only one variable y in Y. The proof can be generalized to a set Y of variables. Without loss of generality, we assume $y = \top$ with probability p'. We show that under any assignment σ_X, the LHS $\neg\Phi$ is equivalent to the RHS $\Theta^{\triangleright(1-p)}, \mathbb{Q}^{p'}\, y, \neg\mathcal{Q}.\phi$. We only consider the case with the outermost threshold quantifier being $\Theta^{>p}$ as the case of $\Theta^{\geq p}$ is similar. Let $\phi' = \phi|_{\sigma_X}$, $\Phi' = \neg\Phi|_{\sigma_X}$, and $\Phi'' = \Theta^{>(1-p)}, \mathbb{Q}^{p'}\, y, \neg\mathcal{Q}.\phi'$, induced under assignment σ_X. We have

$$\begin{aligned}
\Phi' &= \neg(\Pr\left[\mathbb{Q}^{p'}\, y, \mathcal{Q}.\phi'\right] > p) \\
&= \Pr\left[\mathbb{Q}^{p'}\, y, \mathcal{Q}.\phi'\right] \leq p \\
&= (\Pr[\mathcal{Q}.\phi'|_y] \cdot p' + \Pr[\mathcal{Q}.\phi'|_{\neg y}] \cdot (1-p')) \leq p \\
&= ((1 - \Pr[\mathcal{Q}.\phi'|_y]) \cdot p' + \\
&\quad (1 - \Pr[\mathcal{Q}.\phi'|_{\neg y}]) \cdot (1-p')) > 1 - p \\
&= (\Pr[\neg\mathcal{Q}.\phi'|_y] \cdot p' + \\
&\quad \Pr[\neg\mathcal{Q}.\phi'|_{\neg y}] \cdot (1-p')) > 1 - p \\
&= \Theta^{>(1-p)}, \mathbb{Q}^{p'}\, y.\neg\mathcal{Q}.\phi' \\
&= \Phi''.
\end{aligned}$$

□

Finally, the resulting SSAT(Θ) formula is derived by assigning the state variables X_s to the initial state s_0 and converting the matrix into a CNF formula by Tseitin transformation:

$$\mathcal{Q}, \exists X_D.\phi'\,, \tag{13}$$

where ϕ' is the CNF formula, X_D is the set of extra definition variables introduced by the Tseitin transformation, and $\mathcal{Q}$ is pure counting.

Recall that state-of-the-art probabilistic model checkers allow $\mathbb{P}^{=?}$ when the outermost operator is the $\mathbb{P}$-operator. To handle this case, we can simply remove the outermost threshold quantifier of $\mathcal{Q}$, yielding $\mathcal{Q}'$, say, in Eq. (13). The resulting formula is $\mathcal{Q}', \exists X_D.\phi'$.

Parameter Synthesis with SSAT(Θ)

In most applications, the DTMCs are parameterized by several parameters. The probability of a certain BPCTL property is also dependent on the parameters. Take the `Crowds` (Shmatikov 2004) protocol for message transmission, which provides a probabilistic guarantee of the anonymity of the sender. The protocol assumes there are certain numbers of *good people* and *bad people*, where the good people cannot communicate with each other and the bad people would cooperate with each other. To provide the probabilistic anonymity guarantee, the message is passed to a randomly chosen bad person with probability p and to a randomly chosen good person with probability $1 - p$. The only method the bad people can acquire information about the real sender is to observe the identity of the one who passed the message to the bad people. One important parameter is the size of the *good people* or called *crowd size*, which affects the probabilistic bound of anonymity — the greater the crowd size the better anonymity `Crowds` can provide. Under such DTMC, we ask the *parameter synthesis* problem: Given a DTMC and a BPCTL property, how to determine the size of the DTMC, in terms of the parameters, so that the property is maximized?

Note that as long as we have the transition relation of the parameterized DTMC, the parameter synthesis problem can be naturally encoded using SSAT(Θ). We first construct

the transition relation of each instantiated DTMC with a certain parameter value. Suppose the set of variables for the parameter is X_q, which can take values from the finite set of range R. Suppose the maximum value in R is $r_{\max}$ and the minimum value in R is $r_{\min}$. We introduce $\lfloor \log_2 (r_{\max} - r_{\min} + 1) \rfloor + 1$ variables for X_q. We use T_q to denote the transition relation of DTMC with the parameter value parameter being $q \in R$. Then the parametric transition relation is constructed as follows:

$$T(X_q, X_s, X_{s'}) = \bigwedge_{q \in R} ((X_q = q) \rightarrow T_q). \tag{14}$$

We follow the same BPCTL encoding procedure in the previous subsection except that we use the parametric transition relation in Eq. (14) instead. Suppose we have the SSAT(Θ) formula in the form in Eq. (13). Then the parameter synthesis problem can be encoded as:

$$\exists X_q, \text{ꓤ}\, X, \mathcal{Q}, \exists X_D . \phi' \wedge \bigwedge_{q \in R} (X_q = q)\,, \tag{15}$$

where $\mathcal{Q}$ is pure counting and X are some outermost state variables.

