Hierarchical Stochastic SAT and Quality Assessment of Logic Locking

Christoph Scholl
Department of Computer Science, University of Freiburg, Germany

Tobias Seufert
Department of Computer Science, University of Freiburg, Germany

Fabian Siegwolf
Department of Computer Science, University of Freiburg, Germany

Abstract

Motivated by the application of quality assessment of logic locking we introduce Hierarchical Stochastic SAT (HSSAT) which generalizes Stochastic SAT (SSAT). We look into the complexity of HSSAT and for solving HSSAT formulas we provide a prototype solver which computes exact evaluation results (i.e., without any approximation and without any imprecision caused by numerical rounding errors). Finally, we perform an intensive experimental evaluation of our HSSAT solver in the context of quality assessment of logic locking.

2012 ACM Subject Classification Theory of computation → Logic; Theory of computation → Automated reasoning

Keywords and phrases Stochastic Boolean Satisfiability, Hierarchical Stochastic SAT, Binary Decision Diagrams, Decision Procedure

Digital Object Identifier 10.4230/LIPIcs.SAT.2024.24

Supplementary Material *Collection (Source Code and Benchmarks)*: `https://nc.informatik.uni-freiburg.de/index.php/s/PQ9zBAejC2ERCTD` [25]

Funding This work was supported by the German Research Foundation (DFG) within the project SUPRA (SCHO 894/7-1).

1 Introduction

Introducing Hierarchical Stochastic SAT (HSSAT) in this paper is motivated by the *logic locking* technique. Logic locking has been proposed to protect Integrated Circuits (ICs) from unauthorized usage. Such protection techniques have become necessary as the globalization of manufacturing of ICs may lead to trust issues between the various parties involved in the manufacturing process. Logic locking is a technique to prevent counterfeiting and overproduction by an untrusted foundry. Logic locking introduces additional logic to the IC which is connected to gates in the original IC as well as to a set of newly introduced inputs, the so-called *key inputs*, see Fig. 1 for an overview. The key inputs are stored in a tamper-proof memory and – at least ideally – the modified IC (i.e. "locked" IC) produces correct outputs only if the key inputs are set correctly. The correct key is not revealed to the foundry. The foundry just manufactures the ICs according to the mask data provided by the design house and delivers them to the design house. The design house loads a tamper-proof memory which is connected to the key inputs with the correct key value, thus it "activates" or "unlocks" the IC and delivers the activated IC to the end-users [31].

Numerous methods have been proposed for implementing logic locking schemes, see e.g. [23, 1, 21, 9, 22, 38, 37, 36, 40, 39]. Attacks against logic locking assume both an attacker model and an attack model. For the attacker, we assume that the design house is trusted, but the foundry and the end-user may be untrusted. This includes that the design house

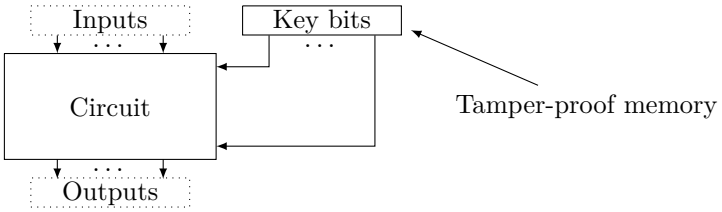

Figure 1 Logic locking.

keeps the unlocking key secret. If the workflow involves transferring the secret key, then the key is secured using cryptographic methods [23, 7] which is an orthogonal problem and is not considered in this paper. For the attack model, most attacks described in the literature assume that the locked netlist is known to the attacker and that the attacker either also has access to the original netlist [23] or at least to an unlocked device [21, 31, 38]. However, an untrusted foundry usually has only access to the mask data and not to the locked *netlist*. Thus, for such an attack it has to re-build the netlist from the mask data which is difficult and very expensive. Reverse engineering of the netlist based on a physical device is even harder or almost impossible for complex designs with small feature sizes and a non-trivial number of design layers. Therefore we assume here that the attacker does not have access to the locked netlist and we ask for the security of logic locking techniques against attacks without knowing the locked netlist. (Nevertheless, our attack model may assume that the attacker has an unlocked working device to check whether key guesses for the locked device have been successful.) Possible weaknesses of logic locking techniques could be: The IC may be unlocked not only with the original (intended) key, but also with other keys, or there may be many keys which "almost unlock" the IC which means that the IC produces for "almost all" input combinations the correct output combinations such that the IC can be used in practice with incorrect keys as well. This danger is real, since current logic locking methods like the one proposed by Yasin et al. [37] achieve resistance against SAT-based attacks [31] by keeping the fraction of input combinations with erroneous output combinations intentionally minimal for each incorrect key.

The goal of this paper is to provide formally precise methods for the quality assessment of logic locking without resorting to imprecise estimations based on simulation like in [19]. This is achieved by reducing the problem to known formalisms like Quantified Boolean Formulas (QBF), (Weighted) Model Counting, Projected (Weighted) Model Counting or Stochastic SAT (SSAT). However it turns out that answering certain interesting questions for quality assessment cannot be expressed naturally by those existing formalisms. For this, we introduce *Hierarchical* Stochastic SAT (HSSAT) which generalizes Stochastic SAT (SSAT). Note that a similar formalism called SSAT(Θ) has been developed by Fan and Jiang [11] in parallel to and independently from our work. Their motivation did not come from an application such as the quality assessment of logic locking, but from the theoretical question of generalizing counting formulas (CFs) that characterize the Counting Hierarchy [34] (as QBFs characterize the Polynomial Hierarchy [30]). We define syntax and semantics of HSSAT, we show that it is PSPACE complete (like QBF and SSAT), and we provide a prototype solver HSSATSolve for HSSAT which is based on ROBDDs [5] and is both algorithmically and numerically exact. In our experimental evaluation we apply our solver to several quality assessment problems. In cases where only (Weighted) Model Counting, Projected (Weighted) Model Counting, or SSAT is needed, we compared it to existing solvers like SharpSAT-TD [15], d4 [17], gpmc [32], arjun-ganak [26, 29], DC-SSAT [18], ClauSSat [6], ElimSSAT [35], and SharpSSAT [10]. The

experiments show that HSSATSolve is capable of answering interesting questions that arise in quality assessment of logic locking and could not be mapped to already existing formalisms; for other questions it often outperforms existing solvers in our application domain.

The paper is structured as follows: We start with basic notations in Sect. 2. In Sect. 3 we introduce details on logic locking, quality assessment of logic locking and mapping it to known problems, whereas in Sect. 4 we discuss Hierarchical Stochastic SAT (HSSAT). We present experimental results in Sect. 5 and conclude with a summary and future research directions in Sect. 6.

2 Preliminaries and Notations

As usual a *Boolean formula* over the variable set X is either 0, 1, some variable $x \in X$, the negation $\neg\phi$ of a Boolean formula ϕ over X, the conjunction $\phi_1 \wedge \phi_2$, or the disjunction $\phi_1 \vee \phi_2$ of two Boolean formulas ϕ_1 and ϕ_2 over X. A *literal* is a variable or its negation. A *clause* is a disjunction of literals, a *cube* is a conjunction of literals. A conjunctive normal form (CNF) is a conjunction of clauses. An *assignment* α over $X' \subseteq X$ is a mapping from X' to $\mathbb{B} = \{0,1\}$ which is called *partial*, if $X' \subset X$, and *full*, if $X' = X$. Sometimes we represent an assignment like $\alpha(x) = 0, \alpha(y) = 1$ by the corresponding cube $\neg xy$. The set of all assignments over X is denoted as $[\![X]\!]$. Given a formula ϕ and an assignment α over X, let $\phi[\alpha]$ denote the result of substituting every occurrence of variables $x \in X$ in ϕ with $\alpha(x)$ and evaluating the obtained boolean expression using the standard rules for the operators $\neg$, $\wedge$, and $\vee$. The Boolean formula ϕ represents a *Boolean function* $f_\phi : [\![X]\!] \to \mathbb{B}$ by $f_\phi(\alpha) = \phi[\alpha]$ for $\alpha \in [\![X]\!]$. 0 (resp. 1) represents the constant 0 (resp. 1) function. The *cofactor* $\phi|_\alpha$ of a Boolean formula ϕ wrt. some (full or partial) assignment α over $X' \subseteq X$ is a formula resulting from substituting every $x \in X'$ in ϕ with $\alpha(x)$.

Boolean functions can also be represented by *Reduced Ordered Binary Decision Diagrams (ROBDDs)* [5], see Fig. 5 for an example of an ROBDD. An ROBDD R over a set X of variables is a directed, acyclic graph $G = (V, E)$ having exactly one root $R.\texttt{root}$ with the following properties: V consists of terminal nodes (represented by squares in Fig. 5) and non-terminal (decision) nodes (represented by circles). The set of terminal nodes is a non-empty subset of $\{\boxed{0}, \boxed{1}\}$. The remaining non-terminal nodes $n \in V$ are labeled with variables $n.\texttt{var} := x \in X$ and they have exactly two outgoing edges, whose targets are denoted by $n.\texttt{low}$ resp. $n.\texttt{high} \in V$. The edge to $n.\texttt{low}$ is called the low edge (represented by a dashed line in Fig. 5), the edge to $n.\texttt{high}$ is called the high edge (represented by a solid line in Fig. 5). An ROBDD R is *ordered* which means that there is a global order $\pi : \{1, \ldots, |X|\} \to X$ such that on each path from the root $R.\texttt{root}$ to a terminal the variable labels of the non-terminal nodes occur in the order $\pi(1), \ldots, \pi(n)$. An ROBDD is *reduced*, i.e., it satisfies the following conditions: There is no pair of non-terminal nodes $n \neq m \in V$ with $n.\texttt{var} = m.\texttt{var}$, $n.\texttt{low} = m.\texttt{low}$, and $n.\texttt{high} = m.\texttt{high}$ ("isomorphism reduction"). For each non-terminal node $n \in V$ it holds $n.\texttt{low} \neq n.\texttt{high}$ ("Shannon reduction"). The reduction property makes ROBDD representations more compact and together with the ordering property it turns ROBDDs into canonical representations for given Boolean functions [5]. The Boolean function $\texttt{eval}(R.\texttt{root}) : [\![X]\!] \to \mathbb{B}$ defined by an ROBDD R can be computed recursively: $\texttt{eval}(\boxed{0}) = 0$, $\texttt{eval}(\boxed{1}) = 1$, and $\texttt{eval}(n) = (\neg n.\texttt{var} \wedge \texttt{eval}(n.\texttt{low})) \vee (n.\texttt{var} \wedge \texttt{eval}(n.\texttt{high}))$ for a non-terminal node n. It easily follows from this rule that the function value of $\texttt{eval}(R.\texttt{root})$ for a full assignment α can be computed by following a path through the ROBDD starting from $R.\texttt{root}$: At each node n on the path one follows the edge to $n.\texttt{low}$ iff $\alpha(n.\texttt{var}) = 0$ and the edge to $n.\texttt{high}$ iff $\alpha(n.\texttt{var}) = 1$. The function value of $\texttt{eval}(R.\texttt{root})$ for assignment α is then given by the reached terminal.

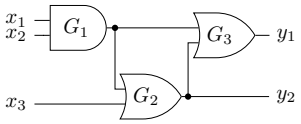

Figure 2 Original circuit.

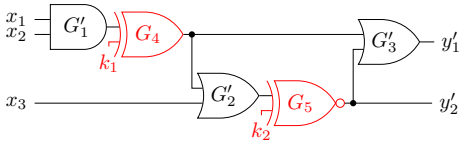

Figure 3 Locked circuit.

