MINTS: Unsupervised Temporal Specifications Miner

Pradeep K. Mahato
Department of Computer Science
The University of British Columbia
BC, Canada
pradeep.mahato@alumni.ubc.ca

Apurva Narayan
Department of Computer Science
The University of British Columbia
BC, Canada
apurva.narayan@ubc.ca

***Abstract*—Specifications for software systems are quite often missing or are obsolete given the evolutionary nature of these systems. Lack of precise software specifications makes the task of debugging and detecting a malfunction of system behavior challenging. Prior works have primarily focused on extracting system specifications in the form of template-based mining frameworks or interactive simulation models. In safety-critical systems where the time of occurrence of events is of prime importance extracting specifications with a quantitative notion of time seems a daunting task.**

This work presents an unsupervised approach to mine timed temporal properties in the form of deterministic finite state machines with a custom-designed trie data structure. Our framework, MINTS learns dominant system specifications from their system traces that are represented as a timed deterministic finite state machine. MINTS is shown to be sound and complete. MINTS scalability and correctness is validated using real-world industry strength traces.

***Index Terms*—Real Time Systems, Specification Mining, Temporal Properties**

I. Introduction

Software programs are shipped with documentation that describes the feature and functionality of the system. A well-documented software contains a detailed description of each module within the software, interaction among the modules, and appropriate use of the modules. Clear and well-defined software documentation help during implementation, refactoring, testing, debugging, and can also be looked back for implementation details in the future [1]. However, quite often [2]–[5] software documentation is not updated periodically making them non-traceable to the current changes. They become stale with incorrect information. Primary reasons involve low enthusiasm among the developers with the documentation and the requirement of huge time and resources to document changes [6]. With the progression of time, missing implementation details create obstructions for maintenance engineers and increases functional incorrectness for complex tasks [7]. Missing or incorrect software documentation becomes even more challenging and important in the safety-critical system where the time of occurrence of events is of prime importance.

Temporal specification mining is an extensively studied domain [8], [9]. Temporal specification mining techniques extract interesting properties with the abstract notion of time from system traces. Extraction of temporal properties from system traces has been widely studied in the past [9]–[13]. System traces record system behavior and interactions with the user. Recently, the notion of mining temporal specification from system traces has received significant attention [10], [12], [13]. More specifically, template-based specification mining has gained considerable prominence [12], [14]. In template-based specification mining, a user-defined template is fed into the model as input that matches similar patterns from the system traces. Specification mining is useful in software engineering for verification [15], generalization of system behavior [1], [10], anomaly detection [12], etc.

Regular expressions are a powerful tools for pattern matching and can be used for template-based specification mining [11]. They can be represented as state machines that are simple and easy to understand. For example, Figure 1 presents a state transition diagram of a coffee vending machine[1] that can be expressed as *Coin* Push* Coin+* where *Coin* and *Push* are events on the machine. Finite Automatons are an abstract representation of Finite State Machines [16]. We use the term Finite State Machine (FSM) and Finite Automata (FA) interchangeably in this paper.

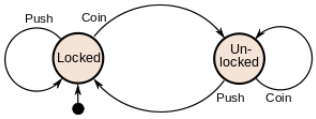

Fig. 1: Finite State Representation of Coin Vending Machine

Timed Automata (TA) [17] is a Finite Automata extended with clock variables and added constraints on the state transitions. State transitions are restricted with time bounds on the events. As a result, the system state changes only when the occurrence of the event is satisfied by the time-bound. TA is primarily used in the field of formal methods and software engineering such as model-testing [18], system verification [15], testing control flow in communication systems [19], computer-aided process engineering [20], etc. TA is also used in safety-critical systems such as medical devices, automobiles, safety

[1]Finite State Machines. Brilliant.org. Retrieved 20:03, April 19, 2021, from https://brilliant.org/wiki/finite-state-machines/

equipment [18], [21], where the occurrence of an event and the timing of the event is of paramount importance. Temporal properties can be extracted from TA [10], [22].

Template-based extraction of temporal properties is difficult and tedious. They are majorly based on expert domain knowledge [23]–[25] or are restrictive due to poor customization during mining [13], [26]. Moreover, due to the evolutionary nature of software development, template-based temporal mining requires a constant update with every software modification.

We propose a novel unsupervised specification mining technique. Our framework takes a set of system traces and builds a custom timed-trie network. Less frequent patterns are filtered out and only the most recurrent patterns are retained in the final system representation. The trie network contains nodes and edges, representative of system state and transition among states respectively. Edges define the necessary event with the clock constraint on the event. Our framework can be used to extract the most frequent temporal patterns or be used as a runtime monitor to detect anomalies.

We begin by defining a trace, state machine with time-constrained event transition, and trie data structure. We then explain the construction of our custom trie along with custom pruning to filter and strengthen the model. Multiple pruning steps are applied to extract only the dominant patterns. We use association rules of support and confidence for pattern extraction.

The major contribution of our work can be summarised as follows:

- Propose our framework MINTS, to mine temporal specifications from a given system trace with time constraints.
- Present two algorithms, one to extract the overall dominant temporal patterns while the other extracts specific behavioral patterns [27].
- For real-world use cases, we present an event prediction algorithm. Our event prediction algorithm is sensitive to both time and sequence of the past event.
- We show our approach is sound, complete, and scalable

The remainder of our paper is organized as follows, Section II provides an overview of the current research in the domain of temporal specification mining. Section III briefly introduces few notations necessary for the understanding of our framework. Section IV explains the various components and working of the framework followed by discussion and experiment in Section V and Section VI respectively. Finally, we conclude with some future work in Section VII.

II. Literature Review

Modeling system using finite state machines is a well-studied topic [11], [24]. In [28], Angluin presented an approach to learn system behavior. It uses a series of queries and counterexample to infer the hidden automaton. It follows a pattern of minimally adequate teacher where the learner tries to find the abstract representation of the system that the teacher is only aware of. Regular language expresses these abstract representations in a relatively simple form [12]. Few recent study [12], [14] used variations of regular languages to model system behavior. [10], [11], uses pattern matching while [14], [29] study the system behavior in the form of regular expressions. In [30], Vaandrager uses a Mealy Machine [31] as a minimally adequate teacher along with a MAPPER that provides a deterministic output for any given input. The primary role of the MAPPER is to perform a translation between input and output states. Similar approaches have also been proposed in [29], [32].

Alternatively, Input-Output Automata (IOAutomata) are used to model the system behavior of a distributed network. IOAutomata uses actions such as input, output, and internal activities to establish the relationship between different components. In [33], the authors use IOAutomata as a teacher and trains the Mealy machine to learn the target black-box model's workflow.

Finite automata can be naturally extended using Timed Automata. Timed Automata (TA) include the clock constraint over the transition edge between each state. In [11], Alur et al. proposed the concept of timed regular expressions. Timed regular expressions are an extension of regular languages with the knowledge of time. In [23], [24], Ulus et al. extended timed regular expression to define timed derivatives. In Formal logic, derivatives are a very neat algebraic tool used to build computation models. The benefit of derivatives is in the speed for sequential computations. Since time series is a form of a continuous stream of non-decreasing discrete timed data, extending derivatives with a set of timed relations is novel [23]. MONTRE [25] uses timed derivatives to perform real-time pattern matching with time bounds. Albeit coming up with pre-defined timed relations is challenging.

