Formal Methods in Computer-Aided Design 2020

SYSLITE: Syntax-Guided Synthesis of PLTL Formulas from Finite Traces

M. Fareed Arif, Daniel Larraz, Mitziu Echeverria, Andrew Reynolds, Omar Chowdhury, Cesare Tinelli
Department of Computer Science, The University of Iowa

***Abstract*—We present an efficient approach to learn past-time linear temporal logic formulas (PLTL) from a set of propositional variables and a sample of finite traces over those variables. The efficiency of our approach can be attributed to a careful encoding of the PLTL formula learning problem as a bit-vector function synthesis problem, and the use of an enhanced Syntax-Guided Synthesis (SyGuS) engine to solve the latter. We implemented our approach in a tool called SYSLITE and empirically evaluated its efficacy with two case studies. In these case studies, we observe that SYSLITE on average enjoys a speedup of 44x over current learning approaches for temporal formulas while learning the expected formulas in the vast majority of cases.**

I. Introduction

We are interested in the problem of synthesizing past-time, propositional linear temporal logic (PLTL) formulas when given an alphabet (*i.e.*,, a set of propositional variables) and a sample of finite traces as inputs. The input sample consists of a set of *positive traces* and a disjoint set of *negative traces*. The synthesized PLTL formulas — containing the usual logical connectives, past-time temporal operators, and propositional variables from the input alphabet — are required to be satisfied by each of the positive traces and falsified by each of the negative traces. In machine learning terms, our goal is to learn classifiers for the input traces. However, in contrast to statistical learning approaches, our setting requires an *exact classifier* for the sample traces, that is, one that rejects no positive traces and accepts no negative ones [1], [2].

The synthesis of PLTL formulas from finite samples has a variety of applications, including security policy mining from logs [3], [4], debugging or understanding the behavior of a system [5], and identifying the root cause of a protocol's misbehavior [6], [7]. The PLTL fragment we consider represents safety properties amenable to efficient runtime verification [8]–[12]. This fragment or its variants have been used to represent security, privacy, and safety properties of systems which can be efficiently enforced through runtime monitoring [11]–[15].

We use PLTL formula synthesis to learn attack signatures for cellular networks such as 3G, 4G LTE, and 5G from a set of *benign* (*i.e.,* positive) and attack (*i.e.,* negative) traces. The cellular network attacks we consider are possible due to the protocol state machine's inability to handle particular out-of-order packets injected over-the-air by an adversary [6], [16]–[20]. Such attack signatures can be characterized by PLTL formulas when considering the relative ordering of packets and their payloads received/sent by the cellular device. One can envision a protocol monitor installed on a mobile device that captures messages from the cellular modem with the goal of detecting particular attack signatures, and notifies the user when such attacks are detected. To our knowledge, there exist no attack notification mechanisms of this kind currently. Efficiently solving the PLTL formula synthesis problem is the first technical step towards building such mechanisms.

Prior work. The prior work most relevant to ours is the one by Neider and Gavran [5]. They present two methods for synthesizing propositional, future-only linear temporal logic (LTL) formulas given an alphabet and a sample of (finitely representable) infinite traces. The first method formulates the LTL formula synthesis problem as a Boolean satisfiability problem and then uses an off-the-shelf SAT solver to solve that problem. Because such SAT-based approach does not scale well, the authors then develop a second method based on decision tree learning where the SAT-based method is used as an oracle to generate predicates for the decision tree. More recently, Riener [21] improves on Neider and Gavran's SAT-based method by precomputing models for shape constraints required by the original method. The approaches followed in these works are not directly applicable to attack signature generation due to one or more of the following reasons: (1) they consider samples with infinite traces only; (2) they synthesize LTL formulas containing only future temporal operators, which are not necessarily monitorable at runtime; (3) they impose certain shape restrictions on the synthesized formula which lead to lengthy formulas.

Exploring possible approaches. Since the prior methods above [5], [21] are not directly applicable to our problem domain, we started by first adapting them to the synthesis of *PLTL* formulas from *finite traces*. In our evaluation, we observed that they either do not scale or do not yield succinct formulas. We then tried to reduce the synthesis problem to a Satisfiability Modulo Theory (SMT) problem where the PLTL syntax is encoded as an algebraic data-type (ADT) and the formula to synthesize is represented by a free variable f with that type. We encoded the requirements of acceptance of the positive traces and rejection of the negative traces as constraints on f and used an SMT solver with finite model finding capabilities [22], [23] to obtain models of the ADT problem. Such models assign to f a datatype value representing a candidate solution to the synthesis problem. Unfortunately, this SMT-based approach is not scalable either, which prompted us to consider an encoding of our synthesis problem as a Syntax-Guided Synthesis (SyGuS) problem [24] over ADTs. Similarly to previous approach, however, the

SyGuS approach proved to be not scalable. The main reason in both cases seems to be that ADTs are user-defined and hence do not benefit from the sort of specialized optimizations that SMT solvers employ for other builtin theories.

Our approach. This brings us to our final approach in which we encode the problem as a SyGuS problem with fixed-size bit-vectors and use a specific SyGuS engine [25] to solve it. In our encoding, we view the projection of a trace of length n over a propositional variable as a bit-vector of size n and then lift the semantics of logical and past temporal connectives to operate over bit-vectors. Such an encoding has the following advantages: (1) since fixed-size bit-vectors are natively supported by the SyGuS solver, we benefit from the solver's various optimization techniques (e.g., rewrite rules) for them; (2) restrictions on the shape of the formula to be learned can be readily added as syntactic constraints on the SyGuS problem; (3) semantics constraints capturing the formula's consistency with sample traces can be efficiently evaluated through direct bit-vector operations on whole traces, unlike prior approaches which operate on each individual state in a trace; (4) with an appropriate term enumeration strategy within the SyGuS solver, it is possible to obtain candidate formulas of minimal size together with other candidates; (5) thanks to the SyGuS solver's symmetry breaking criteria (*i.e.*,, agreement over the sample traces), our approach can enumerate different shapes of formulas while maintaining scalability.

Implementation and evaluation. We have implemented our approach in a novel tool called SYSLITE[1] which uses the CVC4SY SyGuS engine [25]. We also adapted to our setting and implemented the prior methods [5], [21] mentioned earlier and considered them as baselines in our experiments. We evaluated the various approaches based on their scalability and ability to synthesize succinct PLTL formulas.

To verify the generality of our SyGuS approach, in a first case study, we collected a number of PLTL formulas from the literature and considered the behavior they represent as our learning target. For each target formula, we generated random traces and classified them as positive or negative based on whether they satisfied or falsified the formula. We then fed a subset of these classified random traces to both SYSLITE and our implementation of the baseline approaches, and compared the synthesized formulas with the corresponding target formulas. We observed that SYSLITE exhibits an average 60x speedup over the baseline while synthesizing a formula logically equivalent to the target formula in most cases.

In a second case study, we used real-world cellular network traces for 11 known attacks [6], [16]–[20]. We observed that SYSLITE can learn the attack signatures 28x times faster on average than the baseline while still being able to generate succinct attack signatures.

Contributions. To summarize, this paper makes the following technical contributions:

1) We explored a number of possible approaches for PLTL formula learning from samples, including extensions of prior SAT-based approaches originally applied to learning LTL formulas with future operators only. Our empirical evaluation show that none of these approaches scale to realistic trace lengths and numbers of input traces.
2) We propose a new, more scalable learning approach which formulates the learning problem as a SyGuS problem and relies on a high-performance SyGuS engine to generate candidate solutions. Our encoding uses the theory of fixed-size bit-vectors which is natively supported by the underlying SyGuS solver, enabling us to benefit from several specific optimizations.
3) Our PLTL formula learning approach is implemented in a new tool, SYSLITE, which uses the CVC4SY SyGuS engine as a backend. We have empirically evaluated its efficacy on two case studies while considering previous state-of-the-art methods as baselines. The case studies show that SYSLITE on-average enjoys a 44x speed-up over the baselines while, at the same time, being able to learn the expected behavior in almost all cases.

II. TECHNICAL PRELIMINARIES

Many-Sorted First-Order Logic. We rely on the usual notions and terminology of many-sorted first-order logic with equality ($\simeq$). We assume the usual definitions of signature, well-sorted terms, literals, and formulas [26]. A *theory* is a pair $T = (\Sigma, I)$ where Σ is a signature and I is a non-empty class of Σ-interpretations, the *models of* T, that is closed under variable reassignment and isomorphism. A Σ-formula φ is T-*satisfiable* (respectively, T-*unsatisfiable*) if it is satisfied by some (resp., no) interpretation in I. A satisfying interpretation for φ, *models* φ. A formula φ is *valid in* T (or, T-*valid*), written $\models_T \varphi$, if every model of T is a model of φ.