A *Stochastic Boolean Satisfiability* (SSAT) formula Φ in prenex form over variable set X is expressed by

$$\Phi = Q_1 x_1, \ldots, Q_n x_n : \phi, \tag{1}$$

where ϕ is a Boolean formula over $X = \{x_1, \ldots, x_n\}$, for $1 \leq i \leq n$ Q_i is either an *existential* quantifier $\exists$ or a *random* quantifier $\mathbb{R}^{p_i}$ with $p_i \in [0, 1]$. $Q_1 x_1, \ldots, Q_n x_n$ is called the *prefix* and ϕ is called the *matrix* of Φ. The random quantifier $\mathbb{R}^{p_i}$ on variable x_i indicates that $x_i = 1$ with probability $p_i \in [0, 1]$ (resp. $x_i = 0$ with probability $1 - p_i$). The *semantics* of an SSAT formula Φ in the *optimization version* is a satisfying probability $\mathsf{Pr}[\Phi]$ recursively computed as follows:

1. $\mathsf{Pr}[\Phi] = 0$, if the matrix of Φ represents the 0 function,
2. $\mathsf{Pr}[\Phi] = 1$, if the matrix of Φ represents the 1 function,
3. $\mathsf{Pr}[\Phi] = \max(\mathsf{Pr}[\Phi'|_{\neg x_i}], \mathsf{Pr}[\Phi'|_{x_i}])$, if $\Phi = \exists x_i \Phi'$,
4. $\mathsf{Pr}[\Phi] = (1 - p_i) \cdot \mathsf{Pr}[\Phi'|_{\neg x_i}] + p_i \cdot \mathsf{Pr}[\Phi'|_{x_i}]$, if $\Phi = \mathbb{R}^{p_i} x_i \Phi'$.

Note that by the notion $\Phi'|_\alpha$ above we mean the SSAT formula where the prefix of Φ' is unchanged and its matrix ϕ' is replaced by the cofactor $\phi'|_\alpha$.

The *decision version* of an SSAT formula $\Phi = Q_1 x_1, \ldots, Q_n x_n : \phi$ has the form Φ `op` q with `op` $\in \{<, \leq, >, \geq, =, \neq\}$, $q \in [0, 1]$ and it evaluates to 1 (true), if $\mathsf{Pr}[\Phi]$ `op` q holds, and to 0 (false) otherwise.

If all quantifiers are random quantifiers, then SSAT corresponds to (weighted) model counting with $\texttt{weight}(x_i) = p_i$, $\texttt{weight}(\neg x_i) = 1 - p_i$.

The syntax of Quantified Boolean Formulas (QBFs) is as given by Eqn. (1) with the difference that in contrast to SSAT formulas Q_i is either an existential quantifier $\exists$ or a *universal* quantifier $\forall$. For the semantics of QBF, the item **4.** from the semantics definition above has to be replaced by $\mathsf{Pr}[\Phi] = \min(\mathsf{Pr}[\Phi'|_{\neg x_i}], \mathsf{Pr}[\Phi'|_{x_i}])$, if $\Phi = \forall x_i \Phi'$.

3 Logic Locking

As already mentioned in Sect. 1, logic locking changes an existing netlist by adding extra gates and extra key inputs, see Fig. 1. The circuit should only work correctly, if the correct key is applied to the key inputs. In the literature numerous logic locking schemes have been presented, see [23, 1, 21, 9, 22, 38, 37, 36, 40, 39], e.g.. The logic locking methods either modify the whole IC or only critical modules. Here we consider logic locking for combinational circuits without memory elements. For sequential circuits, we assume that logic locking modifies the combinational part.

▶ **Example 1.** Let us consider the very simple example of an original (non-locked) circuit in Fig. 2. Here we assume a simple logic locking method which randomly selects internal signals of the circuit and modifies them with logic locking. In the example the outputs of G_1 and G_2 are selected. An exor gate with key input k_1 is inserted at the output of G_1 and an exnor gate with key input k_2 is inserted at the output of G_2, see Fig. 3. It is easy to see that the key value $(k_1, k_2) = (0, 1)$ unlocks the circuit, since the Boolean functions implemented

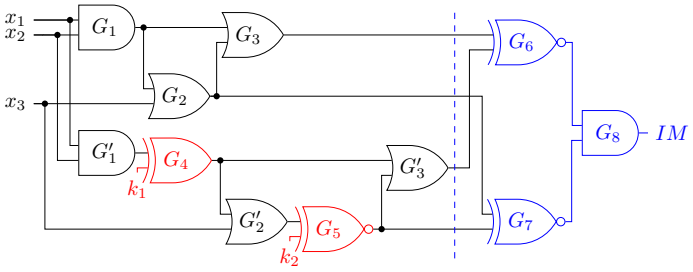

Figure 4 Inverted miter circuit.

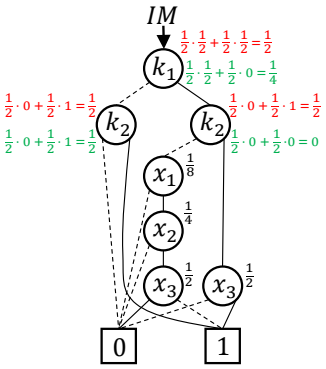

Figure 5 ROBDD for inverted miter circuit.

by the original circuit and the locked circuit with $(k_1, k_2) = (0, 1)$ are the same. In contrast, $(k_1, k_2) = (1, 0)$ does not unlock the circuit, since e.g. for input $(x_1, x_2, x_3) = (1, 1, 1)$ the output of the original circuit is $(y_1, y_2) = (1, 1)$, but the output of the locked circuit is $(y'_1, y'_2) = (0, 0)$.

Now we are interested in a quality assessment of logic lockings for combinational circuits. For such a quality assessment we consider an inverted miter circuit for the original and the locked circuit. As usual, an inverted miter circuit connects the primary inputs of two circuits (in our case of the original and the locked circuit), connects the pairs of corresponding primary outputs of the two circuits with exnor (equivalence) gates, and connects the outputs of all exnor gates at the primary outputs with an and gate. I.e., the inverted miter circuit between the original and the locked circuit outputs a 1 for some assignment to the primary inputs (and key inputs) iff the corresponding outputs of the original and the locked circuit are identical. Fig. 4 shows the inverted miter circuit for the circuits of Figs. 2 and 3. Note that an assessment of logic locking quality is done by the (trusted) design house which owns both the original and the locked netlist. In this section we reduce several quality metrics to problems which are known in the formal reasoning community. The last metric then motivates the definition of a new problem called Hierarchical SSAT.

Key Existence

A very basic question is the question of key existence. If f_{IM} is the Boolean function at the output of the inverted miter, $\vec{X}$ is the input vector and $\vec{K}$ the key bit vector, then there exists an unlocking key iff the QBF

$$\exists \vec{K} \forall \vec{X} : f_{IM} \tag{2}$$

is satisfied (as already observed in [23]). Of course, the QBF should be satisfied, if logic locking was done correctly.

Key Uniqueness

The key existence does not imply security of logic locking however. It could be the case that the existing unlocking key is not the only unlocking key, but there are several unlocking keys. Therefore the uniqueness of the unlocking key is of interest as well. Let $\vec{K}_{\texttt{orig}}$ be the original (intended) unlocking key. Key uniqueness can simply be reduced to the QBF

$$\exists \vec{K} \forall \vec{X} : (f_{IM} \wedge (\vec{K} \neq \vec{K}_{\texttt{orig}})). \tag{3}$$

Fraction of Unlocking Keys

We cannot assume that logic locking methods always produce unique keys. Nevertheless, they may be appropriate in practice, if the number of unlocking keys is not too high. In this case the probability of guessing an unlocking key would be so low that an attack that randomly guesses keys will not have a high chance of success, especially if the size $|\vec{K}|$ is large enough. Thus, we are interested in the fraction of keys which are unlocking. This question can easily be reduced to the formula

$$\text{ꓤ}^{0.5}\vec{K}\forall\vec{X} : f_{IM}. \tag{4}$$

Since existing SSAT solvers often do not support universal quantification, we have to transform Eqn. (4) into a version which removes $\forall$-quantification. This is not a trivial task for arbitrary SSAT formulas containing both $\exists$- and $\forall$-quantifiers, but in our special case it is easy. It is clear that a given fixed key $\vec{K}_{\texttt{fix}}$ *does not* unlock the locked circuit iff there is an input assignment which produces a 1 at the output of the *non-inverted* miter circuit, i.e., iff $\exists\vec{X}\neg f_{IM}|_{\vec{K}=\vec{K}_{\texttt{fix}}}$ holds. Thus, the *fraction* of keys which *do not* unlock the circuit is given by

$$\text{ꓤ}^{0.5}\vec{K}\exists\vec{X} : \neg f_{IM}. \tag{5}$$

If this fraction is given by $p_{\texttt{w}}$, then the fraction of unlocking keys is $p_{\texttt{c}} = 1 - p_{\texttt{w}}$. After multiplying the result by $2^{|\vec{K}|}$ we obtain the number of unlocking keys. If the original key is unlocking and we are interested in the fraction of unlocking keys among the remaining keys, then we simply have to correct $p_{\texttt{c}}$ by computing $\frac{p_{\texttt{c}}\cdot 2^{|\vec{K}|}-1}{2^{|\vec{K}|}-1}$.

Existence of Keys with High Criticality

The existence of (many) keys different from the original key which completely unlock the circuit is of course a security issue. However, from an application perspective it is also critical, if there is a key which is different from the original key and "almost" unlocks the circuit. This is captured by the notion of "criticality" of a key.

▶ **Definition 2.** *The* criticality *of a key is defined as the quotient of the number of input assignments for which the key produces a correct output and the total number of input assignments.*

Given a criticality bound $c \in [0, 1]$, we are interested in the question whether there exists a key different from the original key $\vec{K}_{\texttt{orig}}$ with criticality $> c$. Keys with an extremely high criticality are considered to be dangerous, because with such a key an IC may possibly be operated for a long time without observing an error. Note that there are (as already mentioned in Sect. 1) logic locking methods [37] tailored towards resistance against SAT-based attacks [31] (which make the assumption that the locked netlist is at the attacker's disposal) where all keys have almost maximal criticality. This is of course undesirable from a practical point of view and therefore we consider it to be necessary to include criticality as a metric for assessing the quality of logic locking. For a fixed key $\vec{K}_{\texttt{fix}}$, $\text{ꓤ}^{0.5}\vec{X} : f_{IM}|_{\vec{K}=\vec{K}_{\texttt{fix}}}$ computes the criticality of $\vec{K}_{\texttt{fix}}$ and thus the existence of a key (different from the original one) with criticality $> c$ can be checked by the SSAT formula

$$\left(\exists\vec{K}\text{ꓤ}^{0.5}\vec{X} : (f_{IM} \wedge (\vec{K} \neq \vec{K}_{\texttt{orig}}))\right) > c. \tag{6}$$

According to the semantics definition reviewed in Sect. 2 the existential quantifier $\exists\vec{K}$ performs a maximization over all criticality values $\text{ꓤ}^{0.5}\vec{X} : f_{IM}|_{\vec{K}=\vec{K}_{\texttt{fix}}}$ for different fixed keys $\vec{K}_{\texttt{fix}}$ (the evaluation for the original key is forced to 0 by the additional constraint "$\vec{K} \neq \vec{K}_{\texttt{orig}}$"). This maximal value is finally compared to the criticality bound c.