In [26], the authors propose QMine, a framework for mining quantitative properties from system traces. Quantitative properties are associated with events and numerical measurements in the system traces. These properties can be used to not only understand system behavior but also assists in tasks such as anomaly detection, runtime monitoring, intrusion detection etc. Nevertheless, quantitative regular expressions do not consider timing constraints explicitly. Cutulenco et al. in their work [13] proposed an approach to mine system properties using user-defined templates in the form of timed regular expressions. Yet due to the rapid expansion in the software development lifecycle the necessity to update these templates becomes tedious.

In [34], the authors uses timed k-tail (TkT) to create a timed automaton. A set of traces are collected from a software system and for each trace, an automaton is generated. The automaton uses a relative clock for the clock constraint function and the final step involves merging all the automatons. The clock constraint is then generalized for the merged automaton. The final timed automaton represents the states in the software system. TkT has promising experimental results yet has high time complexity due to dependency on multiple traces. In this work, we present an approach to mine temporal properties with no apriori knowledge about the system behavior. Our approach scales linearly with trace size.

III. Preliminaries

Finite State Machine (FSM) [29] are used to model system interactions among various sub-components. System behavior is generally modeled in terms of a directed graph. $G : \{V, E\}$ where system states are represented by vertices V while edges E represents the relationship between the two connecting states. FSM explains complex system behaviors in the form of explainable graphical representations thereby assisting in troubleshooting for bugs, faster onboarding of new developers, and identifying system specifications among a few. There are two variants of FSM, namely deterministic finite state machines (DFA) and non-deterministic finite state machines (NFA). In NFA, transitions among states are not deterministic in nature. This causes different results with the same set of inputs. However DFAs are deterministic in nature and always reach the same sequence of states. In this work, we focus only on DFA.

Timed Automata [29] are an extension of Finite State Machines with the addition of time constraint over the transition function. More formally, a one-clock timed automata contains a clock variable c along with a set of constraints $\phi_c : \top | c \square i$ where $\square = \{<, =, >, \leq, \geq\}$ and $i \in \mathbb{N}$. We use $\top$ to represent *true* while $\bot$ denotes *false* for $\{\top, \bot\} \in \mathbb{B}$. The clock is set to zero when ϕ_c evaluates to $\top$ or is left unchanged in the case of $\bot$. Timed Automata can be represented as $A = (\Sigma, Q, q_0, F, c, \delta)$, where Σ is the set of alphabets while Q is a finite set of states, $q_0 \in Q$ is the initial state, $F \in Q$ is a non-empty set of accepting states and δ is the transition function along with c as the clock valuation. The transition function (δ) can be expressed as $\delta(q_i, \alpha, c_t) \rightarrow q_j$ where $\{q_i, q_j\} \in Q, \alpha \in \Sigma$ and $c_t : \top$. $\delta(q_i, \alpha, c_t) \rightarrow q_j$ implies that *the system reaches a state q_j from q_i due to an action α within a valid clock constraint, c_t*. Δt denotes the time taken to transition from the present state to the next state. We consider a clock constraint to be valid if the Δt is within the clock interval $[m, n]$ where $\{m, n\} \in \mathbb{N}$.

A path defined as $\Pi : q_0 \xrightarrow{(\alpha_1, c_1)} q_1 \xrightarrow{(\alpha_2, c_2)} q_2 \xrightarrow{(\alpha_3, c_3)} \cdots \rightarrow q_n$, implies *a transition sequence from the initial state q_0 to the final state q_n through the course of multiple valid actions within its time constraints*. A path is called accepting, if the final state is an accepting state. Paths are representative of one complete execution. A trace is a set of path, $= \{\Pi_1, \Pi_2, \cdots\}$ where $\{\Pi_1, \Pi_2, \cdots\}$ are different execution paths from multiple trials.

Clock valuation function, $c_t = c_{t-1} + \delta$ where $\delta \in \mathbb{N}$ denotes continuous non-decreasing unit of time. Moreover, clock valuation can be represented in the context of a *global clock* or a *local clock*. In the case of a *local clock* valuation, after every transition the internal clock is reset to 0. In our work, we use the clock valuation in the context of *local clock* unless otherwise stated.

Trie is a common and famous data structure mostly referred to as an n-ary tree. Each vertex contains multiple edges to its successor vertices. However, a trie data structure does not contain a closed loop. Depth (*D*) of a trie is the maximum distance from the root to its leaf nodes.

$$D = \arg max \Pi_j, \forall j \in \Pi \tag{1}$$

Trie can be used to represent a non-minimal finite state machine. We use the term *non-minimal* due to it being infinitely large for an infinitely large depth. However, with such a representation, we can learn and extract dominant patterns from the system.

A pattern refers to a unique sequence of events. Association rule mining is a process to find unique patterns, correlations among the patterns, frequent patterns, and more. Support and confidence are two common ways to measure association [35]. Support measures the probability of a certain pattern against all possible pattern lists while confidence measures the likelihood of the pattern given part of the pattern has already occurred.

IV. Methodology

Our Timed-Trie (*Trie*$_{timed}$) representation is a special type of Timed-Automaton (TA). *Trie*$_{timed}$ consists of nodes and edges representing states and transitions with clock constraints. This section explains the construction of the *Trie*$_{timed}$. There are 4 components for extracting the *Trie*$_{timed}$ from a system trace. Fig 2 presents the overall framework for our proposed approach. We discuss each component briefly in the following sections.

A. Formation of the Trie$_{timed}$

MINTS builds the *Trie*$_{timed}$ using system traces. System trace is cleaned and pre-processed before feeding as input to MINTS. Processed trace must contain events and the time associated with the events. Alphabet set consists of unique events present in the trace. Additionally, 2 hyper-parameters must also be provided into the framework, that are depth D and threshold $\mathcal{P}$. D describes the maximum depth of the generated *Trie*$_{timed}$ and threshold $\mathcal{P}$ is used for pruning of branches.

Algorithm 1 provides an overview of the steps required to build the *Trie*$_{timed}$. *Trie*$_{timed}$ is constructed progressively. At every step, a window of size d slides over the trace. For each window, the segment of the trace is used to build the *Trie*$_{timed}$. $trace_{sub_d}$ denotes a subtrace formed with a window of size d. A trace can contain multiple window instances of size d, therefore multiple instances of $trace_{sub_d}$ would be formed. Since the window slides by 1 event at a time, the segments overlap with the neighboring window. This iterative use of adjacent segments helps retain the transition of events among the window. Once all the segments of size d are processed, the framework increments the value of d. The construction begins with $d = 1$ and terminates at $d = D$. Line 3 - 5 and Algorithm 2 outlines the above-mentioned steps.