Theory of Fixed-size bit-vectors. The theory $T_{\mathrm{BV}} = (\Sigma_{\mathrm{BV}}, I_{\mathrm{BV}})$ of fixed-size bit-vectors as defined in the SMT-LIB 2 standard [27] consists of the class of interpretations I_{BV} and signature Σ_{BV}, which includes a unique sort for each positive integer n, representing the bit-vector width. We assume that Σ_{BV} includes all *bit-vector constants* for each n, represented here as bit-strings or, to simplify the notation, by the corresponding natural number in $\{0, \ldots, 2^{n-1}\}$. We write a Σ_{BV}-term (or, *bit-vector term*) t of width n as $t_{[n]}$ when we want to specify its bit-width explicitly. We refer to the i-th bit of $t_{[n]}$ as $t[i]$ with $0 \leq i < n$. We consider $t[0]$ as the least significant bit, and $t[n-1]$ as the most significant bit of t, and denote the subvector of t from index j down to i as $t[j:i]$. We will use the following arithmetic bit-vector operators: addition ($+$), arithmetic negation ($-$), and unsigned shift to the left ($\ll$), as well as the following bitwise operators: logical negation ($\sim$), conjunction ($\&$), and disjunction ($|$).

SyGuS Problem. A SyGuS problem for a function f in a theory T consists of (1) *semantic restrictions*, or a specification, given by a (second-order) T-formula of the form $\exists f.\, \varphi$, and (2) *syntactic restrictions* on the definitions for f, given by a context-free grammar R. A *solution for* f is a lambda term $\lambda \boldsymbol{x}.\, e$ of the same type as f, such that (i) the formula

[1] SYSLITE is available at https://github.com/CLC-UIowa/SysLite.

$\varphi\{f \mapsto \lambda \boldsymbol{x}.\ e\}$ is T-valid (modulo beta-reductions) and (ii) the term e is in the language generated by R.

Past-Time Propositional Linear Temporal Logic (PLTL). The formulas we learn are of the form $\Box_f\ \Phi$ where Φ is a PLTL formula and $\Box_f$ is a future temporal operator over finite traces (discussed below).

Definition 1 (Syntax)**.** *The set of well-formed PLTL formulas, denoted as Φ and Ψ, is generated by the following grammar:*

$$\Phi, \Psi \quad ::= \quad \top \quad | \quad \bot \quad | \quad p \quad | \quad \circ^1\, \Phi \quad | \quad \Phi \circ^2 \Psi$$

where p belongs to a non-empty set, or alphabet, *$\mathcal{A}$ of propositional variables, $\circ^1 \in \{\neg, \ominus, \diamondminus, \boxminus\}$, and $\circ^2 \in \{\wedge, \vee, \mathcal{S}\}$. A* core *formula is a formula that does not contain the operators $\vee$, $\diamondminus$, and $\boxminus$. The* size *of a formula Φ, denoted with $|\Phi|$, is the number of its proper subformulas.*

Informally, $\top$ and $\bot$ are the universally true and the universally false formulas, respectively, and $\wedge, \vee$, and $\neg$ are the usual Booleans operators. On the other hand, $\ominus, \diamondminus, \boxminus$, and $\mathcal{S}$ are past temporal operators, respectively read as "yesterday", "once", "historically", and "since. Unary operators have a higher precedence than binary operators, and temporal operators have a higher precedence than logical operators.

We *fix an alphabet* $\mathcal{A}$ for the PLTL formulas in the rest of the paper. The standard PLTL semantics is defined over infinite traces in a Kripke structure [28]. For our purposes, however, it is more useful to define a semantics of PLTL over *finite* traces. A *finite trace σ (of length $n \in \mathbb{N}$ over $\mathcal{A}$)* is a sequence $(\sigma_0, \ldots, \sigma_{n-1})$ of *states* where a state is a total mapping from $\mathcal{A}$ to the set $\{\mathbf{t}, \mathbf{f}\}$ of Boolean values. Let $\sigma = (\sigma_0, \ldots, \sigma_{n-1})$ be a trace of length n. For a propositional variable $p \in \mathcal{A}$ and we denote by $\sigma(p)$ the *projection* of σ over p, that is, the sequence of Boolean values $(\sigma_0(p), \ldots, \sigma_{n-1}(p))$.

Definition 2 (Semantics)**.** *The semantics of PLTL is provided by a ternary satisfiability relation $\models$ defined inductively over core PLTL formulas as follows for all finite traces $\sigma = (\sigma_0, \ldots, \sigma_{n-1})$ and positions $i \in [0, n-1]$.*

- $\sigma, i \models \top$
- $\sigma, i \models p$ *if* $\sigma_i(p) = \mathbf{t}$
- $\sigma, i \models \neg\Phi$ *if* $(\sigma, i) \not\models \Phi$
- $\sigma, i \models \Phi \wedge \Psi$ *if* $(\sigma, i) \models \Phi$ *and* $(\sigma, i) \models \Psi$
- $\sigma, i \models \ominus\Phi$ *if* $i > 0$ *and* $(\sigma, i-1) \models \Phi$
- $\sigma, i \models \Phi\, \mathcal{S}\, \Psi$ *if there is an* $j \in [0, i]$ *such that* $(\sigma, j) \models \Psi$ *and* $(\sigma, k) \models \Phi$ *for all* $k \in [j+1, i]$.

This semantics is extended to the full language of PLTL by treating the additional operators as syntactic sugar according to the following equivalences: $\bot \equiv \neg\top$; $\Phi \vee \Psi \equiv \neg(\neg\Phi \wedge \neg\Psi)$; $\diamondminus\Phi \equiv \top\, \mathcal{S}\, \Phi$; $\boxminus\Phi \equiv \neg\diamondminus\neg\Phi$. We write $\sigma \models \Phi$ as a shorthand for $\sigma, 0 \models \Phi$. Finally, we write $\sigma \models \Box_f\Phi$ to indicate that $\sigma, i \models \Phi$ for all $i \in [0, n-1]$ where n is the length of σ.

III. Problem Definition and Possible Approaches

In this section, we formalize the problem of PLTL formula synthesis from finite samples and discuss potential but inefficient approaches for solving it. We start by introducing the auxiliary notion of *consistency* used in our problem definition.

Definition 3 (Consistency)**.** *A PLTL formula Φ is* consistent *with a finite sample $\mathcal{D} = (\mathcal{P}, \mathcal{N})$ of* positive *finite traces $\mathcal{P}$ and* negative *finite traces $\mathcal{N}$ with $\mathcal{P} \cap \mathcal{N} = \emptyset$ if and only if the following two conditions hold.*

1) $\sigma^+ \models \Box_f\Phi$ *for all traces* $\sigma^+ \in \mathcal{P}$.
2) $\sigma^- \not\models \Box_f\Phi$ *for all traces* $\sigma^- \in \mathcal{N}$.

A formula Φ consistent with $\mathcal{D}$ is *minimal* if no PLTL formula Ψ with $|\Psi| < |\Phi|$ is consistent with $\mathcal{D}$.

Problem Definition 1 (PLTL Formula Synthesis from Finite Samples)**.** *The PLTL formula synthesis problem for a given sample $\mathcal{D} = (\mathcal{P}, \mathcal{N})$ is the problem of finding one or more PLTL formulas Φ that are consistent with $\mathcal{D}$.*

A. Possible Approaches

We considered several natural approaches to the PLTL synthesis problem. Unfortunately, our experimental evaluation revealed that they do not scale well. It is, however, valuable to discuss them here because their weaknesses point to potential performance bottlenecks which any synthesis algorithm must overcome to be effective in practice. We describe a better approach in Section IV.

SAT-based Approaches. We adapted to our context prior SAT-based approaches for learning LTL formulas from samples containing only infinite traces [5], [21]. These approaches look for formulas of increasing size, measured as the *depth* of the formula's abstract syntax tree (AST) which, in essence, guarantees the identification of minimal formulas consistent with a given sample $\mathcal{D}$. As in the approach by Neider and Gavran [5], for a given depth d, the PLTL formula synthesis problem can be posed as the problem of checking the satisfiability of a formula γ^d of propositional logic. The reduction is meant to be such that, γ^d is satisfiable exactly when the original synthesis problem is solvable. Moreover, it is possible to construct a PLTL solution to the synthesis problem from any propositional model of γ^d. The formula γ^d has the form $\gamma^d_{\text{syn}} \wedge \gamma^d_{\text{sem}}$ where γ^d_{syn} tries to captures syntactic restrictions on the expected solution (a well-formed PLTL formula with depth d) whereas γ^d_{sem} captures the semantic restriction that the extracted solution is consistent with the sample.[2] In turn, γ^d_{syn} has the form $\gamma^d_{\text{shape}} \wedge \gamma^d_{\text{label}}$ where models of γ^d_{shape} determine possible AST shapes of depth d (including some infeasible ones) and models of γ^d_{label} assign labels (*i.e.*,, propositions, logical or temporal operators) to the AST nodes. To identify different feasible formulas, this SAT-based approach can be executed in *enumerative* mode by blocking a returned model of γ^d and reissuing a call to the SAT solver with γ^d as well as the blocking formula. Similarly to the original work, this approach does not scale to realistically sized traces or large numbers of them, as we discuss in our evaluation section.

[2] In practice, models of γ^d_{syn} can lead to an overabundance of PLTL solutions since the syntactic restrictions are not strong enough to rule out certain redundancies. Thus, some *a posteriori* filtering is required.

Riener [21] improved on Neider and Gavran's work by precomputing the models of the formula γ^d_{shape} for a given depth d and supplying them with the rest of the formulas in γ^d, in effect trading off input size for execution time. The improved method essentially breaks a number of symmetries, greatly reducing the number of solutions that differ in an insignificant way from each other. It can generate stronger syntactic restrictions by relying on an underlying representation based on chains instead of directed acyclic graphs as in Neider and Gavran. We adapted the method to our context but observed that scalability issues persist, especially, when the alphabet size is larger than 3.

