Model Checking Quantitative Hyperproperties*

Bernd Finkbeiner, Christopher Hahn, and Hazem Torfah

Reactive Systems Group
Saarland University
lastname@react.uni-saarland.de

Abstract. Hyperproperties are properties of sets of computation traces. In this paper, we study quantitative hyperproperties, which we define as hyperproperties that express a bound on the number of traces that may appear in a certain relation. For example, quantitative non-interference limits the amount of information about certain secret inputs that is leaked through the observable outputs of a system. Quantitative non-interference thus bounds the number of traces that have the same observable input but different observable output. We study quantitative hyperproperties in the setting of HyperLTL, a temporal logic for hyperproperties. We show that, while quantitative hyperproperties can be expressed in HyperLTL, the running time of the HyperLTL model checking algorithm is, depending on the type of property, exponential or even doubly exponential in the quantitative bound. We improve this complexity with a new model checking algorithm based on model-counting. The new algorithm needs only logarithmic space in the bound and therefore improves, depending on the property, exponentially or even doubly exponentially over the model checking algorithm of HyperLTL. In the worst case, the new algorithm needs polynomial space in the size of the system. Our Max#Sat-based prototype implementation demonstrates, however, that the counting approach is viable on systems with nontrivial quantitative information flow requirements such as a passcode checker.

1 Introduction

Model checking algorithms [17] are the cornerstone of computer-aided verification. As their input consists of both the system under verification and a logical formula that describes the property to be verified, they uniformly solve a wide range of verification problems, such as all verification problems expressible in linear-time temporal logic (LTL), computation-tree logic (CTL), or the modal μ-calculus. Recently, there has been a lot of interest in extending model checking from standard trace and tree properties to *information flow* policies like observational determinism or quantitative information flow. Such policies are called *hyperproperties* [21] and can be expressed in HyperLTL [18], an extension of LTL

* This work was partly supported by the ERC Grant 683300 (OSARES) and by the German Research Foundation (DFG) in the Collaborative Research Center 1223.

with trace quantifiers and trace variables. For example, *observational determinism* [47], the requirement that any pair of traces that have the same observable input also have the same observable output, can be expressed as the following HyperLTL formula: $\forall\pi.\forall\pi'.\,(\square\pi =_I \pi') \rightarrow (\square\pi =_O \pi')$ For many information flow policies of interest, including observational determinism, there is no longer a need for property-specific algorithms: it has been shown that the standard HyperLTL model checking algorithm [26] performs just as well as a specialized algorithm for the respective property.

The class of hyperproperties studied in this paper is one where, by contrast, the standard model checking algorithm performes badly. We are interested in *quantitative hyperproperties*, i.e., hyperproperties that express a bound on the number of traces that may appear in a certain relation. A prominent example of this class of properties is *quantitative non-interference* [43, 45], where we allow some flow of information but, at the same time, limit the amount of information that may be leaked. Such properties are used, for example, to describe the correct behavior of a password check, where some information flow is unavoidable ("the password was incorrect"), and perhaps some extra information flow is accceptable ("the password must contain a special character"), but the information should not suffice to guess the actual password. In HyperLTL, quantitative non-interference can be expressed [18] as the formula $\forall\pi_0.\,\forall\pi_1 \ldots \forall\pi_{2^c}.\;(\bigwedge_i \square(\pi_i =_I \pi_0)) \rightarrow \left(\bigvee_{i\neq j} \square(\pi_i =_O \pi_j)\right)$. The formula states that there do not exist $2^c + 1$ traces (corresponding to more than c bits of information) with the same observable input but different observable output. The bad performance of the standard model checking algorithm is a consequence of the fact that the $2^c + 1$ traces are tracked simultaneously. For this purpose, the model checking algorithm builds and analyzes a $(2^c + 1)$-fold self-composition of the system.

We present a new model checking algorithm for quantitative hyperproperties that avoids the construction of the huge self-composition. The key idea of our approach is to use *counting* rather than *checking* as the basic operation. Instead of building the self-composition and then *checking* the satisfaction of the formula, we add new atomic propositions and then *count* the number of sequences of evaluations of the new atomic propositions that satisfy the specification. Quantitative hyperproperties are expressions of the following form:

$$\forall\pi_1.\ldots.\forall\pi_k.\,\varphi \rightarrow (\#\sigma : X.\,\psi \triangleleft n),$$

where $\triangleleft \in \{\leq, <, \geq, >, =\}$. The universal quantifiers introduce a set of reference traces against which other traces can be compared. The formulas φ and ψ are HyperLTL formulas. The counting quantifer $\#\sigma : X.\,\psi$ counts the number of paths σ with different valuations of the atomic propositions X that satisfy ψ. The requirement that no more than c bits of information are leaked is the following quantitative hyperproperty:

$$\forall\pi.\,\#\sigma\colon O.\,\square(\pi =_I \sigma) \leq 2^c$$

As we show in the paper, such expressions do not change the expressiveness of the logic; however, they allow us to express quantitative hyperproperties in exponentially more concise form. The counting-based model checking algorithm then maintains this advantage with a logarithmic counter, resulting in exponentially better performance in both time and space.

The viability of our counting-based model checking algorithm is demonstrated on a SAT-based prototype implementation. For quantitative hyperproperties of intrest, such as bounded leakage of a password checker, our algorithm shows promising results, as it significantly outperforms existing model checking approaches.

1.1 Related Work

Quantitative information-flow has been studied extensively in the literature. See, for example, the following selection of contributions on this topic: [43, 34, 19, 14, 1, 32]. Multiple verification methods for quantitative information-flow were proposed for sequential systems. For example, with static analysis techniques [15], approximation methods [35], equivalence relations [22, 3], and randomized methods [35]. Quantitative information-flow for multi-threaded programs was considered in [11].

The study of quantitative information-flow in a reactive setting gained a lot of attention recently after the introduction of hyperproperties [21] and the idea of verifying the self-composition of a reactive system [6] in order to relate traces to each other. There are several possibilities to measure the amount of leakage, such as Shannon entropy [24, 15, 37], guessing entropy [34, 3], and min-entropy [43]. A classification of quantitative information-flow policies as safety and liveness hyperproperties was given in [46]. While several verification techniques for hyperproperties exists [5, 31, 38, 42], the literature was missing general approaches to quantitative information-flow control. SecLTL [25] was introduced as first general approach to model check (quantitative) hyperproperties, before HyperLTL [18], and its corresponding model checker [26], was introduced as a temporal logic for hyperproperties, which subsumes the previous approaches.

Using counting to compute the number of solutions of a given formula is studied in the literature as well and includes many probabilistic inference problems, such as Bayesian net reasoning [36], and planning problems, such as computing robustness of plans in incomplete domains [40]. State-of-the-art tools for propositional model counting are `Relsat` [33] and `c2d` [23]. Algorithms for counting models of temporal logics and automata over infinite words have been introduced in [27, 28, 44]. The counting of projected models, i.e., when some parts of the models are irrelevant, was studied in [2], for which tools such as `#CLASP` [2] and `DSharp_P` [2, 41] exist. Our SAT-based prototype implementation is based on a reduction to a Max#SAT [29] instance, for which a corresponding tool exists.

Among the already existing tools for computing the amount of information leakage, for example, `QUAIL` [8], which analyzes programs written in a specific while-language and `LeakWatch` [12], which estimates the amount of leakage in

Java programs, `Moped-QLeak` [9] is closest to our approach. However, their approach of computing a symbolic summary as an Algebraic Decision Diagram is, in contrast to our approach, solely based on model counting, not maximum model counting.

2 Preliminaries

2.1 HyperLTL

HyperLTL [18] extends linear-time temporal logic (LTL) with trace variables and trace quantifiers. Let AP be a set of *atomic propositions*. A *trace* t is an infinite sequence over subsets of the atomic propositions. We define the set of traces $TR \coloneqq (2^{AP})^\omega$. A subset $T \subseteq TR$ is called a *trace property* and a subset $H \subseteq 2^{TR}$ is called a *hyperproperty*. We use the following notation to manipulate traces: let $t \in TR$ be a trace and $i \in \mathbb{N}$ be a natural number. $t[i]$ denotes the i-th element of t. Therefore, $t[0]$ represents the starting element of the trace. Let $j \in \mathbb{N}$ and $j \geq i$. $t[i, j]$ denotes the sequence $t[i]\ t[i+1] \dots t[j-1]\ t[j]$. $t[i, \infty]$ denotes the infinite suffix of t starting at position i.