However, before incrementing the window size d, all the leaf nodes of the *Trie*$_{timed}$ are processed. The processing step involves evaluation of the clock constraint (c) and is outlined in Algorithm 3. *Trie*$_{timed}$ stores the events on the edges along with the transition time difference from the parent node to the child node. Nodes represent states that the system may

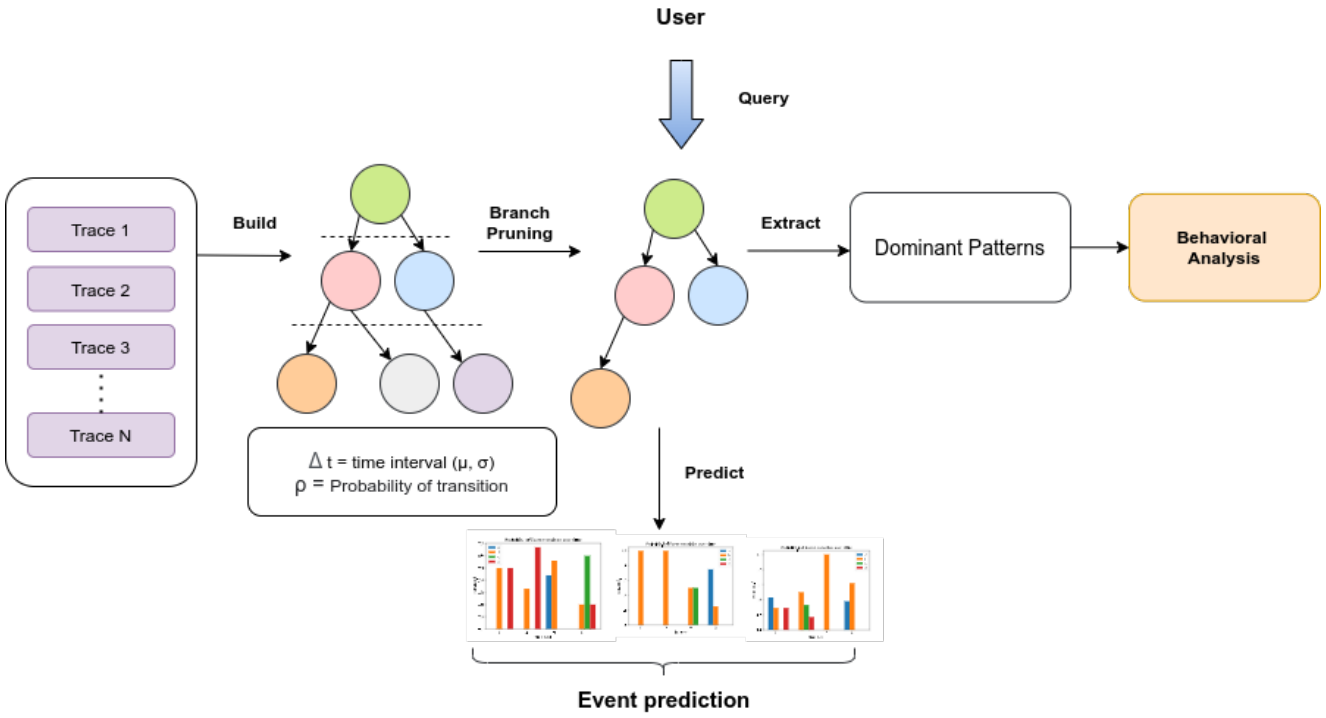

Fig. 2: MINTS Framework

Algorithm 1: buildGraph

Input: trace, D
Result: $Trie_{timed}$

1 $root_{trie} \leftarrow$ NULL
2 **foreach** d *in* $[1, D]$ **do**
3 **foreach** *Segment of trace of length* d ($trace_{sub_d}$) *in trace* **do**
4 traverseAndBuild($root_{trie}, trace_{sub_d}, 0$)
5 **end**
6 evaluateAtDepth($root_{trie}$, d)
7 **end**
8 return $root_{trie}$

undergo during execution. These records of transition time are used to find the time bounds (i.e min and max) for the clock constraint. Although MINTS uses the range defined by 2 standard deviation(σ) across mean(μ) to evaluate the bounds, users can override the evaluation function. Using such a range helps in filtering off the outliers and covers 95 % of the data set. For brevity, in this paper we refer to such a range as *partially complete range* (denoted by $\vdash\dashv$). A direct application of such time-bound is in the field of anomaly detection where events with abnormal transition time deviate significantly.

Transitions that occur outside the time bounds are dropped and only the valid transitions are used to estimate the probability of transition from parent state to the child state for the given event (Line 7) (explained in Section IV-C). It must be noted that MINTS assumes that the system spends approximately the same amount of time in the current state before transitioning to the next state.

Algorithm 2: traverseAndBuild

Input: $node_{trie}$, $trace_{sub_d}$, pos

1 **if** $pos < len(trace_{sub}) - 1$ **then**
2 Find the state node, $node_{child}$ for transition event e at pos in $trace_{sub_d}$
3 **if** Δt *satisfies constraint* c_t **then**
4 Recursively call traverseAndBuild($node_{child}, trace_{sub}, pos + 1$)
5 **else**
6 Reject traversal (to drop the outliers) and return
7 **end**
8 **else**
9 **if** $node_{trie}$ *does not contain state with transition event* e **then**
10 Create and add a state node $newNode_{trie}$ with transition event e as a child to $node_{trie}$
11 **end**
12 **end**

Figure 3 shows a sample illustration of an intermediary step involved during the construction of our timed trie network. For each transition, MINTS records all the Δt transition time steps between the events.

B. Pruning

Before proceeding with the extraction of dominant properties and the probability analysis, we ensure our model is

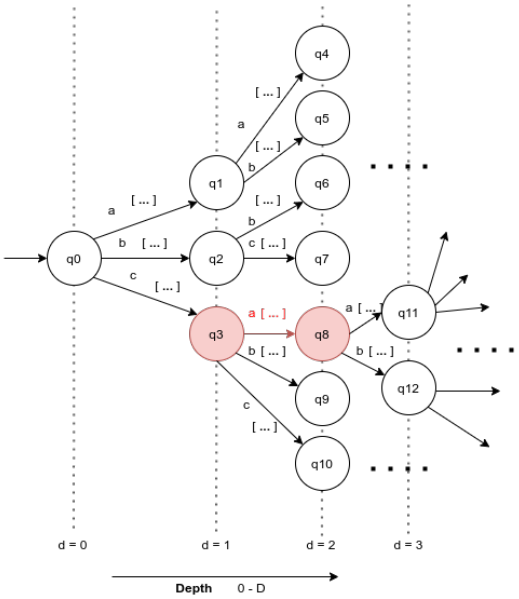

Fig. 3: figure
Sample Representation of $Trie_{timed}$

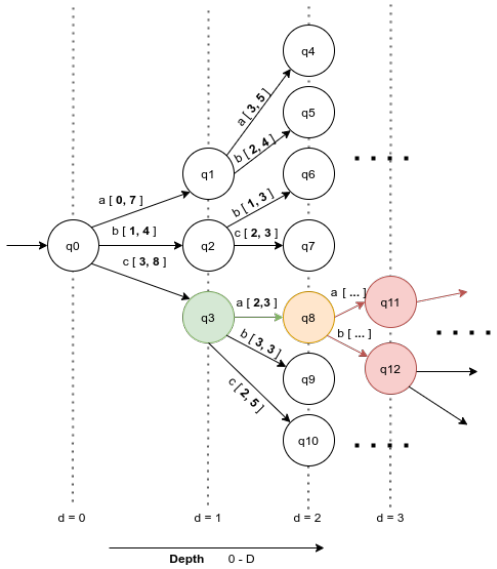

Fig. 4: figure
Clock constraint evaluated

representative of the most prevalent patterns. Therefore, we filter off all the outliers. This is accomplished by only selecting those timed transitions which fall under the empirically evaluated distribution ($\vdash\dashv$).