Finally, we also considered a second approach by Neider and Gavran [5] which combines a classical decision tree learning algorithm with their SAT-based approach. In a first phase, the SAT-based algorithm is executed over k positive and k negative traces to obtain a candidate formula. The approach keeps choosing randomly from $2k$ traces until all the example traces can be separated or a timeout is reached. At that point, it invokes the decision tree learning algorithm which essentially uses the candidate formulas generated in the first phase as possible predicates for the decision tree. Because the decision tree learning algorithm combines these predicates into if-then-else clauses, it only applies to logical languages that are closed under negation. Unfortunately, the presence of the outermost $\Box_f$ operator in our PLTL fragment of interest, makes this fragment not closed under negation and hence this second approach is not applicable to our case.

SMT-based Approach. One of the scalability challenges of SAT-based algorithms can be attributed to the inefficient enumeration of the well-formed PLTL formulas. This is particularly apparent in the approach of Riener [21] who attempts to address this challenge through precomputation. A natural potential solution is to move to an SMT-based approach where the formula to be synthesized is a value of an algebraic data type (ADT) Δ that captures the abstract syntax of well-formed PLTL formulas directly. Each PLTL propositional constant and (logical and temporal) operator is modeled by a corresponding constructor of Δ with the same arity. Traces can be encoded as (partially defined) Boolean maps from propositional constants and trace positions. The PLTL semantics is captured by an *evaluation function*, a recursively defined total function that takes a trace t and a data type d as input and returns true if and only if t satisfies the formula represented by d. The synthesis problem is then encoded by a set of constraints on a fresh constant φ of type Δ, standing for the formula to be synthesized, stating that the evaluation of φ is true for all the positive traces and false for all the negative ones. Synthesizing the PLTL formula thus reduces to asking the SMT solver to find a model of the ADT problem. If it succeeds, the ADT value assigned to φ describes a possible solution. In our evaluation, we observed that such an approach is unfortunately also not scalable, possibly due to the inherent complexity of solving SMT problems over ADTs.

SyGuS-based Approach. We explored next a SyGuS-based approach where the PLTL syntax is encoded as a context-free grammar whereas the consistency with the sample set is given as the specification. Although more scalable than the SMT-based one, this approach is still not sufficiently scalable for our case studies. An analysis of our SyGuS encoding revealed the following two weaknesses whose mitigation led us to our final approach, discussed in the next section. First, since algebraic data types are user-defined, reasoning about them does not benefit from the specialized optimizations (*e.g.*,, rewrite rules, symmetry breaking) available to SMT solvers for other builtin theories such as bit-vectors or linear integer arithmetic. Second, both this and the SMT-based approach require evaluating a candidate solution at each position of each trace in order to guarantee consistency with the sample. Expressing such a constraint requires the use of quantified formulas (with quantification over traces and positions) and recursive function definitions (for the evaluation function) both of which are expensive to reason about.

B. *Lessons learned*

After analyzing the different approaches above to the PLTL synthesis problem, we identified the following performance bottlenecks, which we tried to address in our final approach. First, the SAT-based approaches produce constraints with a lot of symmetries and hence many redundant solutions, a substantial bottleneck. Except for the SyGus-based approach, none of these approches apply symmetry breaking optimizations to rule out or reduce the generation of formulas similar to previously generated ones, substantially hampering the generation of truly diverse PLTL formulas consistent with the input sample. Finally, all the approaches attempt to achieve sample-consistency through (quantified or explicit) constraints on *individual* trace positions, thus missing out on full-trace-level optimizations, which are crucial to scalability.

Examples of our SMT-based and SyGuS-based encodings can be found in the longer version of this paper [29].

IV. PLTL Synthesis with SyGuS

In this section, we present an efficient approach for synthesizing a PLTL formula consistent with a finite sample $\mathcal{D}$ using a SyGuS solver over the theory of fixed-sized bit-vectors. It relies on the observation that a PLTL formula over finite traces of length at most n can be encoded as a function over bit-vectors of size n. Thus, the problem of synthesizing a PLTL formula is reduced to the synthesis of a bit-vector function.

Similarly to a bit-vector encoding presented by Baresi et al. [30], we use bit-vectors of size $n > 0$ to represent the truth values of PLTL formulae at positions $[0, n-1]$ of a given trace of length n. More precisely, for each atomic proposition $p \in \mathcal{A}$, we use a bit-vector variable $\overleftarrow{p}_{[n]}$ such that $\overleftarrow{p}_{[n]}[i]$ captures the value of proposition p at all instants i from 0 to $n-1$. The bit-vector representation of $\bot$ for length n, denoted with $\overleftarrow{\bot}_{[n]}$, is the bit-vector constant 0 of size n, while the bit-vector representation of $\top$, denoted with $\overleftarrow{\top}_{[n]}$, is the value of $\sim\overleftarrow{\bot}_{[n]}$. For any other PLTL formula Φ, we describe the value of Φ at positions 0 through $n-1$ in a trace by the bit-vector obtained by recursively performing operations on the bit-vectors corresponding to the sub-formulas of Φ. The operations performed depend on the structure of Φ and follow the transformations shown in Table I.

TABLE I. Translation of a PLTL formulas to bit-vector terms.

Φ	$\overleftarrow{\Phi}$	unfolded bit-vector encoding
$\neg\Psi$	$\sim\overleftarrow{\Psi}$	$\sim\overleftarrow{\Psi}$
$\Psi_1 \wedge \Psi_2$	$\overleftarrow{\Psi_1} \,\&\, \overleftarrow{\Psi_2}$	$\overleftarrow{\Psi_1} \,\&\, \overleftarrow{\Psi_2}$
$\Psi_1 \vee \Psi_2$	$\overleftarrow{\Psi_1} \mid \overleftarrow{\Psi_2}$	$\overleftarrow{\Psi_1} \mid \overleftarrow{\Psi_2}$
$\ominus\Psi$	$\circledtilde\overleftarrow{\Psi}$	$\ll \overleftarrow{\Psi}$
$\diamondminus\Psi$	$\widehat{\diamondminus}\overleftarrow{\Psi}$	$-\overleftarrow{\Psi} \mid \overleftarrow{\Psi}$
$\boxminus\Psi$	$\widehat{\boxminus}\overleftarrow{\Psi}$	$\sim(1+\overleftarrow{\Psi}) \,\&\, \overleftarrow{\Psi}$
$\Psi_1 \,\mathcal{S}\, \Psi_2$	$\overleftarrow{\Psi_1}\overleftarrow{\mathcal{S}}\overleftarrow{\Psi_2}$	$\overleftarrow{\Psi_2} \mid (\sim((\overleftarrow{\Psi_1} \mid \overleftarrow{\Psi_2}) + \overleftarrow{\Psi_2}) \,\&\, \overleftarrow{\Psi_1})$

Table I also introduces new bit-vector operators, $\circledtilde, \widehat{\diamondminus}, \widehat{\boxminus}$, and $\overleftarrow{\mathcal{S}}$ denoting, respectively, the bit-vector encodings for the temporal operators $\ominus, \diamondminus, \boxminus$, and $\mathcal{S}$. To establish the correctness of the connection between the bit-vector encoding and the semantics of PLTL (see Theorem 1) and to explain the example we use the following notation: for a propositional variable $p \in \mathcal{A}$ and a trace σ of length n, $\overleftarrow{\sigma(p)}$ denotes the bit-vector representation of $\sigma(p)$, that is, for all $i \in [0, n-1]$, $\overleftarrow{\sigma(p)}[i] = 1$ if $\sigma_i(p) = \mathbf{t}$, and $\overleftarrow{\sigma(p)}[i] = 0$ if $\sigma_i(p) = \mathbf{f}$.

To see more concretely how the translation works we explain, for instance, the correspondence between the unary PLTL operator $\diamondminus$ (read: true at least once in the present or past) and its bit-vector counterpart $\widehat{\diamondminus}$ with an example.

Example 1. Let σ be a trace of length 6 where propositional variable p is true only at positions 3 and 4. The projection $\sigma(p)$ is represented by the bit vector `011000` with the most significant (*i.e.,*, leftmost) bit corresponding to $\sigma_5(p)$, the next most significant bit corresponding to $\sigma_4(p)$, and so on. So $\overleftarrow{\sigma(p)} =$ `011000`. Intuitively, the valuation of $\diamondminus p$ over σ should then be represented by the bit-vector `111000`. To verify that let $\overleftarrow{p}_{[6]}$ be the bit-vector variable corresponding to p. According to our translation, $\overleftarrow{\diamondminus p} = \widehat{\diamondminus}(\overleftarrow{p}) = -\overleftarrow{p} \mid \overleftarrow{p} = -\overleftarrow{p}_{[6]} \mid \overleftarrow{p}_{[6]}$ where $\mid$ is bitwise disjunction and $-$ is arithmetic negation (two's complement). If we evaluate the resulting bit-vector term with the valuation $\alpha = \{\overleftarrow{p}_{[6]} \mapsto \texttt{011000}\}$ we get

$$\begin{aligned}\alpha(-\overleftarrow{p}_{[6]} \mid \overleftarrow{p}_{[6]}) &= -\texttt{011000} \mid \texttt{011000} \\ &= \texttt{101000} \mid \texttt{011000} = \texttt{111000}\end{aligned}$$

as expected. □

Theorem 1. *Let Φ be a PLTL formula over the alphabet $\mathcal{A} = \{p_1, \ldots, p_m\}$ and let σ be a trace of length n over $\mathcal{A}$. Then,*

$$\sigma \models \Box_f \Phi \quad \textit{iff} \quad \models_{T_{\mathrm{BV}}} \overleftarrow{\Phi}\,\{\bar{p} \mapsto \bar{\sigma}\} \simeq \overleftarrow{\top}_{[n]}$$

where $\bar{p} = (\overleftarrow{p_1}_{[n]}, \ldots, \overleftarrow{p_m}_{[n]})$ and $\bar{\sigma} = (\overleftarrow{\sigma(p_1)}, \ldots, \overleftarrow{\sigma(p_m)})$.