HyperLTL Syntax. Let $\mathcal{V}$ be an infinite supply of trace variables. The syntax of HyperLTL is given by the following grammar:

$$\begin{aligned} \psi &::= \exists \pi.\ \psi \mid \forall \pi.\ \psi \mid \varphi \\ \varphi &::= a_\pi \mid \neg\varphi \mid \varphi \vee \varphi \mid \bigcirc \varphi \mid \varphi\, \mathcal{U}\, \varphi \end{aligned}$$

where $a \in AP$ is an atomic proposition and $\pi \in \mathcal{V}$ is a trace variable. Note that atomic propositions are indexed by trace variables. The quantification over traces makes it possible to express properties like "on all traces ψ must hold", which is expressed by $\forall \pi.\ \psi$. Dually, one can express that "there exists a trace such that ψ holds", which is denoted by $\exists \pi.\ \psi$. The derived operators $\diamondsuit$, $\square$, and $\mathcal{W}$ are defined as for LTL. We abbreviate the formula $\bigwedge_{x \in X} (x_\pi \leftrightarrow x_{\pi'})$, expressing that the traces π and π' are equal with respect to a set $X \subseteq AP$ of atomic propositions, by $\pi =_X \pi'$. Furthermore, we call a trace variable π free in a HyperLTL formula if there is no quantification over π and we call a HyperLTL formula φ closed if there exists no free trace variable in φ.

HyperLTL Semantics. A HyperLTL formula defines a *hyperproperty*, i.e., a set of sets of traces. A set T of traces satisfies the hyperproperty if it is an element of this set of sets. Formally, the semantics of HyperLTL formulas is given with respect to a *trace assignment* Π from $\mathcal{V}$ to TR, i.e., a partial function mapping trace variables to actual traces. $\Pi[\pi \mapsto t]$ denotes that π is mapped to t, with everything else mapped according to Π. $\Pi[i, \infty]$ denotes the trace assignment

that is equal to $\Pi(\pi)[i, \infty]$ for all π.

$$
\begin{array}{lll}
\Pi \models_T \exists \pi.\psi & \text{iff} & \text{there exists } t \in T \ : \ \Pi[\pi \mapsto t] \models_T \psi \\
\Pi \models_T \forall \pi.\psi & \text{iff} & \text{for all } t \in T \ : \ \Pi[\pi \mapsto t] \models_T \psi \\
\Pi \models_T a_\pi & \text{iff} & a \in \Pi(\pi)[0] \\
\Pi \models_T \neg\psi & \text{iff} & \Pi \not\models_T \psi \\
\Pi \models_T \psi_1 \vee \psi_2 & \text{iff} & \Pi \models_T \psi_1 \text{ or } \Pi \models_T \psi_2 \\
\Pi \models_T \bigcirc \psi & \text{iff} & \Pi[1, \infty] \models_T \psi \\
\Pi \models_T \psi_1 \mathcal{U} \psi_2 & \text{iff} & \text{there exists } i \geq 0 : \Pi[i, \infty] \models_T \psi_2 \\
 & & \text{and for all } 0 \leq j < i \text{ we have } \Pi[j, \infty] \models_T \psi_1
\end{array}
$$

We say a set of traces T *satisfies* a HyperLTL formula φ if $\Pi \models_T \varphi$, where Π is the empty trace assignment.

2.2 System model

A *Kripke structure* is a tuple $K = (S, s_0, \delta, AP, L)$ consisting of a set of states S, an initial state $s_0 \in S$, a transition function $\delta : S \to 2^S$, a set of *atomic propositions* AP, and a *labeling function* $L : S \to 2^{AP}$, which labels every state with a set of atomic propositions. We assume that each state has a successor, i.e., $\delta(s) \neq \emptyset$. This ensures that every run on a Kripke structure can always be extended to an infinite run. We define a *path* of a Kripke structure as an infinite sequence of states $s_0 s_1 \cdots \in S^\omega$ such that s_0 is the initial state of K and $s_{i+1} \in \delta(s_i)$ for every $i \in \mathbb{N}$. We denote the set of all paths of K that start in a state s with $Paths(K, s)$. Furthermore, $Paths^*(K, s)$ denotes the set of all path prefixes and $Paths^\omega(K, s)$ the set of all path suffixes. A *trace* of a Kripke structure is an infinite sequence of sets of atomic propositions $L(s_0), L(s_1), \cdots \in (2^{AP})^\omega$, such that s_0 is the initial state of K and $s_{i+1} \in \delta(s_i)$ for every $i \in \mathbb{N}$. We denote the set of all traces of K that start in a state s with $TR(K, s)$. We say that a Kripke structure K *satisfies* a HyperLTL formula φ if its set of traces satisfies φ, i.e., if $\Pi \models_{TR(K,s_0)} \varphi$, where Π is the empty trace assignment.

2.3 Automata over infinite words

In our construction we use automata over infinite words. A *Büchi automaton* is a tuple $\mathcal{B} = (Q, Q_0, \delta, \Sigma, F)$, where Q is a set of states, Q_0 is a set of initial states, $\delta : Q \times \Sigma \to 2^Q$ is a transition relation, and $F \subset Q$ are the accepting states. A run of $\mathcal{B}$ on an infinite word $w = \alpha_1 \alpha_2 \cdots \in \Sigma^\omega$ is an infinite sequence $r = q_0 q_1 \cdots \in Q^\omega$ of states, where $q_0 \in Q_0$ and for each $i \geq 0$, $q_{i+1} = \delta(q_i, \alpha_{i+1})$. We define $\mathbf{Inf}(r) = \{q \in Q \mid \forall i \exists j > i.\ r_j = q\}$. A run r is called accepting if $\mathbf{Inf}(r) \cap F \neq \emptyset$. A word w is accepted by $\mathcal{B}$ and called a *model* of $\mathcal{B}$ if there is an accepting run of $\mathcal{B}$ on w.

Furthermore, an *alternating automaton*, whose runs generalize from sequences to trees, is a tuple $\mathcal{A} = (Q, Q_0, \delta, \Sigma, F)$. Q, Q_0, Σ, and F are defined as above

and $\delta : Q \times \Sigma \to \mathbb{B}^+Q$ being a transition function, which maps a state and a symbol into a Boolean combination of states. Thus, a run(-tree) of an alternating Büchi automaton $\mathcal{A}$ on an infinite word w is a Q-labeled tree. A word w is accepted by $\mathcal{A}$ and called a *model* if there exists a run-tree T such that all paths p trough T are accepting, i.e., $\mathbf{Inf}(p) \cap F \neq \emptyset$.

A strongly connected component (SCC) in $\mathcal{A}$ is a maximal strongly connected component of the graph induced by the automaton. An SCC is called *accepting* if one of its states is an accepting state in $\mathcal{A}$.

3 Quantitative Hyperproperties

Quantitative Hyperproperties are properties of sets of computation traces that express a bound on the number of traces that may appear in a certain relation. In the following, we study quantitative hyperproperties that are specified in terms of HyperLTL formulas. We consider expressions of the following general form:

$$\forall \pi_1, \ldots, \pi_k.\ \varphi \to (\#\sigma : A.\ \psi \triangleleft n)$$

Both the universally quantified variables $\pi_1, \ldots, \pi_k$ and the variable σ after the *counting* operator $\#$ are trace variables; φ is a HyperLTL formula over atomic propositions AP and free trace variables $\pi_1 \ldots \pi_k$; $A \subseteq AP$ is a set of atomic propositions; ψ is a HyperLTL formula over atomic propositions AP and free trace variables $\pi_1 \ldots \pi_k$ and, additionally σ. The operator $\triangleleft \in \{<, \leq, =, >, \geq\}$ is a comparison operator; and $n \in \mathbb{N}$ is a natural number.