More formally, a normal distribution is defined and stored within each node's clock constraint c (Refer algorithm 3). The normal distribution is evaluated using all recorded transitions at this transition step and thereby the outliers are filtered off to reduce the formation of transition for the next time step. It is common to use an acceptable transition time to be within the $\vdash\dashv$ (refer line 6 in algorithm 3). We do acknowledge, that this step, in turn, could reduce precision (in terms of validity of event transitions), however, the primary objective of MINTS

Algorithm 3: evaluateAtDepth

Input: $node_{trie}$, d
1 **while** *desired depth is not reached* **do**
2 | Keep traversing down
3 **end**
4 Evaluate mean (μ) and variance (σ) using the list $\langle time - list \rangle$
5 Update the permissible range $\langle t_{min}, t_{max} \rangle$ for $node_{trie}^{depth}$ with the range containing 2 standard deviations (σ) of $\langle time - list \rangle$ ($node_{trie}^{depth} \leftarrow \langle \mu - 2\sigma, \mu + 2\sigma \rangle$)
6 Update $node_{trie}^{depth}$ with the transitions where Δt is within range $\langle t_{min}, t_{max} \rangle$
7 Compute the probability of transition ($\mathbb{P}_e^t$)

is to extract the dominant patterns.

Figure 5 shows the steps involved in measuring the clock constraint[2]. Please note, the interval constraint objective function can be changed by the user and it is not necessarily fixed on $\vdash\dashv$. Figure 4 displays the graph with clock constraint c after evaluation. Moreover, since the clock constraint is evaluated before the construction of the network at the next depth, only a partial trie network is filled with c.

C. Event Prediction

The real-world performance of a model measures its effectiveness. Given a sequence of events, many machine learning models, such as Recurrent Neural networks (RNN), Long-Short Term Memory (LSTM) [36], can be used to predict future events. Although these models have high accuracy, they lack the capability to represent the learned behavior [12]. While FSM can represent the system workflow with nodes and edges, modeling the system workflow requires domain knowledge. MINTS builds a TA that can be used for various tasks such as event prediction.

Algorithm 4: probabilityOfTransition

Input: $node_{trie}$, $trace_{sub_d}$
1 **while** *desired depth is not reached* **do**
2 | Keep traversing down with $trace_{sub_d}$
3 **end**
4 Computer the overall time-interval for all event, $e \in \Sigma$
5 **foreach** t *in* $\langle time - interval \rangle$ **do**
6 | **foreach** *event* $e \in \Sigma$ **do**
7 | | *Compute probability of transition at time* t
8 | **end**
9 **end**
10 *Return the overall* Transition Probability *for each event in the* $\langle time - interval \rangle$
11

Algorithm 4 outlines our event prediction approach. $Trie_{timed}$ stores the total count of valid transitions from each

[2]Normal Distribution Image Credit: By Jhguch at en.wikipedia, CC BY-SA 2.5 https://commons.wikimedia.org/w/index.php?curid=14524285

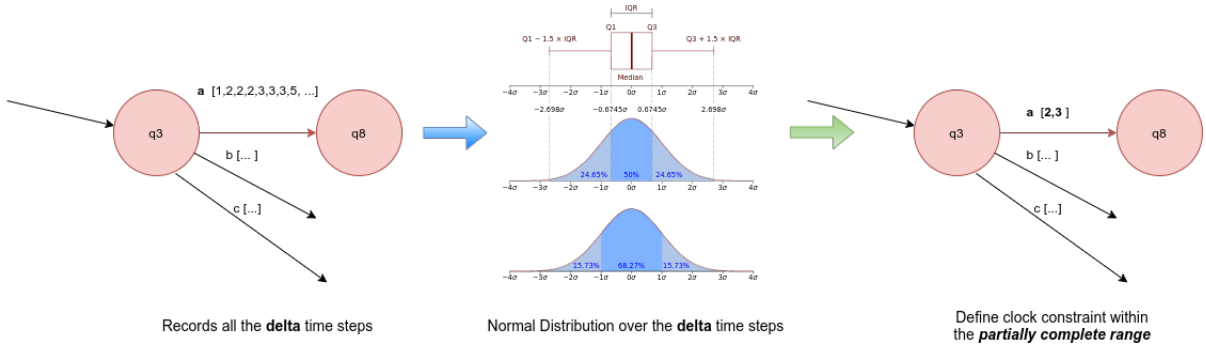

Fig. 5: Determination of clock constraint

state (except the final states). The transition probability for each event (i.e edge) is computed and stored in the node. The transition probability assists the framework to predict the next event. Moreover, since all the transition times are recorded, MINTS is capable of predicting an event with Δt time step in the future.

Figure 6 presents the output from event prediction module of the $Trie_{timed}$.

D. Dominant Property extraction

1) Additional Pruning: Although our model, at this stage, has been pruned once (refer section IV-B), we employ additional indirect pruning to avoid extraction of irrelevant properties. We classify such pruning as indirect since the approach does not explicitly prune the branches of our $Trie_{timed}$, yet, it only permits traversal over fewer paths. The term *does not explicitly prune the branches* refer to ignoring traversal over such branches and not the removal of the same.

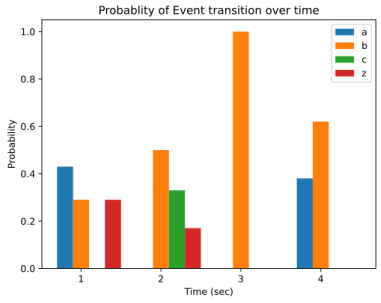

Fig. 6: Event Prediction

Our model takes a hyper-parameter, $\mathcal{P}$ that determines the threshold over the probability of transition. Only those branches, from the root node, are considered which contain a minimum $\mathbb{P}_e^t$ above $\mathcal{P}$.

$$\mathbb{P}_e^t = \arg min_j \mathcal{P}^j, \forall j \in \Pi \tag{2}$$

The method is effectively a depth-first traversal from the root node.

Algorithm 5 provides a brief overview of the steps involved in pruning the tree. The main idea is to remove all the stems

Algorithm 5: pruneTree

Input: $node_{trie}$, $\mathcal{P}$

```
foreach child of node_trie do
    if P^child_node_trie < P then
        Prune the stem i.e child
    else
        Recursively call pruneTree(child, P)
    end
end
```

which has $\mathbb{P}_e^t$ less than the user defined $\mathcal{P}$. This helps in exposing the dominant patterns.

Name	Property Template
Response	[-P]*; (P; [-S]*; S; [-P]*)*
Alternating	[-P,S]*; (P; [-P,S]*; S; [-P,S]*)*
MultiEffect	[-P,S]*; (P; [-P,S]*; S; [-P]*)*
MultiCause	[-P,S]*; (P; [-S]*; S; [-P,S]*)*

TABLE I: Temporal Properties used for mining in MINTS

2) Temporal properties: Temporal patterns have been well studied by Yang and Evans [27]. In their work, 8 most common and relevant patterns have been suggested for temporal property mining based on their behavioural importance. For demonstration, we have evaluated our algorithm with 4 such patterns while the option of extending such properties with user-defined properties has been provided. Table I presents the temporal properties that were used for the evaluation of our MINTS framework. Note that we refer to these temporal properties as behavioral patterns for the system. The time bounds for the given property template applies within the central enclosed sequence of event. P refers to the causing event while the effect S is reflected in the subsequent event. For example, for *Response* temporal pattern, a possible time-bounded expression could be $[-P]*; (P; [-S]*; S\langle x, y\rangle; [-P]*)*$ where x and y are the *start* and *end* of the time interval.

V. Discussion

1) Complexity Analysis: Algorithm 6 is used for the construction of *Trie*$_{timed}$. The program takes as input, the **trace**

along with the window length of subsequent events, d. We denote D as depth since it defines the maximum height of our $Trie_{timed}$.