Proof. By induction on the structure of Φ. See Arif et al. [29] for a full proof. □

We now describe how we use the bit-vector encoding above to reduce the problem of synthesizing a PLTL formula consistent with a sample into a SyGuS problem over bit-vectors. More precisely, given propositional variables $p_i \in \mathcal{A}$, with $1 \le i \le m$, and a sample $\mathcal{D} = (\mathcal{P}, \mathcal{N})$ whose longest trace has length n, we seek to synthesize a bit-vector function $f(\overleftarrow{p_1}_{[n]}, \ldots, \overleftarrow{p_m}_{[n]})$ such that if $\lambda\overleftarrow{p_1}_{[n]}, \ldots, \lambda\overleftarrow{p_m}_{[n]}.\ e$ is a solution for the SyGuS problem, then there exists a PLTL formula Φ consistent with $\mathcal{D}$ whose bit-vector encoding is e (that is, $\overleftarrow{\Phi} = e$).

To meet the requirements on f, we start by imposing the syntactic restrictions expressed by this context-free grammar:

$$\Psi \quad ::= \quad \overleftarrow{\top}_{[n]} \quad \mid \quad \overleftarrow{\bot}_{[n]} \quad \mid \quad \overleftarrow{p}_{[n]} \quad \mid \quad \circ^1\,\Psi \quad \mid \quad \Psi \circ^2 \Psi$$

where $\overleftarrow{p}$ is $\overleftarrow{p_j}_{[n]}$ for some $j \in [0, m]$, $\circ^1 \in \{\sim, \circledtilde, \widehat{\diamondminus}, \widehat{\boxminus}\}$ are the unary operators, and $\circ^2 \in \{\&, |, \overleftarrow{\mathcal{S}}\}$ are the binary operators. Notice that, although $\circledtilde, \widehat{\diamondminus}, \widehat{\boxminus}$, and $\overleftarrow{\mathcal{S}}$ do not belong to the theory of bit-vectors, they can be defined using a bit-vector function in the SyGuS problem (see Table I).

In addition, the function f is subject to the following semantic restrictions where $|\sigma|$ denotes the length of trace σ:

1) $\bigwedge_{\sigma \in \mathcal{P}} f(\overleftarrow{\sigma(p_1)}, \ldots, \overleftarrow{\sigma(p_m)})[|\sigma| - 1 : 0] \simeq \overleftarrow{\top}_{[n]}[|\sigma| - 1 : 0]$
2) $\bigwedge_{\sigma \in \mathcal{N}} f(\overleftarrow{\sigma(p_1)}, \ldots, \overleftarrow{\sigma(p_m)})[|\sigma| - 1 : 0] \not\simeq \overleftarrow{\top}_{[n]}[|\sigma| - 1 : 0]$

The two constraints enforce the consistency of the solution respectively with the positive traces and the negative traces. Notice that, since an input may include traces of different length, we compare only the relevant positions for each trace.

V. Implementation and Evaluation of SYSLITE

In this section, we discuss the implementation of SysLite and our empirical evaluation of it based on two case studies.

A. SysLite *Implementation*

SysLite is a wrapper around the syntax-guided synthesis solver cvc4sy which is part of the SMT solver cvc4 [31] and now incorporates additional optimizations for PLTL synthesis. cvc4sy supports various theories, including that of fixed-size bit-vectors, used in our encoding, and implements several specialized synthesis algorithms for various types of synthesis conjectures [32]. We rely on its support for enumerative counterexample-guided inductive synthesis (CEGIS) which was recently improved with several novel strategies [33].

In enumerative CEGIS [34], candidate solutions are generated based on some ordering, typically on term size. In our setting, a candidate solution is a function whose definition involves the bit-vector symbols from Section IV. cvc4sy uses advanced techniques to aggressively reduce the number of candidate solutions it generates. In particular, it uses fast incomplete techniques based on term rewriting to avoid generating candidate solutions s' that are logically equivalent to some previous candidate s. This form of *symmetry breaking*, is critical for the scalability of enumerative approaches [32]. Our encoding of PLTL formulas as bit-vector constraints was motivated by the intention to capitalize on CVC4SY's infrastructure for establishing the equivalence of bit-vector terms developed to accelerate SyGuS enumeration [35].

For synthesis conjectures (*i.e.*,, semantic restrictions) $\exists f.\, \varphi$ where all applications of f in φ have concrete values as arguments, CVC4SY can apply a stronger version of symmetry breaking that considers *equivalence under examples*. Suppose the concrete inputs for f in φ are $c_1, \ldots, c_n$. Using this technique, while constructing a new candidate solution for f, the solver disregards any term t' that over the inputs $c_1, \ldots, c_n$ evaluates exactly as some previously disregarded term t. For example, the terms $x \;\&\, y$ and x take the same value over the inputs $(0001, 0001), (0000, 0001), (1010, 1110)$ for (x, y). Hence, one of them ($x \;\&\, y$, due to its larger size) will be excluded from consideration in candidate solutions since it is equivalent to x for all relevant inputs as specified in the conjecture. In practice, this heuristic is traditionally applied when the synthesis conjecture specifies a set of input/output pairs for the function f to synthesize (with constraints of the form $f(c_i) = o_i$). We have generalized symmetry breaking in CVC4SY to apply the heuristics to any conjecture $\exists f.\, \varphi$ where f is applied to concrete inputs, even when φ is not just a conjunction of input/output constraints. In our specific context, this enables symmetry breaking constraints for the negative traces, and also allows us to have traces of different length in the same problem.

Since the evaluation of terms on concrete examples is a major bottleneck in syntax-guided synthesis solvers, we have additionally implemented in CVC4SY several low-level optimizations for quickly computing the result of PLTL terms on concrete inputs. Thanks to our encoding of PLTL formulas as bit-vector constraints, we can capitalize on the data structures in the core of CVC4 for representing and efficiently evaluating bit-vectors terms. Our experiments confirm that this is critical to achieving scalability for the synthesis tasks in question.

The enumeration strategy itself (by formula size) remains a major bottleneck in our approach when behavior consistent with the training set cannot be captured by a small formula. In contrast, capturing behavior that spans distant states on a trace is not, per se, problematic because evaluation times for a given candidate solution grow linearly with trace length.

B. Empirical Analysis Criteria and Configuration

Research questions. In our evaluation of SysLite, we aimed to answer two research questions. Compared to a baseline:

RQ_1. How effective is SysLite in synthesizing succinct, diverse, and accurate PLTL formulas?

RQ_2. How scalable is SysLite?

Case studies. We address the above questions in the context of the two case studies presented in Sections V-C and V-D, respectively. The first focuses on RQ_1 whereas the second focuses on RQ_2 based on SysLite's ability to synthesize attack signatures from real cellular network traces.

Baseline. We compare SysLite against a baseline represented by our own implementation of the (first) SAT-based method by Neider and Gavran [5]. We use our own implementation and not theirs because the latter applies to traditional LTL, as opposed to PLTL. We do not discuss here the other approaches we tried, that is, Reiner's SAT-based approach [21], our encodings to algebraic data types, as well as DFA learning approaches, specifically, RPNI [36], since they proved either not scalable or ineffective. We point out that, in the second case study (V-D), the passive DFA learning approach does scale significantly better than SysLite with trace length and number of traces. However, the produced signatures are of significantly worse quality in all considered benchmarks (e.g., have F1 score as low as 0.35 for RLF report attack). Moreover, the quality of the DFA signatures does not necessarily improve with a larger set of traces or longer traces over the signatures produced by SysLite. In other words, SysLite can learn better quality signatures with fewer and shorter traces. Furthermore, recall that in this case study the objective is to generate attack monitors that execute on a mobile phone. A PLTL formula of size n can be monitored with just $2n$ *bits* of memory [8]. In contrast, the learned DFA equivalent to a PLTL formula can have $O(2^n)$ states [37]–[39]. The memory footprint of such a high number of states per signature makes DFA-based monitors infeasible in practice, especially, when many attacks are being monitored at the same time.

Sample sizes. For both of our case studies, we considered sample sizes 50, 100, 250, 500, and 1250. For Case Study I, traces were generated randomly and have length 10 whereas for Case Study II the traces were collected from a cellular network and have length 100. We chose on purpose data sets with an equal number of positive and negative traces. An imbalanced dataset, due for example to an uneven distribution of positive and negative traces for the target behavior (which we did observe in some of the benchmarks), can negatively impact the quality of the synthesized formula by not restricting the search space enough to learn the desired behavior early in the search. Oversampling, on the other hand, does not impact the quality of the synthesized formula, although it can obviously impact training time.