For a given set of traces T and a valuation of the trace variables $\pi_1, \ldots, \pi_k$, the term $\#\sigma : A.\ \psi$ computes the number of traces σ in T that differ in their valuation of the atomic propositions in A and satisfy ψ. The expression $\#\sigma : A.\ \psi \triangleleft n$ is *true* iff the resulting number satisfies the comparison with n. Finally, the complete expression $\forall \pi_1, \ldots, \pi_k.\ \varphi \to (\#\sigma : A.\ \psi \triangleleft n)$ is *true* iff for all combinations $\pi_1, \ldots, \pi_k$ of traces in T that satisfy φ, the comparison $\#\sigma : A.\ \psi \triangleleft n$ is satisfied.

Example 1 (Quantitative non-interference). Quantitative information-flow policies [30, 13, 34, 20] allow the flow of a bounded amount of information. One way to measure leakage is with *min-entropy* [43], which quantifies the amount of information an attacker can gain given the answer to a single guess about the secret. The *bounding problem* [45] for min-entropy is to determine whether that amount is bounded from above by a constant 2^c, corresponding to c bits. We assume that the program whose leakage is being quantified is deterministic, and assume that the secret input to that program is uniformly distributed. The bounding problem then reduces to determining that there is no tuple of $2^c + 1$ distinguishable traces [43, 45]. Let $O \subseteq AP$ be the set of observable outputs. A simple quantitative information flow policy is then the following quantitative hyperproperty, which bounds the number of distinguishable outputs to 2^c, corresponding to a bound of c bits of information:

$$\#\sigma : O.\ \mathit{true} \leq 2^c$$

A slightly more complicated information flow policy is quantitative non-interference. In quantitative non-interference, the bound must be satisfied for every individual input. Let $I \subseteq AP$ be the observable inputs to the system. Quantitative non-interference is the following quantitative hyperproperty[1]:

$$\forall \pi.\, \#\sigma \colon O.\, (\square(\pi =_I \sigma)) \leq 2^c$$

For each trace π in the system, the property checks whether there are more than 2^c traces σ that have the same observable input as π but different observable output.

Example 2 (Deniability). A program satisfies *deniability* (see, for example, [7, 10]) when there is no proof that a certain input occurred from simply observing the output, i.e., given an output of a program one cannot derive the input that lead to this output. A deterministic program satisfies deniability when each output can be mapped to at least two inputs. A quantitative variant of deniability is when we require that the number of corresponding inputs is larger than a given threshold. Quantitative deniability can be specified as the following quantitative Hyperproperty:

$$\forall \pi.\, \#\sigma \colon I.\, (\square(\pi =_O \sigma)) > n$$

For all traces π of the system we count the number of sequences σ in the system with different input sequences and the same output sequence of π, i.e., for the fixed output sequence given by π we count the number of input sequences that lead to this output.

4 Model Checking Quantitative Hyperproperties

We present a model checking algorithm for quantitative hyperproperties based on model counting. The advantage of the algorithm is that its runtime complexity is independent of the bound n and thus avoids the n-fold self-composition necessary for any encoding of the quantitative hyperproperty in HyperLTL.

Before introducing our novel counting-based algorithm, we start by a translation of quantitative hyperproperties into formulas in HyperLTL and establishing an exponential lower bound for its representation.

4.1 Standard model checking algorithm: encoding quantitative hyperproperties in HyperLTL

The idea of the reduction is to check a lower bound of n traces by existentially quantifying over n traces, and to check an upper bound of n traces by *universally* quantifying over $n+1$ traces. The resulting HyperLTL formula can be verified using the standard model checking algorithm for HyperLTL [18].

[1] We write $\pi =_A \pi'$ short for $\pi_A = \pi'_A$ where π_A is the A-projection of π

Theorem 1. *Every quantitative hyperproperty* $\forall \pi_1, \dots, \pi_k.\ \psi_\iota \to (\#\sigma : A.\ \psi \lhd n)$ *can be expressed as a HyperLTL formula. For* $\lhd \in \{\leq\}(\{<\})$*, the HyperLTL formula has* $n + k + 1$*(resp.* $n + k$*) universal trace quantifiers in addition to the quantifiers in* ψ_ι *and* ψ*. For* $\lhd \in \{\geq\}(\{>\})$*, the HyperLTL formula has* k *universal trace quantifiers and* n *(resp.* $n + 1$*) existential trace quantifiers in addition to the quantifiers in* ψ_ι *and* ψ*. For* $\lhd \in \{=\}$*, the HyperLTL formula has* $k + n + 1$ *universal trace quantifiers and* n *existential trace quantifiers in addition to the quantifiers in* ψ_ι *and* ψ*.*

Proof. For $\lhd \in \{\leq\}$, we encode the quantitative hyperproperty $\forall \pi_1, \dots, \pi_k.\ \psi_\iota \to (\#\sigma : A.\ \psi \lhd n)$ as the following HyperLTL formula:

$$\forall \pi_1, \dots, \pi_k.\ \forall \pi'_1, \dots, \pi'_{n+1}.\ \left(\psi_\iota \wedge \bigwedge_{i \neq j} \Diamond(\pi'_i \neq_A \pi'_j) \right) \to \left(\bigvee_i \neg \psi[\sigma \mapsto \pi'_i] \right)$$

where $\psi[\sigma \mapsto \pi'_i]$ is the HyperLTL formula ψ with all occurrences of σ replaced by π'_i. The formula states that there is no tuple of $n + 1$ traces $\pi'_1, \dots, \pi'_{n+1}$ different in the evaluation of A, that satisfy ψ. In other words, for every $n + 1$ tuple of traces $\pi'_1, \dots, \pi'_{n+1}$ that differ in the evaluation of A, one of the paths must violate ψ. For $\lhd \in \{<\}$, we use the same formula, with $\forall \pi'_1, \dots, \pi'_n$ instead of $\forall \pi'_1, \dots, \pi'_{n+1}$.

For $\lhd \in \{\geq\}$, we encode the quantitative hyperproperty analogously as the HyperLTL formula

$$\forall \pi_1, \dots, \pi_k.\ \exists \pi'_1, \dots, \pi'_n.\ \psi_\iota \to \left(\bigwedge_{i \neq j} \Diamond(\pi'_i \neq_A \pi'_j) \right) \wedge \left(\bigwedge_i \psi[\sigma \mapsto \pi'_i] \right)$$

The formula states that there exist paths $\pi'_1, \dots, \pi'_n$ that differ in the evaluation of A and that all satisfy ψ. For $\lhd \in \{>\}$, we use the same formula, with $\exists \pi'_1, \dots, \pi'_{n+1}$ instead of $\forall \pi'_1, \dots, \pi'_n$. Lastly, for $\lhd \in \{=\}$, we encode the quantitative hyperproperty as a conjunction of the encodings for $\leq$ and for $\geq$.

Example 3 (Quantitative non-interference in HyperLTL). As discussed in Example 1, quantitative non-interference is the quantitative hyperproperty

$$\forall \pi.\ \#\sigma \colon O.\ \square(\pi =_I \sigma) \leq 2^c,$$

where we measure the amount of leakage with min-entropy [43]. The bounding problem for min-entropy asks whether the amount of information leaked by a system is bounded by a constant 2^c where c is the number of bits. This is encoded in HyperLTL as the requirement that there are no $2^c + 1$ traces distinguishable in their output:

$$\forall \pi_0.\ \forall \pi_1 \dots \forall \pi_{2^c}.\ \left(\bigwedge_i \square(\pi_i =_I \pi_0) \right) \to \left(\bigvee_{i \neq j} \square(\pi_i =_O \pi_j) \right).$$

This formula is equivalent to the formalization of quantitative non-interference given in [26].

Model checking quantitative hyperproperties via the reduction to HyperLTL is very expensive. In the best case, when $\triangleleft \in \{\leq, <\}$, ψ_ι does not contain existential quantifiers, and ψ does not contain universal quantifiers, we obtain an HyperLTL formula without quantifier alternations, where the number of quantifiers grows linearly with the bound n. For m quantifiers, the HyperLTL model checking algorithm [26] constructs and analyzes the m-fold self-composition of the Kripke structure. The running time of the model checking algorithm is thus exponential in the bound. If $\triangleleft \in \{\geq, >, =\}$, the encoding additionally introduces a quantifier alternation. The model checking algorithm checks quantifier alternations via a complementation of Büchi automata, which adds another exponent, resulting in an overall doubly exponential running time.

