Temporal Mode-Checking for Runtime Monitoring of Privacy Policies

Omar Chowdhury[§], Limin Jia[§], Deepak Garg[†], and Anupam Datta[§]

Carnegie Mellon University[§], Max Planck Institute for Software Systems[†]
{omarc, liminjia, danupam}@cmu.edu[§], dg@mpi-sws.org[†]

Abstract. Fragments of first-order temporal logic are useful for representing many practical privacy and security policies. Past work has proposed two strategies for checking event trace (audit log) compliance with policies: online monitoring and offline audit. Although online monitoring is space- and time-efficient, existing techniques insist that satisfying instances of all subformulas of the policy be amenable to caching, which limits expressiveness when some subformulas have infinite support. In contrast, offline audit is brute force and can handle more policies but is not as efficient. This paper proposes a new online monitoring algorithm that caches satisfying instances when it can, and falls back to the brute force search when it cannot. Our key technical insight is a new flow- and time-sensitive static check of variable groundedness, called the *temporal mode check*, which determines subformulas for which such caching is feasible and those for which it is not and, hence, guides our algorithm. We prove the correctness of our algorithm and evaluate its performance over synthetic traces and realistic policies.

Keywords: Mode checking, runtime monitoring, metric first-order temporal logic, privacy policy.

1 Introduction

Many organizations routinely collect sensitive personal information like medical and financial records to carry out business operations and to provide services to clients. These organizations must handle sensitive information in compliance with applicable privacy legislation like the Health Insurance Portability and Accountability Act (HIPAA) [1] and the Gramm-Leach-Bliley Act (GLBA) [2]. Violations attract substantial monetary and even criminal penalties [3]. Hence, developing mechanisms and automatic tools to check privacy policy compliance in organizations is an important problem.

The overarching goal of this paper is to improve the state of the art in checking whether an event trace or audit log, which records relevant events of an organization's data handling operations, is compliant with a given privacy policy. At a high-level, this problem can be approached in two different ways. First, logs may be recorded and compliance may be checked *offline*, when demanded by an audit authority. Alternatively, an *online* program may monitor privacy-relevant events, check them against the prevailing privacy policy and report violations on the fly. Both approaches have been considered in literature: An algorithm for offline compliance checking has been proposed by a subset of the authors [4], whereas online monitoring has been the subject of extensive work by other researchers [5–11].

These two lines of work have two common features. First, they both assume that privacy policies are represented in first-order temporal logic, extended with explicit time. Such extensions have been demonstrated adequate for representing the privacy requirements of both HIPAA and GLBA [12]. Second, to ensure that only finitely many instances of quantifiers are tested during compliance checking, both lines of work use static policy checks to restrict the syntax of the logic. The specific static checks vary, but always rely on assumptions about finiteness of predicates provided by the policy designer. Some work, e.g. [5, 8–11], is based on the *safe-range check* [5], which requires syntactic subformulas to have finite support independent of each other; other work, e.g. [4, 7], is based on the *mode check* from logic programming [13–15], which is more general and can propagate variable groundedness information across subformulas.

Both lines of work have their relative advantages and disadvantages. An online monitor can cache policy-relevant information from logs on the fly (in so-called *summary structures*) and discard the remaining log immediately. This saves space. It also saves time because the summary structures are organized according to the policy formula so lookups are quicker than scans of the log in the offline method. However, online monitoring algorithms proposed so far require that all subformulas of the policy formula be amenable to caching. Furthermore, many real policies, including several privacy requirements of HIPAA and GLBA, are not amenable to such caching. In contrast, the offline algorithm proposed in our prior work [4] uses brute force search over a stored log. This is inefficient when compared to an online monitor, but it can handle all privacy requirements of HIPAA and GLBA. In this work, we combine the space- and time-efficiency of online monitoring with the generality of offline monitoring: We extend existing work in online monitoring [5] for privacy policy violations with a brute force search fallback based on offline audit for subformulas that are not amenable to caching. Like the work of Basin *et al.* [5], our work uses policies written in metric first-order temporal logic (MFOTL) [16].

Our key technical innovation is what we call the *temporal mode check*, a new static check on formulas to ensure finiteness of quantifier instantiation in our algorithm. Like a standard mode check, the temporal mode check is flow-sensitive: It can propagate variable groundedness information across subformulas. Additionally, the temporal mode check is *time-sensitive*: It conservatively approximates whether the grounding substitution for a variable comes from the future or the past. This allows us to classify all subformulas into those for which we build summary structures during online monitoring (we call such formulas buildable or `B-formulas`) and those for which we do not build summary structures and, hence, use brute force search.

As an example, consider the formula $\Box \exists x, y, z.(\mathsf{p}(x) \wedge \diamondminus \mathsf{q}(x, y) \wedge \diamondminus \mathsf{r}(x, z))$, which means that in all states, there exist x, y, z such that $\mathsf{p}(x)$ holds and in some past states $\mathsf{q}(x, y)$ and $\mathsf{r}(x, z)$ hold. Assume that p and q are finite predicates and that r is infinite, but given a ground value for its first argument, the second argument has finite computable support. One possible efficient strategy for monitoring this formula is to build summary structures for p and q and in each state where an x satisfying p exists, to quickly *lookup* the summary structure for q to find a past state and a y such that $\diamondminus \mathsf{q}(x, y)$ holds, and to *scan* the log brute force to find a past state and z such that $\diamondminus \mathsf{r}(x, z)$ holds. Note that doing so requires marking p and q as `B-formulas`, but r as not a `B-formula` (because z can be computed only after x is known, but x is known from satisfaction of p, which happens in the *future* of r). Unlike the safe-range check or the standard mode check, our new temporal mode check captures this information correctly and our monitoring algorithm, **précis**, implements this strategy. No existing work on online monitoring can handle this formula because r cannot be summarized [5–11]. The work on offline checking can handle this formula [4], it does not build summary structures and is needlessly inefficient on q.

We prove the correctness of **précis** over formulas that pass the temporal mode check and analyze its asymptotic complexity. We also empirically evaluate the performance of **précis** on synthetically generated traces, with respect to privacy policies derived from HIPAA and GLBA. The goal of our experiment is to demonstrate that incrementally maintaining summary structures for `B-formulas` of the policy can improve the performance of policy compliance checking relative to a baseline of pure brute force search. This baseline algorithm is very similar to the offline monitoring algorithm of [4], called **reduce**. In our experiments, we observe marked improvements in running time over **reduce**, e.g., up to 2.5x-6.5x speedup for HIPAA and up to 1.5x speed for GLBA, even with very conservative (unfavorable) assumptions about disk access. Even though these speedups are not universal (online monitoring optimistically constructs summary structures and if those structures are not used later then computation is wasted), they do indicate that temporal mode checking and our monitoring algorithm could have substantial practical benefit for privacy policy compliance.

Due to space restrictions, we defer the correctness proof of **précis** and several other details to a technical report [17].

2 Policy Specification Logic

Our policy specification logic, $\mathcal{GMP}$, is a fragment of MFOTL [16, 18] with restricted universal quantifiers. The syntax of $\mathcal{GMP}$ is shown below.

$$\textbf{(Policy formula)}\ \varphi ::= \mathsf{p}(\boldsymbol{t}) \mid \top \mid \bot \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \exists \boldsymbol{x}.\varphi \mid \forall \boldsymbol{x}.(\varphi_1 \rightarrow \varphi_2)$$
$$\varphi_1 \,\mathcal{S}_{\mathbb{I}} \varphi_2 \mid \diamondminus_{\mathbb{I}} \varphi \mid \boxminus_{\mathbb{I}} \varphi \mid \ominus_{\mathbb{I}} \varphi \mid \varphi_1 \,\mathcal{U}_{\mathbb{I}} \varphi_2 \mid \Diamond_{\mathbb{I}} \varphi \mid \Box_{\mathbb{I}} \varphi \mid \bigcirc_{\mathbb{I}} \varphi$$

The letter t denotes terms, which are *constants* or *variables* (x, y, etc.). Bold-faced roman letters like $\boldsymbol{t}$ denote sequences or vectors. Policy formulas are denoted by φ, α, and β. Universal quantifiers have a restricted form $\forall \boldsymbol{x}.\varphi_1 \rightarrow \varphi_2$. A *guard* [19] φ_1 is required as explained further in Section 3.