Average Criticality of Keys

Again, it is not very dangerous, if there are only a few keys with high criticality, since the probability is not high that an attacker guesses a critical key. So we are also interested in the *average criticality* of all possible keys. This can be computed by the SSAT formula

$$\text{ꓤ}^{0.5}\vec{K}\text{ꓤ}^{0.5}\vec{X} : f_{IM}, \tag{7}$$

since the quantification $\text{ꓤ}^{0.5}\vec{K}$ just averages over all possible criticalities of the key values. Eqn. (7) may be interpreted as a model counting problem.

Fraction of Keys with High Criticality

A high average criticality may be regarded as a security problem at first sight. Unfortunately, it is not a perfect metric for assessing the quality of logic locking, since there may be several reasons for a high average criticality. Let us look into two extreme cases: In case 1 all keys (except the original one) have criticality 0.5. This leads to a (rather high) average criticality of (about) 0.5 (the original key contributes 1, all other keys 0.5 to the average). In case 2, one half of the keys have criticality 1, the other half have criticality 0. Here the average criticality is again 0.5. Case 1 is not really critical, since the user most probably cannot work with a key with criticality 0.5, since half of the input assignments produce erroneous values. The probability of guessing a key with criticality > 0.5 is as low as possible ($\frac{1}{2^{|\vec{K}|}}$). On the other hand, case 2 with almost the same average criticality is highly critical from an application point of view. On average, every second guessed key completely unlocks the circuit.

This observation shows that we should look for a different criticality-based metric. We would like to ensure that the *number* of keys (or the *fraction* of keys) with a criticality larger than a given criticality bound $c \in [0, 1]$ is as low as possible. If we choose the criticality bound c in a way that the circuit is supposed to be of no use, if it is operated with a key with criticality $\leq c$, then this fraction immediately gives the probability of guessing a key which can be used in practice. From a conceptional point of view, the task is to compute for each fixed key $\vec{K}_{\texttt{fix}}$ the probability $\text{ꓤ}^{0.5}\vec{X} : f_{IM}|_{\vec{K}=\vec{K}_{\texttt{fix}}}$ (i.e. the criticality of $\vec{K}_{\texttt{fix}}$), evaluate "$\text{ꓤ}^{0.5}\vec{X} : f_{IM}|_{\vec{K}=\vec{K}_{\texttt{fix}}} > c$" for the given criticality bound c, and then just to compute the fraction of keys for which this evaluation returns "true". In the end we have to compare this fraction again with a bound which specifies how large the computed fraction is allowed to be. However, this approach does not seem to be feasible, since it leads to a number of SSAT (or model counting) problems which is exponential in the number of key bits $\vec{K}$. The problem cannot naturally be reduced to an SSAT formula, since the comparison with the criticality bound c has to be performed for each key value individually instead of a single comparison as in Eqn. (6). This situation leads us to the definition of a new generalization of SSAT called *Hierarchical* Stochastic SAT (HSSAT). HSSAT allows to hierarchically include several comparison operators within the formula instead of only one at the end as in the decision version of SSAT.

4 Hierarchical Stochastic SAT

4.1 Definition of HSSAT

Here we define our new formalism of HSSAT which has been motivated in the previous section. In contrast to usual definitions of SSAT we also allow universal quantifiers in HSSAT, since (1) allowing them does not change the complexity class of HSSAT and (2) we can easily process universal quantifiers in our solver as well, so there is no reason to forbid universal quantifiers.

▶ **Definition 3** (Syntax of HSSAT formulas). *A Boolean Formula ϕ over variable set X is also an HSSAT formula Φ with matrix $\texttt{matrix}(\Phi) = \phi$, the set $\texttt{FV}(\Phi) = X$ of free variables, the set $\texttt{BV}(\Phi) = \emptyset$ of bound variables, the quantifier order $\pi(\Phi) : \{1, \dots, |\texttt{BV}(\Phi)|\} \to \texttt{BV}(\Phi)$, and hierarchy level $\texttt{hlevel}(\Phi) = 0$.*

Now let Φ be an arbitrary HSSAT formula with matrix $\texttt{matrix}(\Phi)$, the set $\texttt{FV}(\Phi)$ of free variables, the set $\texttt{BV}(\Phi)$ of bound variables, the quantifier order $\pi(\Phi) : \{1, \dots, |BV(\Phi)|\} \to BV(\Phi)$, and hierarchy level $\texttt{hlevel}(\Phi)$. Let $x \in \texttt{FV}(\Phi)$, $p, q \in [0, 1]$, $\texttt{op} \in \{<, \leq, >, \geq, =, \neq\}$. Then the following formulas are HSSAT formulas as well:

(a) *$\Phi' = (\exists x \Phi)$ is an HSSAT formula with $\texttt{FV}(\Phi') = \texttt{FV}(\Phi) \setminus \{x\}$, $\texttt{BV}(\Phi') = \texttt{BV}(\Phi) \cup \{x\}$, $\texttt{hlevel}(\Phi') = \texttt{hlevel}(\Phi)$, $\texttt{hlevel}(x) = \texttt{hlevel}(\Phi)$, $\texttt{quantor}(x) = \exists$.*

(b) *$\Phi' = (\forall x \Phi)$ is an HSSAT formula with $\texttt{FV}(\Phi') = \texttt{FV}(\Phi) \setminus \{x\}$, $\texttt{BV}(\Phi') = \texttt{BV}(\Phi) \cup \{x\}$, $\texttt{hlevel}(\Phi') = \texttt{hlevel}(\Phi)$, $\texttt{hlevel}(x) = \texttt{hlevel}(\Phi)$, $\texttt{quantor}(x) = \forall$.*

(c) *$\Phi' = (\text{ᴙ}^p x \Phi)$ is an HSSAT formula with $\texttt{FV}(\Phi') = \texttt{FV}(\Phi) \setminus \{x\}$, $\texttt{BV}(\Phi') = \texttt{BV}(\Phi) \cup \{x\}$, $\texttt{hlevel}(\Phi') = \texttt{hlevel}(\Phi)$, $\texttt{hlevel}(x) = \texttt{hlevel}(\Phi)$, $\texttt{quantor}(x) = \text{ᴙ}^p$.*

(d) *$\Phi' = (\Phi\ \texttt{op}\ q)$ is an HSSAT formula with $\texttt{FV}(\Phi') = \texttt{FV}(\Phi)$, $\texttt{BV}(\Phi') = \texttt{BV}(\Phi)$, $\texttt{hlevel}(\Phi') = \texttt{hlevel}(\Phi) + 1$, $\texttt{op}(\texttt{hlevel}(\Phi) + 1) = \texttt{op}$, $\texttt{prob}(\texttt{hlevel}(\Phi) + 1) = q$.*

In all cases (a) - (d) we have $\texttt{matrix}(\Phi') = \texttt{matrix}(\Phi)$. In cases (a) - (c) the quantifier order is $\pi(\Phi') : \{1, \dots, |BV(\Phi')|\} \to BV(\Phi')$ with $\pi(\Phi')(1) = x$, $\pi(\Phi')(i) = \pi(\Phi)(i-1)$ for all $2 \leq i \leq |BV(\Phi')|$. In case (d) $\pi(\Phi') = \pi(\Phi)$.

An HSSAT formula Φ is called closed, *if $\texttt{FV}(\Phi) = \emptyset$.*

In a closed HSSAT formula all variables in the matrix are bound by $\exists$, $\forall$, or ᴙ quantifiers. We define the semantics only for closed HSSAT formulas and assume that non-closed formulas are made closed by using leading existential quantifiers.

Note that SAT formulas can be seen as closed HSSAT formulas of hierarchy level 0 with $\exists$-quantifiers only, QBFs as closed HSSAT formulas of hierarchy level 0 with only $\exists$- and $\forall$-quantifiers, SSAT formulas in the optimization version as closed HSSAT formulas of hierarchy level 0 with only $\exists$- and ᴙ-quantifiers, and SSAT formulas in the decision version as closed HSSAT formulas of hierarchy level 1 with only $\exists$- and ᴙ-quantifiers.

Before we look into the semantics of HSSAT, we consider how to express the motivating example from the last section with an HSSAT formula.

Fraction of Keys with High Criticality

The problem of deciding whether the fraction of keys with criticality $> c$, $c \in [0, 1]$ is larger than $d \in [0, 1]$ (see Sect. 3) can be reduced to the following closed HSSAT formula:

$$((\text{ᴙ}^{0.5} \vec{K} ((\text{ᴙ}^{0.5} \vec{X} : f_{IM}) > c)) > d). \tag{8}$$

▶ **Example 4.** For Example 1 (see Figs. 2, 3, 4) the corresponding closed HSSAT formula is

$$\Phi := \left[\left(\text{ᴙ}^{0.5} k_1 \text{ᴙ}^{0.5} k_2 \left[\left(\text{ᴙ}^{0.5} x_1 \text{ᴙ}^{0.5} x_2 \text{ᴙ}^{0.5} x_3 : f_{IM}\right) > c\right]\right) > d\right].^1 \tag{9}$$

Φ is a closed formula with an empty set $\texttt{FV}(\Phi)$ of free variables, the bound variables are $\texttt{BV}(\Phi) = \{k_1, k_2, x_1, x_2, x_3\}$. For all variables $x \in \texttt{BV}(\Phi)$ $\texttt{quantor}(x) = \forall^{0.5}$. The quantifier order is given by $\pi(\Phi)(1) = k_1$, $\pi(\Phi)(2) = k_2, \ldots, \pi(\Phi)(5) = x_3$. The hierarchy level $\texttt{hlevel}(\Phi)$ is 2, $\texttt{hlevel}(x_1) = \texttt{hlevel}(x_2) = \texttt{hlevel}(x_3) = 0$, $\texttt{hlevel}(k_1) = \texttt{hlevel}(k_2) = 1$. The probability value for comparison at hierarchy level 1 is $\texttt{prob}(1) = c$ and at hierarchy level 2 it is $\texttt{prob}(2) = d$. The operands at hierarchy levels 1 and 2 are $>$: $\texttt{op}(1) = \texttt{op}(2) = >$.

The formula checks whether the fraction of keys with criticality greater than c is greater than d.

The semantics of HSSAT formulas can be formally defined using an evaluation function $\textsc{Eval}(\cdot)$. The result is a value from $[0, 1]$ which may be interpreted as a probability value or – if the HSSAT formula is a comparison with some number from $[0, 1]$, i.e., if we are in case (d) of Def. 3 – as a logical value 0 (false) or 1 (true).

▶ **Definition 5** (Semantics of HSSAT formulas). *Let Φ' be a closed HSSAT formula. Φ' is evaluated by a function $\textsc{Eval}(\cdot)$.*

(1) *If $\mathit{BV}(\Phi') = \emptyset$, $\mathit{hlevel}(\Phi') = 0$, and $f_{\Phi'} = 0$, then $\textsc{Eval}(\Phi') = 0$.*
(2) *If $\mathit{BV}(\Phi') = \emptyset$, $\mathit{hlevel}(\Phi') = 0$, and $f_{\Phi'} = 1$, then $\textsc{Eval}(\Phi') = 1$.*
(3) *If $\Phi' = \exists x \Phi$, then $\textsc{Eval}(\Phi') = \max(\textsc{Eval}(\Phi|_{\neg x}), \textsc{Eval}(\Phi|_x))$.*
(4) *If $\Phi' = \forall x \Phi$, then $\textsc{Eval}(\Phi') = \min(\textsc{Eval}(\Phi|_{\neg x}), \textsc{Eval}(\Phi|_x))$.*
(5) *If $\Phi' = \forall^p x \Phi$, then $\textsc{Eval}(\Phi') = (1-p) \cdot \textsc{Eval}(\Phi|_{\neg x}) + p \cdot \textsc{Eval}(\Phi|_x)$.*
(6) *If $\Phi' = (\Phi \ \mathit{op} \ q)$, then $\textsc{Eval}(\Phi') = 1$ in case $\textsc{Eval}(\Phi)$ op q holds, $\textsc{Eval}(\Phi') = 0$ otherwise.*

The notion $\Phi|_\alpha$ above means that the matrix of Φ is replaced with its cofactor $(\mathit{matrix}(\Phi))|_\alpha$.