The model checking algorithm we introduce in the next section avoids the n-fold self-composition needed in the model checking algorithm of HyperLTL and its complexity is independent of the bound n.

4.2 Counting-based model checking algorithm

A Kripke structure $K = (S, s_0, \tau, AP, L)$ violates a quantitative hyperproperty

$$\varphi = \forall \pi_1, \ldots, \pi_k.\ \psi_\iota \rightarrow (\#\sigma : A.\psi \triangleleft n)$$

if there is a k-tuple $t = (\pi_1, \ldots, \pi_k)$ of traces $\pi_i \in TR(K)$ that satisfies the formula

$$\exists \pi_1, \ldots, \pi_k.\ \psi_\iota \wedge (\#\sigma : A.\ \psi \,\overline{\triangleleft}\, n)$$

where $\overline{\triangleleft}$ is the negation of the comparison operator $\triangleleft$. The tuple t then satisfies the property ψ_ι and the number of $(k+1)$-tuples $t' = (\pi_1, \ldots, \pi_k, \sigma)$ for $\sigma \in TR(K)$ that satisfy ψ and differ pairwise in the A-projection of σ satisfies the comparison $\overline{\triangleleft}\ n$ (The A-projection of a sequence σ is defined as the sequence $\sigma_A \in (2^A)^\omega$, such that for every position i and every $a \in A$ it holds that $a \in \sigma_A[i]$ if and only if $a \in \sigma[i]$). The tuples t' can be captured by the automaton composed of the product of an automaton $A_{\psi_\iota \wedge \psi}$ that accepts all $k+1$ of traces that satisfy both ψ_ι and ψ and a $k+1$-self composition of K. Each accepting run of the product automaton presents $k+1$ traces of K that satisfy $\psi_\iota \wedge \psi$. On top of the product automaton, we apply a special counting algorithm which we explain in detail in Section 4.4 and check if the result satisfies the comparison $\overline{\triangleleft}\ n$.

Algorithm 1 gives a general picture of our model checking algorithm. The algorithm has two parts. The first part applies if the relation $\overline{\triangleleft}$ is one of $\{\geq, >\}$. In this case, the algorithm checks whether a sequence over AP_ψ (propositions in ψ) corresponds to infinitely many sequences over A. This is done by checking whether the product automaton B has a so-called *doubly pumped lasso*(DPL), a subgraph with two connected lassos, with a unique sequence over AP_ψ and different sequences over A. Such a doubly pumped lasso matches the same sequence over AP_ψ with infinitely many sequences over A (more in section 4.4). If no doubly pumped lasso is found, a projected model counting algorithm is applied in the second part of the algorithm in order to compute either the maximum or the minimum value, corresponding to the comparison operator $\overline{\triangleleft}$. In the next subsections, we explain the individual parts of the algorithm in detail.

Algorithm 1 Counting-based Model Checking of Quantitative Hyperproperties

Input: Quantitative Hyperproperty $\varphi = \forall \pi_1 \dots \pi_k.\ \psi_\iota \rightarrow (\#\sigma : A.\psi \triangleleft n)$, Kripke Structure $K = (S, s_0, \tau, AP, L)$
Output: $K \models \varphi$
1: $B = QHLTL2BA(K, \pi_1, \dots, \pi_k, \psi_\iota \wedge \psi)$
2: /*Check Infinity*/
3: **if** $\triangleleft \in \{\geq, >\}$ **then**
4: $ce = DPL(B)$
5: **if** $ce \neq \bot$ **then**
6: **return** ce
7: /*Apply Projected Counting Algorithm*/
8: **if** $\triangleleft \in \{\geq, >\}$ **then**
9: $ce = MaxCount(B, n, \triangleleft)$
10: **else**
11: $ce = MinCount(B, n, \triangleleft)$
12: **return** ce

4.3 Büchi automata for quantitative hyperproperties

For a quantitative hyperproperty $\varphi = \forall \pi_1 \dots \pi_k.\ \psi_\iota \rightarrow (\#\sigma : A.\psi \triangleleft n)$ and a Kripke structure $K = (S, s_0, \tau, AP, L)$, we first construct an alternating automaton $A_{\psi_\iota \wedge \psi}$ for the HyperLTL property $\psi_\iota \wedge \psi$. Let $A_{\psi_1} = (Q_1, q_{0,1}, \Sigma_2, \delta_1, F_1)$ and $A_{\psi_2} = (Q_2, q_{0,2}, \Sigma_2, \delta_2, F_2)$ be alternating automata for subformulas ψ_1 and ψ_2. Let $\Sigma = 2^{AP_\varphi}$ where AP_φ are all indexed atomic propositions that appear in φ. $A_{\psi_\iota \wedge \psi}$ is constructed using following rules[2]:

$\varphi = a_\pi$	$A_\varphi = (\{q_0\}, q_0, \Sigma, \delta, \emptyset)$ where $\delta(q_0, \alpha) = (a_\pi \in \alpha)$
$\varphi = \neg a_\pi$	$A_\varphi = (\{q_0\}, q_0, \Sigma, \delta, \emptyset)$ where $\delta(q_0, \alpha) = (a_\pi \notin \alpha)$
$\varphi = \psi_1 \wedge \psi_2$	$A_\varphi = (Q_1 \cup Q_2 \cup \{q_0\}, q_0, \Sigma, \delta, F_1 \cup F_2)$ where $\delta(q, \alpha) = \delta_1(q_{0,1}, \alpha) \wedge \delta_2(q_{0,2}, \alpha)$ and $\delta(q, \alpha) = \delta_i(q, \alpha)$ when $q \in Q_i$ for $i \in \{1, 2\}$
$\varphi = \psi_1 \vee \psi_2$	$A_\varphi = (Q_1 \cup Q_2 \cup \{q_0\}, q_0, \Sigma, \delta, F_1 \cup F_2)$ where $\delta(q, \alpha) = \delta_1(q_{0,1}, \alpha) \vee \delta_2(q_{0,2}, \alpha)$ and $\delta(q, \alpha) = \delta_i(q, \alpha)$ when $q \in Q_i$ for $i \in \{1, 2\}$
$\varphi = \bigcirc \psi_1$	$A_\varphi = (Q_1 \cup \{q_0\}, q_0, \Sigma, \delta, F_1)$ where $\delta(q, \alpha) = q_{0,1}$ and $\delta(q, \alpha) = \delta_1(q, \alpha)$ for $q \in Q_1$
$\varphi = \psi_1 \,\mathcal{U}\, \psi_2$	$A_\varphi = (Q_1 \cup Q_2 \cup \{q_0\}, q_0, \Sigma, \delta, F_1 \cup F_2)$ where $\delta(q_0, \alpha) = \delta_2(q_{0,2}, \alpha) \vee (\delta_1(q_{0,1}, \alpha) \wedge q_0)$ and $\delta(q, \alpha) = \delta_i(q, \alpha)$ when $q \in Q_i$ for $i \in \{1, 2\}$
$\varphi = \psi_1 \,\mathcal{R}\, \psi_2$	$A_\varphi = (Q_1 \cup Q_2 \cup \{q_0\}, q_0, \Sigma, \delta, F_1 \cup F_2 \cup \{q_0\})$ where $\delta(q_0, \alpha) = \delta_2(q_{0,2}, \alpha) \wedge (\delta_1(q_{0,1}, \alpha) \vee q_0)$ and $\delta(q, \alpha) = \delta_i(q, \alpha)$ when $q \in Q_i$ for $i \in \{1, 2\}$

For a quantified formula $\varphi = \exists \pi.\psi_1$, we construct the product automaton of the Kripke structure K and the Büchi automaton of ψ_1. Here we reduce the alphabet of the automaton by projecting all atomic proposition in AP_π away:

$\varphi = \exists \pi.\psi_1$	$A_\varphi = (Q_1 \times S \cup \{q_0\}, \Sigma \setminus AP_\pi, \delta, F_1 \times S)$ where $\delta(q_0, \alpha) = \{(q', s') \mid q' \in \delta_1(q_{0,1}, \alpha \cup \alpha'), s' \in \tau(s_0), (L(s_0))_\pi =_{AP_\pi} \alpha'\}$ and $\delta((q, s), \alpha) = \{(q', s') \mid q' \in \delta_1(q, \alpha \cup \alpha'), s' \in \tau(s), (L(s))_\pi =_{AP_\pi} \alpha'\}$

Given the Büchi automaton for the hyperproperty $\psi_\iota \wedge \psi$ it remains to construct the product with the $k+1$-self composition of K. The transitions of the automaton are defined over labels from $\Sigma = 2^{AP^*}$ where $AP^* = AP_\sigma \cup \bigcup_i AP_{\pi_i}$.