Policy formulas include both past temporal operators ($\diamondminus$, $\boxminus$, $\mathcal{S}$, $\ominus$) and future temporal operators ($\Diamond$, $\Box$, $\mathcal{U}$, $\bigcirc$). Each temporal operator has an associated time interval $\mathbb{I}$ of the form $[lo, hi]$, where $lo, hi \in \mathbb{N}$ and $lo \leq hi$. The interval selects a sub-part of the trace in which the immediate subformula is interpreted. For example, $\diamondminus_{[2,6]}\varphi$ means that at some point between 2 and 6 time units in the past, φ holds. For past temporal operators, we allow the higher limit (hi) of $\mathbb{I}$ to be ∞. We omit the interval when it is $[0,\infty]$. Policies must be *future-bounded*: both limits (lo and hi) of intervals associated with future temporal operators must be finite. $\mathcal{GMP}$ is not closed under negation due to the absence of the duals of operators $\mathcal{S}$ and $\mathcal{U}$. However, these operators do not arise in the practical privacy policies we have investigated.

Formulas are interpreted over a timed event trace (or, log) $\mathcal{L}$. Given a possibly-infinite domain of terms $\mathcal{D}$, each element of $\mathcal{L}$—the ith element is denoted $\mathcal{L}_i$—maps each ground atom $\mathsf{p}(\boldsymbol{t})$ for $\boldsymbol{t} \in \boldsymbol{\mathcal{D}}$ to either true or false. Each position $\mathcal{L}_i$ is associated with a time stamp, $\tau_i \in \mathbb{N}$, which is used to interpret intervals in formulas. We use τ to represent the sequence of time stamps, each of which is a natural number. For any arbitrary $i, j \in \mathbb{N}$ with $i > j$, $\tau_i > \tau_j$ (monotonicity). The environment η maps free variables to values in $\mathcal{D}$. Given an execution trace $\mathcal{L}$ and a time stamp-sequence τ, a position $i \in \mathbb{N}$ in the trace, an environment η, and a formula φ, we write $\mathcal{L}, \tau, i, \eta \models \varphi$ to mean that φ is satisfied in the ith position of $\mathcal{L}$ with respect to η and τ. The definition of $\models$ is standard and can be found in the technical report [17].

Example policy. The following $\mathcal{GMP}$ formula represents a privacy rule from clause §6802(a) of the U.S. privacy law GLBA [2]. It states that a financial institution can disclose to a non-affiliated third party any non-public personal information (*e.g.*, name, SSN) if such financial institution provides (within 30 days) or has provided, to the consumer, a notice of the disclosure.

$$
\begin{aligned}
&\forall p_1, p_2, q, m, t, u, d.\ (\ \mathsf{send}(p_1^-, p_2^-, m^-) \land \mathsf{contains}(m^+, q^-, t^-) \land \mathsf{info}(m^+, d^-, u^-) \rightarrow \\
&\quad \mathsf{inrole}(p_1^-, institution^+) \land \mathsf{nonAffiliate}(p_2^+, p_1^+) \land \mathsf{consumerOf}(q^-, p_1^+) \land \mathsf{attrIn}(t, npi) \\
&\qquad \land \diamondminus(\exists m_1.\mathsf{send}(p_1^-, q^-, m_1^-) \land \mathsf{noticeOfDisclosure}(m_1^+, p_1^+, p_2^+, q^+, t^+))\ ) \lor \\
&\qquad\quad \diamond_{[0,30]}\exists m_2.\mathsf{send}(p_1^-, q^-, m_2^-) \land \mathsf{noticeOfDisclosure}(m_2^+, p_1^+, p_2^+, q^+, t^+)\ )
\end{aligned}
$$

3 Temporal Mode Checking

We review mode-checking and provide an overview of our key insight, temporal mode-checking. Then, we define temporal mode-checking for $\mathcal{GMP}$ formally.

Mode-checking. Consider a predicate $\mathsf{addLessEq}(x, y, a)$, meaning $x + y \leq a$, where x, y, and a range over $\mathbb{N}$. If we are given ground values for x and a, then the number of substitutions for y for which $\mathsf{addLessEq}(x, y, a)$ holds is finite. In this case, we may say that $\mathsf{addLessEq}$'s argument position 1 and 3 are input positions (denoted by '+') and argument position 2 is an output position (denoted by '−'), denoted $\mathsf{addLessEq}(x^+, y^-, a^+)$. Such a specification of inputs and outputs is called a *mode-specification.* The meaning of a mode-specification for a predicate is that if we are given ground values for arguments in the input positions, then the number of substitutions for the variables in the output positions that result in a satisfied relation is finite. For instance, $\mathsf{addLessEq}(x^+, y^+, a^-)$ is not a valid mode-specification. Mode analysis (or mode-checking) lifts input-output specifications on predicates to input-output specification on formulas. It is commonly formalized as a judgment $\chi_{in} \vdash \varphi : \chi_{out}$, which states that given a grounding substitution for variables in χ_{in}, there is at most a finite set of substitutions for variables in χ_{out} that could together satisfy φ. For instance, consider the formula $\varphi \equiv \mathsf{p}(x) \wedge \mathsf{q}(x, y)$. Given the mode-specification $\mathsf{p}(x^-)$ and $\mathsf{q}(x^+, y^-)$ and a left-to-right evaluation order for conjunction, φ passes mode analysis with $\chi_{in} = \{\}$ and $\chi_{out} = \{x, y\}$. Mode analysis guides an algorithm to obtaining satisfying substitutions. In our example, we first obtain substitutions for x that satisfy $\mathsf{p}(x)$. Then, we plug ground values for x in $\mathsf{q}(x, y)$ to get substitutions for y. However, if the mode-specification is $\mathsf{p}(x^+)$ and $\mathsf{q}(x^+, y^-)$, then φ will fail mode analysis unless x is already ground (i.e., $x \in \chi_{in}$).

Mode analysis can be used to identify universally quantified formulas whose truth is finitely checkable. We only need to restrict universal quantifiers to the form $\forall \boldsymbol{x}.(\varphi_1 \rightarrow \varphi_2)$, and require that $\boldsymbol{x}$ be in the output of φ_1 and that φ_2 be well-moded (x may be in its input). To check that $\forall \boldsymbol{x}.(\varphi_1 \rightarrow \varphi_2)$ is true, we first find the values of $\boldsymbol{x}$ that satisfy φ_1. This is a finite set because $\boldsymbol{x}$ is in the output of φ_1. We then check that for each of these $\boldsymbol{x}$'s, φ_2 is satisfied.

Overview of temporal mode-checking. Consider the policy $\varphi_p \equiv \mathsf{p}(x^-) \wedge \diamondminus \mathsf{q}(x^+, y^-)$ and consider the following obvious but inefficient way to monitor it: We wait for $\mathsf{p}(x)$ to hold for some x, then we look back in the trace to find a position where $\mathsf{q}(x, y)$ holds for some y. This is mode-compliant (we only check q with its input x ground) but requires us to traverse the trace backward whenever $\mathsf{p}(x)$ holds for some x, which can be slow.

Ideally, we would like to incrementally build a summary structure for $\diamondminus \mathsf{q}(x, y)$ containing all the substitutions for x and y for which the formula holds as the monitor processes each new trace event. When we see $\mathsf{p}(x)$, we could quickly look through the summary structure to check whether a relation of the form $\mathsf{q}(x, y)$ for the specific x and any y exists. However, note that building such a structure may be *impossible* here. Why? The mode-specification $\mathsf{q}(x^+, y^-)$ tells us only that we will obtain a finite set of satisfying substitutions when x is already ground. However, in this example, the ground x comes from p, which holds in the *future* of q, so the summary structure may be infinite and, hence, unbuildable. In contrast, if the mode-specification of q is $\mathsf{q}(x^-, y^-)$, then we can build the summary structure because, independent of whether or not x is ground, only a finite number of substitutions can satisfy q. In this example, we would label $\diamondminus \mathsf{q}(x, y)$ *buildable* or a `B-formula` when the mode-specification is $\mathsf{q}(x^-, y^-)$ and a non-`B-formula` when the mode-specification is $\mathsf{q}(x^+, y^-)$.