4.2 Complexity of HSSAT

Now we come to the complexity of the HSSAT problem. It is pretty easy to see that HSSAT is PSPACE complete – just as SSAT.

▶ **Lemma 6.** *HSSAT is PSPACE hard.*

Proof. Since we allow $\forall$-quantifiers in HSSAT, each QBF is a closed HSSAT formula of hierarchy level 0 with $\exists$- and $\forall$-quantifiers only. Thus the hardness proof easily follows from the PSPACE hardness of QBF. If $\forall$-quantifiers would not be allowed in HSSAT, a polynomial time reduction from QBF would replace in a QBF Φ all $\forall$-quantifiers by $\forall^p$-quantifiers with arbitrary $p \in (0, 1)$, leading to some Φ', and it would consider "$\Phi' = 1$". The QBF Φ is satisfied iff $\Phi' = 1$ evaluates to 1 (true). (This is just as in the reduction for SSAT [18].) ◀

▶ **Lemma 7.** *HSSAT is in PSPACE.*

Proof (Sketch). Similar to the SSAT case, Def. 5 immediately suggests an evaluation of a closed HSSAT formula Φ with quantified variables from X by an "implicit depth-first traversal of the decision tree" of all assignments over X. The decision tree considers the variables in the order they occur in the prefix of Φ. Of course, it is not necessary to store the decision tree explicitly, but it is only necessary to store the currently considered path. Whenever a node has been evaluated, it is not necessary anymore to store the evaluation values for its successors. Thus, it is never needed to store more than $O(|X|)$ values during the depth-first traversal. Moreover, it is easy to see that the size of the number representations occurring during the evaluation is polynomially restricted by the input size of the problem instance. Thus, HSSAT is in PSPACE. ◀

[1] Parentheses according to Def. 3 which are clear from the context are omitted.

Algorithm 1 evalHSSAT.

Input: HSSAT formula Φ with $\texttt{matrix}(\Phi) = \texttt{ROBDD}\ R$, $\texttt{FV}(\Phi) = \emptyset$
Output: rational number evaluating Φ
1: **return** evalEdge(evalNode($R.\texttt{root}$), $\texttt{hlevel}(\Phi)$, $\texttt{hlevel}(R.\texttt{root})$);

Algorithm 2 evalNode.

Input: ROBDD node n
Output: rational number evaluating n
1: **if** $n = \boxed{0}$ **then return** 0;
2: **if** $n = \boxed{1}$ **then return** 1;
3: **if** $n.\texttt{value} \neq \texttt{undefined}$ **then return** $n.\texttt{value}$;
4: $p_{\texttt{low}}$:= evalEdge(evalNode($n.\texttt{low}$), $\texttt{hlevel}(n)$, $\texttt{hlevel}(n.\texttt{low})$);
5: $p_{\texttt{high}}$:= evalEdge(evalNode($n.\texttt{high}$), $\texttt{hlevel}(n)$, $\texttt{hlevel}(n.\texttt{high})$);
6: **if** $\texttt{quantor}(n.\texttt{var}) = \exists$ **then** $p := \max(p_{\texttt{low}}, p_{\texttt{high}})$;
7: **if** $\texttt{quantor}(n.\texttt{var}) = \forall$ **then** $p := \min(p_{\texttt{low}}, p_{\texttt{high}})$;
8: **if** $\texttt{quantor}(n.\texttt{var}) = \text{Ɐ}^q$ **then** $p := (1-q) \cdot p_{\texttt{low}} + q \cdot p_{\texttt{high}}$;
9: $n.\texttt{value} := p$;
10: **return** p;

Lemma 6 and Lemma 7 imply Theorem 8.

▶ **Theorem 8.** *HSSAT is PSPACE complete.*

4.3 Solving HSSAT

Here we present a prototype algorithm for solving HSSAT which is based on the computation of ROBDDs for the matrix of the formula. The prototype algorithm also gives an indication of how to design a generalization of DPLL-based SSAT algorithms like DC-SSAT [18], Prime [24], or SharpSSAT [10] to HSSAT.

We will start with an algorithm which first builds for an HSSAT formula Φ an ROBDD $R = (V, E)$ for $\texttt{matrix}(\Phi)$ with the variable order $\pi := \pi(\Phi)$. Then the evaluation of Φ is reduced to an evaluation of the ROBDD R by Algs. 1, 2, and 3. Alg. 2 is a recursive algorithm computing the evaluation values at the different ROBDD nodes and Alg. 3 takes care of edges in the ROBDD, especially long edges crossing several levels in the variable order. Alg. 1 just reads out the correct value from the root of the ROBDD. To simplify notations in the definition of Algs. 1 and 2 we define for $n \in V$: $\texttt{hlevel}(n) = 0$, if $n \in \{\boxed{0}, \boxed{1}\}$, $\texttt{hlevel}(n) = \texttt{hlevel}(n.var)$ otherwise. We will consider the correctness of the algorithms first and we will show an example afterwards.

For the time being, we assume that the HSSAT formula does not contain hierarchical comparisons with values $p \in [0, 1]$, i.e., we first neglect case (d) of Def. 3. This means that the hierarchy level of Φ as well as of all variables is 0. It is easy to check that in this case Alg. 3 does not have any effect, i.e., it returns its input probability without any change.

I.e., with our initial assumption the algorithm works like an ROBDD-based algorithm for SSAT. We briefly discuss its correctness. First we assume that the "Shannon reductions are reverted" in the ROBDD. This means the following: As long as there exists a "long" edge from a node n to node $m = n.\texttt{dir}$, $\texttt{dir} \in \{\texttt{low}, \texttt{high}\}$, with either $m \in \{\boxed{0}, \boxed{1}\}$ and $\pi^{-1}(n.\texttt{var}) < |X|$ or $\pi^{-1}(m.\texttt{var}) > \pi^{-1}(n.\texttt{var}) + 1$, we introduce a new successor n' of n

Algorithm 3 evalEdge.

Input: rational number $p_{\texttt{target}}$ evaluating the target node of an edge, hierarchy level $hl_{\texttt{source}}$ of source node, hierarchy level $hl_{\texttt{target}}$ of target node of the edge
Output: rational number evaluating the edge between source and target node

```
1: p := p_target; hl := hl_target;
2: while hl < hl_source do
3:     if p op(hl + 1) prob(hl + 1) then p := 1 else p := 0;
4:     hl := hl + 1;
5: return p;
```

with $n.\texttt{dir} = n'$, $n'.\texttt{low} = m$, $n'.\texttt{high} = m$. We perform a similar transformation for the root of R, if $\pi^{-1}(R.\texttt{root}) > 1$. The resulting ROBDD is then essentially a decision tree with variable order π (with the only difference that there are shared nodes in the ROBDD due to "isomorphism reduction"). Now it is clear that Alg. 2 does exactly the same evaluation steps as given in Def. 5, since following an edge from some node n labelled with $n.\texttt{var}$ in the ROBDD exactly corresponds to a cofactor computation (remember that Alg. 3 does not have an effect for now). Alg. 2 additionally caches values which have already been computed in the variable $n.\texttt{value}$. It only remains to show that re-introducing Shannon reductions does not change the evaluation. Consider a node n with $n.\texttt{low} = n.\texttt{high}$. From $v = \max(v, v) = \min(v, v) = (1 - p) \cdot v + p \cdot v$ it easily follows by case distinction with the cases $\texttt{quantor}(n.\texttt{var}) = \exists, \forall, ⅋^p$ that the evaluation of n gives the same value as the evaluation of $n.\texttt{low} = n.\texttt{high}$, i.e., removing n does not change the evaluation.

Now that it is clear that the algorithm is correct, if Φ does not contain hierarchical comparisons according to case (d) of Def. 3, we only have to consider what changes, if we re-introduce case (d). Intuitively, Rule (6) of Def. 5 with $\Phi' = (\Phi\ \texttt{op}\ q)$ and hierarchy level $\texttt{hlevel}(\Phi') = hl$ says the following: If we are at a node n with hierarchy level $\texttt{hlevel}(n) = hl$ and read a value v from a node m with hierarchy level $\texttt{hlevel}(m) = hl - 1$, then we should not take v itself, but we have to replace v with the outcome of the comparison $v\ \texttt{op}\ q$ which is 0 or 1. The **while** loop in Alg. 3 just accounts for the case that there may be "long edges" from a node n with $\texttt{hlevel}(n) = hl_n$ to node m with $\texttt{hlevel}(m) = hl_m$ and $hl_n \geq hl_m + 1$ in the ROBDD, crossing several hierarchy levels. Then several evaluations are needed, comparing with $\texttt{op}(hl_m + 1)\ \texttt{prob}(hl_m + 1)$ first, and finally with $\texttt{op}(hl_n)\ \texttt{prob}(hl_n)$. In a similar way, the call to evalEdge in Alg. 1 considers the case that $\texttt{hlevel}(\Phi) > \texttt{hlevel}(R.\texttt{root})$, i.e., the outer formula contains comparisons according to case (d) of Def. 3.

▶ **Theorem 9.** *Algorithm 1 correctly evaluates an HSSAT formula Φ according to Def. 5.*

▶ **Example 10.** Consider the ROBDD in Fig. 5 for the inverted miter circuit in Fig. 4. Let us consider the HSSAT formula $\left[\left(⅋^{0.5} k_1 ⅋^{0.5} k_2 \left[\left(⅋^{0.5} x_1 ⅋^{0.5} x_3 ⅋^{0.5} x_3 : f_{IM}\right) \geq c\right]\right) \geq 0.3\right]$, see also Example 4. Let us first assume $c = \frac{1}{2}$. The nodes in the ROBDD are annotated with their evaluation values. The evaluation for the x_i-nodes just uses the formula $\frac{1}{2} \cdot \text{evalNode}(n.\texttt{low}) + \frac{1}{2} \cdot \text{evalNode}(n.\texttt{high})$. However, the evaluation of the right k_2-node, e.g., has to consider the comparison with $c = \frac{1}{2}$. For the $\texttt{low}$–successor we have $\frac{1}{8} < \frac{1}{2}$, thus we have to replace its value by 0. For the $\texttt{high}$–successor we have $\frac{1}{2} \geq \frac{1}{2}$, thus we have to replace its value by 1. This leads to $\frac{1}{2} \cdot 0 + \frac{1}{2} \cdot 1 = \frac{1}{2}$ at the right k_2-node. The upper (red) annotations at the k_i-nodes give the node values for $c = \frac{1}{2}$ with the final value of $\frac{1}{2}$ at the root, leading to 1 in the final comparison with 0.3. The lower (green) annotations at the k_i-nodes give the node values for $c = 0.99$ with the final value of $\frac{1}{4}$ at the root, leading to 0 in the final comparison with 0.3.