Training and testing configuration. We used the standard Pareto-principle of classifier evaluation which suggests an (80%, 20%) partition of the provided sample set into training and testing datasets, respectively. By considering a synthesized PLTL formula Φ as a classifier for the traces in the testing set, its quality can be measured in terms of *precision* (the percentage of correctly classified traces among all traces classified as positive by Φ), *recall* (the ratio of correctly classified positive traces over the total number of positive traces) and their harmonic mean (F1 score). Moreover, the evaluation method also performs cross-validation. It considers the first five solutions generated by SysLite and by the baseline, selecting the formula (or formulas, in case of ties) with the highest F1 score.

In Case Study I, one could imagine directly comparing the *closeness* of a synthesized formula to the target formula, for instance by considering the Jaccard distance of the sets of satisfying traces, up to some fixed length n, for each formula. We did not do it since it is prohibitively expensive for requiring the enumeration of all such traces. A better approach might be to estimate closeness by adapting model counting techniques to this setting, something we leave to future work.

Evaluation infrastructure. We performed all our evaluations on a 3.40GHz Intel(R) Xeon(R) E3-1240 CPU running CentOS (Linux Kernel 3.10.0-1062.9.1) on 16GB RAM. We set a 3600 second timeout for each learning task.

C. Case Study I: PLTL Formulae from Literature

The purpose of this case study was to measure SysLite's effectiveness in synthesizing succinct and accurate formulas from a sample set of traces. For this, we first collected a few representative PLTL formulas from the literature (see Table II). For each of them, we collected a sample consisting of randomly generated traces and then checked if SysLite and the baseline were able to learn the original formula or a logically equivalent one. We had both synthesis approaches generate up to 5 candidate formulas before a given timeout.

TABLE II. target formulas from the literature.

Literature Formula	PLTL Formula
Chinese Wall Policy [11]	$\Box_f((\text{access\_org1\_records} \Rightarrow \neg\diamondminus(\text{access\_org2\_records})) \wedge (\text{access\_org2\_records} \Rightarrow \neg\diamondminus(\text{access\_org1\_records})))$
Bank Transaction Policy [11]	$\Box_f(\text{Transaction\_over\_threshold\_performed} \Rightarrow \diamondminus(\text{Transaction\_over\_threshold\_approved}))$
Secure File [11]	$\Box_f((\text{secure\_file\_open} \Rightarrow (\ominus(\boxminus(\neg(\text{secure\_file\_open}))) \vee \ominus(\neg\text{secure\_file\_open}\,\mathcal{S}\,\text{secure\_file\_closed}))))$
Financial Institute Policy [11]	$\Box_f(\text{grant} \Rightarrow \ominus(\neg\text{grant}\,\mathcal{S}\,\text{request}))$
GLBA-6802 [12], [15]	$\Box_f(\text{institution\_discloses\_to\_affiliate\_customers\_npi} \Rightarrow (\neg\text{customer\_opt\_out}\,\mathcal{S}\,\text{notice\_of\_disclosure}))$
HIPPA-164508A2 [12], [15]	$\Box_f(\text{covered\_entity\_discloses\_patient\_psych\_notes} \Rightarrow (\neg\text{authorization\_psych\_notes\_revoked})\,\mathcal{S}\,\text{receive\_patient\_authorization\_psych\_notes})$
HIPPA-164508A3 [12], [15]	$\Box_f(\text{covered\_entity\_discloses\_patient\_info\_for\_marketing} \Rightarrow \diamondminus(\text{receive\_patient\_authorization\_marketing}))$
Dynamic Separ. of Duty [11]	$\Box_f(\text{member\_activates\_role1} \Rightarrow (\ominus(\boxminus(\neg\text{member\_activates\_role2})) \vee \ominus(\neg\text{member\_activates\_role2}\,\mathcal{S}\,\text{member\_deactivates\_role2})))$

Trace generation: Given a target formula φ from Table II, a desired trace length ℓ, and a desired sample size $2n$, our trace generation process uses a cryptographically-secure pseudorandom number generator to produce a sample set $\mathcal{P}$ of n positive traces and a sample set $\mathcal{N}$ of n negative traces, all of length ℓ. It generates a trace σ of length ℓ by randomly assigning truth values to φ's propositional variables for each of the ℓ states of σ. The trace goes in the set $\mathcal{P}$ or $\mathcal{N}$ depending on whether it satisfies φ or not, as long as the set in question contains less than n traces; otherwise, it is discarded. Note that, depending on the target formula φ, we may have to oversample for positive or negative traces.

Measuring quality of synthesized formulas. To evaluate the quality of the synthesized formulas, in addition to relying on the usual statistical measures (i.e., precision, recall, and F1 score) on the test dataset, we considered logical equivalence with the target formula (i.e., being satisfied by exactly the same set of possible traces) as another metric of effectiveness. We used the GOAL tool [40] to check for equivalence in PLTL.

Quality of synthesized formulas. Our results on the synthesized formulas' quality (i.e., equivalence to target formula) and count are summarized in Table III. For each run of SysLite and the baseline for a particular dataset and a target formula, we select the highest-ranked formula after cross validation[3] among those synthesized in the allotted time, if any. For each original (target) formula, column **Count** reports the total of number selected formulas across the 5 training sets of different size. For instance, a value of $2/5$ indicates that the algorithm was able to synthesize formulas for 2 of the 5 training sets. Column **Quality** reports how many of the selected formulas are logically equivalent to the target formula.

TABLE III. Case Study I: Quality of Synthesis Methods.

	SysLite		SAT	
target Formula	**Count**	**Quality**	**Count**	**Quality**
Chinese Wall Policy [11]	5/5	1/5	4/5	0/5
Bank Transaction Policy [11]	5/5	5/5	4/5	4/5
Secure File [11]	5/5	5/5	0/5	0/5
Financial Institute [11]	5/5	5/5	2/5	1/5
GLBA-6802 [12], [15]	5/5	5/5	1/5	2/5
HIPPA-164508A2 [12], [15]	5/5	5/5	1/5	0/5
HIPPA-164508A3 [12], [15]	5/5	5/5	4/5	4/5
Dynamic Separation of Duty [11]	2/5	0/5	2/5	0/5
Total:	37/40 (92%)	31/40 (76%)	18/40 (45%)	11/40 (27%)

Our evaluation confirms that SysLite can learn the target formula or an equivalent one for each of the five random sample sets in almost all cases. The only exceptions are the Dynamic Separation of Duty formula, for which SysLite generates two formulas neither of which is equivalent to the target formula, and the Chinese Wall Policy formula, for which it generates one formula and only for the sample set of size 1250. To put things in perspective, however, note that since the Chinese Wall Policy formula has two variables and traces have length 10, a set of 1250 traces covers just 0.1% of the set of all possible 4^{10} traces. Remarkably, SysLite is able to learn the right formula with much smaller sample sets in all the other cases, with perfect precision, recall, and F1 scores.

Looking at the baseline approach, it performs gracefully with a few simple target formulas such as Bank Transaction Policy and HIPAA-164508A3. However, it cannot synthesize any candidates for the Secure File target formula. Moreover, its synthesized formulas for HIPPA-164508A2, Dynamic Separation of Duty, and Chinese Wall Policy are not equivalent to the target. See Arif et al. [29] for detailed results.

Scalability. The training results for case study I are shown in Figure 1. The X-axis of the graph represents the different training set sizes: 80% of 50, 100, 250, 500, and 1250, while the Y-axis (in log-scale) represents the training time in seconds. Cross validation times are not shown because they are uniform and negligible. The horizontal red line on the top of the graph represents the timeout (3600 seconds). In the graph, we only show results for the 3 target formulas for which the SAT-method performs best. See [29] for complete results.

In our evaluation, SysLite was able to generate results for almost all combinations of target formula and training set size while exhibiting an average 60x speedup over the baseline. The exception, already mentioned, is the Dynamic Separation of Duty formula where it timed-out on the training sets with more than 100 traces. This is likely due to the large size of the formulas to be synthesized which requires

[3] In this case study, we did not observe any ties after cross-validation.

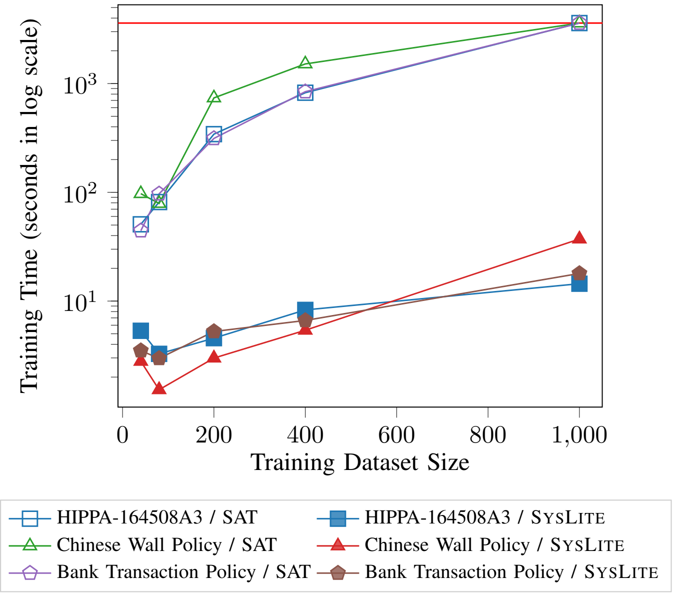

Fig. 1. Training Results of Case Study I.