With conventional mode analysis, φ_p is well-moded under both mode-specifications of q. Consequently, in order to decide whether φ_p is a `B-formula`, we need a refined analysis which takes into account the fact that, with the mode-specification $\mathsf{q}(x^+, y^-)$, information about grounding of x flows *backward* in time from p to q and, hence, $\diamondminus \mathsf{q}(x, y)$ is not a `B-formula`. This is precisely what our temporal mode-check accomplishes: It tracks whether an input substitution comes from the past/current state, or from the future. By doing so, it provides enough information to determine which subformulas are `B-formulas`.

$$\boxed{\chi_C \vdash_{\mathbf{B}} \varphi : \chi_O}$$

$$\frac{\forall k \in I(\mathsf{p}).fv(t_k) \subseteq \chi_C \qquad \chi_O = \bigcup_{j \in O(\mathsf{p})} fv(t_j)}{\chi_C \vdash_{\mathbf{B}} \mathsf{p}(t_1, \dots, t_n) : \chi_O} \text{ B-PRE}$$

$$\frac{\chi_C \vdash_{\mathbf{B}} \varphi_1 : \chi_1 \qquad \chi_C \cup \chi_1 \vdash_{\mathbf{B}} \varphi_2 : \chi_2 \qquad \chi_O = \chi_1 \cup \chi_2}{\chi_C \vdash_{\mathbf{B}} \varphi_1 \wedge \varphi_2 : \chi_O} \text{ B-AND}$$

$$\frac{\{\} \vdash_{\mathbf{B}} \varphi_2 : \chi_1 \qquad \chi_1 \vdash_{\mathbf{B}} \varphi_1 : \chi_2 \qquad \chi_O = \chi_1}{\chi_C \vdash_{\mathbf{B}} \varphi_1 \mathcal{S}_{\mathbb{I}} \varphi_2 : \chi_O} \text{ B-SINCE}$$

$$\boxed{\chi_C, \chi_F \vdash \varphi : \chi_O}$$

$$\frac{\forall k \in I(\mathsf{p}).fv(t_k) \subseteq (\chi_C \cup \chi_F) \qquad \chi_O = \bigcup_{j \in O(\mathsf{p})} fv(t_j)}{\chi_C, \chi_F \vdash \mathsf{p}(t_1, \dots, t_n) : \chi_O} \text{ PRE}$$

$$\frac{\{\} \vdash_{\mathbf{B}} \varphi_2 : \chi_1 \qquad \chi_1, \chi_C \cup \chi_F \vdash \varphi_1 : \chi_2 \qquad \chi_O = \chi_1}{\chi_C, \chi_F \vdash \varphi_1 \mathcal{S}_{\mathbb{I}} \varphi_2 : \chi_O} \text{ SINCE-1}$$

$$\frac{\chi_C \vdash_{\mathbf{B}} \varphi_2 : \chi_1 \qquad \chi_C, \chi_F \cup \chi_1 \vdash \varphi_1 : \chi_2 \qquad \chi_O = \chi_1}{\chi_C, \chi_F \vdash \varphi_1 \mathcal{U}_{\mathbb{I}} \varphi_2 : \chi_O} \text{ UNTIL-1}$$

$$\frac{\begin{array}{c} \chi_C, \chi_F \vdash \varphi_1 : \chi_1 \qquad \{\boldsymbol{x}\} \subseteq \chi_1 \\ fv(\varphi_1) \subseteq \chi_C \cup \chi_F \cup \{\boldsymbol{x}\} \qquad fv(\varphi_2) \subseteq (\chi_C \cup \chi_1 \cup \chi_F) \\ \chi_C, \chi_F \cup \chi_1 \vdash \varphi_2 : \chi_2 \end{array}}{\chi_C, \chi_F \vdash \forall \boldsymbol{x}.(\varphi_1 \rightarrow \varphi_2) : \{\}} \text{ UNIV-1}$$

Fig. 1: Selected rules of temporal mode-checking

Formally, our temporal mode-checking has two judgments: $\chi_C \vdash_{\mathbf{B}} \varphi : \chi_O$ and $\chi_C, \chi_F \vdash \varphi : \chi_O$. The first judgment assumes that substitutions for χ_C are available from the past or at the current time point; any subformula satisfying such a judgment is labeled as a `B-formula`. The second judgment assumes that substitutions for χ_C are available from the past or at current time point, but those for χ_F will be available in future. A formula satisfying such a judgment is not a `B-formula` but can be handled by brute force search. Our implementation of temporal mode analysis first tries to check a formula by the first judgment, and falls back to the second when it fails. The formal rules for mode analysis (described later) allow for both possibilities but do not prescribe a preference. At the top-level, φ is *well-moded* if $\{\}, \{\} \vdash \varphi : \chi_O$ for some χ_O.

To keep things simple, we do not build summary structures for future formulas such as $\alpha \mathcal{U}_{\mathbb{I}} \beta$, and do not allow future formulas in the judgment form $\chi_C \vdash_{\mathbf{B}} \varphi : \chi_O$ (however, we do build summary structures for nested past-subformulas of future formulas). To check $\alpha \mathcal{U}_{\mathbb{I}} \beta$, we wait until the upper limit of $\mathbb{I}$ is exceeded and then search backward. As an optimization, one may build conservative summary structures for future formulas, as in some prior work [5].

Recognizing* `B-formulas`*. We list selected rules of temporal mode-checking in Figure 1. Rule B-Pre, which applies to an atom $\mathsf{p}(t_1, \dots, t_n)$, checks that all variables in input positions of p are in χ_C. The output χ_O is the set of variables in output positions of p. ($I(\mathsf{p})$ and $O(\mathsf{p})$ are the sets of input and output positions of p, respectively.) The rule for conjunctions $\varphi_1 \wedge \varphi_2$ first checks φ_1 and then checks φ_2, propagating variables in the output of φ_1 to the input of φ_2. These two rules are standard in mode-checking. The new, interesting rule is B-Since for the formula $\varphi_1 \mathcal{S}_{\mathbb{I}} \varphi_2$. Since structures for φ_1 and φ_2 could be built at time points earlier than the current time, the premise simply ignores the input χ_C. The first premise of B-since checks φ_2 with an empty input. Based on the semantics of temporal logic, φ_1 needs to be true on the trace after φ_2, so all variables ground by φ_2 (i.e., χ_1) are available as "current" input in φ_1. As an example, $\{\} \vdash_{\mathbf{B}} \top \mathcal{S}\, \mathsf{q}(x^-, y^-) : \{x, y\}$.

Temporal mode-checking judgement. In the mode-checking judgment $\chi_C, \chi_F \vdash \varphi : \chi_O$, we separate the set of input variables for which substitutions are available at the current time point or from the past (χ_C) from the set of variables for which substitutions are available from the future (χ_F). The distinction is needed because sub-derivations of the form $\chi'_C \vdash_{\mathbf{B}} \varphi' : \chi'_O$ should be passed only the former variables as input.

Rule Pre for atoms checks that variables in input positions are in the union of χ_C and χ_F. There are four rules for $\varphi_1 \mathcal{S}_{\mathbb{I}} \varphi_2$, accounting for the buildability/non-buildability of each of the two subformulas. We show only one of these four rules, Since-1, which applies when φ_2 is a `B-formula` but φ_1 is not. In this case, φ_2 will be evaluated (for creating the summary structure) at time points earlier than $\varphi_1 \mathcal{S} \varphi_2$ and, therefore, cannot use variables in χ_C or χ_F as input (see Figure 2). When checking φ_1, variables in the output of φ_2 (called χ_1), χ_C and χ_F are all inputs, but those in χ_C or χ_F come from the future. The entire formula is not a `B-formula` as φ_1 is not.

Similarly, there are four rules for $\varphi_1 \mathcal{U}_{\mathbb{I}} \varphi_2$, of which we show only one, Until-1. This rule applies when φ_2 is a `B-formula`, but φ_1 is not. Its first premise checks that φ_2 is a `B-formula` with input χ_C. Our algorithm checks φ_1 only when φ_2 is true, so the outputs χ_1 of φ_2 are available as input for φ_1. In checking φ_1, both χ_1 and χ_F may come from the future.

The first premise of rule UNIV-1 checks that the guard φ_1 is well-moded with some output χ_1. The second premise, $\{\boldsymbol{x}\} \subseteq \chi_1$, ensures that the guard φ_1 can be satisfied only for a finite number of substitutions for $\boldsymbol{x}$, which is necessary to feasibly check φ_2. The third premise, $fv(\varphi_1) \subseteq (\chi_C \cup \chi_F \cup \{\boldsymbol{x}\})$,

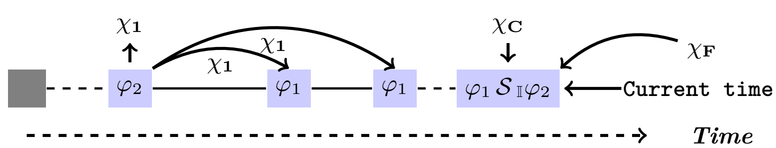

Fig. 2: Example: Temporal information in mode checking $\varphi_1 \mathcal{S}_{\mathbb{I}} \varphi_2$

ensures that no variables other than $\boldsymbol{x}$ are additionally grounded by checking φ_1. The fourth premise, $fv(\varphi_2) \subseteq (\chi_C \cup \chi_F \cup \chi_1)$, ensures that all free variables in φ_2 are already grounded by the time φ_2 needs to be checked. The final premise ensures the well-modedness of φ_2. The third and fourth premises are technical conditions, needed for the soundness of our algorithm.