Improvements

Here we consider two improvements to the basic algorithm.

The first improvement concerns the variable order in the ROBDD. For this improvement we define quantifier blocks for HSSAT, similar to SSAT or QBF. For an HSSAT formula Φ with bound variables $\texttt{BV}(\Phi) = X$, we define a partition $P = \{X_1, \ldots, X_k\}$ on X by the following rule: P is the (unique) partition where for all $1 \leq i \leq |X| - 1$ $\pi(\Phi)(i)$ and $\pi(\Phi)(i+1)$ are in the same set X_j iff $\texttt{quantor}(\pi(\Phi)(i)) = \texttt{quantor}(\pi(\Phi)(i+1))$ and $\texttt{hlevel}(\pi(\Phi)(i)) = \texttt{hlevel}(\pi(\Phi)(i+1))$ (random quantors $\mathbb{R}^p$ and $\mathbb{R}^q$ are considered to be equal even if $p \neq q$). Thus P partitions the variables in the prefix into groups of consecutive variables with the same quantor and the additional condition that comparisons according to case (d) of Def. 3 introduce cuts into those groups. It is easy to see by a simple computation that the value of an HSSAT Φ does not change, if neighboring variables $\pi(\Phi)(i)$ and $\pi(\Phi)(i+1)$ belonging to the same group X_j are exchanged in Φ. Thus, all variables within some group X_j can be arbitrarily exchanged without changing the value of Φ. This also means that for the ROBDD-based evaluation we do not need to choose exactly the variable order $\pi(\Phi)$, but we can choose a variable order which is "*compatible with* $\pi(\Phi)$". "Compatible" means here that variables within groups X_j of $\{X_1, \ldots, X_k\}$ can be arbitrarily exchanged in the variable order of the ROBDD without affecting the value of the evaluation. In our implementation we choose $\pi(\Phi)$ as the initial order when we build the ROBDD for $\texttt{matrix}(\Phi)$, but we activate the dynamic reordering technique of "group sifting" [20] which dynamically tries to change the variable order within the blocks with the goal of minimizing the number of needed ROBDD nodes to represent $\texttt{matrix}(\Phi)$.

The second improvement comes into play when the matrix of the HSSAT formula Φ is given in CNF, especially when it was produced from a circuit representation by a transformation like Tseitin transformation [33]. In our implementation we use the "interpolation-based gate extraction" by Slivovsky [27] which is based on the replacement of so-called "defined variables" by their "definition".

▶ **Definition 11** ([27]). *Let ϕ be a Boolean formula over X, $X' \subseteq X$. $x \in X$ is* defined *in terms of X' in ϕ iff $\alpha(x) = \beta(x)$ for any two full assignments α and β that satisfy ϕ and agree on X'. A definition of x by X' in ϕ is a formula ψ over X' with $\alpha(x) = \psi[\alpha]$ for any assignment α that satisfies ϕ.*

Slivovsky [27] looks into QBF and considers existential defined variables and universal defining variables. We adjust the approach of [27] to HSSAT by the following lemma:

▶ **Lemma 12.** *Let x be an existential variable of a closed HSSAT formula Φ and for all hl with $1 \leq hl \leq \texttt{hlevel}(x)$ the equation "0 $\texttt{op}(hl)$ $\texttt{prob}(hl)$" does* not *hold. Let x be defined in terms of $\{y \in \texttt{BV}(\Phi) \mid \pi^{-1}(y) < \pi^{-1}(x)\}$ in $\texttt{matrix}(\Phi)$ and let ψ be a corresponding definition. Then $\textsc{Eval}(\Phi) = \textsc{Eval}(\Phi[x := \psi])$ where $\Phi[x := \psi]$ results from Φ by replacing x with ψ in the matrix and omitting $\exists x$ in the prefix.*

On the one hand, Lemma 12 does not only allow universal variables to the left of existential variables as defining variables, but *all* variables to the left. (A similar approach has been already proposed in [27] as an improvement for the restricted case of QBFs.) On the other hand, we need an additional restriction on the choice of defined existential variables that is specific to *Hierarchical* Stochastic SAT. We prove the correctness of Lemma 12 in Appendix A.

The transformation of Lemma 12 is useful, since the replaced variables do not occur as input variables in the ROBDD later on and the computed ROBDDs are smaller in most cases. Semantic gate extraction has already been used in the context of SSAT solving [35]. Similar ideas have also been used in [16, 14] for model counting.

5 Experiments

We implemented our prototype solver HSSATSolve with the ROBDD package CUDD [28] using dynamic group sifting [20] for optimizing the variable orders. For all number representations we use rational numbers from the GNU Multiple Precision Arithmetic Library (GMP) [12]. All experiments were performed on one core of an Intel Xeon CPU E5-2650v2 CPU with 2.6 GHz using 16 GB of memory with a timeout of 30 CPU minutes. All benchmarks and the code of our tool can be found at [25].

For our experiments we added logic locking to the combinational ISCAS'85 benchmarks [4] and to the combinational parts of the ITC'99 benchmarks [8]. For logic locking we used the method from [23] which randomly selects signals in the circuit and replaces them randomly either with exor or exnor gates having a key bit as a side input (see also Figs. 2 and 3). We considered key bit lengths of 4, 8, 16, 32, and 64. We generated the inverted miters of the original and the locked circuits and generated HSSAT formulas of types ∀∃, ∀, ∃∀, and H. Formulas of type ∀∃ compute the fraction of unlocking keys (Eqn. (5)), formulas of type ∀ the average criticality of keys (Eqn. (7)), formulas of type ∃∀ the existence of keys with high criticality (Eqn. (6)), and formulas of type H the fraction of keys with high criticality (Eqn. (8)). For all random quantifiers we considered input probabilities of 0.5.

To be able to evaluate the general quality of our prototype solver we first compared it with existing tools. For this we translated the formulas of types ∀∃, ∀, and ∃∀ into CNFs by Tseitin transformation [33]. Formulas of type ∀ are Model Counting problems, formulas of type ∀∃ are Projected Model Counting problems. For their comparisons, we considered the best exact solvers of the Model Counting Competition 2023 [13]. We used SharpSAT-TD [15], d4 [17], gpmc [32], and arjun-ganak [26, 29]. Formulas of type ∃∀ are SSAT formulas. Here we compared with DC-SSAT [18], ClauSSat [6], ElimSSAT [35], and SharpSSAT [10].

For formulas with matrix in CNF we use a version HSSATSolve$^{\texttt{cnf}}$ of our tool. Here we first apply the tool UNIQUE [27] which was adapted according to Lemma 12 to transform the CNF into a circuit format, then we optimize the circuit using ABC [2, 3], and finally we build ROBDDs for the matrix using CUDD with dynamic group sifting activated. Then we evaluate the ROBDDs using Alg. 1. For formulas with matrix in circuit format we use a version HSSATSolve$^{\texttt{circ}}$ of our tool where the application of UNIQUE is of course omitted.

Fig. 6 shows a cactus plot for the comparisons using formulas of type ∀∃, Fig. 7 a cactus plot for the comparisons using formulas of type ∀, and Fig. 8 a cactus plot for the comparisons using formulas of type ∃∀.

Overall, it turns out that our prototype solver competes well with the already existing tools on formulas from our application domain of logic locking. Comparing HSSATSolve$^{\texttt{cnf}}$ and HSSATSolve$^{\texttt{circ}}$ the results show that HSSATSolve$^{\texttt{circ}}$ clearly outperforms HSSATSolve$^{\texttt{cnf}}$. The gap between HSSATSolve$^{\texttt{cnf}}$ and HSSATSolve$^{\texttt{circ}}$ proves that the semantic gate extraction procedure of UNIQUE is apparently not able to reconstruct the original circuit structure from the CNF – although for our benchmarks this is easily possible with a simple syntactic gate extraction method. Based on those observations we believe that it will be possible to extend UNIQUE with certain improvements, but this will be the subject of future work.

There are apparent differences between the results of Figs. 6 and 7, e.g.. Whereas gpmc and arjun-ganak solve more formulas than HSSATSolve$^{\texttt{circ}}$ on formulas of type ∀∃ (Fig. 6), the situation is the other way round for SharpSAT-td and d4 on formulas of type ∀ (Fig. 7). In particular, HSSATSolve$^{\texttt{circ}}$ solves more formulas of type ∀ than formulas of type ∀∃. To analyze the reason for this, we added an experiment with HSSATSolve$^{\texttt{circ,ns}}$ which is a

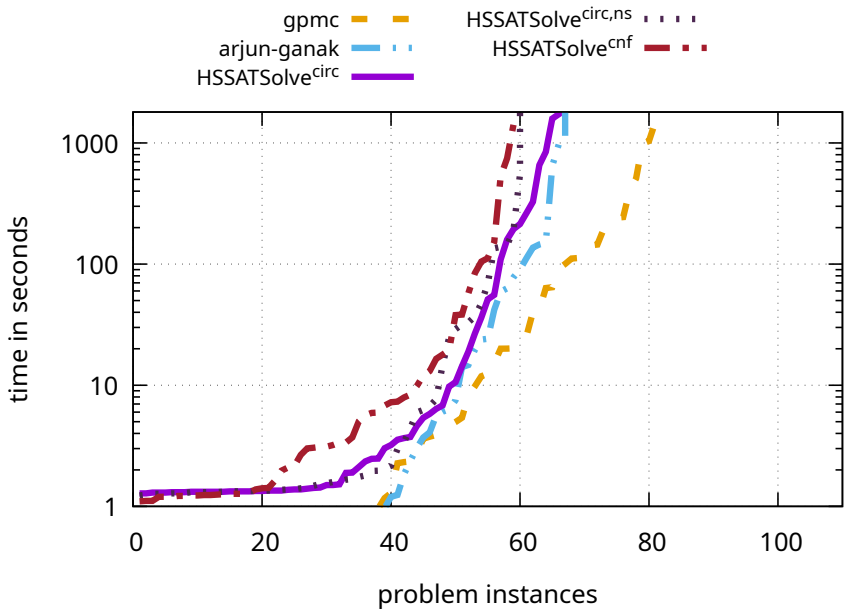

Figure 6 Type ЯЭ.

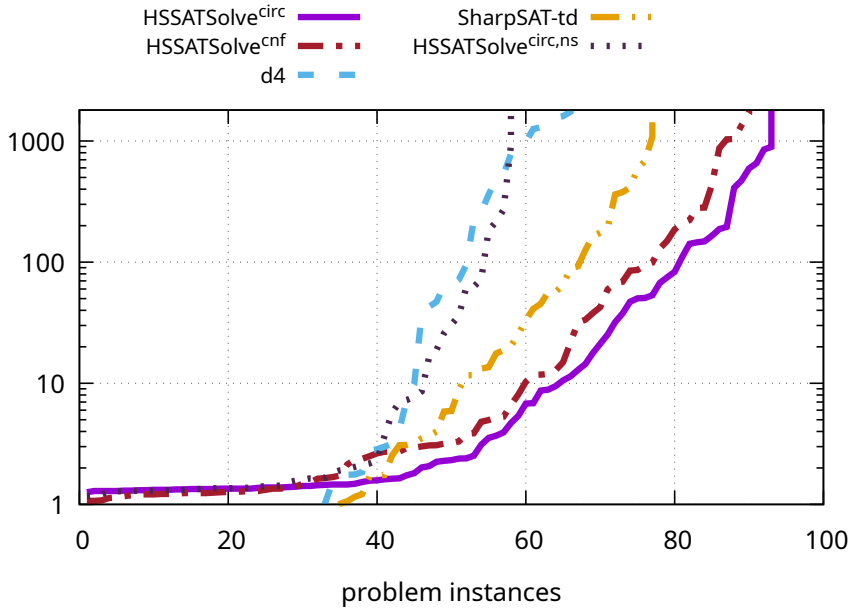

Figure 7 Type Я.