SysLite to enumerate internally a very large number of terms. The baseline method was unable to generate any formula and timed-out, even for the smallest sample (of 50 traces) for the Secure File formula. For a few of the other target formulas, it failed to synthesize a candidate even for the small sample sets (of size 50 and 100). For example, in HIPPA-164508A2 policy it failed to synthesize any formula for sample size larger than 50 traces; for the Dynamic Separation of Duty and Financial Institute it was unable to deal with sample sets with more than 100 traces. These scalability problems are the main cause of its low formula-quality scores (shown in Table III) and low statistical measures scores (not shown).

D. Case Study II: 4G LTE Attack Signature Generation

Our second case study focused on synthesizing *attack signatures*, represented as PLTL formulas, for cellular networks from a set of *benign* (i.e., positive) and *attack* (i.e., negative) traces. Once again, we considered the scalability and effectiveness of SysLite versus the SAT-based baseline. The choice of this application domain was motivated by the vital role cellular networks play in a modern nation's infrastructure, which makes them a frequent target for malicious attacks [6], [16]–[18], [41], [42].

As with any protocol, the cellular network protocol allows only specific orderings of messages (packets) sent or received by a cellular device, and predicates over their payload (e.g., the sequence number is in a range). For a given type of attack, the synthesized attack signature is expected to be satisfied, ideally, by *all and only* the benign protocol executions, those not containing an attack. This way, one can deploy a runtime monitor [43] for each attack type that checks whether the current execution violates (i.e., falsifies) the attack signature and issues an alert as soon as it detects a violation. Currently, there are no mechanisms that can achieve this goal efficiently. Being able to automatically synthesize effective attack signatures is the natural first step towards that. In light of this, our case study focused on 11 known, representative attacks that are detectable from the vantage point of a cellular device (see Table IV). These attacks target weaknesses of the cellular network protocol in the Non-Access Stratum (NAS) layer, responsible for communication between a cellular device and the core network, and the Radio Resource Control (RRC) layer, responsible for the communication between a device and the base station [6], [16]–[20]. While other attacks exist [7], [16]–[18], [44]–[51], they are not detectable from a device's point of view and thus are not relevant to our case study.

TABLE IV. Table summarizing the attacks used for evaluation of 4G LTE Attack Signature Generation. (● = NAS Protocol Layer, ○ = RRC Protocol Layer)

Name of Attack	SysLite-synthesized Attack Signature	PL
Numb Attack [6]	$\Box_f(\mathsf{authentication\_reject} \Rightarrow \ominus(\mathsf{authentication\_response}))$	●
Authentication Failure [6]	$\Box_f(\neg(\mathsf{authentication\_failure}))$	●
IMSI Cracking Attack Against 4G [16]	$\Box_f(\neg(\mathsf{paging\_IMSI\_and\_TMSI}))$	●
IMSI Catching [16]	$\Box_f(\neg(\mathsf{identity\_request\_IMSI}))$	●
Measurement Report [17]	$\Box_f(\mathsf{measurementReport} \Rightarrow (\neg(\mathsf{rrcConnectionSetup})\,\mathcal{S}\,\mathsf{securityModeComplete}))$	○
RLF Report [17]	$\Box_f(\mathsf{ueInformationResponse} \Rightarrow (\neg(\mathsf{rrcConnectionRequest})\,\mathcal{S}\,\mathsf{securityModeCommand}))$	○
AKA Bypass Attack [18]	$\Box_f(\mathsf{rrcConnectionReconfiguration} \Rightarrow (\neg(\mathsf{rrcConnectionSetupComplete})\,\mathcal{S}\,\mathsf{securityModeCommand}))$	○
Malformed Identity Request [19]	$\Box_f(\neg(\mathsf{identity\_request\_malformed}))$	●
Null Encryption Chosen by MME	$\Box_f(\neg(\mathsf{MME\_null\_encryption\_chosen}))$	●
EMM Information Spoofing [20]	$\Box_f(\neg(\mathsf{emm\_information\_insecure}))$	○
Paging with IMSI [16]	$\Box_f(\neg(\mathsf{paging\_IMSI} \vee \mathsf{paging\_IMSI\_and\_TMSI}))$	●

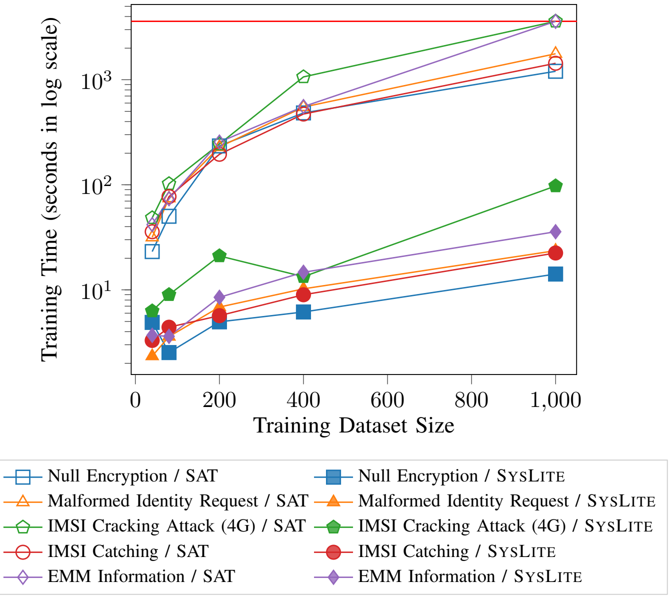

Fig. 2. Training Results of Case Study II.

Trace gathering. We now discuss how we gathered benign traces and generated attack traces through testbed experiments.

Benign Traces: We collected benign traces by random sampling traces from a crowd-sourced platform to which users world-wide submit their cellular network traces through an Android app called MobileInsight [52]. Our collected traces include 1892 NAS layer traces containing about 52K messages and 2045 RRC layer traces containing about 1.5M messages. We cleaned up the traces so that each contained 100 states as this is sufficient for the attacks we considered.

Malicious Traces: To collect malicious traces, we first implemented each attack and its variants using srsLTE [53] and software-defined radios in a testbed. srsLTE is an open-source cellular network stack which permits the modification of different components of the network. We then extracted the attack traces with SCAT [54], a desktop application capable of extracting 4G LTE modem traffic exposed by certain devices through a USB interface. Finally, we inserted one or more copies of the attack traces at arbitrary positions of some arbitrarily chosen benign traces to obtain our set of malicious traces. The latter is meant to mimic real-world scenarios in which attacks span a few sessions of the protocol.

Quality of the synthesized attack signatures. In this case study, our quality criteria were signature succinctness and correctness in capturing the attack. We consider an attack signature to be succinct if it can concisely capture the attack's root cause without including superfluous events (e.g., messages received/sent) or conditions (e.g., predicates over message payload). Visual inspection of the signatures returned by SysLite and the baseline shows that those generated by SysLite, shown in Table IV, are more succinct.

Looking at correctness, all the attack signatures synthesized by either the SAT-based baseline or SysLite for the NAS layer have a perfect (100%) precision, recall and F1 on the testing set. However, the baseline is able to synthesize signature only with samples of small size. For all the RRC layer attacks, SysLite is able to score perfectly on the test dataset based on the statistical measures. The baseline, however, does not achieve a 100% precision, recall, and F1 score as it cannot synthesize any signature for the Measurement Report attack. We have manually vetted the synthesized attack signatures by both SysLite and the baseline based on our expertise in cellular security and observed that the generated signatures correctly identified (i.e., rejected) traces containing attacks.

Scalability. The scalability results for Case Study II are shown in Figure 2. The graph's X-axis shows the sizes of the different training sets we used whereas the Y-axis (in log-scale) reports the corresponding training time in seconds. The timeout value is shown as a red horizontal line. For ease of exposition, we show only the training results for 3 NAS and 2 RRC layers attacks. For the rest of attacks, the results follow a similar trend. See Arif et al. [29] for complete results.

We conjecture that the performance of the baseline is comparable with that of SysLite when learning attack signatures on the NAS protocol layer because it induces attacks spanning only a single protocol session. Thus, the patterns are relatively easier to learn. On the other hand, for the RRC layer attacks, the sequences of attack steps can be complex and spread over multiple sessions, thus making it challenging to learn (see [29]). Indeed, the baseline timed out more frequently while synthesizing multi-session attacks from RRC traffic. In case of the Measurement Report attack, the baseline timed out for all sample sizes and did not yield any signature. In contrast, and as illustrated in Figure 2, we observed that the SysLite is scalable and efficient in synthesizing multi-session attacks signatures, exhibiting on average a 28x speedup over the baseline. We stress that scalability is essential in this context to promptly generate attack signatures for newly discovered attacks before attackers can cause substantial damage.