4 Runtime Monitoring Algorithm

Our policy compliance algorithm **`précis`** takes as input a well-moded $\mathcal{GMP}$ policy φ, monitors the system trace as it grows, builds summary structures for nested `B-formula`s and reports a violation as soon as it is detected.

We write σ to denote a substitution, a finite map from variables to values in the domain $\mathcal{D}$. The identity substitution is denoted $\bullet$ and $\sigma_\perp$ represents an invalid substitution. For instance, the result of joining ($\bowtie$) two substitutions σ_1 and σ_2 that do not agree on the values of shared variables is $\sigma_\perp$. We say that σ' extends σ, written $\sigma' \geq \sigma$, if the domain of σ' is a superset of the domain of σ and they agree on mappings of variables that are in the domain of σ. We summarize relevant algorithmic functions below.

`précis`(φ) is the top-level function (Algorithm 1).

`checkCompliance`$(\mathcal{L}, i, \tau, \pi, \varphi)$ checks whether events in the ith position of the trace $\mathcal{L}$ satisfy φ, given the algorithm's internal state π and the time stamps τ. State π contains up-to-date summary structures for all `B-formulas` of φ.

`uSS`$(\mathcal{L}, i, \tau, \pi, \varphi)$ incrementally updates summary structures for `B-formula` φ when log position i is seen. It assumes that the input π is up-to-date w.r.t. earlier log positions and it returns the state with the updated summary structure for φ. (**`uSS`** abbreviates **`updateSummaryStructures`**).

`sat`$(\mathcal{L}, i, \tau, \mathsf{p}(\boldsymbol{t}), \sigma)$ returns the set of all substitutions σ_1 for free variables in $\mathsf{p}(\boldsymbol{t})$ that make $\mathsf{p}(\boldsymbol{t})\sigma_1$ true in the ith position of $\mathcal{L}$, given σ that grounds variables in the input positions of p. Here, $\sigma_1 \geq \sigma$.

`ips`$(\mathcal{L}, i, \tau, \pi, \sigma, \varphi)$ generalizes **`sat`** from atomic predicates to policy formulas. It takes the state π as an input to look up summary structures when `B-formulas` are encountered.

Top-level monitoring algorithm. Algorithm 1 (**`précis`**), the top-level monitoring process, uses two pointers to log entries: *curPtr* points to the last entry in the log $\mathcal{L}$, and *evalPtr* points to the position at which we next check whether φ is satisfied. Naturally, *curPtr* $\geq$ *evalPtr*. The gap between these two pointers is determined by the intervals occurring in future temporal operators in φ. For example, with the policy $\diamondsuit_{[lo,hi]}\beta$, β can be evaluated at log position i only after a position $j \geq i$ with $\tau_j - \tau_i \geq hi$ has been observed. We define a simple function $\Delta(\varphi)$ that computes a coarse but finite upper bound on the maximum time the monitor needs to wait before φ can be evaluated (see [17] for details).

The algorithm **`précis`** first initializes relevant data structures and labels `B-formulas` using mode analysis (lines 1-2). The main body of the **`précis`** is a trace-event triggered loop. In each iteration of the loop, **`précis`**: (1) updates the summary structures in π based on the newly available log entries (lines 6-7),

Algorithm 1 The **précis** algorithm

Require: A $\mathcal{GMP}$ policy φ

$\pi \leftarrow \emptyset$; $\quad curPtr \leftarrow 0$; $\quad evalPtr \leftarrow 0$; $\quad \mathcal{L} \leftarrow \emptyset$; $\quad \tau \leftarrow \emptyset$;
`Mode-check` φ`. Label all B-formulas of` φ`.`
while $(true)$ **do**
    `Wait until new events are available`
    `Extend` $\mathcal{L}$ `and` τ `with new entries`
    for all (`B-formulas` φ_s of φ in ascending formula size) **do**
        $\pi \leftarrow$ `uSS`$(\mathcal{L}, curPtr, \tau, \pi, \varphi_s)$ `//update summary structures`
    while $(evalPtr \leq curPtr)$ **do**
        if $(\tau_{curPtr} - \tau_{evalPtr} \geq \Delta(\varphi))$ **then**
            $tVal \leftarrow$ `checkCompliance`$(\mathcal{L}, evalPtr, \tau, \pi, \varphi)$
            if $tVal = false$ **then**
                `Report violation on` $\mathcal{L}$ `position` $evalPtr$
            $evalPtr \leftarrow evalPtr + 1$
        else
            break
    $curPtr \leftarrow curPtr + 1$

and (2) evaluates the policy at positions where it can be fully evaluated, i.e., where the difference between the entry's time point and the current time point $(curPtr)$ exceeds the maximum delay $\Delta(\varphi)$. Step (1) uses the function **`uSS`** and step (2) uses the function **`checkCompliance`**. **`checkCompliance`** is a wrapper for **`ips`** that calls **`ips`** with $\bullet$ as the input substitution. If **`ips`** returns an empty set of satisfying substitutions, **`checkCompliance`** returns false, signaling a violation at the current time point, else it returns true.

Finding substitutions for policy formulas. The recursive function **`ips`** returns the set of substitutions that satisfy a formula at a given log position, given a substitution for the formula's input variables. Selected clauses of the definition of **`ips`** are shown in Figure 3. When the formula is an atom, **`ips`** invokes **`sat`**, an abstract wrapper around specific implementations of predicates. When the policy is a universally quantified formula, **`ips`** is called on the guard φ_1 to find the guard's satisfying substitutions Σ_1. Then, **`ips`** is called to check that φ_2 is true for all substitutions in Σ_1. If the latter fails, **`ips`** returns the empty set of substitutions to signal a violation, else it returns $\{\sigma_{in}\}$.

When a `B-formula` $\alpha \, \mathcal{S}_{\mathbb{I}} \beta$ is encountered, all its satisfying substitutions have already been computed and stored in π. Therefore, **`ips`** simply finds these substitutions in π (expression $\pi.\mathcal{A}(\alpha \, \mathcal{S}_{\mathbb{I}} \beta)(i).\mathbb{R}$), and discards those that are inconsistent with σ_{in} by performing a join ($\bowtie$). For the non-`B-formula` $\alpha \, \mathcal{S}_{\mathbb{I}} \beta$, **`ips`** calls itself recursively on the sub-formulas α and β, and computes the substitutions brute force.

Incrementally updating summary structures. We explain how we update summary structures for formulas of the form $\varphi_1 \, \mathcal{S}_{\mathbb{I}} \varphi_2$ here. Updates for $\ominus_{\mathbb{I}} \varphi$, $\boxminus_{\mathbb{I}} \varphi$, and $\diamondminus_{\mathbb{I}} \varphi$ are similar and can be found in the technical report [17].

$$\mathtt{ips}(\mathcal{L}, i, \tau, \pi, \sigma_{\text{in}}, \mathsf{p}(\boldsymbol{t})) \quad = \quad \mathtt{sat}(\mathcal{L}, i, \tau, \mathsf{p}(\boldsymbol{t}), \sigma_{\text{in}})$$

$$\begin{array}{l} \mathtt{ips}(\mathcal{L}, i, \tau, \pi, \sigma_{\text{in}}, \\ \forall \boldsymbol{x}.(\varphi_1 \rightarrow \varphi_2)) \end{array} \quad = \quad \begin{array}{l} \text{let} \quad \varSigma_1 \leftarrow \mathtt{ips}(\mathcal{L}, i, \tau, \pi, \sigma_{\text{in}}, \varphi_1) \\ \text{return} \begin{cases} \emptyset & \text{if } \exists \sigma_c \in \varSigma_1.(\mathtt{ips}(\mathcal{L}, i, \tau, \pi, \sigma_c, \varphi_2) = \emptyset) \\ \{\sigma_{\text{in}}\} & \text{otherwise} \end{cases} \end{array}$$

$$\mathtt{ips}(\mathcal{L}, i, \tau, \pi, \sigma_{\text{in}}, \alpha \, \mathcal{S}_{\mathbb{I}} \beta) \;=\; \begin{cases} \textbf{If } \alpha \, \mathcal{S}_{\mathbb{I}} \beta \text{ is a } \texttt{B-formula} \textbf{ then} \\ \quad \textbf{return } \sigma_{\text{in}} \bowtie \pi.\mathcal{A}(\alpha \, \mathcal{S}_{\mathbb{I}} \beta)(i).\mathbb{R} \\ \textbf{Else} \\ \text{let} \quad S_\beta \leftarrow \{\langle \sigma, k \rangle | k = \max l.((0 \leq l \leq i) \wedge ((\tau_i - \tau_l) \in \mathbb{I}) \\ \qquad\qquad \wedge \sigma \in \mathtt{ips}(\mathcal{L}, l, \tau, \pi, \sigma_{\text{in}}, \beta))\} \\ \quad S_{R_1} \leftarrow \{\sigma | \langle \sigma, i \rangle \in S_\beta \wedge 0 \in \mathbb{I}\} \\ \quad S_{R_2} \leftarrow \{\bowtie \sigma_l^\alpha \neq \sigma_\perp | \exists \langle \sigma_\beta, k \rangle \in S_\beta . k < i \wedge \\ \qquad\qquad \forall l.(k < l \leq i \rightarrow \sigma_l^\alpha \in \mathtt{ips}(\mathcal{L}, l, \tau, \pi, \sigma_\beta, \varphi_1))\} \\ \quad \textbf{return } S_{R_1} \cup S_{R_2} \end{cases}$$