Finally, we remark that both the BPCTL model checking and the parameter synthesis problems cannot be naturally encoded as regular SSAT since they involve multiple threshold operations.

Experimental Results

We implemented our SSAT(Θ) solver, named `ClauSSat` (Θ),[1] in the C++ language by extending `ClauSSat` (Chen, Huang, and Jiang 2021). We note that we also implemented Algorithm 1 as a baseline for comparison. However, since the baseline version solved none of the instances in our experiments, its results are not included in our discussion. The experiments were conducted on a Linux machine with 2.2 GHz Intel Xeon CPU and 128 GB RAM. Two benchmark sets were experimented for evaluation. The first set includes instances from the case study of parameter synthesis on the DTMC `Crowds` (Shmatikov 2004) protocol.[2]

The second set includes instances converted from SSAT formulas. A 1000-second time limit was imposed on solving each instance.

Evaluation on Instances of Parameter Synthesis

To create the benchmark instances of parameter synthesis, we adopted the `Crowds` (Shmatikov 2004) protocol.

Range	k	Formula		Run Time (s)
		#Cls	#Vars	
[2..4]	1	6112	2203	9.13
	2	6763	2428	0.07
	3	8697	3113	83.78
	4	9318	3337	554.75
	5	10093	3626	TO
[2..6]	1	6923	2489	25.00
	2	9961	3541	277.54
	3	10561	3761	394.02
	4	11625	4139	658.14
	5	11786	4207	TO

Table 1: Results on instances of parameter synthesis.

A DTMC model was first specified in the prism model format (Kwiatkowska, Norman, and Parker 2002), then converted to a transition relation in a bit-vector form of the QF-BV SMT format, then further bit-blasted using the SMT solver `Boolector` (Niemetz, Preiner, and Biere 2014).[3] Given the BPCTL property to be checked and the transition relation, the SSAT(Θ) formula was created based on the proposed encoding method.

Recall that `Crowds` is a protocol for providing anonymity of the actual sender. The protocol assumes that there is only one actual sender among t potential senders, for t being the crowd size. We define the "safe$_i$" property as follows.

$$\text{safe}_i = \mathbb{P}^{>0.5}[\mathbb{F}^{\leq m}(\text{observe}_i < 1)]\,, \tag{16}$$

where observe$_i$ is an integer state variable and $(\text{observe}_i < 1)$ asserts the actual sender i is not observed. It asserts that the probability that the adversary does not observe the actual sender i in the future m-steps is greater than 0.5.

We say that the system is in a safe state for sender i if the system state satisfies property *safe$_i$*.[4] For the BPCTL property to be checked, we extend the *safe$_i$* property to

$$k\text{-step safe}_i = \mathbb{P}^{=?}[\mathbb{F}^{\leq k}(\text{safe}_i)]\,. \tag{17}$$

Thereby, with Eq. (15), the problem is to search for a crowd size value such that the *k-step safe$_i$* property is maximized starting from the initial state of the DTMC. In the experiment, we checked "safe$_0$" under $m = 5$. All the created formulas share the same prefix form $\exists$-ꓤ-Θ-ꓤ-$\exists$.

The results are shown in Table 1, where columns "Range," "k," and "Run Time (s)" report the range of the parameter values, k of the k-step safe property, and the time spent for solving each case, respectively, and the numbers of clauses and variables of the SSAT(Θ) formula are reported in columns "#Cls" and "#Vars," respectively. The cases that

[1] Available at https://github.com/NTU-ALComLab/ClauSSat-Theta.

[2] We note that although BPCTL model checking for DTMC can be encoded in SSAT(Θ), the converted instances are often too large to be solved. For this problem, there are dedicated model checkers, such as `Prism` (Kwiatkowska, Norman, and Parker 2002), `Storm` (Dehnert et al. 2017), and `Epmc` (Hahn et al. 2014), for more direct and effective solving. Hence, we focused on evaluating instances with complex queries that cannot be handled by existing model checkers.

[3] We disabled the `-vs` option in `Boolector` as it may perform aggressive SAT-based encoding that might not be sound in SSAT(Θ). The instances were only simplified using Boolean constraint propagation.