[2] The construction follows the one presented in [26] with a slight modification on the labeling of transitions. Labeling over atomic proposition instead of the states of the Kripke structure suffices, as any nondeterminism in the Kripke structure is inherently resolved, because we quantify over trace not paths

$A_{\psi_\iota \wedge \psi}$. This is necessary to identify which transition was taken in each copy of K, thus, mirroring a tuple of traces in K. For each of the variables $\pi_1, \dots \pi_k$ and σ we use following rule:

$\varphi = \exists \pi . \psi_1$	$A_\varphi = (Q_1 \times S \cup \{q_0\}, \Sigma, \delta, F_1 \times S)$ where $\delta(q_0, \alpha) = \{(q', s') \mid q' \in \delta_1(q_{0,1}, \alpha), s' \in \tau(s_0), (L(s_0))_\pi =_{AP_\pi}\}$ and $\delta((q, s), \alpha) = \{(q', s') \mid q' \in \delta_1(q, \alpha), s' \in \tau(s), (L(s))_\pi =_{AP_\pi}\}$

Finally, we transform the resulting alternating automaton to an equivalent Büchi automaton following the construction of Miyano and Hayashi [39].

4.4 Counting models of ω-Automata

Computing the number of words accepted by a Büchi automaton can be done by examining its accepting lassos. Consider, for example, the Büchi automata over the alphabet $2^{\{a\}}$ in Figure 1. The automaton on the left has one accepting lasso $(q_0)^\omega$ and thus has only one model, namely $\{a\}^\omega$. The automaton on the right has infinitely many accepting lassos $(q_0\{\})^i\{a\}(q_1(\{\} \vee \{a\}))^\omega$ that accept infinitely many different words all of the from $\{\}^*\{a\}(\{\} \vee \{a\})^\omega$. Computing

Fig. 1. Büchi automata with one model (left) and infinitely many models (right).

the models of a Büchi automaton is insufficient for model checking quantitative hyperproperties as we are not interested in the total number of models. We rather *maximize*, respectively *minimize*, over sequences of subsets of atomic propositions *the number of projected models* of the Büchi automaton. For instance, consider the automaton given in Figure 2. The automaton has infinitely

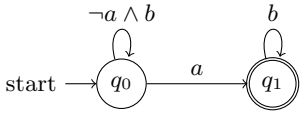

Fig. 2. A two-state Büchi automaton, such that there exist exactly two $\{b\}$-projected models for each $\{a\}$-projected sequence.

many models. However, the maximum number of sequences $\sigma_b \in 2^{\{b\}}$ that correspond to accepting lassos in the automaton with a unique sequence $\sigma_a \in 2^{\{a\}}$ is two: For example, let n be a natural number. For any model of the automaton and for each sequence $\sigma_a := \{\}^n\{a\}(\{\})^\omega$ the automaton accepts the following two sequences: $\{b\}^n\{\}\{b\}^\omega$ and $\{b\}^\omega$. Formally, given a Büchi automaton $\mathcal{B}$ over

AP and a set A, such that $A \subseteq AP$, an *A-projected model* (or projected model over A) is defined as a sequence $\sigma_A \in (2^A)^\omega$ that results in the A-projection of an accepting sequence $\sigma \in (2^{AP})^\omega$.

In the following, we define the maximum model counting problem over automata and give an algorithm for solving the problem. We show how to use the algorithm for model checking quantitative hyperproperties.

Definition 1 (Maximum Model Counting over Automata (MMCA)). *Given a Büchi automaton B over an alphabet 2^{AP} for some set of atomic propositions AP and sets $X, Y, Z \subseteq AP$ the maximum model counting problem is to compute*

$$\max_{\sigma_Y \in (2^Y)^\omega} |\{\sigma_X \in (2^X)^\omega \mid \exists \sigma_Z \in (2^Z)^\omega.\ \sigma_X \cup \sigma_Y \cup \sigma_Z \in L(B)\}|$$

where $\sigma \cup \sigma'$ is the point-wise union of σ and σ'.

As a first step in our algorithm, we show how to check whether the maximum model count is equal to infinity.

Definition 2 (Doubly Pumped Lasso). *For a graph G, a doubly pumped lasso in G is a subgraph that entails a cycles C_1 and another different cycle C_2 that is reachable from C_1.*

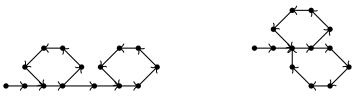

Fig. 3. Forms of doubly pumped lassos.

In general, we distinguish between two types of doubly pumped lassos as shown in Figure 3. We call the lassos with periods C_1 and C_2 the lassos of the doubly pumped lasso. A doubly pumped lasso of a Büchi automaton B is one in the graph structure of B. The doubly pumped lasso is called accepting when C_2 has an accepting state. A more generalized formalization of this idea is given in the following theorem.

Theorem 2. *Let $B = (Q, q_0, \delta, 2^{AP}, F)$ be a Büchi automaton for some set of atomic propositions $AP = X \cup Y \cup Z$ and let $\sigma' \in (2^Y)^\omega$. The automaton B has infinitely many $X \cup Y$-projected models σ with $\sigma =_Y \sigma'$ if and only if B has an accepting doubly pumped lasso with lassos ρ and ρ' such that: 1) ρ is an accepting lasso 2) $tr(\rho) =_Y tr(\rho') =_Y \sigma'$ 3) The period of ρ' shares at least one state with ρ and 4) $tr(\rho) \neq_X tr(\rho')$.*

To check whether there is a sequence $\sigma' \in (2^Y)^\omega$ such that the number of $X \cup Y$-projected models σ of B with $\sigma =_Y \sigma'$ is infinite, we search for a doubly pumped lasso satisfying the constraints given in Theorem 2. This can be done by applying the following procedure:

Given a Büchi automaton $B = (Q, q_0, 2^{AP}, \delta, F)$ and sets $X, Y, Z \subseteq AP$, we construct the following product automaton $B_\times = (Q_\times, q_{\times,0}, 2^{AP} \times 2^{AP}, \delta_\times, F_\times)$ where: $Q_\times = Q \times Q$, $q_{\times,0} = (q_0, q_0)$, $\delta_\times = \{(s_1, s_2) \xrightarrow{(\alpha,\alpha')} (s_1', s_2') \mid s_1 \xrightarrow{\alpha} s_2, s_1' \xrightarrow{\alpha'} s_2', \alpha =_Y \alpha'\}$ and $F_\times = Q \times F$. The automaton B has infinitely many models σ' if there is an accepting lasso $\rho = (q_0, q_0)(\alpha_1, \alpha_1') \dots ((q_j, q_j')(\alpha_{j+1}, \alpha_{j+1}') \dots (q_k, q_k')(\alpha_{k+1}, \alpha_{k+1}'))$ in $B_\times$ such that: $\exists h \leq j.\ q_h' = q_j$, i.e., B has lassos ρ_1 and ρ_2 that share a state in the period of ρ_1 and $\exists h > j.\ \alpha_h \neq_X \alpha_h'$, i.e., the lassos differ in the evaluation of X in a position after the shared state and thus allows infinitely many different sequence over X for the a sequence over Y. The lasso ρ simulates a doubly pumped lasso in B satisfying the constraints of Theorem 2.

Theorem 3. *Given an alternating Büchi automaton $A = (Q, q_0, \delta, 2^{AP}, F)$ for a set of atomic propositions $AP = X \cup Y \cup Z$, the problem of checking whether there is a sequence $\sigma' \in (2^Y)^\omega$ such that A has infinitely many $X \cup Y$-projected models σ with $\sigma =_Y \sigma'$ is* PSPACE*-complete.*

The lower and upper bound for the problem can be given by a reduction from and to the satisfiability problem of LTL [4]. Due to the finite structure of Büchi automata, if the number of models of the automaton exceed the exponential bound $2^{|Q|}$, where Q is the set of states, then the automaton has infinitely many models.