Fig. 3: Definition of the $\mathtt{ips}$ function, selected clauses

For each B-formula of the form $\alpha \, \mathcal{S}_{[lo,hi]} \beta$, we build three structures: $\mathbb{S}_\beta$, $\mathbb{S}_\alpha$, and $\mathbb{R}$. The structure $\mathbb{S}_\beta$ contains a set of pairs of form $\langle \sigma, k \rangle$ in which σ represents a substitution and $k \in \mathbb{N}$ is a position in $\mathcal{L}$. Each pair of form $\langle \sigma, k \rangle \in \mathbb{S}_\beta$ represents that for all $\sigma' \geq \sigma$, the formula $\beta\sigma'$ is true at position k of $\mathcal{L}$. The structure $\mathbb{S}_\alpha$ contains a set of pairs of form $\langle \sigma, k \rangle$, each of which represents that for all $\sigma' \geq \sigma$ the formula $\alpha\sigma'$ has been true from position k until the current position in $\mathcal{L}$. The structure $\mathbb{R}$ contains a set of substitutions, which make $(\alpha \, \mathcal{S}_{[lo,hi]} \beta)$ true in the current position of $\mathcal{L}$. We use $\mathbb{R}^i$ (similarly for other structures too) to represent the structure $\mathbb{R}$ at position i of $\mathcal{L}$. We also assume $\mathbb{S}_\beta^{(-1)}$, $\mathbb{S}_\alpha^{(-1)}$, and $\mathbb{R}^{(-1)}$ to be empty (the same applies for other structures too). We show here how the structures $\mathbb{S}_\beta$ and $\mathbb{R}$ are updated. We defer the description of update of $\mathbb{S}_\alpha$ to the technical report [17].

To update the structure $\mathbb{S}_\beta$, we first calculate the set $\varSigma_\beta$ of substitutions that make β true at i by calling $\mathtt{ips}$. Pairing all these substitutions with the position i yields S^β_{new}. Next, we compute the set S^β_{remove} of all old $\langle \sigma, k \rangle$ pairs that do not satisfy the interval constraint $[lo, hi]$ (i.e., for which $\tau_i - \tau_k > hi$). The updated structure $\mathbb{S}^i_\beta$ is then obtained by taking a union of S^β_{new} and the old structure $\mathbb{S}_\beta^{(i-1)}$, and removing all the pairs in the set S^β_{remove}.

$$\begin{array}{ll|ll} \varSigma_\beta & \leftarrow \mathtt{ips}(\mathcal{L}, i, \tau, \pi, \bullet, \beta) & S^\beta_{\text{remove}} & \leftarrow \{\langle \sigma, k \rangle \mid \langle \sigma, k \rangle \in \mathbb{S}_\beta^{(i-1)} \wedge (\tau_i - \tau_k) > hi\} \\ S^\beta_{\text{new}} & \leftarrow \{\langle \sigma, i \rangle \mid \sigma \in \varSigma_\beta\} & \mathbb{S}^i_\beta & \leftarrow (\mathbb{S}_\beta^{(i-1)} \cup S^\beta_{\text{new}}) \setminus S^\beta_{\text{remove}} \end{array}$$

To compute the summary structure $\mathbb{R}$ for $\alpha \, \mathcal{S}_{\mathbb{I}} \beta$ at i, we first compute the set S_{R_1} of all substitutions for which the formula β is true in the ith position and the interval constraint is respected by the position i. Then we compute S_{R_2} as the join $\sigma \bowtie \sigma_1$ of substitutions σ for which β was satisfied at some prior position k, and substitutions σ_1 for which α is true from position $k+1$ to i. The updated structure $\mathbb{R}^i$ is the union of S_{R_1} and S_{R_2}.

$$S_{R_1} \leftarrow \{\sigma \mid \langle\sigma, i\rangle \in \mathbb{S}^i_\beta \wedge 0 \in [lo, hi]\}$$
$$S_{R_2} \leftarrow \{\sigma \bowtie \sigma_1 \mid \exists k, j.\langle\sigma, k\rangle \in \mathbb{S}^i_\beta \wedge (k \neq i) \wedge (\tau_i - \tau_k \in [lo, hi]) \wedge \langle\sigma_1, j\rangle \in \mathbb{S}^i_\alpha \wedge (j \leq (k+1)) \wedge \sigma \bowtie \sigma_1 \neq \sigma_\perp\}$$
$$\mathbb{R}^i \leftarrow S_{R_1} \cup S_{R_2}$$

Optimizations. When all temporal sub-formulas of φ are `B-formulas`, *curPtr* and *evalPtr* proceed in synchronization and only the summary structure for position *curPtr* needs to be maintained. When φ contains future temporal formulas but all past temporal sub-formulas of φ are `B-formulas`, then we need to maintain only the summary structures for positions in [*evalPtr*, *curPtr*], but the rest of the log can be discarded immediately. When φ contains at least one past temporal subformula that is not a `B-formula` we need to store the slice of the trace that contains all predicates in that non-`B-formula`.

The following theorem states that on well-moded policies, **`précis`** terminates and is correct. The theorem requires that the internal state π be *strongly consistent* at *curPtr* with respect to the log $\mathcal{L}$, time stamp sequence τ, and policy φ. Strong consistency means that the state π contains sound and complete substitutions for all `B-formulas` of φ for all trace positions in $[0, curPtr]$ (see [17]).

Theorem 1 (Correctness of `précis`). *For all $\mathcal{GMP}$ policies φ, for all evalPtr, curPtr $\in \mathbb{N}$, for all traces $\mathcal{L}$, for all time stamp sequences τ, for all internal states π, for all empty environments η_0 such that (1) π is strongly consistent at curPtr with respect to $\mathcal{L}$, τ, and φ, (2) $curPtr \geq evalPtr$ and $\tau_{curPtr} - \tau_{evalPtr} \geq \Delta(\varphi)$, and (3) $\{\}, \{\} \vdash \varphi : \chi_O$ where $\chi_O \subseteq fv(\varphi)$, it is the case that* ***`checkCompliance`***$(\mathcal{L}, evalPtr, \tau, \pi, \varphi)$ *terminates and if* ***`checkCompliance`***$(\mathcal{L}$, *evalPtr*, τ, $\pi, \varphi) = tVal$, *then* $(tVal = true) \leftrightarrow \exists\sigma.(\mathcal{L}, \tau, evalPtr, \eta_0 \models \varphi\sigma)$.

Proof. By induction on the policy formula φ (see [17]).

***Complexity of* `précis`.** The runtime complexity of one iteration of **`précis`** for a given policy φ is $|\varphi| \times$ (complexity of the **`uSS`** function) + (complexity of **`ips`** function), where $|\varphi|$ is the policy size. We first analyze the runtime complexity of **`ips`**. Suppose the maximum number of substitutions returned by a single call to **`sat`** (for any position in the trace) is $\mathbb{F}$ and the maximum time required by **`sat`** to produce one substitution is $\mathbb{A}$. The worst case runtime of **`ips`** occurs when all subformulas of φ are non-`B-formulas` of the form $\varphi_1 \mathcal{S} \varphi_2$ and in that case the complexity is $\mathcal{O}((\mathbb{A} \times \mathbb{F} \times \mathbb{L})^{\mathcal{O}(|\varphi|)})$ where $\mathbb{L}$ denotes the length of the trace. **`uSS`** is invoked only for `B-formulas`. From the definition of mode-checking, all sub-formulas of a `B-formula` are also `B-formulas`. This property of `B-formulas` ensures that when **`uSS`** calls **`ips`**, the worst case behavior of **`ips`** is not encountered. The overall complexity of **`uSS`** is $\mathcal{O}(|\varphi| \times (\mathbb{A} \times \mathbb{F})^{\mathcal{O}(|\varphi|)})$. Thus, the runtime complexity of each iteration of the **`précis`** function is $\mathcal{O}((\mathbb{A} \times \mathbb{F} \times \mathbb{L})^{\mathcal{O}(|\varphi|)})$.