Algorithm 6: Abstract Representation of $Trie_{timed}$ Generation

Input: trace, D
Result: $Trie_{timed}$
1 **foreach** *depth d in D* **do**
2 | $subtrace \leftarrow$ { set of all segments of length d from trace }
3 | **foreach** $subtrace_i$ *in* $subtrace$ **do**
4 | | traverse and build the $Trie_{timed}$
5 | **end**
6 | evaluate and prune the weak stems at depth d
7 **end**
8 return $root_{trie}$

For each depth $d \in$ [1, D], we traverse and build our $Trie_{timed}$. For a given depth d, line 2 extracts all subtraces of length d from the input **trace**. This operation takes $\mathcal{O}(n-d)$ time while traversal and building of the $Trie_{timed}$(line 3 - 5), for level d, takes constant time i.e $\mathcal{O}(1)$.

We also store Δt, i.e the relative time for the current action from the previous state. Line 6 performs pruning for stems at depth d using the $\vdash\dashv$ range. This time bound can be defined as $[\mu - x\sigma, \mu + x\sigma]$, where μ and σ are the average time step and the standard deviation over the stored list of Δt while x is a constant used to define the range, for example $x = 1$ defines a range within 1 standard deviation while 2 define our *partially complete range*($\vdash\dashv$).

This takes constant time, $\mathcal{O}(1)$ however since there are Σ^d stems at depth d, thus the entire operation takes $\mathcal{O}(\Sigma^d)$ time.

$$\begin{aligned} \text{Total time for step } d &= n - d + n - d + \Sigma^d \\ &= 2n - 2d + \Sigma^d \\ &\approx n - d + \Sigma^d \end{aligned} \tag{3}$$

Thus, the overall time complexity until depth D can be expressed as

$$\begin{aligned} \text{Total time} &= \sum_{d=1}^{D} (n - d + \Sigma^d) \\ &= \sum_{d=1}^{D} n - \sum_{d=1}^{D} d + \sum_{d=1}^{D} \Sigma^d \\ &\approx nD - D^2 + \Sigma^D \end{aligned} \tag{4}$$

Although the equation 4 contains an exponential term Σ^D, it must be noted that with each pruning step at depth d, the next iteration for step size $d+1$ would skip traversing those stems that were already being pruned at step d. Moreover, a large value of D does increase the uncertainty of the learned behavior. However, the worst-case complexity do stay at $\mathcal{O}(nD - D^2 + \Sigma^D)$.

2) Feature Comparison: Table II provides a brief overview of few specification mining algorithms[3]. UPPAAL-SMC [37] (an extension of UPPAAL) is a Statistical Model Checking tool and is widely used for real-time verification of systems. In UPPAAL-SMC, the real-time system must be modeled before performing specification verification. Linear-time Temporal Logic (LTL) is a propositional logic modeling approach with timed modalities. Modalities can be of two types, future modalities (always, until, eventually, next) and past modalities (historically, since, once, previously). Texada [10] is a popular LTL Specification mining tool. However, Texada lacks representation of the extracted specification as State Machines. The approach presented in [14] and [29] are fundamentally based on the concept of Active Learning and extract a neat representation of the system in the form of a State Machine. Yet, the time complexity of these algorithms is very high to be considered for practical use. MONTRE [25] is a rich tool for timed pattern monitoring albeit it requires a monitoring template as input. TRE Automaton [13] and QMine [26] suffer from a similar problem with a requirement of a mining template. MINTS successfully overcomes all the above-mentioned shortcomings while being highly scalable for real-world applications.

3) Correctness: In this section, we formally show the correctness of our algorithm. Let *trace* denote the system trace while $Trie^D_{timed}$ represent the extracted Timed Automaton in the form of our custom timed Trie data structure and *D* denotes the depth of $Trie^D_{timed}$, then $Trie^D_{timed}$ is formally correct

Proof We will prove by induction that $Trie^D_{timed}$ is formally correct.

Base case: When $D = 1$, $Trie^1_{timed}$ has only 1 root node (or the initial state) and Σ children where Σ denotes the alphabet set.

Induction Hypothesis: Let $Trie^d_{timed}$ is correct where $1 < d < D$

Induction Step: Since $Trie^d_{timed}$ is a Trie data structure (also referred to as prefix tree), it can be stated that the formed tree at depth d corresponds to all prefixes (or sub-segments) of length $[1, d]$. It must be noted that not all sub-segments of length d must be present in $Trie^d_{timed}$ however the reverse is true. This is because, during pruning, MINTS drops the segments that are rarely traversed. Nevertheless, all the prefixes formed from the $[1, d]$ must always be present in *trace*.

In the next iteration for $d+1$, MINTS extracts all the segments of size $d+1$. It then proceeds with appending leaf nodes at depth $d+1$ provided the prefix of the segment is valid and can be traversed from the root node. Subsequently, MINTS performs pruning to remove any path traversal that were rarely traversed. Thus the final $Trie^{d+1}_{timed}$ is formed and

[3]Only the most relevant algorithms are compared

Algorithm (Authors names used if algorithm has no name)	Features						
	Pattern Extraction	Timed Pattern Extraction	Pattern Monitoring	Timed Pattern Monitoring	DFA Extraction	Timed DFA Extraction	Probability Estimation
UPPAAL SMC [37]			✓	✓			✓
Texada [10]	✓		✓				
Weiss et al. [14]			✓	✓	✓		
J. An et al. [29]					✓	✓	
MONTRE [25]			✓	✓			
Cutulenco et al. [13]	✓	✓	✓	✓			
QMine [26]	✓		✓				
MINTS	✓	✓	✓	✓	✓	✓	✓

TABLE II: Comparison of various specification miners

is correct since all the prefixes present in $Trie^{d+1}_{timed}$ can be found in *trace*.

Therefore, $Trie^{d}_{timed}$ is formally correct for all $d \in [1, D]$.

VI. Experiment

In this section, we discuss the performance, scalability, and efficiency of our approach. We used a real-world dataset along with a synthetic dataset to evaluate our framework. Our implementation is available online[4]. All experiments were carried out in a single quad-core Intel i7-2630QM CPU clocked at 2.0 GHz with 8 GB system memory and running on Ubuntu 18.04 (64 bit) using GCC version 7.5.0 and Python 3. Unless otherwise stated, each experiment was replicated 10 times, and the mean value was recorded.

A. Extraction and Analysis of Hexacopter Temporal Properties

Many safety-critical systems, such as medical devices, electronic automobiles, IOT systems, use Blackberry QNX Operating System. ONX is a real-time Unix-like operating system primarily used in embedded devices. It is equipped with *tracelogger*[5] which assist in recording interaction within and across the operating system. Since *tracelogger* operates at the kernel level, it is a useful utility to track system interrupts, inter-process communication, thread callbacks, and many more. We used *tracelogger* to capture such information from an operational hexacopter that uses QNX operating system internally. The dataset contains over 600 thousand events. After a few basic prepossessing, like conversion of $.kev$ files to $.csv$, replacing the escape characters in the event with some dummy placeholder, and sorting in non-decreasing order of the timestamp, we applied our approach to extract interesting properties. For property extraction, we considered only the Event and the associated timestamp while discarding the remaining features.

For evaluation and extraction of such properties, we used 4 temporal properties as shown in Table I. For brevity, we extracted only top 3 properties from each temporal pattern, however, MINTS would present all possible temporal properties from the system trace and rank them according to their counts. We also extracted dominant patterns which were prevalent in the entire system along with their support and confidence measure.