Lemma 1. *For any Büchi automaton B, the number of models of B is less or equal to $2^{|Q|}$ otherwise it is ∞.*

Proof. Assume the number of models is larger than $2^{|Q|}$ then there are more than $2^{|Q|}$ accepting lassos in B. By the pigeonhole principle, two of them share the same $2^{|Q|}$-prefix. Thus, either they are equal or we found doubly pumped lasso in B.

Corollary 1. *Let a Büchi automaton B over a set of atomic propositions AP and sets $X, Y \subseteq AP$. For each sequence $\sigma_Y \in (2^Y)^\omega$ the number of $X \cup Y$-projected models σ with $\sigma =_Y \sigma_Y$ is less or equal than $2^{|Q|}$ otherwise it is ∞.*

From Corollary 1, we know that if no sequence $\sigma_Y \in (2^Y)^\omega$ matches to infinitely many $X \cup Y$-projected models then the number of such models is bound by $2^{|Q|}$. Each of these models has a run in B which ends in an accepting strongly connected component. Also from Corollary 1, we know that every model has a lasso run of length $|Q|$. For each finite sequence w_Y of length $|w_Y| = |Q|$ that reaches an accepting strongly connected component, we count the number $X \cup Y$-projected words w of length $|Q|$ with $w =_Y w_Y$ and that end in an accepting strongly connected component. This number is equal to the maximum model counting number.

Algorithm 2 describes the procedure. An algorithm for the minimum model counting problem is defined in similar way. The algorithm works in a backwards

Algorithm 2 Maximum Model Counting

Input: $B = (Q, q_0, 2^{AP}, \delta, F)$, disjoint $X, Y, Z \subseteq AP$, $n \in \mathbb{N}$
Output: $\#_{X,Y,Z}(B) > n$

```
1: SCC = acceptingSCC(B)
2: i = 1
3: W = ⋃_{S∈SCC} S
4: while i ≤ |Q| do
5:     i = i + 1
6:     for q ∈ W do
7:         for (q', α, q) do
8:             W' = W' ∪ {q'}
9:             for σ ∈ Π(q) do
10:                Π'(q') = Π'(q') ∪ {α_{Y∪X} · σ}
11:    W = W'
12:    W' = ∅
13:    for q ∈ W do
14:        Π'(q) = ∅
15: return max_{X,Y,Z} Π(q_0) ≥ n
```

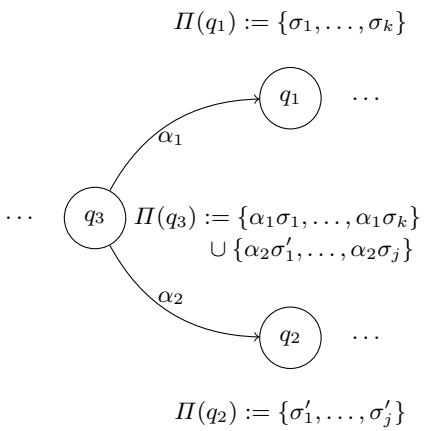

Fig. 4. Maximum Model Counting Algorithm (left) and a Sketch of a step in this algorithm (right): Current elements of our working set are $q_1, q_2 \in W$ and $q_3 \in W'$. If $i = 0$, i.e., we are in the first step of the algorithm, then q_1 and q_2 are states of accepting SCCs.

fashion starting with states of accepting strongly connected components. In each iteration i, the algorithm maps each state of the automaton with $X \cup Y$-projected words of length i that reach an accepting strongly connected component. After $|Q|$ iterations, the algorithm determines from the mapping of initial state q_0 a Y-projected word of length $|Q|$ with the maximum number of matching $X \cup Y$-projected words.

Theorem 4. *The decisional version of the maximum model counting problem over automata (MMCA), i.e. the question whether the maximum is greater than a given natural number n, is in $NP^{\#P}$.*

Proof. Let a Büchi automaton over an alphabet 2^{AP} for a set of atomic propositions AP and sets $AP_X, AP_Y, AP_Z \subseteq AP$ and a natural number n be given. We construct a nondeterministic Turing Machine M with access to a $\#P$-oracle as follows: M guesses a sequence $\sigma_Y \in 2^{AP_Y}$. It then queries the oracle, to compute a number c, such that $c = |\{\sigma_X \in (2^{AP_X})^\omega \mid \exists \sigma_Z \in (2^{AP_Z})^\omega.\ \sigma_X \cup \sigma_Y \cup \sigma_Z \in L(B)\}|$, which is a $\#P$ problem [27]. It remains to check whether $n > c$. If so, M accepts.

The following theorem summarizes the main findings of this section, which establish, depending on the property, an exponentially or even doubly exponentially better algorithm (in the quantitative bound) over the existing model checking algorithm for HyperLTL.

Theorem 5. *Given a Kripke structure K and a quantitative hyperproperty φ with bound n, the problem whether $K \models \varphi$ can be decided in logarithmic space in the quantitative bound n and in polynomial space in the size of K.*

5 A Max#Sat-based Approach

For existential HyperLTL formulas ψ_ι and ψ, we give a more practical model checking approach by encoding the automaton-based construction presented in Sections 4 into a propositional formula.

Given a Kripke structure $K = (S, s_0, \tau, AP_K, L)$ and a quantitative hyperproperty $\varphi = \forall \pi_1, \dots, \pi_k.\ \psi_\iota \rightarrow (\#\sigma : A.\ \psi) \triangleleft n$ over a set of atomic propositions $AP_\varphi \subseteq AP_K$ and bound μ, our algorithm constructs a propositional formula ϕ such that, every satisfying assignment of ϕ uniquely encodes a tuple of lassos $(\pi_1, \dots, \pi_k, \sigma)$ of length μ in K, where $(\pi_1, \dots, \pi_k)$ satisfies ψ_ι and $(\pi_1, \dots, \pi_k, \sigma)$ satisfies ψ. To compute the values $\max_{(\pi_1,\dots,\pi_k)} |\{\sigma_A \mid (\pi_1, \dots, \pi_k, \sigma) \models \psi_\iota \wedge \psi\}|$ (in case $\triangleleft \in \{\leq, <\}$) or $\min_{(\pi_1,\dots,\pi_k)} |\{\sigma_A \mid (\pi_1, \dots, \pi_k, \sigma) \models \psi_\iota \wedge \psi\}|$ (in case $\triangleleft \in \{\geq, >\}$), we pass ϕ to a maximum model counter, respectively, to a minimum model counter with the appropriate sets of counting and maximization, respectively, minimization propositions. From Lemma 1 we know that it is enough to consider lasso of length exponential in the size of φ. The size of ϕ is thus exponential in the size of φ and polynomial in the size of K.

The construction resembles the encoding of the bounded model checking approach for LTL [16]. Let $\psi_\iota = \exists \pi'_1 \dots \pi'_{k'}.\ \psi'_\iota$ and $\psi = \exists \pi''_1 \dots \pi''_{k''}.\ \psi''$ and let AP_{ψ_ι} and AP_ψ be the sets of atomic propositions that appear in ψ_ι and ψ respectively. The propositional formula ϕ is given as a conjunction of the following propositional formulas: $\phi = \bigwedge_{i \leq k} [\![K]\!]^\mu_{\pi_i} \wedge [\![K]\!]^\mu_\sigma \wedge [\![\psi_\iota]\!]^0_\mu \wedge [\![\psi]\!]^0_\mu$ where:

- μ is length of considered lassos and is equal to $\mu = 2^{|\psi'_\iota \wedge \psi''|} * |S|^{k+k'+k''+1} + 1$ which is one plus the size of the product automaton constructed from the $k + k' + k'' + 1$ self-composition and the automaton for $\psi_\iota \wedge \psi$. The "plus one" is to additionally check whether the number of models is infinite.
- $[\![K]\!]^k_\pi$ is the encoding of the transition relation of the copy of K where atomic propositions are indexed with π and up to an unrolling of length k. Each state of K can be encoded as an evaluation of a vector of $\log |S|$ unique propositional variables. The encoding is given by the propositional formula $I(\vec{v}^\pi_0) \wedge \bigwedge_{i=0}^{k-1} \tau(\vec{v}^\pi_i, \vec{v}^\pi_{i+1})$ which encodes all paths of K of length k. The formula $I(\vec{v}^\pi_0)$ defines the assignment of the initial state. The formulas $\tau(\vec{v}^\pi_i, \vec{v}^\pi_{i+1})$ define valid transitions in K from the ith to the $(i+1)$st state of a path.
- $[\![\psi_\iota]\!]^0_k$ and $[\![\psi]\!]^0_k$ are constructed using the following rules[3]:

	$i < k$	$i = k$
$[\![a_\pi]\!]^i_k$	a^i_π	$\bigvee_{j=0}^{k-1}(l_j \wedge a^j_\pi)$
$[\![\neg a_\pi]\!]^i_k$	$\neg a^i_\pi$	$\bigvee_{j=0}^{k-1}(l_j \wedge \neg a^j_\pi)$
$[\![\bigcirc \varphi_1]\!]^i_k$	$[\![\varphi_1]\!]^{i+1}_k$	$\bigvee_{j=0}^{k-1}(l_j \wedge [\![\varphi_1]\!]^j_k)$
$[\![\varphi_1 \,\mathcal{U}\, \varphi_2]\!]^i_k$	$[\![\varphi_2]\!]^i_k \vee ([\![\varphi_1]\!]^i_k \wedge [\![\varphi_1 \,\mathcal{U}\, \varphi]\!]^{i+1}_k)$	$\bigvee_{j=0}^{k-1}(l_j \wedge \langle \varphi_1 \,\mathcal{U}\, \varphi_2 \rangle^j_k)$
$\langle \varphi_1 \,\mathcal{U}\, \varphi_2 \rangle^i_k$	$[\![\varphi_2]\!]^i_k \vee ([\![\varphi_1]\!]^i_k \wedge \langle \varphi_1 \,\mathcal{U}\, \varphi \rangle^{i+1}_k)$	false
$[\![\varphi_1 \,\mathcal{R}\, \varphi_2]\!]^i_k$	$[\![\varphi_2]\!]^i_k \wedge ([\![\varphi_1]\!]^i_k \vee [\![\varphi_1 \,\mathcal{R}\, \varphi]\!]^{i+1}_k)$	$\bigvee_{j=0}^{k-1}(l_j \wedge \langle \varphi_1 \,\mathcal{R}\, \varphi_2 \rangle^j_k)$
$\langle \varphi_1 \,\mathcal{R}\, \varphi_2 \rangle^i_k$	$[\![\varphi_2]\!]^i_k \wedge ([\![\varphi_1]\!]^i_k \vee \langle \varphi_1 \,\mathcal{R}\, \varphi \rangle^{i+1}_k)$	true

[3] We omitted the rules for boolean operators for the lack of space

in case of an existential quantifier over a trace variable π, we add a copy of the encoding of K with new variables distinguished by π:

$[\![\exists \pi.\varphi_1]\!]_k^i$	$[\![K]\!]_\pi^k \wedge [\![\varphi_1]\!]_k^i$

We define sets $X = \{a_\sigma^i \mid a \in A, i \leq k\}$, $Y = \{a^i \mid a \in AP_\psi \setminus A, i \leq k\}$ and $Z = P \setminus X \cup Y$, where P is the set of all propositions in ϕ. The maximum model counting problem is then $MMC(\phi, X, Y, Z)$.

5.1 Experiments

We have implemented the Max#Sat-based model checking approach from the last section. We compare the Max#Sat-based approach to the expansion-based approach using HyperLTL [26]. Our implementation uses the MaxCount tool [29]. We use the option in MaxCount that enumerates, rather than approximates, the number of assignments for the counting variables. We furthermore instrumented the tool so that it terminates as soon as a sample is found that exceeds the given bound. If no sample is found after one hour, we report a timeout.

Table 1 shows the results on a parameterized benchmark obtained from the implementation of an 8bit passcode checker. The parameter of the benchmark is the bound on the number of bits that is leaked to an adversary, who might, for example, enter passcodes in a brute-force manner. In all instances, a violation is found. The results show that the Max#Sat-based approach scales significantly better than the expansion-based approach.

Benchmark	**Specification**	MCHyper			MCQHyper			
		#Latches	#Gates	Time(sec)	#var.	#max	#count	Time(sec).
Pwd_8bit	1bit_leak	9	55	0.3	97	16	2	1
	2bit_leak			0.4	176	32	4	1
	3bit_leak			1.3	336	64	8	2
	4bit_leak			97	656	128	16	4
	5bit_leak			TO	1296	256	32	8
	6bit_leak			TO	2576	512	64	335
	8bit_leak			TO	10256	2048	256	TO

Table 1. Comparison between the expansion-based approach (MCHyper) and the Max#Sat-based approach (MCQHyper). #max is the number of maximization variables (set Y). #count is the number of the counting variables (set X). TO indicates a time-out after 1 hour.

6 Conclusion

We have studied quantitative hyperproperties of the form $\forall \pi_1, \ldots, \pi_k.\ \varphi \rightarrow (\#\sigma : A.\ \psi \triangleleft n)$, where φ and ψ are HyperLTL formulas, and $\#\sigma : A.\varphi \triangleleft n$ compares the number of traces that differ in the atomic propositions A and satisfy ψ to a threshold n. Many quantitative information flow policies of practical interest, such as quantitative non-interference and deniability, belong to this class of properties. Our new counting-based model checking algorithm for quantitative hyperproperties performs at least exponentially better in both time and space

in the bound n than a reduction to standard HyperLTL model checking. The new counting operator makes the specifications exponentially more concise in the bound, and our model checking algorithm solves the concise specifications efficiently.

We also showed that the model checking problem for quantitative hyperproperties can be solved with a practical Max#SAT-based algorithm. The SAT-based approach outperforms the expansion-based approach significantly for this class of properties. An additional advantage of the new approach is that it can handle properties like deniability, which cannot be checked by MCHyper because of the quantifier alterntion.

References

1. Mário S. Alvim, Miguel E. Andrés, and Catuscia Palamidessi. Quantitative information flow in interactive systems. *Journal of Computer Security*, 20(1):3–50, 2012.
2. Rehan Abdul Aziz, Geoffrey Chu, Christian J. Muise, and Peter J. Stuckey. #∃sat: Projected model counting. In *Theory and Applications of Satisfiability Testing - SAT 2015 - 18th International Conference, Austin, TX, USA, September 24-27, 2015, Proceedings*, pages 121–137, 2015.
3. Michael Backes, Boris Köpf, and Andrey Rybalchenko. Automatic discovery and quantification of information leaks. In *30th IEEE Symposium on Security and Privacy (S&P 2009), 17-20 May 2009, Oakland, California, USA*, pages 141–153, 2009.
4. Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking (Representation and Mind Series)*. The MIT Press, 2008.
5. Anindya Banerjee and David A. Naumann. Stack-based access control and secure information flow. *J. Funct. Program.*, 15(2):131–177, 2005.
6. Gilles Barthe, Pedro R. D'Argenio, and Tamara Rezk. Secure information flow by self-composition. *Mathematical Structures in Computer Science*, 21(6):1207–1252, 2011.
7. Vincent Bindschaedler, Reza Shokri, and Carl A. Gunter. Plausible deniability for privacy-preserving data synthesis. *PVLDB*, 10(5):481–492, 2017.
8. Fabrizio Biondi, Axel Legay, Louis-Marie Traonouez, and Andrzej Wasowski. QUAIL: A quantitative security analyzer for imperative code. In *Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings*, pages 702–707, 2013.
9. Rohit Chadha, Umang Mathur, and Stefan Schwoon. Computing information flow using symbolic model-checking. In *34th International Conference on Foundation of Software Technology and Theoretical Computer Science, FSTTCS 2014, December 15-17, 2014, New Delhi, India*, pages 505–516, 2014.
10. Anrin Chakraborti, Chen Chen, and Radu Sion. Datalair: Efficient block storage with plausible deniability against multi-snapshot adversaries. *PoPETs*, 2017(3):179, 2017.
11. Han Chen and Pasquale Malacaria. Quantitative analysis of leakage for multi-threaded programs. In *Proceedings of the 2007 Workshop on Programming Languages and Analysis for Security, PLAS 2007, San Diego, California, USA, June 14, 2007*, pages 31–40, 2007.