5 Implementation and Evaluation

This section reports an experimental evaluation of the **`précis`** algorithm. All measurements were made on a 2.67GHz Intel Xeon CPU X5650 running Debian

GNU/Linux 7 (Linux kernel 3.2.48.1.amd64-smp) on 48GB RAM, of which at most 2.2GB is used in our experiments. We store traces in a SQLite database. Each n-ary predicate is represented by a $n+1$ column table whose first n columns store arguments that make the predicate true on the trace and the last column stores the trace position where the predicate is true. We index each table by the columns corresponding to input positions of the predicate. We experiment with randomly generated synthetic traces. Given a $\mathcal{GMP}$ policy and a target trace length, at each trace point, our synthetic trace generator randomly decides whether to generate a policy-compliant action or a policy violating action. For a compliant action, it recursively traverses the syntax of the policy and creates trace actions to satisfy the policy. Disjunctive choices are resolved randomly. Non-compliant actions are handled dually. The source code and traces used in the experiments are available from the authors' homepages.

Our goal is to demonstrate that incrementally maintaining summary structures for `B-formulas` can improve the performance of policy compliance checking. Our baseline for comparison is a variant of **`précis`** that does not use any summary structures and, hence, checks temporal operators by brute force scanning. This baseline algorithm is very similar to the **`reduce`** algorithm of prior work [4] and, indeed, in the sequel we refer to our baseline as **`reduce`**. For the experimental results reported here, we deliberately hold traces in an in-memory SQLite database. This choice is conservative; using a disk-backed database improves **`précis`**' performance relative to **`reduce`** because **`reduce`** accesses the database more intensively (our technical report contains comparative evaluation using a disk-backed database and confirms this claim [17]). Another goal of our experiment is to identify how **`précis`** scales when larger summary structures must be maintained. Accordingly, we vary the upper bound hi in intervals $[lo, hi]$ in past temporal operators.

We experiment with two privacy policies that contain selected clauses of HIPAA and GLBA, respectively. As **`précis`** and **`reduce`** check compliance of non-`B-formulas` similarly, to demonstrate the utility of building summary structures, we ensure that the policies contain `B-formulas` (in our HIPAA policy, 7 out of 8 past temporal formulas are `B-formulas`; for GLBA the number is 4 out of 9). Our technical report [17] lists the policies we used. Figure 4 show our evaluation times for the HIPAA privacy policy for the following upper bounds on the past temporal operators: 100, 1000, 3000, and ∞. Points along the x-axis are the size of the trace and also the number of privacy-critical events checked. The y-axis represents the *average* monitoring time per event. We plot four curves for each bound: (1) The time taken by **`précis`**, (2) The time taken by **`reduce`**, (3) The time spent by **`précis`** in building and accessing summary structures for `B-formulas`, and (4) The time spent by **`reduce`** in evaluating `B-formulas`. For all trace positions $i \in \mathbb{N}$, $\tau_{i+1} - \tau_i = 1$.

The difference between (1) and (3), and (2) and (4) is similar at all trace lengths because it is the time spent on non-buildable parts of the policy, which is similar in **`précis`** and **`reduce`**. For the policy considered here, **`reduce`** spends most time on `B-formulas`, so construction of summary structures improves per-

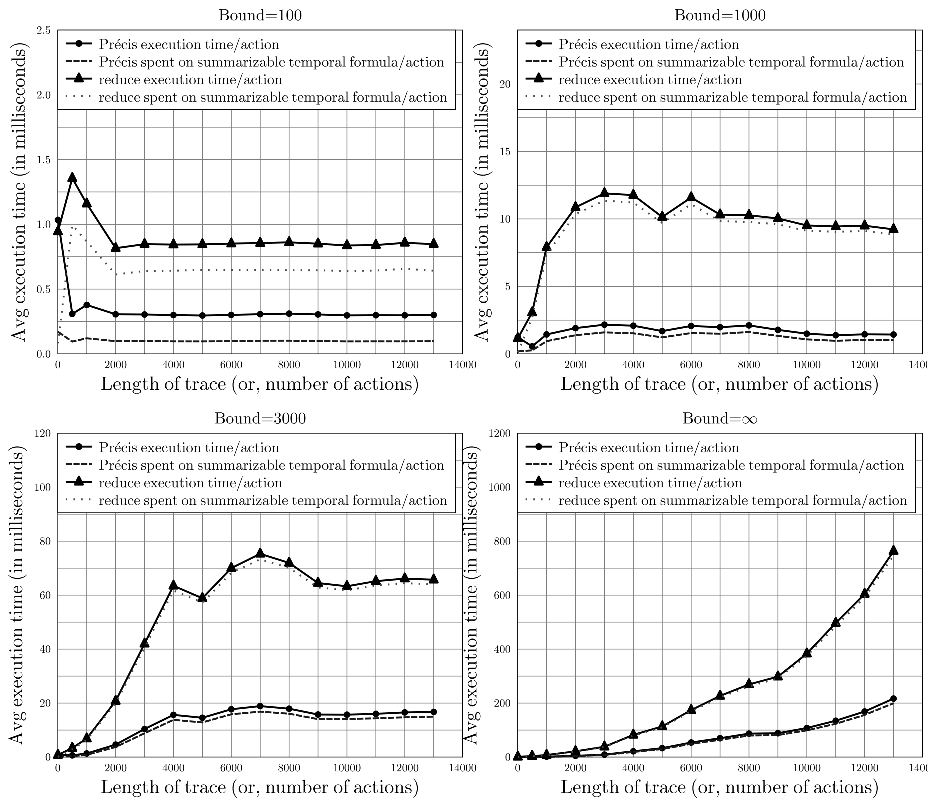

Fig. 4: **Experimental results (HIPAA)**

formance. For trace lengths greater than the bound, the curves flatten out, as expected. As the bound increases, the average execution time for **`reduce`** increases as the algorithm has to look back further on the trace, and so does the relative advantage of **`précis`**. Overall, **`précis`** achieves a speedup up of 2.5x-6.5x over **`reduce`** after the curves flatten out in the HIPAA policy. The results for GLBA, not shown here but discussed in our technical report [17] are similar, with speedups of 1.25x to 1.5x. The technical report also describes the amount of memory needed to store summary structures in **`précis`**. Briefly, this number grows proportional to the minimum of trace length and policy bound. The maximum we observe (for trace length 13000 and bound ∞) is 2.2 GB, which is very reasonable. This can be further improved by compression.

Algorithms	Incomplete states allowed?	Mode of operation	Summary structures (past formulas)	Summary structures (future formulas)
`précis`	`no`	`online`	`yes`	`no`
`reduce` [4]	`yes`	`offline`	`no`	`no`
Chomicki [8, 9] Krukow *et al.* [10]	`no`	`online`	`yes`	`no`
Bauer *et al.* [11]	`yes`	`online`	`yes`	`no`
Basin *et al.* [5, 7]	`no`	`online`	`yes`	`yes`
Basin *et al.* [6]	`yes`	`online`	`yes`	`yes`
Bauer *et al.* [20]	`no`	`online`	`(automata)`*	`(automata)`*

Table 1: Comparison of design choices in **`précis`** and prior work using first-order temporal logic for privacy compliance. *Automata-based approaches have no explicit notion of summary structures.

6 Related Work

Runtime monitoring of *propositional* linear temporal logic (pLTL) formulas [21], regular expressions, finite automata, and other equivalent variants has been studied in literature extensively [22–48]. However, pLTL and its variants are not sufficient to capture the privacy requirements of legislation like HIPAA and GLBA. To address this limitation, many logics and languages have been proposed for specifying privacy policies. Some examples are P3P [49, 50], EPAL [51, 52], Privacy APIs [53], LPU [54, 55], past-only fragment of first-order temporal logic (FOTL) [10, 11], predLTL [56], pLogic [57], PrivacyLFP [12], MFOTL [5–7], the guarded fragment of first-order logic with explicit time [4], and P-RBAC [58]. Our policy language, $\mathcal{GMP}$, is more expressive than many existing policy languages such as LPU [54, 55], P3P [49, 50], EPAL [51, 52], and P-RBAC [58].