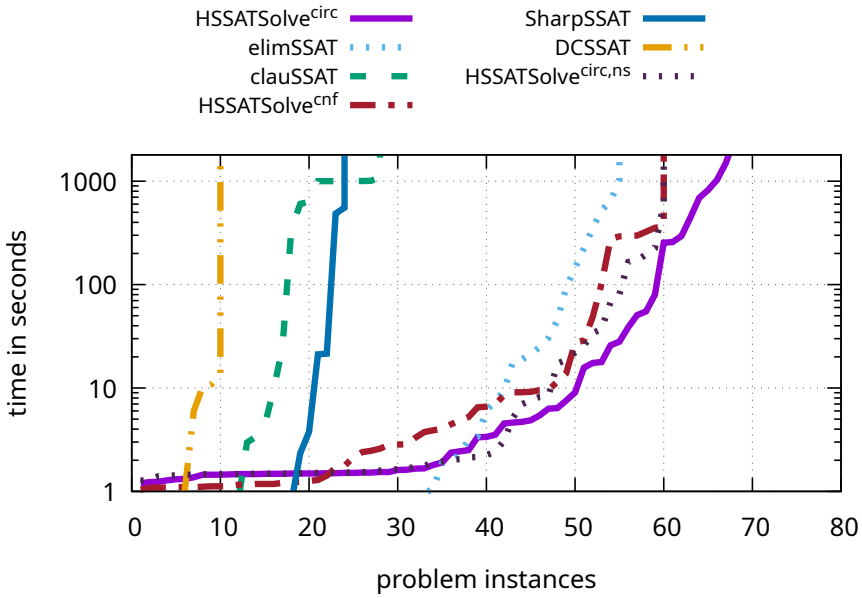

Figure 8 Type ∃∀.

version of HSSATSolve$^{\texttt{circ}}$ where group sifting during ROBDD construction is deactivated. One can see that the cactus plots for HSSATSolve$^{\texttt{circ,ns}}$ are pretty similar in Figs. 6, 7, and 8. This can be easily explained: The variables in the formulas of types ∀∃, ∀, and ∃∀ occur in the same order, see Eqns. (5), (7), and (6), they only differ from their quantifier types. The matrices of the formulas are (almost) identical ($\neg f_{IM}$ for Eqn. (5), f_{IM} for Eqn. (7), $f_{IM} \wedge (\vec{K} \neq \vec{K}_{\texttt{orig}})$ for Eqn. (6)). Thus, HSSATSolve$^{\texttt{circ,ns}}$ builds similar ROBDDs with the same variable order which is not changed during the construction. Since the run times of our approach are dominated by the ROBDD construction and evaluation is very fast in comparison, the differences between the results of HSSATSolve$^{\texttt{circ}}$ in Figs. 6 and 7 can be explained by the fact that group sifting with a single group of variables in formulas of type ∀ (Fig. 7) has more degrees of freedom than with two groups of variables for formulas of type ∀∃ (Fig. 6).

For our formulas of type ∃∀, HSSATSolve$^{\texttt{circ}}$ outperforms the existing solvers, see Fig. 8. Among the other solvers ElimSSAT performs best. ElimSSAT is well suited for our formulas with input probabilities of 0.5. In general, it is only suited for input probabilities which are a sum of negative powers of two [35]. If we have other probabilities for the inputs $\vec{X}$ in Eqn. (6), then the input probabilities have to be rounded and ElimSSAT produces errors of unknown and uncontrollable size. HSSATSolve can work with arbitrary rational numbers as input probabilities and it works with unlimited precision.

Formulas of type H compute the fraction of keys with high criticality, see Eqn. (8). Here we first investigate the effect of group sifting and of using logic synthesis by ABC before building ROBDDs. In HSSATSolve$^{\texttt{circ,nABC}}$ ABC is omitted, in HSSATSolve$^{\texttt{circ,ns}}$ group sifting is omitted, and in HSSATSolve$^{\texttt{circ,nsnABC}}$ both techniques are omitted. For this evaluation we fixed the criticality bound c to 0.999. (Note that there is basically no run time difference for our method with different criticality bounds. Different values of c do change the value of the result, of course, but not the evaluation steps which have to be performed by Alg. 1.) Results for formulas of type H can be found in Fig. 9. The results show that the

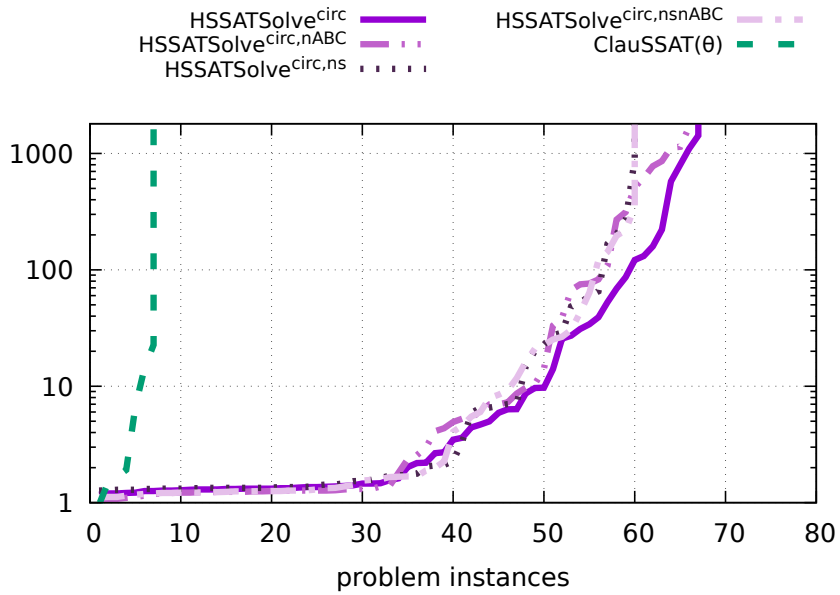

Figure 9 Type H.

effect of ABC is less significant than the effect of group sifting. Whereas ABC apparently accelerates the overall run times of the tool, it does not change the number of benchmarks which are solved before the timeout. Group sifting helps on the other hand, as already shown in the other experiments. Formulas of type H also lie in the class SSAT(Θ) recently proposed by Fan and Jiang [11]. They presented the prototype tool ClauSSat(Θ) for solving SSAT(Θ) formulas. However, Fig. 9 shows that ClauSSat(Θ) is currently not well suited for formulas from our application domain.

In Figs. 10 and 11 we demonstrate how our tool can be used for quality assessment in logic locking.

In a first experiment the criticality bound c of the keys is again fixed to 0.999 (which would mean for an application that we consider a key only as highly critical, if it produces correct outputs for at least 99,9% of the possible input combinations). We investigate how increasing the key lengths influences the average fraction of keys with high criticality. Fig. 10 shows results for benchmark circuits b01, b03, b05, b06, and b13. For the chosen logic locking method and the considered circuits the fraction of keys with high criticality rapidly decreases with increasing key lengths (chosen in steps from 4 to 32). Since there is at least one unlocking key, the fraction of highly critical keys can of course not be lower than $\frac{1}{2^{kl}}$ with key length kl. The results show e.g. that for circuit b05 the logic locking is not absolutely perfect, since for key length 16 there is not only one, but there are 4 critical keys (out of $2^{16} = 65536$ possible keys), for key length 32 there are 32 critical keys (out of 2^{32} possible keys).

Fig. 11 shows (for a fixed key length of 16) how the fraction of keys with high criticality changes, if we change our notion of "high criticality". If we choose a criticality bound $c = 80\%$, then circuit c3540, e.g., still has 1138 keys (a fraction of approximately 1.7%) with high criticality. Nevertheless, all of the considered circuits reach the lowest possible fraction of $\frac{1}{2^{16}}$ for key length 16, if we consider a key only as critical, if it unlocks the circuit completely ($c = 1.0$).

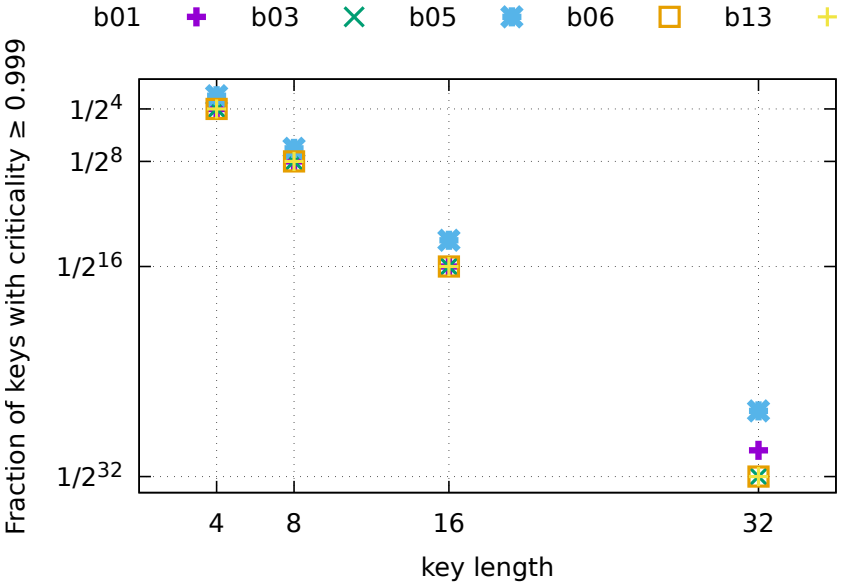

Figure 10 Fraction of keys with criticality ≥ 0.999, key lengths 4, 8, 16, 32.

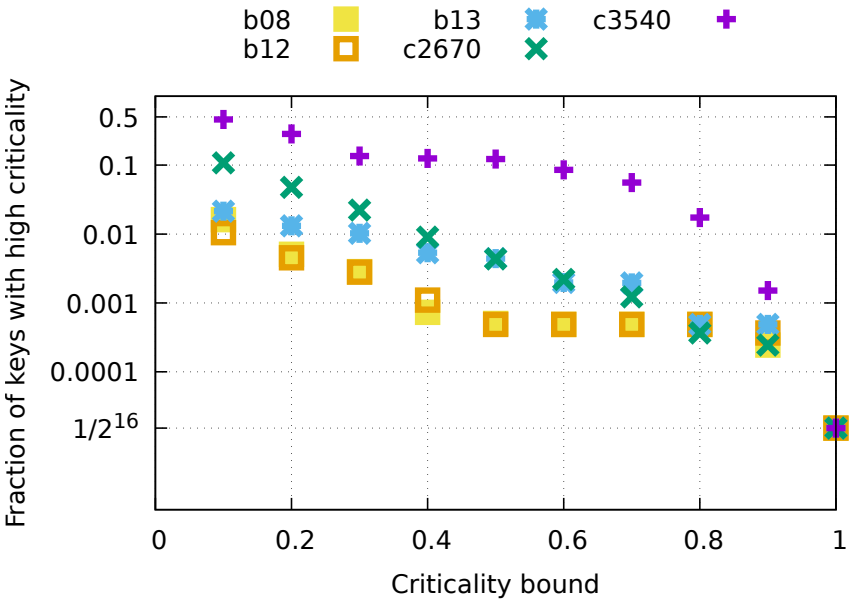

Figure 11 Fraction of keys with criticality $\geq 0.1, 0.2, \ldots, 0.8, 0.9, 1.0$, key length 16.