A total of 384 properties were extracted from the system trace among which Table III presents 5 interesting patterns. We kept the $\mathcal{P} = 0.45$ and $D = 10$. We represent events and their temporal relationship as $E1[a, b]E2$ where $\{E1, E2\} \in \Sigma$ and $\{a, b\} \in \mathbb{N}$. For example `[-INF,0]THWAITPAGE [647,1173]THRUNNING [677, 6843]REC_PULSE` represents `THWAITPAGE` as the initial event followed by `THRUNNING` within a clock constraint c of $[647, 1173]$ milliseconds and is then followed by `REC_PULSE` within a clock constraint c of $[677, 6843]$ milliseconds. In this case, `REC_PULSE` is the final event in the path $\Pi : q_0 \xrightarrow{(\alpha_0, c_1)} q_1 \xrightarrow{(\alpha_1, c_2)} q_2 \xrightarrow{(\alpha_2, c_3)} q_3$, where $\alpha_0 =$ `THWAITPAGE`, $\alpha_1 =$ `THRUNNING`, $\alpha_2 =$ `REC_PULSE`, $c_1 = [0, 0]$, $c_2 = [647, 1173]$ and $c_3 = [677, 6843]$.

Few insights from the study of these patterns suggests `MSG_WRITEV` operations having significantly higher execution time compared to `MSG_REPLY` event while `THREPLY` takes fewer milliseconds to initiate after the event `SND_MESSAGE`. These relationship are usually an expected behaviour since write operations do take longer duration compared to read and threads always communicate asynchronously which causes latency in receiving a response. Deviation from a conventional pattern can assist in detecting anomaly in a system.

The measurement for support is *low* (this is because in our experiment, over 95% of the support measures were lower than 3 decimal places). This is primarily due to the presence of clock constraint over the transition edges in the $Trie_{timed}$ where a pattern of depth D must satisfy all the constraints in its path. This filters majority of possible similar patterns from the constructed timed-trie. Meanwhile, the confidence measures have a value of almost ≈ 1 that explains the constant internal behavior of the Hexcopter system. For example, an event `SND_MESSAGE` will always be followed by `THREPLY` and its confidence would be 1. MINTS also provides the capability to extract patterns based on a user-defined threshold

[4]https://github.com/Idsl-group/MINTS

[5]https://www.qnx.com/developers/docs/7.0.0/index.html#com.qnx.doc.neutrino.utilities/topic/t/tracelogger.html

Serial	Pattern	Count	Support	Confidence
1	[-INF,0]SND_MESSAGE [1373,9332]THREPLY	16234	0.029	0.988
2	[-INF,0]SYNC_CONDVAR_WAIT_82 [3222,156765]MSG_SENDV_11 [0,0]SND_MESSAGE [576,5867]THREPLY	2042	0.003	0.999
3	[-INF,0]THWAITPAGE [647,1773]THRUNNING [677,6843]REC_PULSE	1744	0.003	0.914
4	[-INF,0]MSG_READV_16 [26259,85259]MSG_REPLYV_15	623	0.007	0.257
5	[-INF,0]MSG_WRITEV_17 [2087,15713]MSG_WRITEV_17 [3487,20070]SND_PULSE_EXE	113	0.000	1.000

TABLE III: Hexacopter Dominant Properties

for support and confidence.

Table IV captures few of the temporal properties that MINTS detected (a total of 190 temporal matches were found). These results certainly help draw critical insights related to the behavior of the system, such as responsive relationship for events like `SND_MESSAGE to THREPLY` or `THREADY to MSG_REPLYV_15` to name a few. It also provides the temporal relationship across different temporal properties.

We believe such properties would be extremely useful for system experts and analysts in understanding along with troubleshooting system behavior. Moreover, establishing a relationship between numerous events/components with time does impart a better temporal understanding of the system.

B. Evaluation using Synthetic Trace

Real-world traces provide practical use cases for a framework. We measure the scalability of the framework on synthetic traces. We used multiple synthetic traces by varying the three parameters namely, alphabet size(Σ), length of pattern during mining (as depth D of the trie), and length of the trace (L). We mined for the dominant and temporal patterns (refer Table I) while keeping $\mathcal{P} = 0.50$. Below we describe each setup and its finding in more detail.

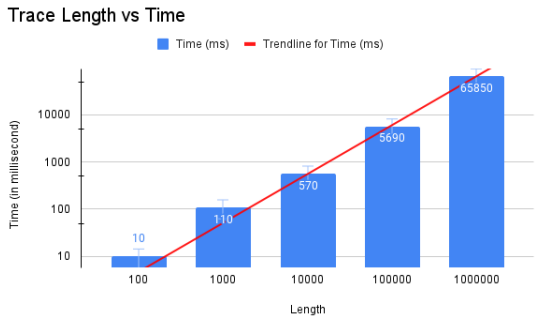

Fig. 7: Trace length vs extraction time

Setup 1: Real-world traces are large. Understanding the relationship of the trace length L over time, while keeping depth and alphabet size constant, is important to determine the scalability of a framework.

Figure 7 presents the relationship of trace length over time. It is evident that MINTS scales linearly with time while varying the trace length.

Setup 2: In this setup, we were interested in understanding the strength of the framework while mining for longer patterns. The intent is to learn long temporal patterns. Therefore, while keeping the length of trace constant at 100000 events and comprising of 10 unique events ($\Sigma = 10$), we altered D from $[3, 30]$. It must be noted that prediction accuracy decreases as future time horizon increases. Therefore, we kept the upper bound of $D = 30$.

As evident from Figure 8, MINTS grows almost linearly over time by varying the length of the extracted pattern. This does corroborate our theoretical finding of $\approx \mathcal{O}(\Sigma^D + nD)$. The effect of exponentiation over Σ is minimal since Σ in practice is a small value.

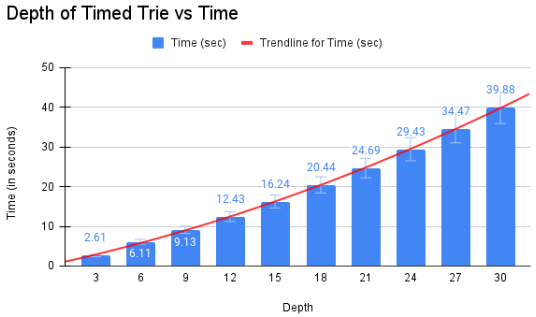

Fig. 8: Depth of $Trie_{timed}$

Setup 3: Finally, we generated multiple synthetic traces of length 100000 events while altering the alphabet size. Depth D was kept at 5. Such examination helps understand the efficiency of our framework in situations involving a large number of events. Figure 9 displays the relationship of varying the number of unique events against time.

As apparent from Figure 9, MINTS grows linearly with respect to time as shown in Section V-1. Therefore, it can be stated that MINTS do indeed exhibit $\approx \mathcal{O}(\Sigma)$ while D and L are kept constant.