In Table 1, we summarize design choices in **`précis`** and other existing work on privacy policy compliance checking using first-order temporal logics. The column "Incomplete states allowed?" indicates whether the work can handle some form of incompleteness in observation about states. Our own prior work [4] presents the algorithm **`reduce`** that checks compliance of a mode-checked fragment of FOL policies with respect to potentially incomplete logs. This paper makes the mode check time-aware and adds summary structures to **`reduce`**, but we assume that our event traces have complete information in all observed states.

Bauer *et al.* [11] present a compliance-checking algorithm for the (non-metric) past fragment of FOTL. $\mathcal{GMP}$ can handle both past and future (metric) temporal operators. However, Bauer *et al.* allow counting operators, arbitrary computable functions, and partial observability of events, which we do not allow. They allow a somewhat simplified guarded universal quantification where the guard is a single predicate. In $\mathcal{GMP}$, we allow the guard of the universal quantification to be a complex $\mathcal{GMP}$ formula. For instance, the following formula cannot be expressed in the language proposed by Bauer *et al.* but $\mathcal{GMP}$ mode checks it: $\forall x, y.\, (\mathsf{q}(x^+, y^+)\, \mathcal{S}\, \mathsf{p}(x^-, y^-)) \to \mathsf{r}(x^+, y^+)$. Moreover, Bauer *et al.* only consider closed formulas and also assume that each predicate argument position is output. We do not insist on these restrictions. In further development, Bauer *et al.* [20], propose an automata-based, incomplete monitoring algorithm for a fragment of FOTL called LTL^{FO}. They consider non-safety policies (unbounded future operators), which we do not consider.

Basin *et al.* [5] present a runtime monitoring algorithm for a fragment of MFOTL. Our summary structures are directly inspired by this work and the work of Chomicki [8, 9]. We improve expressiveness through the possibility of brute force search similar to [4], when subformulas are not amenable to summarization. Basin *et al.* build summary structures for future operators, which we do not (such structures can be added to our monitoring algorithm). In subsequent work, Basin *et al.* [6] extend their runtime monitoring algorithm to handle incomplete logs and inconsistent logs using a three-valued logic, which we do not consider. In more recent work, Basin *et al.* [7] extend the monitoring algorithm to handle aggregation operators and function symbols, which $\mathcal{GMP}$ does not include. These extensions are orthogonal to our work.

Our temporal mode check directly extends mode checking from [4] by adding time-sensitivity, although the setting is different— [4] is based on first-order logic with an explicit theory of linear time whereas we work with MFOTL. The added time-sensitivity allows us to classify subformulas into those that can be summarized and those that must be brute forced. Some prior work, e.g. [5–11], is based on the safe-range check instead of the mode check. The safe-range check is less expressive than a mode check. For example, the safe-range check does not accept the formula $\mathsf{q}(x^+, y^+, z^-)\,\mathcal{S}\,\mathsf{p}(x^-, y^-)$, but our temporal mode check does (however, the safe-range check will accept the formula $\mathsf{q}(x^-, y^-, z^-)\,\mathcal{S}\,\mathsf{p}(x^-, y^-)$). More recent work [7] uses a static check intermediate in expressiveness between the safe-range check and a full-blown mode check.

7 Conclusion

We have presented a privacy policy compliance-checking algorithm for a fragment of MFOTL. The fragment is characterized by a novel temporal mode-check, which, like a conventional mode-check, ensures that only finitely many instantiations of quantifiers are tested but is, additionally, time-aware and can determine which subformulas of the policy are amenable to construction of summary structures. Using information from the temporal mode-check, our algorithm **`précis`** performs best-effort runtime monitoring, falling back to brute force search when summary structures cannot be constructed. Empirical evaluation shows that summary structures improve performance significantly, compared to a baseline without them.

Acknowledgement

This work was partially supported by the AFOSR MURI on "Science of Cybersecurity", the National Science Foundation (NSF) grants CNS 1064688, CNS 0964710, and CCF 0424422, and the HHS/ONC grant HHS90TR0003/01. The authors thank anonymous reviewers, Sagar Chaki, Andreas Gampe, and Murillo Pontual for their helpful comments and suggestions.

References

1. Health Resources and Services Administration: Health insurance portability and accountability act (1996) Public Law 104-191.
2. Senate Banking Committee: Gramm-Leach-Bliley Act (1999) Public Law 106-102.
3. Roberts, P.: HIPAA Bares Its Teeth: $4.3m Fine For Privacy Violation Available at https://threatpost.com/en_us/blogs/hipaa-bares-its-teeth-43m-fine-privacy-violation-022311.
4. Garg, D., Jia, L., Datta, A.: Policy auditing over incomplete logs: Theory, implementation and applications. In: Proceedings of the 18th ACM Conference on Computer and Communications Security. CCS '11, New York, NY, USA, ACM (2011) 151–162
5. Basin, D., Klaedtke, F., Müller, S.: Monitoring security policies with metric first-order temporal logic. In: Proceedings of the 15th ACM Symposium on Access Control Models and Technologies. SACMAT '10, New York, NY, USA, ACM (2010) 23–34
6. Basin, D., Klaedtke, F., Marinovic, S., Zălinescu, E.: Monitoring compliance policies over incomplete and disagreeing logs. In Qadeer, S., Tasiran, S., eds.: Runtime Verification. Volume 7687 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2013) 151–167
7. Basin, D., Klaedtke, F., Marinovic, S., Zălinescu, E.: Monitoring of temporal first-order properties with aggregations. In Legay, A., Bensalem, S., eds.: Runtime Verification. Volume 8174 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2013) 40–58
8. Chomicki, J.: Efficient checking of temporal integrity constraints using bounded history encoding. ACM Trans. Database Syst. **20**(2) (June 1995) 149–186
9. Chomicki, J., Niwiński, D.: On the feasibility of checking temporal integrity constraints. In: Proceedings of the Twelfth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems. PODS '93, New York, NY, USA, ACM (1993) 202–213
10. Krukow, K., Nielsen, M., Sassone, V.: A logical framework for history-based access control and reputation systems. J. Comput. Secur. **16**(1) (January 2008) 63–101
11. Bauer, A., Gore, R., Tiu, A.: A first-order policy language for history-based transaction monitoring. In Leucker, M., Morgan, C., eds.: Proceedings of the 6th International Colloquium on Theoretical Aspects of Computing (ICTAC). Volume 5684 of Lecture Notes in Computer Science., Berlin, Heidelberg, Springer-Verlag (August 2009) 96–111
12. DeYoung, H., Garg, D., Jia, L., Kaynar, D., Datta, A.: Experiences in the logical specification of the hipaa and glba privacy laws. In: Proceedings of the 9th Annual ACM Workshop on Privacy in the Electronic Society. WPES '10, New York, NY, USA, ACM (2010) 73–82
13. Apt, K., Marchiori, E.: Reasoning about prolog programs: From modes through types to assertions. Formal Aspects of Computing **6**(1) (1994) 743–765
14. Dembinski, P., Maluszynski, J.: And-parallelism with intelligent backtracking for annotated logic programs. In: Proceedings of the 1985 Symposium on Logic Programming, Boston, Massachusetts, USA, July 15-18, 1985, IEEE-CS (1985) 29–38
15. Mellish, C.S.: The automatic generation of mode declarations for Prolog programs. Department of Artificial Intelligence, University of Edinburgh (1981)
16. Koymans, R.: Specifying real-time properties with metric temporal logic. Real-Time Systems **2**(4) (1990) 255–299