VI. Related Work

The problem of Learning LTL formulas consistent with a given set of traces is an instance of the so called language learning from the informant problem [1], [5], [21]. Unlike prior approaches for Signal Temporal Logic (STL) formula learning [55]–[58] and LTL specification mining [59], [60], these exact learning methods do not require any user-provided templates. Alternatively, for attack monitor synthesis, one can envision using active/passive learning to learn a regular language representation (e.g., DFA [61]–[64], NFA [65], alternating automaton [66]) of attack signatures. Monitoring such regular language representations with language recognizers (e.g., DFA) may require exponentially more states than PLTL.

Also, these regular language learning methods are not scalable as an automaton requires an explicit state representation of the behavior to-be-learned. LTL formulas, in contrast, are an efficient alternative for capturing behavior as it offers a more succinct and interpretable representation. Efforts on synthesis of reactive synthesis design [67] and counterexample-guided inductive synthesis [68] are complementary to the approaches we discuss here.

VII. Conclusion

We have presented an efficient approach for synthesizing PLTL formulas from a set of finite traces. The approach reduces the problem to a bit-vector function synthesis problem and then uses an enhanced version of the cvc4sy SyGuS solver to solve the latter. The reduction to bit-vector function synthesis proves critical for performance not only because cvc4sy implements specific optimization for bit-vectors but also because it allows us to efficiently express the requirements capturing the consistency of the solution with the samples. The conventional wisdom that SyGuS solvers are more efficient for problems over natively supported theories compared to reductions to other SMT theories (such as algebraic datatypes) or to SAT is corroborated by our experimental evaluation.

Possible directions for future work include understanding the impact of grammar representation (i.e., which temporal operators to be included in the syntactic specification of the SyGuS problem) in the efficiency of PLTL formula synthesis as well as extending the current approach to synthesizing past, propositional metric temporal logic.

Acknowledgments

This work was partially funded by DARPA YFA contract no. D19AP00039 and by NSF grant no. 1656926. The authors are grateful to Daniel Nader, Ivan Gravan, Rajarshi Roy, and the anonymous reviewers for their helpful feedback.

References

[1] Colin De la Higuera. *Grammatical inference: learning automata and grammars*. Cambridge University Press, 2010.

[2] Nader H Bshouty. Exact learning via the monotone theory. In *Proceedings of 1993 IEEE 34th Annual Foundations of Computer Science*, pages 302–311. IEEE, 1993.

[3] Scott D. Stoller and Thang Bui. Mining hierarchical temporal roles with multiple metrics. *Journal of Computer Security*, 26(1):121–142, 2018.

[4] Zhongyuan Xu and Scott D. Stoller. Mining attribute-based access control policies from logs. In Vijay Atluri and Guenther Pernul, editors, *Proceedings of the 28th Annual IFIP WG 11.3 Working Conference on Data and Applications Security and Privacy (DBSec 2014)*, volume 8566 of *Lecture Notes in Computer Science*, pages 276–291. Springer-Verlag, 2014.

[5] Daniel Neider and Ivan Gavran. Learning linear temporal properties. In *2018 Formal Methods in Computer Aided Design (FMCAD)*, pages 1–10. IEEE, 2018.

[6] Syed Rafiul Hussain, Omar Chowdhury, Shagufta Mehnaz, and Elisa Bertino. LTEInspector: A Systematic Approach for Adversarial Testing of 4G LTE. In *25th Annual Network and Distributed System Security Symposium, NDSS, San Diego, CA, USA, February 18-21*, 2018.

[7] Syed Rafiul Hussain, Mitziu Echeverria, Imtiaz Karim, Omar Chowdhury, and Elisa Bertino. 5greasoner: A property-directed security and privacy analysis framework for 5g cellular network protocol. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 669–684, 2019.

[8] Klaus Havelund and Grigore Roşu. Synthesizing monitors for safety properties. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 342–356. Springer, 2002.

[9] Andreas Bauer, Martin Leucker, and Christian Schallhart. Comparing ltl semantics for runtime verification. *Journal of Logic and Computation*, 20(3):651–674, 2010.

[10] Shaohui Wang, Anaheed Ayoub, Oleg Sokolsky, and Insup Lee. Runtime verification of traces under recording uncertainty. In *International Conference on Runtime Verification*, pages 442–456. Springer, 2011.

[11] David Basin, Felix Klaedtke, and Samuel Müller. Monitoring security policies with metric first-order temporal logic. In *Proceedings of the 15th ACM Symposium on Access Control Models and Technologies*, SACMAT '10, page 23–34, New York, NY, USA, 2010. Association for Computing Machinery.

[12] Omar Chowdhury, Limin Jia, Deepak Garg, and Anupam Datta. Temporal mode-checking for runtime monitoring of privacy policies. In Armin Biere and Roderick Bloem, editors, *Computer Aided Verification*, pages 131–149, Cham, 2014. Springer International Publishing.

[13] Deepak Garg, Limin Jia, and Anupam Datta. Policy auditing over incomplete logs: Theory, implementation and applications. In *Proceedings of the 18th ACM Conference on Computer and Communications Security*, CCS '11, page 151–162, New York, NY, USA, 2011. Association for Computing Machinery.

[14] Omar Chowdhury, Andreas Gampe, Jianwei Niu, Jeffery von Ronne, Jared Bennatt, Anupam Datta, Limin Jia, and William H. Winsborough. Privacy promises that can be kept: A policy analysis method with application to the hipaa privacy rule. In *Proceedings of the 18th ACM Symposium on Access Control Models and Technologies*, SACMAT '13, page 3–14, New York, NY, USA, 2013. Association for Computing Machinery.

[15] Henry DeYoung, Deepak Garg, Limin Jia, Dilsun Kaynar, and Anupam Datta. 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, page 73–82, New York, NY, USA, 2010. Association for Computing Machinery.

[16] Syed Rafiul Hussain, Mitziu Echeverria, Omar Chowdhury, Ninghui Li, and Elisa Bertino. Privacy Attacks to the 4G and 5G Cellular Paging Protocols Using Side Channel Information. In *26th Annual Network and Distributed System Security Symposium, NDSS, San Diego, CA, USA, February 24-27, 2019*, 2019.

[17] Altaf Shaik, Jean-Pierre Seifert, Ravishankar Borgaonkar, N. Asokan, and Valtteri Niemi. Practical Attacks Against Privacy and Availability in 4G/LTE Mobile Communication Systems. In *23nd Annual Network and Distributed System Security Symposium, NDSS, San Diego, CA, USA, February 21-24*, 2016.

[18] Hongil Kim, Jiho Lee, Lee Eunkyu, and Yongdae Kim. Touching the Untouchables: Dynamic Security Analysis of the LTE Control Plane. In *Proceedings of the IEEE Symposium on Security & Privacy (SP)*. IEEE, 2019.

[19] Benoit Michau and Christophe Devine. How to Not Break LTE Crypto. In *ANSSI Symposium sur la sécurité des technologies de l'information et des communications (SSTIC)*, 2016.

[20] Shinjo Park, Altaf Shaik, Ravishankar Borgaonkar, and Jean-Pierre Seifert. White Rabbit in Mobile: Effect of Unsecured Clock Source in Smartphones. In *Proceedings of the 6th Workshop on Security and Privacy in Smartphones and Mobile Devices*, pages 13–21. ACM, 2016.

[21] Heinz Riener. Exact synthesis of LTL properties from traces. In *2019 Forum for Specification and Design Languages (FDL)*, pages 1–6. IEEE, 2019.

[22] Andrew Reynolds, Cesare Tinelli, Amit Goel, and Sava Krstić. Finite model finding in smt. In *International Conference on Computer Aided Verification*, pages 640–655. Springer, 2013.

[23] Andrew Reynolds, Jasmin Christian Blanchette, Simon Cruanes, and Cesare Tinelli. Model finding for recursive functions in smt. In *International Joint Conference on Automated Reasoning*, pages 133–151. Springer, 2016.

[24] Rajeev Alur, Rastislav Bodik, Garvit Juniwal, Milo MK Martin, Mukund Raghothaman, Sanjit A Seshia, Rishabh Singh, Armando Solar-Lezama, Emina Torlak, and Abhishek Udupa. *Syntax-guided synthesis*. IEEE, 2013.

[25] Andrew Reynolds, Haniel Barbosa, Andres Nötzli, Clark Barrett, and Cesare Tinelli. cvc 4 sy: smart and fast term enumeration for syntax-guided synthesis. In *International Conference on Computer Aided Verification*, pages 74–83. Springer, 2019.

[26] Herbert B. Enderton. *A mathematical introduction to logic*. Academic Press, 2 edition, 2001.

[27] Clark Barrett, Aaron Stump, and Cesare Tinelli. The SMT-LIB Standard: Version 2.0. In A. Gupta and D. Kroening, editors, *Proceedings of the 8th International Workshop on Satisfiability Modulo Theories (Edinburgh, UK)*, 2010.

[28] S. Kripke. Semantical Considerations on Modal Logic. *Acta Phil. Fennica*, 16:83–94, 1963.

[29] M. Fareed Arif, Daniel Larraz, Mitziu Echeverria, Andrew Reynolds, Omar Chowdhury, and Cesare Tinelli. SYSLITE: syntax-guided synthesis of PLTL formulas from finite traces. Technical report, Department of Computer Science, The University of Iowa, August 2020. Available at https://github.com/CLC-UIowa/SySLite.