As discussed in Section V-1, experimental analysis establishes the scalability of our framework. Figure 10 provides a sample illustration of the learned timed model from our

Type	From-To Event	Pattern	Count	Support	Confidence
Response Pattern	SND_MESSAGE - THREPLY	[-INF,0]SND_MESSAGE [1373,9332]THREPLY	16234	0.029	0.988
	THREADY - MSG_REPLYV_15	[-INF,0]THREADY [371,1246]MSG_REPLYV_15	11069	0.029	0.367
	EVENT_2 - MSG_RECEIVEV_14	[-INF,0]EVENT_0 [0,0]MSG_RECEIVEV_14	1496	0.002	0.499
Alternating Pattern	THRECEIVE - THRUNNING	[-INF,0]THRECEIVE [306,640]THRUNNING	9040	0.068	0.996
	SND_PULSE_EXE - REC_PULSE	[-INF,0]SND_PULSE_EXE [2109,9901]REC_PULSE	6886	0.015	0.110
	SYNC_CONDVAR_WAIT_82 - MSG_SENDV_11	[-INF,0]SYNC_CONDVAR_WAIT_82 [3222,156765]MSG_SENDV_11	2061	0.003	0.453
Multi Cause	TIMER_TIMEOUT_75 - EVENT_1	[-INF,0]TIMER_TIMEOUT_75 [1330,22081]EVENT_1	1490	0.002	0.452
	MSG_READV_16 - MSG_REPLYV_15	[-INF,0]MSG_READV_16 [26259,85259]MSG_REPLYV_15	623	0.007	0.257
	CONNECT_FLAGS_43 - MSG_SENDV_11	[-INF,0]CONNECT_FLAGS_43 [824,3063]MSG_SENDV_11	36	0.000	0.494
Multi Effect	TIMER_TIMEOUT_75 - EVENT_1	[-INF,0]TIMER_TIMEOUT_75 [1330,22081]EVENT_1	1490	0.002	0.452
	REC_PULSE - BUFFER	[-INF,0]REC_PULSE [925,2049]BUFFER	28	0.001	0.013
	THWAITPAGE - THRUNNING	[-INF,0]THWAITPAGE [647,1773]THRUNNING	768	0.003	1.000

TABLE IV: Hexacopter Temporal Properties

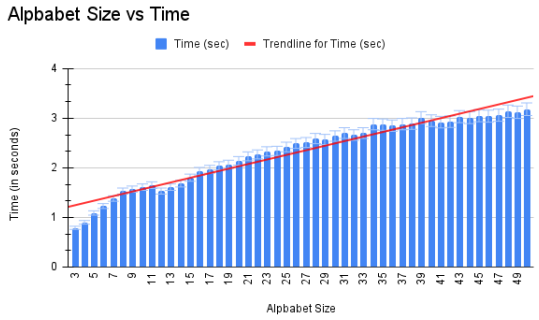

Fig. 9: Increasing alphabet size

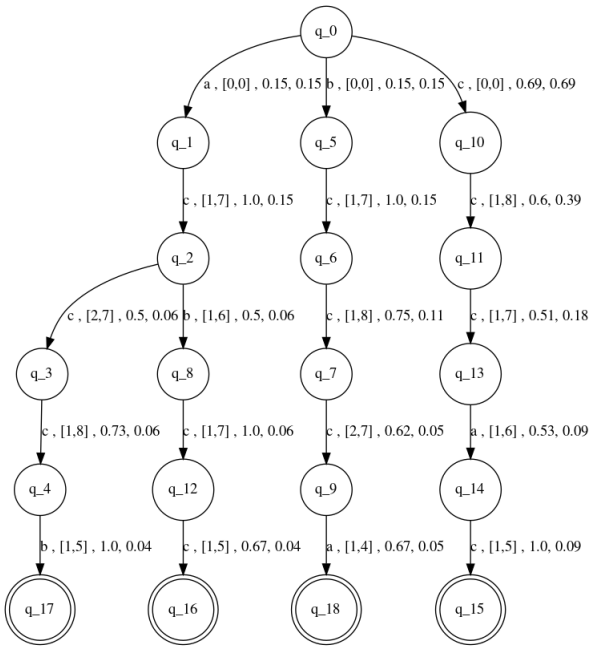

Fig. 10: Sample illustration of $Trie_{timed}$ with depth as 5

approach. While the nodes represents the states, the edges represents $\langle \alpha, c, \mathbb{P}^{q_i}_{q_{i-1}}, \mathbb{P}_{q_i} \rangle$ where α is the event, c is the clock constraint with the interval $[a, b] \forall a, b \in \mathbb{N}$, $\mathbb{P}^{q_i}_{q_{i-1}}$ as the transition probability from previous state(q_{i-1}) to the current state(q_i) and $\mathbb{P}_{q_i}$ as the transition possibility to the current state considering the all possible transitions.

VII. Conclusion

In this paper, we propose an approach to learn and represent the behavior of a system from its traces as a special timed automaton (TA) using timed-trie data structure. We first introduced our framework, MINTS, which mines temporal patterns from system traces. We then explain the working of MINTS and extracted two sets of patterns, (i) Dominant Patterns (ii) Specific Behavioral Patterns. Mining pattern signatures can also be extended by the user by defining the minimum threshold ($\mathcal{P}$) for the patterns along with support and confidence. Next, we described the use of MINTS in predicting the future event, given the user already knows the past event sequence. We believe such insights would most certainly be helpful for system experts. We then discussed the time and space complexity of our framework and validated it using synthetic traces. Real-world traces from a Hexacopter was used to extract dominant properties and the underlying temporal relationships among system events. These properties can be represented as a timed finite state machine and with the feasibility to convert into COTA (complete one-clock deterministic automata).

Moving forward, we aim to provide a simplified representation of the $Trie_{timed}$. To the best of our knowledge, the merging of two timed automata is not well studied (although Ullman and Hopcroft studied merging of finite-state machines [38]). At present, our approach suffers from event sparsity (when the time between consecutive events varies greatly). With a small trace size and large time step between two contiguous events, our approach would not be able to learn significant properties. Nevertheless, we believe MINTS can be used in safety-critical applications and by researchers to learn the temporal properties of a system. To our knowledge, this is the first attempt to develop a completely unsupervised approach.

REFERENCES

[1] D. LO, "Specification mining: Methodologies, theories and applications," 2006. [Online]. Available: https://www.comp.nus.edu.sg/~khoosc/tmp/david-lo-thesisprop.pdf

[2] T. C. Lethbridge, J. Singer, and A. Forward, "How software engineers use documentation: the state of the practice," *IEEE software*, vol. 20, no. 6, pp. 35–39, 2003.

[3] S. C. de Souza, N. Anquetil, and K. de Oliveira, "A study of the documentation essential to software maintenance." ACM, 2005, pp. 68–75.

[4] L. Moreno, "Summarization of complex software artifacts." ACM, 2014, pp. 654–657.

[5] A. S. Khan and M. K. Mattsson, "Management of documentation and maintainability in the context of software handover," vol. 1. IEEE, 2012, pp. 238–243.

[6] S. C J and A. Mahendran, "Software documentation management issues and practices: A survey," *Indian Journal of Science and Technology*, vol. 9, 05 2016.