6 Conclusions and Future Work

We introduced the problem of Hierarchical Stochastic SAT, we presented a prototype solver for HSSAT, and we used the solver to investigate interesting questions in quality assessment for logic locking methods. For some subclasses of HSSAT formulas for which solvers already exist, the new solver compares favorably with the existing solvers. In the future, we plan to use our new tool to compare the quality of different existing logic locking schemes. We also plan to develop HSSAT algorithms going beyond ROBDD construction. Finally, we plan to add certification to the solver.

References

1 Alex Baumgarten, Akhilesh Tyagi, and Joseph Zambreno. Preventing IC piracy using reconfigurable logic barriers. *IEEE Des. Test Comput.*, 27(1):66–75, 2010. doi:10.1109/MDT.2010.24.

2 Berkeley Logic Synthesis and Verification Group. ABC: A system for sequential synthesis and verification. available at https://people.eecs.berkeley.edu/~alanmi/abc/, 2019.

3 Robert K. Brayton and Alan Mishchenko. ABC: an academic industrial-strength verification tool. In Tayssir Touili, Byron Cook, and Paul B. Jackson, editors, *Computer Aided Verification, 22nd International Conference, CAV 2010, Edinburgh, UK, July 15-19, 2010. Proceedings*, volume 6174 of *Lecture Notes in Computer Science*, pages 24–40. Springer, 2010. doi:10.1007/978-3-642-14295-6_5.

4 Franc Brglez and Hideo Fujiwara. A neutral netlist of 10 combinational benchmark circuits and a target translator in fortran. In *Proceedings of IEEE International Symposium Circuits and Systems (ISCAS 85)*, pages 677–692. IEEE Press, Piscataway, N.J., 1985.

5 Randal E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Computers*, 35(8):677–691, 1986. doi:10.1109/TC.1986.1676819.

6 Pei-Wei Chen, Yu-Ching Huang, and Jie-Hong R. Jiang. A sharp leap from quantified boolean formula to stochastic boolean satisfiability solving. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, pages 3697–3706. AAAI Press, 2021. doi:10.1609/AAAI.V35I5.16486.

7 Gustavo K. Contreras, Md. Tauhidur Rahman, and Mohammad Tehranipoor. Secure split-test for preventing IC piracy by untrusted foundry and assembly. In *2013 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems, DFTS 2013, New York City, NY, USA, October 2-4, 2013*, pages 196–203. IEEE Computer Society, 2013. doi:10.1109/DFT.2013.6653606.

8 Fulvio Corno, Matteo Sonza Reorda, and Giovanni Squillero. Rt-level itc'99 benchmarks and first ATPG results. *IEEE Des. Test Comput.*, 17(3):44–53, 2000. doi:10.1109/54.867894.

9 Sophie Dupuis, Papa-Sidi Ba, Giorgio Di Natale, Marie-Lise Flottes, and Bruno Rouzeyre. A novel hardware logic encryption technique for thwarting illegal overproduction and hardware trojans. In *2014 IEEE 20th International On-Line Testing Symposium, IOLTS 2014, Platja d'Aro, Girona, Spain, July 7-9, 2014*, pages 49–54. IEEE, 2014. doi:10.1109/IOLTS.2014.6873671.

10 Yu-Wei Fan and Jie-Hong R. Jiang. Sharpssat: A witness-generating stochastic boolean satisfiability solver. In Brian Williams, Yiling Chen, and Jennifer Neville, editors, *Thirty-Seventh AAAI Conference on Artificial Intelligence, AAAI 2023, Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence, IAAI 2023, Thirteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2023, Washington, DC, USA, February 7-14, 2023*, pages 3949–3958. AAAI Press, 2023. doi:10.1609/AAAI.V37I4.25509.

11 Yu-Wei Fan and Jie-Hong R. Jiang. Unifying decision and function queries in stochastic boolean satisfiability. In Michael J. Wooldridge, Jennifer G. Dy, and Sriraam Natarajan, editors, *Thirty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2024, Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence, IAAI 2024, Fourteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2014, February 20-27, 2024, Vancouver, Canada*, pages 7995–8003. AAAI Press, 2024. doi:10.1609/AAAI.V38I8.28637.

12 The gnu multiple precision arithmetic library. https://gmplib.org/, 2024.

13 Markus Hecher and Johannes Fichte. Model counting competition 2023. https://mccompetition.org/, 2023.

14 Alexander Ivrii, Sharad Malik, Kuldeep S. Meel, and Moshe Y. Vardi. On computing minimal independent support and its applications to sampling and counting. *Constraints An Int. J.*, 21(1):41–58, 2016. doi:10.1007/S10601-015-9204-Z.

15 Tuukka Korhonen and Matti Järvisalo. Integrating tree decompositions into decision heuristics of propositional model counters (short paper). In Laurent D. Michel, editor, *27th International Conference on Principles and Practice of Constraint Programming, CP 2021, Montpellier, France (Virtual Conference), October 25-29, 2021*, volume 210 of *LIPIcs*, pages 8:1–8:11. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.CP.2021.8.

16 Jean-Marie Lagniez, Emmanuel Lonca, and Pierre Marquis. Improving model counting by leveraging definability. In Subbarao Kambhampati, editor, *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*, pages 751–757. IJCAI/AAAI Press, 2016. URL: http://www.ijcai.org/Abstract/16/112.

17 Jean-Marie Lagniez and Pierre Marquis. An improved decision-dnnf compiler. In Carles Sierra, editor, *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, pages 667–673. ijcai.org, 2017. doi:10.24963/IJCAI.2017/93.

18 Stephen M. Majercik and Byron Boots. DC-SSAT: A divide-and-conquer approach to solving stochastic satisfiability problems efficiently. In Manuela M. Veloso and Subbarao Kambhampati, editors, *Proceedings, The Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference, July 9-13, 2005, Pittsburgh, Pennsylvania, USA*, pages 416–422. AAAI Press / The MIT Press, 2005. URL: http://www.aaai.org/Library/AAAI/2005/aaai05-066.php.

19 Marcel Merten, Sebastian Huhn, and Rolf Drechsler. Quality assessment of rfet-based logic locking protection mechanisms using formal methods. In *IEEE European Test Symposium, ETS 2022, Barcelona, Spain, May 23-27, 2022*, pages 1–2. IEEE, 2022. doi:10.1109/ETS54262.2022.9810459.

20 Shipra Panda and Fabio Somenzi. Who are the variables in your neighborhood. In Richard L. Rudell, editor, *Proceedings of the 1995 IEEE/ACM International Conference on Computer-Aided Design, ICCAD 1995, San Jose, California, USA, November 5-9, 1995*, pages 74–77. IEEE Computer Society / ACM, 1995. doi:10.1109/ICCAD.1995.479994.

21 Jeyavijayan Rajendran, Youngok K. Pino, Ozgur Sinanoglu, and Ramesh Karri. Security analysis of logic obfuscation. In Patrick Groeneveld, Donatella Sciuto, and Soha Hassoun, editors, *The 49th Annual Design Automation Conference 2012, DAC '12, San Francisco, CA, USA, June 3-7, 2012*, pages 83–89. ACM, 2012. doi:10.1145/2228360.2228377.

22 Jeyavijayan Rajendran, Huan Zhang, Chi Zhang, Garrett S. Rose, Youngok K. Pino, Ozgur Sinanoglu, and Ramesh Karri. Fault analysis-based logic encryption. *IEEE Trans. Computers*, 64(2):410–424, 2015. doi:10.1109/TC.2013.193.

23 Jarrod A. Roy, Farinaz Koushanfar, and Igor L. Markov. EPIC: ending piracy of integrated circuits. In Donatella Sciuto, editor, *Design, Automation and Test in Europe, DATE 2008, Munich, Germany, March 10-14, 2008*, pages 1069–1074. ACM, 2008. doi:10.1109/DATE.2008.4484823.

24 Ricardo Salmon and Pascal Poupart. On the relationship between satisfiability and markov decision processes. In Amir Globerson and Ricardo Silva, editors, *Proceedings of the Thirty-Fifth Conference on Uncertainty in Artificial Intelligence, UAI 2019, Tel Aviv, Israel, July 22-25, 2019*, volume 115 of *Proceedings of Machine Learning Research*, pages 1105–1115. AUAI Press, 2019. URL: `http://proceedings.mlr.press/v115/salmon20a.html`.

25 Christoph Scholl, Tobias Seufert, and Fabian Siegwolf. Hierarchical SSAT: Benchmarks and solver HSSATSolve. `https://nc.informatik.uni-freiburg.de/index.php/s/PQ9zBAejC2ERCTD`, 2024.

26 Shubham Sharma, Subhajit Roy, Mate Soos, and Kuldeep S. Meel. GANAK: A scalable probabilistic exact model counter. In Sarit Kraus, editor, *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, pages 1169–1176. ijcai.org, 2019. `doi:10.24963/IJCAI.2019/163`.

27 Friedrich Slivovsky. Interpolation-based semantic gate extraction and its applications to QBF preprocessing. In Shuvendu K. Lahiri and Chao Wang, editors, *Computer Aided Verification - 32nd International Conference, CAV 2020, Los Angeles, CA, USA, July 21-24, 2020, Proceedings, Part I*, volume 12224 of *Lecture Notes in Computer Science*, pages 508–528. Springer, 2020. `doi:10.1007/978-3-030-53288-8_24`.

28 Fabio Somenzi. Efficient manipulation of decision diagrams. *Int. J. Softw. Tools Technol. Transf.*, 3(2):171–181, 2001. `doi:10.1007/S100090100042`.

29 Mate Soos and Kuldeep S. Meel. Arjun: An efficient independent support computation technique and its applications to counting and sampling. In Tulika Mitra, Evangeline F. Y. Young, and Jinjun Xiong, editors, *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design, ICCAD 2022, San Diego, California, USA, 30 October 2022 - 3 November 2022*, pages 71:1–71:9. ACM, 2022. `doi:10.1145/3508352.3549406`.

30 Larry J. Stockmeyer. The polynomial-time hierarchy. *Theor. Comput. Sci.*, 3(1):1–22, 1976. `doi:10.1016/0304-3975(76)90061-X`.

31 Pramod Subramanyan, Sayak Ray, and Sharad Malik. Evaluating the security of logic encryption algorithms. In *IEEE International Symposium on Hardware Oriented Security and Trust, HOST 2015, Washington, DC, USA, 5-7 May, 2015*, pages 137–143. IEEE Computer Society, 2015. `doi:10.1109/HST.2015.7140252`.

32 Ryosuke Suzuki, Kenji Hashimoto, and Masahiko Sakai. Improvement of projected model-counting solver with component decomposition using SAT solving in components. Technical Report SIG-FPAI-103-B506, JSAI Technical Report, March 2017. in Japanese.

33 G. S. Tseitin. On the complexity of derivation in propositional calculus. In Jörg H. Siekmann and Graham Wrightson, editors, *Automation of Reasoning: 2: Classical Papers on Computational Logic 1967–1970*, pages 466–483. Springer Berlin Heidelberg, 1983.

34 Klaus W. Wagner. The complexity of combinatorial problems with succinct input representation. *Acta Informatica*, 23(3):325–356, 1986. `doi:10.1007/BF00289117`.