[4] The safe$_i$ property can be encoded as an $\exists$-ꓤ-$\exists$ quantified SSAT formula.

Family	Instance	#R	Formula		Run Time (s)	
			#Cls	#Vars	SSAT	SSAT(Θ)
tlc	depth-7	2	7491	2809	0.56	0.47
	depth-8	2	8430	3160	0.73	0.61
	depth-9	2	9360	3511	0.92	0.78
gttt	2_2_0010	9	2958	1133	10.05	26.85
	2_2_001020	9	2814	1085	144.16	310.16
	2_2_000111	9	3214	1165	145.36	298.59
Robot	8	39	30444	8892	41.97	61.14
	9	45	34078	9880	50.32	34.35
	10	50	37712	10868	63.78	286.01
stracomp	25.9	25	2278	779	0.45	309.74
	30.9	30	2754	934	0.38	TO
	35.9	35	3227	1089	0.37	TO

Table 2: Results of evaluation on general SSAT(Θ) instances converted from SSAT instances.

failed to be solved within the time limit are denoted with "TO." Unsurprisingly, the running time increases drastically as the k increases since the transition relation and the variables have to be duplicated, as mentioned in the previous section. When comparing the solving time of the two different configurations of range, we notice that the formulas failed to be solved within the time limit when $k = 5$ for both configurations. We observed that for the $[2..4]$ configuration, the run time seems to increase more acutely as k increases than the $[2..6]$ configuration. As two (resp. three) existential variables are allocated (for X_p mentioned in the subsection of Parameter Synthesis) for the range $[2..4]$ (resp. $[2..6]$), it would be interesting to investigate the effect of the number of these outermost existential variables on the solving efficiency for the parameter synthesis benchmarks.

Finally, we remark that although the current SSAT(Θ) solver can solve most of the generated instances, the step size k and the range size of the parameter values are relatively small, and the efficiency is sensitive to the increase of the range size. In order to alleviate the high computation complexity, it may be crucial to develop specialized SSAT(Θ) solver and preprocessing techniques.

Evaluation on Instances of SSAT Extension

To study the impact of the threshold quantifier on SSAT instances, we assess the solver's efficiency by evaluating SSAT(Θ) instances generated from existing SSAT benchmarks, taken from (Chen, Huang, and Jiang 2021).[5] The conversion was done by randomly selecting a random quantification block $\exists V$, bi-partitioning variables V into V_1 and V_2, and inserting a randomly generated threshold quantifier $\Theta^{\triangleright p}$ so that the quantification becomes $\exists V_1, \Theta^{\triangleright p}, \exists V_2$. The bi-partition of variables V is also determined randomly. Specifically, we selected four families of benchmarks where all the cases in each family are solved by `ClauSSat` within the time limit. To examine the effect of the threshold quantifier, we compare the performance on the SSAT(Θ) instances with that on the original SSAT instances. The results are shown in Table 2, where three representative instances for each family are listed due to space limit. The column "#R" represents the original number of random-quantified variables in the inserted quantification and the column "SSAT" (resp. "SSAT(Θ)") reports the time spent on solving the SSAT (resp. SSAT(Θ)) instance.

Comparing the performance on SSAT and SSAT(Θ) instances, we observed that in most cases within the families, inserting threshold quantifiers deteriorates the performance as expected. It is expected because the threshold quantifier increases the number of quantification levels, and the current implementation does not equip efficient pruning techniques for the alternation ∃-Θ-∃. Also, observe that the number of random-quantified variables may play a role in affecting the run time due to the increased complexity introduced by the threshold quantifier. This effect is especially significant in the family `stracomp`, where all the listed SSAT instances can be solved in less than one second, while their SSAT(Θ) counterparts fail to be solved within the time limit as the number of random-quantified variables attains or exceeds 30. However, there is an exception in instance 9 of the `Robot` family, whose SSAT(Θ) instance, in contrast, takes less time to be solved. Some other factors contributing to the hardness of computation remain to be further investigated.

Conclusions and Future Work

This work presents SSAT(Θ), which unifies the decision and function queries of SSAT by augmenting SSAT with the threshold quantifier. SSAT(Θ) subsumes the counting formulas and can encode problems in the Polynomial and Counting Hierarchies. For a practical case study, we encode BPCTL model checking and the parameter synthesis problem of DTMCs into SSAT(Θ). Experiments demonstrate the feasibility of extending `ClauSSat` for solving SSAT(Θ) instances. For future work, we plan an extension to DSSAT (Lee and Jiang 2021).

[5] Available at https://github.com/NTU-ALComLab/ClauSSat.

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.