[7] E. Tryggeseth, "Report from an experiment: Impact of documentation on maintenance," *Empirical Software Engineering*, vol. 2, no. 2, pp. 201–207, Jun 1997. [Online]. Available: https://doi.org/10.1023/A:1009778023863

[8] L. Lamport, "Time, clocks, and the ordering of events in a distributed system," *Commun. ACM*, vol. 21, no. 7, p. 558–565, Jul. 1978. [Online]. Available: https://doi.org/10.1145/359545.359563

[9] M. B. Dwyer, G. S. Avrunin, and J. C. Corbett, "Patterns in property specifications for finite-state verification," in *Proceedings of the 21st International Conference on Software Engineering*, ser. ICSE '99. New York, NY, USA: Association for Computing Machinery, 1999, p. 411–420. [Online]. Available: https://dl.acm.org/doi/10.1145/302405.302672

[10] C. Lemieux, D. Park, and I. Beschastnikh, "General ltl specification mining," in *Proceedings of the 30th IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE '15. IEEE Press, 2015, p. 81–92. [Online]. Available: https://doi.org/10.1109/ASE.2015.71

[11] R. Alur and D. L. Dill, "A theory of timed automata," *Theoretical Computer Science*, vol. 126, no. 2, pp. 183–235, 1994. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0304397594900108

[12] T.-D. B. Le and D. Lo, "Deep specification mining," in *Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis*, ser. ISSTA 2018. New York, NY, USA: Association for Computing Machinery, 2018, p. 106–117. [Online]. Available: https://doi.org/10.1145/3213846.3213876

[13] A. Narayan, G. Cutulenco, Y. Joshi, and S. Fischmeister, "Mining timed regular specifications from system traces," *ACM Trans. Embed. Comput. Syst.*, vol. 17, no. 2, Jan. 2018. [Online]. Available: https://doi.org/10.1145/3147660

[14] G. Weiss, Y. Goldberg, and E. Yahav, "Extracting automata from recurrent neural networks using queries and counterexamples," in *Proceedings of the 35th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, J. Dy and A. Krause, Eds., vol. 80, 10–15 Jul 2018, pp. 5247–5256.

[15] S. Edelkamp and S. Schrödl, "Chapter 16 - automated system verification," in *Heuristic Search*, S. Edelkamp and S. Schrödl, Eds. San Francisco: Morgan Kaufmann, 2012, pp. 701–736. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B978012372512700016X

[16] J. C. Martin, *Introduction to Languages and the Theory of Computation*, 1st ed. USA: McGraw-Hill, Inc., 1990.

[17] J. Bengtsson and W. Yi, *Timed Automata: Semantics, Algorithms and Tools*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 87–124.

[18] W. Weimer and G. C. Necula, "Mining temporal specifications for error detection," in *Tools and Algorithms for the Construction and Analysis of Systems*, N. Halbwachs and L. D. Zuck, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 461–476.

[19] R. Dssouli, A. Khoumsi, M. Elqortobi, and J. Bentahar, "Chapter three - testing the control-flow, data-flow, and time aspects of communication systems: A survey," ser. Advances in Computers, A. M. Memon, Ed. Elsevier, 2017, vol. 107, pp. 95–155. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0065245817300281

[20] S. Subbiah, C. Schoppmeyer, and S. Engell, "Efficient scheduling of batch plants using reachability tree search for timed automata with lower bound computations," in *21st European Symposium on Computer Aided Process Engineering*, ser. Computer Aided Chemical Engineering, E. Pistikopoulos, M. Georgiadis, and A. Kokossis, Eds. Elsevier, 2011, vol. 29, pp. 930–934. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780444537119501863

[21] K. Mamouras, M. Raghothaman, R. Alur, Z. G. Ives, and S. Khanna, "Streamqre: Modular specification and efficient evaluation of quantitative queries over streaming data," in *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI 2017. New York, NY, USA: Association for Computing Machinery, 2017, p. 693–708. [Online]. Available: https://dl.acm.org/doi/10.1145/3062341.3062369

[22] P. Bouyer, F. Laroussinie, N. Markey, J. Ouaknine, and J. Worrell, *Timed Temporal Logics*. Cham: Springer International Publishing, 2017, vol. 10460, pp. 211–230.

[23] D. Ulus, T. Ferrère, E. Asarin, and O. Maler, "Online timed pattern matching using derivatives," in *Tools and Algorithms for the Construction and Analysis of Systems*, M. Chechik and J.-F. Raskin, Eds. Springer Berlin Heidelberg, 2016.

[24] D. Ulus, T. Ferrère, E. Asarin, and O. Maler, "Timed pattern matching," in *Formal Modeling and Analysis of Timed Systems*. Cham: Springer International Publishing, 2014, pp. 222–236.

[25] D. Ulus, "Montre: A tool for monitoring timed regular expressions," in *Computer Aided Verification*, R. Majumdar and V. Kunčak, Eds. Cham: Springer International Publishing, 2017, pp. 329–335.

[26] P. K. Mahato and A. Narayan, "Qmine: A framework for mining quantitative regular expressions from system traces," in *2020 IEEE 20th International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, 2020, pp. 370–377.

[27] J. Yang and D. Evans, "Dynamically inferring temporal properties," in *Proceedings of the 5th ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering*, ser. PASTE '04. New York, NY, USA: Association for Computing Machinery, 2004, p. 23–28. [Online]. Available: https://doi.org/10.1145/996821.996832

[28] "Learning regular sets from queries and counterexamples," *Information and Computation*, vol. 75, no. 2, pp. 87–106, 1987.

[29] J. An, M. Chen, B. Zhan, N. Zhan, and M. Zhang, "Learning one-clock timed automata," in *Tools and Algorithms for the Construction and Analysis of Systems*, A. Biere and D. Parker, Eds. Cham: Springer International Publishing, 2020, pp. 444–462.

[30] F. Vaandrager, "Model learning," *Commun. ACM*, vol. 60, no. 2, p. 86–95, Jan. 2017. [Online]. Available: https://doi.org/10.1145/2967606

[31] M. Shahbaz and R. Groz, "Inferring mealy machines," in *FM 2009: Formal Methods*, A. Cavalcanti and D. R. Dams, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 207–222.

[32] F. Aarts, B. Jonsson, and J. Uijen, "Generating models of infinite-state communication protocols using regular inference with abstraction," in *Testing Software and Systems*, A. Petrenko, A. Simão, and J. C. Maldonado, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 188–204.

[33] F. Aarts and F. Vaandrager, "Learning i/o automata," in *CONCUR 2010 - Concurrency Theory*, P. Gastin and F. Laroussinie, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 71–85.

[34] F. Pastore, D. Micucci, and L. Mariani, "Timed k-tail: Automatic inference of timed automata," in *2017 IEEE International Conference on Software Testing, Verification and Validation (ICST)*, 2017, pp. 401–411.

[35] C. Astudillo, M. Bardeen, and N. Cerpa, "Editorial: Data mining in electronic commerce - support vs. confidence," *J. Theor. Appl. Electron. Commer. Res.*, vol. 9, no. 1, p. I–VII, Jan. 2014. [Online]. Available: https://doi.org/10.4067/S0718-18762014000100001

[36] N. K. Ahmed, A. F. Atiya, N. E. Gayar, and H. El-Shishiny, "An empirical comparison of machine learning models for time series forecasting," *Econometric Reviews*, vol. 29, no. 5-6, pp. 594–621, 2010. [Online]. Available: https://doi.org/10.1080/07474938.2010.481556

[37] A. David, K. G. Larsen, A. Legay, M. Mikučionis, and D. B. Poulsen, "Uppaal smc tutorial," *International Journal on Software Tools for Technology Transfer*, vol. 17, no. 4, pp. 397–415, Aug 2015. [Online]. Available: https://doi.org/10.1007/s10009-014-0361-y

[38] J. E. Hopcroft, R. Motwani, and J. D. Ullman, "Introduction to automata theory, languages, and computation, 2nd edition," *SIGACT News*, vol. 32, no. 1, p. 60–65, Mar. 2001. [Online]. Available: https://doi.org/10.1145/568438.568455