17. Chowdhury, O., Jia, L., Garg, D., Datta, A.: Temporal mode-checking for runtime monitoring of privacy policies. Technical Report CMU-CyLab-14-005, Cylab, Carnegie Mellon University, Pittsburgh, Pennsylvania (May 2014)
18. Alur, R., Henzinger, T.: Logics and models of real time: A survey. In Bakker, J., Huizing, C., Roever, W., Rozenberg, G., eds.: Real-Time: Theory in Practice. Volume 600 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (1992) 74–106
19. Andréka, H., Németi, I., van Benthem, J.: Modal languages and bounded fragments of predicate logic. Journal of Philosophical Logic **27**(3) (1998) 217–274
20. Bauer, A., Küster, J.C., Vegliach, G.: From propositional to first-order monitoring. In Legay, A., Bensalem, S., eds.: Runtime Verification. Volume 8174 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2013) 59–75
21. Pnueli, A.: The temporal logic of programs. In: Proceedings of the 18th Annual Symposium on Foundations of Computer Science. SFCS '77, Washington, DC, USA, IEEE Computer Society (1977) 46–57
22. Roşu, G.: On Safety Properties and Their Monitoring. Technical Report UIUCDCS-R-2007-2850, Department of Computer Science, University of Illinois at Urbana-Champaign (2007)
23. Büchi, J.R.: On a Decision Method in Restricted Second-Order Arithmetic. In: International Congress on Logic, Methodology, and Philosophy of Science, Stanford University Press (1962) 1–11
24. Hussein, S., Meredith, P.O., Roşu, G.: Security-policy monitoring and enforcement with JavaMOP. In: ACM SIGPLAN Seventh Workshop on Programming Languages and Analysis for Security (PLAS'12). (2012) 3:1–3:11
25. Meredith, P., Roşu, G.: Runtime verification with the RV system. In: First International Conference on Runtime Verification (RV'10). Volume 6418 of Lecture Notes in Computer Science., Springer (2010) 136–152
26. Meredith, P., Roşu, G.: Efficient parametric runtime verification with deterministic string rewriting. In: Proceedings of 28th IEEE/ACM International Conference. Automated Software Engineering (ASE'13), IEEE/ACM (May 2013) NA
27. Pellizzoni, R., Meredith, P., Caccamo, M., Roşu, G.: Hardware runtime monitoring for dependable cots-based real-time embedded systems. In: Proceedings of the 29th IEEE Real-Time System Symposium (RTSS'08). (2008) 481–491
28. Meredith, P., Jin, D., Chen, F., Roşu, G.: Efficient monitoring of parametric context-free patterns. In: Proceedings of the 23rd IEEE/ACM International Conference on Automated Software Engineering(ASE '08), IEEE/ACM (2008) 148–157
29. Roşu, G., Havelund, K.: Synthesizing dynamic programming algorithms from linear temporal logic formulae. Technical report, Research Institute for Advanced Computer Science (2001)
30. Roşu, G., Havelund, K.: Rewriting-based techniques for runtime verification. Automated Software Engineering **12**(2) (2005) 151–197
31. Havelund, K., Roşu, G.: Efficient monitoring of safety properties. Int. J. Softw. Tools Technol. Transf. **6**(2) (August 2004) 158–173
32. Roşu, G., Chen, F., Ball, T.: Synthesizing monitors for safety properties – this time with calls and returns –. In: Workshop on Runtime Verification (RV'08). Volume 5289 of Lecture Notes in Computer Science., Springer (2008) 51–68
33. Leucker, M., Schallhart, C.: A brief account of runtime verification. The Journal of Logic and Algebraic Programming **78**(5) (2009) 293 – 303 The 1st Workshop on Formal Languages and Analysis of Contract-Oriented Software (FLACOS'07).

34. Roşu, G., Bensalem, S.: Allen Linear (Interval) Temporal Logic –Translation to LTL and Monitor Synthesis. In: Proceedings of 18th International Conference on Computer Aided Verification (CAV'06). Volume 4144 of Lecture Notes in Computer Science., Springer (2006) 263–277
35. d'Amorim, M., Roşu, G.: Efficient monitoring of ω-languages. In: Proceedings of 17th International Conference on Computer-aided Verification (CAV'05). Volume 3576 of Lecture Notes in Computer Science., Springer (2005) 364 – 378
36. Basin, D., Jugé, V., Klaedtke, F., Zălinescu, E.: Enforceable security policies revisited. ACM Trans. Inf. Syst. Secur. **16**(1) (June 2013) 3:1–3:26
37. Bauer, L., Ligatti, J., Walker, D.: More enforceable security policies. In Cervesato, I., ed.: Foundations of Computer Security: proceedings of the FLoC'02 workshop on Foundations of Computer Security, Copenhagen, Denmark, DIKU Technical Report (25–26 July 2002) 95–104
38. Schneider, F.B.: Enforceable security policies. ACM Trans. Inf. Syst. Secur. **3**(1) (February 2000) 30–50
39. Giannakopoulou, D., Havelund, K.: Automata-based verification of temporal properties on running programs. In: Automated Software Engineering, 2001. (ASE 2001). Proceedings. 16th Annual International Conference on. (Nov 2001) 412–416
40. Martinell, F., Matteucci, I.: Through modeling to synthesis of security automata. Electron. Notes Theor. Comput. Sci. **179** (2007) 31–46
41. Huisman, M., Tamalet, A.: A formal connection between security automata and jml annotations. In: FASE '09: Proceedings of the 12th International Conference on Fundamental Approaches to Software Engineering, Berlin, Heidelberg, Springer-Verlag (2009) 340–354
42. Ligatti, J., Bauer, L., Walker, D.: Run-time enforcement of nonsafety policies. ACM Trans. Inf. Syst. Secur. **12**(3) (January 2009) 19:1–19:41
43. Ligatti, J., Bauer, L., Walker, D.: Edit automata: enforcement mechanisms for run-time security policies. Int. J. Inf. Sec. '05
44. Bauer, A., Leucker, M., Schallhart, C.: Monitoring of real-time properties. In Arun-Kumar, S., Garg, N., eds.: Proceedings of the 26th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS). Volume 4337 of Lecture Notes in Computer Science., Berlin, Heidelberg, Springer-Verlag (December 2006)
45. Bauer, A., Leucker, M., Schallhart, C.: The good, the bad, and the ugly, but how ugly is ugly? In Sokolsky, O., Tasiran, S., eds.: Proceedings of the 7th International Workshop on Runtime Verification (RV). Volume 4839 of Lecture Notes in Computer Science., Berlin, Heidelberg, Springer-Verlag (November 2007) 126–138
46. Bauer, A., Leucker, M., Schallhart, C.: Comparing LTL semantics for runtime verification. Logic and Computation **20**(3) (2010) 651–674
47. Bauer, A., Leucker, M., Schallhart, C.: Runtime verification for LTL and TLTL. ACM Trans. Softw. Eng. Methodol. **20**(4) (September 2011) 14:1–14:64
48. Bauer, A., Falcone, Y.: Decentralised LTL monitoring. In Giannakopoulou, D., Méry, D., eds.: FM 2012: Formal Methods. Volume 7436 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2012) 85–100
49. Cranor, L., Langheinrich, M., Marchiori, M., Presler-Marshall, M., Reagle, J.M.: The platform for privacy preferences 1.0 (p3p1.0) specification. World Wide Web Consortium, Recommendation REC-P3P-20020416 (April 2002)
50. Reagle, J., Cranor, L.F.: The platform for privacy preferences. Commun. ACM **42**(2) (February 1999) 48–55

51. Ashley, P., Hada, S., Karjoth, G., Powers, C., Schunter, M.: Enterprise Privacy Authorization Language (EPAL). Technical report, IBM Research, Rüschlikon (2003)
52. Karjoth, G., Schunter, M.: A privacy policy model for enterprises. In: Proceedings of the 15th IEEE Workshop on Computer Security Foundations. CSFW '02, Washington, DC, USA, IEEE Computer Society (2002) 271–281
53. May, M.J., Gunter, C.A., Lee, I.: Privacy apis: Access control techniques to analyze and verify legal privacy policies. In: Proceedings of the 19th IEEE Workshop on Computer Security Foundations. CSFW '06, Washington, DC, USA, IEEE Computer Society (2006) 85–97
54. Barth, A., Datta, A., Mitchell, J.C., Nissenbaum, H.: Privacy and contextual integrity: Framework and applications. In: Proceedings of the 2006 IEEE Symposium on Security and Privacy. SP '06, Washington, DC, USA, IEEE Computer Society (2006) 184–198
55. Barth, A., Mitchell, J., Datta, A., Sundaram, S.: Privacy and utility in business processes. In: Proceedings of the 20th IEEE Computer Security Foundations Symposium. CSF '07, Washington, DC, USA, IEEE Computer Society (2007) 279–294
56. Dinesh, N., Joshi, A., Lee, I., Sokolsky, O.: Checking traces for regulatory conformance. In Leucker, M., ed.: Runtime Verification. Volume 5289 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2008) 86–103
57. Lam, P., Mitchell, J., Sundaram, S.: A formalization of hipaa for a medical messaging system. In Fischer-Hübner, S., Lambrinoudakis, C., Pernul, G., eds.: Trust, Privacy and Security in Digital Business. Volume 5695 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2009) 73–85
58. Ni, Q., Bertino, E., Lobo, J., Brodie, C., Karat, C.M., Karat, J., Trombeta, A.: Privacy-aware role-based access control. ACM Trans. Inf. Syst. Secur. **13**(3) (July 2010) 24:1–24:31