References

Chen, P.-W.; Huang, Y.-C.; and Jiang, J.-H. R. 2021. A Sharp Leap from Quantified Boolean Formula to Stochastic Boolean Satisfiability Solving. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 3697–3706.

Chou, Y.-M.; Chen, Y.-C.; Wang, C.-Y.; and Huang, C.-Y. 2016. MajorSat: A SAT Solver to Majority Logic. In *Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC)*, 480–485.

Ciesinski, F.; and Größer, M. 2004. *On probabilistic computation tree logic*, volume 2925. Springer.

Dehnert, C.; Junges, S.; Katoen, J.-P.; and Volk, M. 2017. A Storm Is Coming: A Modern Probabilistic Model Checker. In *Proceedings of the International Conference on Computer Aided Verification (CAV)*, 592–600.

Fan, Y.-W.; and Jiang, J.-H. R. 2023. SharpSSAT: A Witness-Generating Stochastic Boolean Satisfiability Solver. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 3949–3958.

Ghosh, B.; Basu, D.; and Meel, K. S. 2021. Justicia: A Stochastic SAT Approach to Formally Verify Fairness. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 7554–7563.

Hahn, E. M.; Li, Y.; Schewe, S.; Turrini, A.; and Zhang, L. 2014. iscasMc: A Web-Based Probabilistic Model Checker. In *Proceedings of the International Symposium of Formal Methods (FM)*, 312–317.

Hsieh, C.-H.; and Jiang, J.-H. R. 2022. Encoding Probabilistic Graphical Models into Stochastic Boolean Satisfiability. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 1834–1842.

Kwiatkowska, M.; Norman, G.; and Parker, D. 2002. PRISM: Probabilistic symbolic model checker. In *Proceedings of the International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*, 200–204.

Lee, N.-Z.; and Jiang, J.-H. R. 2021. Dependency Stochastic Boolean Satisfiability: A Logical Formalism for NEXPTIME Decision Problems with Uncertainty. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 3877–3885.

Lee, N.-Z.; Wang, Y.-S.; and Jiang, J.-H. R. 2017. Solving Stochastic Boolean Satisfiability under Random-Exist Quantification. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 688–694.

Lee, N.-Z.; Wang, Y.-S.; and Jiang, J.-H. R. 2018. Solving Exist-Random Quantified Stochastic Boolean Satisfiability via Clause Selection. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 1339–1345.

Littman, M. L.; Goldsmith, J.; and Mundhenk, M. 1998. The Computational Complexity of Probabilistic Planning. *Journal of Artificial Intelligence Research (JAIR)*, 9(1): 1–36.

Littman, M. L.; Majercik, S. M.; and Pitassi, T. 2001. Stochastic Boolean Satisfiability. *Journal of Automated Reasoning (JAR)*, 27(3): 251–296.

Majercik, S. M.; and Boots, B. 2005. DC-SSAT: A Divide-and-Conquer Approach to Solving Stochastic Satisfiability Problems Efficiently. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 416–422.

Majercik, S. M.; and Littman, M. L. 2003. Contingent Planning under Uncertainty via Stochastic Satisfiability. *Artificial Intelligence (AI)*, 147(1-2): 119–162.

Niemetz, A.; Preiner, M.; and Biere, A. 2014. Boolector 2.0. *Journal on Satisfiability, Boolean Modeling and Computation (JSAT)*, 9(1): 53–58.

Oztok, U.; Choi, A.; and Darwiche, A. 2016. Solving PP^{PP}-Complete Problems Using Knowledge Compilation. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning (KR)*, 94–103.

Papadimitriou, C. H. 1985. Games Against Nature. *Journal of Computer and System Sciences (JSCC)*, 31(2): 288–301.

Salmon, R.; and Poupart, P. 2020. On the Relationship Between Satisfiability and Markov Decision Processes. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI)*, 1105–1115.

Shmatikov, V. 2004. Probabilistic Model Checking of an Anonymity System. *Journal of Computer Security (JCS)*, 12(3-4): 355–377.

Stockmeyer, L. J. 1976. The Polynomial-Time Hierarchy. *Theoretical Computer Science (TCS)*, 3(1): 1–22.

Wagner, K. W. 1986. The Complexity of Combinatorial Problems with Succinct Input Representation. *Acta Informatica*, 23(3): 325–356.

Wang, H.-R.; Tu, K.-H.; Jiang, J.-H. R.; and Scholl, C. 2022. Quantifier Elimination in Stochastic Boolean Satisfiability. In *Proceedings of the International Conference on Theory and Applications of Satisfiability Testing (SAT)*, 23:1–23:17.