[30] Luciano Baresi, Mohammad Mehdi Pourhashem Kallehbasti, and Matteo Rossi. Efficient scalable verification of LTL specifications. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, volume 1, pages 711–721. IEEE, 2015.

[31] Clark Barrett, Christopher L. Conway, Morgan Deters, Liana Hadarean, Dejan Jovanovic, Tim King, Andrew Reynolds, and Cesare Tinelli. CVC4. In *Computer Aided Verification - 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings*, pages 171–177, 2011.

[32] Andrew Reynolds, Morgan Deters, Viktor Kuncak, Cesare Tinelli, and Clark W. Barrett. Counterexample-guided quantifier instantiation for synthesis in SMT. In *Computer Aided Verification - 27th International Conference, CAV 2015, San Francisco, CA, USA, July 18-24, 2015, Proceedings, Part II*, pages 198–216, 2015.

[33] Andrew Reynolds, Haniel Barbosa, Andres Nötzli, Clark W. Barrett, and Cesare Tinelli. cvc4sy: Smart and fast term enumeration for syntax-guided synthesis. In *Computer Aided Verification - 31st International Conference, CAV 2019, New York City, NY, USA, July 15-18, 2019, Proceedings, Part II*, pages 74–83, 2019.

[34] Armando Solar-Lezama, Liviu Tancau, Rastislav Bodik, Sanjit Seshia, and Vijay Saraswat. Combinatorial sketching for finite programs. *SIGPLAN Not.*, 41(11):404–415, October 2006.

[35] Andres Nötzli, Andrew Reynolds, Haniel Barbosa, Aina Niemetz, Mathias Preiner, Clark W. Barrett, and Cesare Tinelli. Syntax-guided rewrite rule enumeration for SMT solvers. In *Theory and Applications of Satisfiability Testing - SAT 2019 - 22nd International Conference, SAT*

2019, Lisbon, Portugal, July 9-12, 2019, Proceedings, pages 279–297, 2019.

[36] José Oncina and Pedro Garcia. Inferring regular languages in polynomial updated time. In *Pattern recognition and image analysis: selected papers from the IVth Spanish Symposium*, pages 49–61. World Scientific, 1992.

[37] Shufang Zhu, Geguang Pu, and Moshe Y. Vardi. First-order vs. second-order encodings for ltlf-to-automata translation, 2019.

[38] Ashok K. Chandra, Dexter C. Kozen, and Larry J. Stockmeyer. Alternation. *J. ACM*, 28(1):114–133, January 1981.

[39] Giuseppe De Giacomo and Moshe Y. Vardi. Linear temporal logic and linear dynamic logic on finite traces. In *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*, IJCAI '13, page 854–860. AAAI Press, 2013.

[40] Yih-Kuen Tsay, Yu-Fang Chen, Ming-Hsien Tsai, Kang-Nien Wu, and Wen-Chin Chan. Goal: A graphical tool for manipulating büchi automata and temporal formulae. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 466–471. Springer, 2007.

[41] Adrian Dabrowski, Nicola Pianta, Thomas Klepp, Martin Mulazzani, and Edgar Weippl. Imsi-catch me if you can: Imsi-catcher-catchers. In *Proceedings of the 30th annual computer security applications Conference*, pages 246–255, 2014.

[42] Syed Rafiul Hussain, Mitziu Echeverria, Ankush Singla, Omar Chowdhury, and Elisa Bertino. Insecure connection bootstrapping in cellular networks: the root of all evil. In *Proceedings of the 12th Conference on Security and Privacy in Wireless and Mobile Networks*, pages 1–11, 2019.

[43] Viktor Schuppan and Armin Biere. Shortest counterexamples for symbolic model checking of ltl with past. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 493–509. Springer, 2005.

[44] Iosif Androulidakis. Intercepting mobile phone calls and short messages using a gsm tester. In *International Conference on Computer Networks*, pages 281–288. Springer, 2011.

[45] Myrto Arapinis, Loretta Mancini, Eike Ritter, Mark Ryan, Nico Golde, Kevin Redon, and Ravishankar Borgaonkar. New privacy issues in mobile telephony: fix and verification. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 205–216, 2012.

[46] Byeongdo Hong, Sangwook Bae, and Yongdae Kim. Guti reallocation demystified: Cellular location tracking with changing temporary identifier. In *NDSS*, 2018.

[47] Katharina Kohls, David Rupprecht, Thorsten Holz, and Christina Pöpper. Lost traffic encryption: fingerprinting lte/4g traffic on layer two. In *Proceedings of the 12th Conference on Security and Privacy in Wireless and Mobile Networks*, pages 249–260, 2019.

[48] Denis Foo Kune, John Koelndorfer, Nicholas Hopper, and Yongdae Kim. Location leaks on the gsm air interface. *ISOC NDSS (Feb 2012)*, 2012.

[49] Ulrike Meyer and Susanne Wetzel. A man-in-the-middle attack on umts. In *Proceedings of the 3rd ACM workshop on Wireless security*, pages 90–97, 2004.

[50] David Rupprecht, Katharina Kohls, Thorsten Holz, and Christina Pöpper. Breaking lte on layer two. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 1121–1136. IEEE, 2019.

[51] David Rupprecht, Katharina Kohls, Thorsten Holz, and Christina Pöpper. Imp4gt: Impersonation attacks in 4g networks. In *ISOC Network and Distributed System Security Symposium (NDSS)*. ISOC, February 2020.

[52] Yuanjie Li, Chunyi Peng, Zengwen Yuan, Jiayao Li, Haotian Deng, and Tao Wang. Mobileinsight: Extracting and analyzing cellular network information on smartphones. In *Proceedings of the 22nd Annual International Conference on Mobile Computing and Networking*, MobiCom '16, pages 202–215, New York, NY, USA, 2016. ACM.

[53] Ismael Gomez-Miguelez, Andres Garcia-Saavedra, Paul D Sutton, Pablo Serrano, Cristina Cano, and Doug J Leith. srsLTE: An Open-source Platform for LTE Evolution and Experimentation. In *Proceedings of the Tenth ACM International Workshop on Wireless Network Testbeds, Experimental Evaluation, and Characterization*, pages 25–32, 2016.

[54] Byeongdo Hong, Shinjo Park, Hongil Kim, Dongkwan Kim, Hyunwook Hong, Hyunwoo Choi, Jean-Pierre Seifert, Sung-Ju Lee, and Yongdae Kim. Peeking over the cellular walled gardens-a method for closed network diagnosis. *IEEE Transactions on Mobile Computing*, 17(10):2366–2380, 2018.

[55] Eugene Asarin, Alexandre Donzé, Oded Maler, and Dejan Nickovic. Parametric identification of temporal properties. In *International Conference on Runtime Verification*, pages 147–160. Springer, 2011.

[56] Zhaodan Kong, Austin Jones, and Calin Belta. Temporal logics for learning and detection of anomalous behavior. *IEEE Transactions on Automatic Control*, 62(3):1210–1222, 2016.

[57] Prashant Vaidyanathan, Rachael Ivison, Giuseppe Bombara, Nicholas A DeLateur, Ron Weiss, Douglas Densmore, and Calin Belta. Grid-based temporal logic inference. In *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, pages 5354–5359. IEEE, 2017.

[58] Ezio Bartocci, Luca Bortolussi, and Guido Sanguinetti. Learning temporal logical properties discriminating ecg models of cardiac arrhytmias. *arXiv preprint arXiv:1312.7523*, 2013.

[59] Wenchao Li, Lili Dworkin, and Sanjit A Seshia. Mining assumptions for synthesis. In *Ninth ACM/IEEE International Conference on Formal Methods and Models for Codesign (MEMPCODE2011)*, pages 43–50. IEEE, 2011.

[60] Caroline Lemieux, Dennis Park, and Ivan Beschastnikh. General ltl specification mining (t). In *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 81–92. IEEE, 2015.

[61] Dana Angluin. Learning regular sets from queries and counterexamples. *Information and computation*, 75(2):87–106, 1987.

[62] Georgios Giantamidis, Stavros Tripakis, and Stylianos Basagiannis. Learning moore machines from input–output traces. *International Journal on Software Tools for Technology Transfer*, pages 1–29, 2019.

[63] Marijn J. H. Heule and Sicco Verwer. Exact dfa identification using sat solvers. In *Proceedings of the 10th International Colloquium Conference on Grammatical Inference: Theoretical Results and Applications*, page 66–79, Berlin, Heidelberg, 2010. Springer-Verlag.

[64] Daniel Neider and Nils Jansen. Regular model checking using solver technologies and automata learning. In *NASA Formal Methods Symposium*, pages 16–31. Springer, 2013.

[65] Benedikt Bollig, Peter Habermehl, Carsten Kern, and Martin Leucker. Angluin-style learning of nfa. In *Twenty-First International Joint Conference on Artificial Intelligence*, 2009.

[66] Dana Angluin, Sarah Eisenstat, and Dana Fisman. Learning regular languages via alternating automata. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

[67] Nir Piterman, Amir Pnueli, and Yaniv Sa'ar. Synthesis of reactive (1) designs. In *International Workshop on Verification, Model Checking, and Abstract Interpretation*, pages 364–380. Springer, 2006.

[68] Rajeev Alur, Rishabh Singh, Dana Fisman, and Armando Solar-Lezama. Search-based program synthesis. *Communications of the ACM*, 61(12):84–93, 2018.