12. Tom Chothia, Yusuke Kawamoto, and Chris Novakovic. Leakwatch: Estimating information leakage from java programs. In *Computer Security - ESORICS 2014 - 19th European Symposium on Research in Computer Security, Wroclaw, Poland, September 7-11, 2014. Proceedings, Part II*, pages 219–236, 2014.
13. David Clark, Sebastian Hunt, and Pasquale Malacaria. Quantified interference for a while language. *Electr. Notes Theor. Comput. Sci.*, 112:149–166, 2005.
14. David Clark, Sebastian Hunt, and Pasquale Malacaria. Quantitative information flow, relations and polymorphic types. *J. Log. Comput.*, 15(2):181–199, 2005.
15. David Clark, Sebastian Hunt, and Pasquale Malacaria. A static analysis for quantifying information flow in a simple imperative language. *Journal of Computer Security*, 15(3):321–371, 2007.
16. Edmund Clarke, Armin Biere, Richard Raimi, and Yunshan Zhu. Bounded model checking using satisfiability solving. *Form. Methods Syst. Des.*, 19(1):7–34, July 2001.
17. Edmund M. Clarke and E. Allen Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In *Logics of Programs, Workshop, Yorktown Heights, New York, May 1981*, pages 52–71, 1981.
18. Michael R. Clarkson, Bernd Finkbeiner, Masoud Koleini, Kristopher K. Micinski, Markus N. Rabe, and César Sánchez. Temporal logics for hyperproperties. In *Proceedings of POST*, volume 8414 of *LNCS*, pages 265–284. Springer, 2014.
19. Michael R. Clarkson, Andrew C. Myers, and Fred B. Schneider. Belief in information flow. In *18th IEEE Computer Security Foundations Workshop, (CSFW-18 2005), 20-22 June 2005, Aix-en-Provence, France*, pages 31–45, 2005.
20. Michael R. Clarkson, Andrew C. Myers, and Fred B. Schneider. Quantifying information flow with beliefs. *Journal of Computer Security*, 17(5):655–701, 2009.
21. Michael R. Clarkson and Fred B. Schneider. Hyperproperties. *Journal of Computer Security*, 18(6):1157–1210, 2010.
22. Ellis S Cohen. Information transmission in sequential programs. *Foundations of Secure Computation*, pages 297–335, 1978.
23. Adnan Darwiche. New advances in compiling CNF into decomposable negation normal form. In *Proceedings of the 16th Eureopean Conference on Artificial Intelligence, ECAI'2004, including Prestigious Applicants of Intelligent Systems, PAIS 2004, Valencia, Spain, August 22-27, 2004*, pages 328–332, 2004.
24. Dorothy E. Denning. *Cryptography and Data Security*. Addison-Wesley, 1982.
25. Rayna Dimitrova, Bernd Finkbeiner, Máté Kovács, Markus N. Rabe, and Helmut Seidl. Model checking information flow in reactive systems. In *Verification, Model Checking, and Abstract Interpretation - 13th International Conference, VMCAI 2012, Philadelphia, PA, USA, January 22-24, 2012. Proceedings*, pages 169–185, 2012.
26. Bernd Finkbeiner, Markus N. Rabe, and César Sánchez. Algorithms for model checking HyperLTL and HyperCTL*. In *Proceedings of CAV*, volume 9206 of *LNCS*, pages 30–48. Springer, 2015.
27. Bernd Finkbeiner and Hazem Torfah. Counting models of linear-time temporal logic. In Adrian Horia Dediu, Carlos Martín-Vide, José Luis Sierra-Rodríguez, and Bianca Truthe, editors, *Language and Automata Theory and Applications - 8th International Conference, LATA 2014, Madrid, Spain, March 10-14, 2014. Proceedings*, volume 8370 of *Lecture Notes in Computer Science*, pages 360–371. Springer, 2014.
28. Bernd Finkbeiner and Hazem Torfah. The density of linear-time properties. In Deepak D'Souza and K. Narayan Kumar, editors, *Automated Technology for Veri-*

fication and Analysis, pages 139–155, Cham, 2017. Springer International Publishing.
29. Daniel J. Fremont, Markus N. Rabe, and Sanjit A. Seshia. Maximum model counting. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA.*, pages 3885–3892, 2017.
30. James W. Gray, III. Toward a mathematical foundation for information flow security. In *Proc. IEEE Symposium on Security and Privacy*, pages 210–34, May 1991.
31. Christian Hammer and Gregor Snelting. Flow-sensitive, context-sensitive, and object-sensitive information flow control based on program dependence graphs. *Int. J. Inf. Sec.*, 8(6):399–422, 2009.
32. James W. Gray III. Toward a mathematical foundation for information flow security. In *IEEE Symposium on Security and Privacy*, pages 21–35, 1991.
33. Roberto J. Bayardo Jr. and Robert Schrag. Using CSP look-back techniques to solve real-world SAT instances. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence and Ninth Innovative Applications of Artificial Intelligence Conference, AAAI 97, IAAI 97, July 27-31, 1997, Providence, Rhode Island.*, pages 203–208, 1997.
34. Boris Köpf and David A. Basin. An information-theoretic model for adaptive side-channel attacks. In *Proceedings of the 2007 ACM Conference on Computer and Communications Security, CCS 2007, Alexandria, Virginia, USA, October 28-31, 2007*, pages 286–296, 2007.
35. Boris Köpf and Andrey Rybalchenko. Approximation and randomization for quantitative information-flow analysis. In *Proceedings of the 23rd IEEE Computer Security Foundations Symposium, CSF 2010, Edinburgh, United Kingdom, July 17-19, 2010*, pages 3–14, 2010.
36. Michael L. Littman, Stephen M. Majercik, and Toniann Pitassi. Stochastic boolean satisfiability. *J. Autom. Reasoning*, 27(3):251–296, 2001.
37. Pasquale Malacaria. Assessing security threats of looping constructs. In *Proceedings of the 34th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2007, Nice, France, January 17-19, 2007*, pages 225–235, 2007.
38. Dimiter Milushev and Dave Clarke. Incremental hyperproperty model checking via games. In *Secure IT Systems - 18th Nordic Conference, NordSec 2013, Ilulissat, Greenland, October 18-21, 2013, Proceedings*, pages 247–262, 2013.
39. Satoru Miyano and Takeshi Hayashi. Alternating finite automata on -words. *Theoretical Computer Science*, 32(3):321 – 330, 1984.
40. Daniel Morwood and Daniel Bryce. Evaluating temporal plans in incomplete domains. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, July 22-26, 2012, Toronto, Ontario, Canada.*, 2012.
41. Christian J. Muise, Sheila A. McIlraith, J. Christopher Beck, and Eric I. Hsu. Dsharp: Fast d-dnnf compilation with sharpsat. In *Advances in Artificial Intelligence - 25th Canadian Conference on Artificial Intelligence, Canadian AI 2012, Toronto, ON, Canada, May 28-30, 2012. Proceedings*, pages 356–361, 2012.
42. Andrew C. Myers. Jflow: Practical mostly-static information flow control. In *POPL '99, Proceedings of the 26th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, San Antonio, TX, USA, January 20-22, 1999*, pages 228–241, 1999.
43. G. Smith. On the foundations of quantitative information flow. In *Proc. Conference on Foundations of Software Science and Computation Structures*, pages 288–302, March 2009.

44. Hazem Torfah and Martin Zimmermann. The complexity of counting models of linear-time temporal logic. *Acta Informatica*, Nov 2016.
45. H. Yasuoka and T. Terauchi. On bounding problems of quantitative information flow. In *Proc. European Symposium on Research in Computer Security*, pages 357–372, September 2010.
46. Hirotoshi Yasuoka and Tachio Terauchi. Quantitative information flow as safety and liveness hyperproperties. *Theor. Comput. Sci.*, 538:167–182, 2014.
47. Steve Zdancewic and Andrew C. Myers. Observational determinism for concurrent program security. In *Proceedings of CSF*, page 29. IEEE Computer Society, 2003.