35 Hao-Ren Wang, Kuan-Hua Tu, Jie-Hong Roland Jiang, and Christoph Scholl. Quantifier elimination in stochastic boolean satisfiability. In Kuldeep S. Meel and Ofer Strichman, editors, *25th International Conference on Theory and Applications of Satisfiability Testing, SAT 2022, August 2-5, 2022, Haifa, Israel*, volume 236 of *LIPIcs*, pages 23:1–23:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. `doi:10.4230/LIPICS.SAT.2022.23`.

36 Yang Xie and Ankur Srivastava. Mitigating SAT attack on logic locking. In Benedikt Gierlichs and Axel Y. Poschmann, editors, *Cryptographic Hardware and Embedded Systems - CHES 2016 - 18th International Conference, Santa Barbara, CA, USA, August 17-19, 2016, Proceedings*, volume 9813 of *Lecture Notes in Computer Science*, pages 127–146. Springer, 2016. `doi:10.1007/978-3-662-53140-2_7`.

37 Muhammad Yasin, Bodhisatwa Mazumdar, Jeyavijayan (JV) Rajendran, and Ozgur Sinanoglu. Sarlock: SAT attack resistant logic locking. In William H. Robinson, Swarup Bhunia, and Ryan Kastner, editors, *2016 IEEE International Symposium on Hardware Oriented Security and Trust, HOST 2016, McLean, VA, USA, May 3-5, 2016*, pages 236–241. IEEE Computer Society, 2016. `doi:10.1109/HST.2016.7495588`.

38 Muhammad Yasin, Jeyavijayan (JV) Rajendran, Ozgur Sinanoglu, and Ramesh Karri. On improving the security of logic locking. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 35(9):1411–1424, 2016. doi:10.1109/TCAD.2015.2511144.

39 Muhammad Yasin, Abhrajit Sengupta, Mohammed Thari Nabeel, Mohammed Ashraf, Jeyavijayan Rajendran, and Ozgur Sinanoglu. Provably-secure logic locking: From theory to practice. In Bhavani Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, pages 1601–1618. ACM, 2017. doi:10.1145/3133956.3133985.

40 Muhammad Yasin, Abhrajit Sengupta, Benjamin Carrión Schäfer, Yiorgos Makris, Ozgur Sinanoglu, and Jeyavijayan Rajendran. What to lock?: Functional and parametric locking. In Laleh Behjat, Jie Han, Miroslav N. Velev, and Deming Chen, editors, *Proceedings of the on Great Lakes Symposium on VLSI 2017, Banff, AB, Canada, May 10-12, 2017*, pages 351–356. ACM, 2017. doi:10.1145/3060403.3060492.

A Proof of Lemma 12

Proof. Let Φ be a closed HSSAT formula with matrix $\texttt{matrix}(\Phi) = \phi$ which is recursively evaluated according to Def. 5 and let x be an existential variable in its prefix.

Let α be an assignment over $\{y \in \texttt{BV}(\Phi) \mid \pi^{-1}(y) < \pi^{-1}(x)\}$. Here we use the notation $\Phi|_\alpha$ for the closed HSSAT formula which replaces $\texttt{matrix}(\Phi)$ by its cofactor $\texttt{matrix}(\Phi)|_\alpha$, removes all comparisons according to case (d) of Def. 3 at hierarchy levels $> \texttt{hlevel}(x)$, and removes all variables $y_i \in \{y \in \texttt{BV}(\Phi) \mid \pi^{-1}(y) < \pi^{-1}(x)\}$ together with their quantifiers $Q_i \in \{\exists, \forall, \mathbb{H}^{p_i}\}$ from the prefix of Φ.

$\Phi|_\alpha = (\exists x \Phi')$ is a subproblem occurring during the recursive evaluation of Φ according to Def. 5 with $\texttt{matrix}(\Phi|_\alpha) = \phi|_\alpha$.

Let x be defined in terms of $\{y \in \texttt{BV}(\Phi) \mid \pi^{-1}(y) < \pi^{-1}(x)\}$ in ϕ and for all hl with $1 \leq hl \leq \texttt{hlevel}(x)$ the equation "$0\ \texttt{op}(hl)\ \texttt{prob}(hl)$" does not hold.

We make a case distinction wrt. the cofactors $(\phi|_\alpha)|_x$ and $(\phi|_\alpha)|_{\neg x}$ of $\phi|_\alpha$:

- Case 1: Both $(\phi|_\alpha)|_{\neg x}$ and $(\phi|_\alpha)|_x$ do not represent the constant 0 function.
  This means that there are two full assignments $\beta : \texttt{BV}(\Phi) \to \{0,1\}$ and $\beta' : \texttt{BV}(\Phi) \to \{0,1\}$ with $\beta(y) = \beta'(y) = \alpha(y)$ for all $y \in \texttt{BV}(\Phi)$ with $\pi^{-1}(y) < \pi^{-1}(x)$, $\beta(x) = 1$, $\beta'(x) = 0$, and $\beta[\phi] = \beta'[\phi] = 1$. This is a contradiction to the assumption that x is defined in terms of $\{y \in \texttt{BV}(\Phi) \mid \pi^{-1}(y) < \pi^{-1}(x)\}$ in ϕ, since β and β' both satisfy ϕ and agree on $\{y \in \texttt{BV}(\Phi) \mid \pi^{-1}(y) < \pi^{-1}(x)\}$, but $\beta(x) \neq \beta'(x)$. Thus, the case assumption cannot hold, if x is defined in terms of $\{y \in \texttt{BV}(\Phi) \mid \pi^{-1}(y) < \pi^{-1}(x)\}$ in ϕ.
- Case 2: $(\phi|_\alpha)|_{\neg x}$ represents the constant 0 function, $(\phi|_\alpha)|_x$ not.
  Since $\texttt{matrix}(\Phi'|_{\neg x}) = (\phi|_\alpha)|_{\neg x}$ represents the constant 0 function and for all hl with $1 \leq hl \leq \texttt{hlevel}(x)$ the equation $0\ \texttt{op}(hl)\ \texttt{prob}(hl)$ does not hold, we obtain $\textsc{Eval}(\Phi'|_{\neg x}) = 0$. We show $\textsc{Eval}(\Phi'|_{\neg x}) = 0$ by proving $\textsc{Eval}(\Psi) = 0$ for all subproblems Ψ of $\Phi'|_{\neg x}$ occurring during the recursive evaluation of $\Phi'|_{\neg x}$ according to Def. 5. We show this by induction over the sum $s := \texttt{hlevel}(\Psi) + |\texttt{BV}(\Psi)|$. If $s = 0$, then either case (1) or case (2) of Def. 5 applies to Ψ. Since $f_{(\phi|_\alpha)|_{\neg x}} = 0$ by case assumption, case (2) of Def. 5 cannot occur. We are in case (1) and $\textsc{Eval}(\Psi) = 0$.
  For cases (3), (4), and (5) of Def. 5 we have $\Psi = \exists z \Psi'$, $\Psi = \forall z \Psi'$, or $\Psi = \mathbb{H}^p z \Psi'$. We can assume $\textsc{Eval}(\Psi|_{\neg z}) = \textsc{Eval}(\Psi|_z) = 0$ by induction hypothesis and thus $\textsc{Eval}(\Psi) = 0$.
  For case (6) we consider $\Psi = (\Psi'\ \texttt{op}(\texttt{hlevel}(\Psi))\ \texttt{prob}(\texttt{hlevel}(\Psi)))$. We can assume by induction hypothesis that $\textsc{Eval}(\Psi') = 0$. Now we need our additional restriction on the choice of defined existential variables to ensure that the comparison according to Def. 5,

case (6), is not able to change the value 0 of $\textsc{Eval}(\Psi')$ into a different value. We have $1 \leq \texttt{hlevel}(\Psi) \leq \texttt{hlevel}(x)$, therefore 0 $\texttt{op}(\texttt{hlevel}(\Psi))$ $\texttt{prob}(\texttt{hlevel}(\Psi))$ does not hold and thus $\textsc{Eval}(\Psi) = 0$.
Now we have $\textsc{Eval}(\Phi'|_{\neg x}) = 0$. According to Def. 5

$$\textsc{Eval}(\exists x \Phi') = \max(\textsc{Eval}(\Phi'|_{\neg x}), \textsc{Eval}(\Phi'|_x)) = \max(0, \textsc{Eval}(\Phi'|_x)) = \textsc{Eval}(\Phi'|_x).$$

Altogether we have

$$\textsc{Eval}(\Phi|_\alpha) = \textsc{Eval}(\exists x \Phi') = \textsc{Eval}(\Phi'|_x). \tag{10}$$

Consider a definition ψ of x by $\{y \in \texttt{BV}(\Phi) \mid \pi^{-1}(y) < \pi^{-1}(x)\}$ in ϕ and a full assignment $\beta : \texttt{BV}(\Phi) \to \{0,1\}$ with $\beta(y) = \alpha(y)$ for all $y \in \texttt{BV}(\Phi)$ with $\pi^{-1}(y) < \pi^{-1}(x)$, $\beta(x) = 1$ and $\beta[\phi] = 1$. Such an assignment β exists, since $(\phi|_\alpha)|_x$ does not represent the constant 0 function. Since ψ is a definition, we have $\psi[\beta] = \beta(x) = 1$. Since ψ only depends on variables from $\{y \in \texttt{BV}(\Phi) \mid \pi^{-1}(y) < \pi^{-1}(x)\}$, we also have $\psi[\alpha] = \beta(x) = 1$. Thus, we have

$$\textsc{Eval}(\Phi[x := \psi]|_\alpha) = \textsc{Eval}(\Phi'[x := \psi[\alpha]]) = \textsc{Eval}(\Phi'|_x). \tag{11}$$

From Eqns. (10) and (11) we conclude

$$\textsc{Eval}(\Phi|_\alpha) = \textsc{Eval}(\Phi[x := \psi]|_\alpha). \tag{12}$$

- Case 3: $(\phi|_\alpha)|_x$ represents the constant 0 function, $(\phi|_\alpha)|_{\neg x}$ not.
  We conclude $\textsc{Eval}(\Phi|_\alpha) = \textsc{Eval}(\Phi[x := \psi]|_\alpha)$ similar to Case 2.
- Case 4: Both $(\phi|_\alpha)|_{\neg x}$ and $(\phi|_\alpha)|_x$ represent the constant 0 function.
  Similar to Case 2 we obtain $\textsc{Eval}(\Phi'|_{\neg x}) = 0$ and $\textsc{Eval}(\Phi'|_x) = 0$. This leads to

$$\textsc{Eval}(\Phi|_\alpha) = \textsc{Eval}(\exists x \Phi') = 0. \tag{13}$$

We have

$$\textsc{Eval}(\Phi[x := \psi]|_\alpha) = \textsc{Eval}(\Phi'[x := \psi[\alpha]]) = 0 \tag{14}$$

for arbitrary formulas ψ over $\{y \in \texttt{BV}(\Phi) \mid \pi^{-1}(y) < \pi^{-1}(x)\}$ and thus also for each definition of x.

Case 1 cannot occur and in the remaining cases we have $\textsc{Eval}(\Phi|_\alpha) = \textsc{Eval}(\Phi[x := \psi]|_\alpha)$ for arbitrary assignments α over $\{y \in \texttt{BV}(\Phi) \mid \pi^{-1}(y) < \pi^{-1}(x)\}$. This proves $\textsc{Eval}(\Phi) = \textsc{Eval}(\Phi[x := \psi])$. ◀