2019 IEEE International Conference on Systems, Man and Cybernetics (SMC)
Bari, Italy. October 6-9, 2019

Mining Time for Timed Regular Specifications

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

Sebastian Fischmeister
Dept. of Electrical and Computer Engg.
University of Waterloo
ON, Canada
sebastian.fischmeister@uwaterloo.ca

Abstract—**Dynamic evaluation of a computer program is done by analyzing the data generated by it during the execution. The program is considered correct and efficient if it meets a set of pre-specified temporal properties. Temporal properties define the order of occurrence and timing constraints on event occurrence. These properties become all the more important in the case of safety-critical real-time systems where a delayed response to a request may lead to a fault in the system. In the context of real-time systems, it is thus desirable to mine timing information and a set of dominant properties from system execution traces for testing, verification, anomaly detection, and debugging purposes.**

We propose a framework to automatically mine timing characteristics for properties that are in the form of timed regular expressions (TREs) from system traces. The framework estimates a set of dominant and most frequently occurring properties and their timing characteristics. The framework is evaluated on traces from industrial safety-critical real-time applications (a deployed autonomous hexacopter system) using traces with more than 1 Million entries.

I. Introduction

Mining temporal specifications from system traces is a popular area of research [1]. Many programs do not have well specified temporal specifications therefore mining specifications is valuable since they support numerous activities for dynamic evaluation and analysis of softwares in anomaly detection [2], verification [3] etc.

The specification mining algorithms identify a set of properties that are satisfied by traces while meeting certain criteria [4]. Quite often the timing characteristic of these properties is provided statically to the mining algorithms [4]. Analysts do not usually have concrete information about the timing behavior of these temporal properties. Therefore, it becomes important to not only mine specifications in the form of Timed Regular Expressions (TREs) but also mine the time intervals as specified in timed regular properties.

II. Related Work

Mined specification help in numerous task for analyzing the programs. For instance, they help in automated verification as well specified specification allow one to represent program behavior concisely [5]. There has been recently thrust in developing specification mining frameworks with custom specification formats such as LTL [6] or TREs [4].

There are numerous other tools available for mining invariants. Their usage is restricted to specific applications. There have been extensions to Daikon [7] for developing specific packages such as Agitator [8].The other common packages are DIDUCE [9] and IODINE [10]. DIDUCE is used for identifying root causes in JAVA programs whereas IODINE is used for inferring invariants from hardware design descriptions.

Pattern matching and frequency of occurrence of patterns is a key concept that lies beneath numerous specification mining frameworks such as the Peracotta [11], [12], [13]. Research on mining temporal specifications has mostly been focusing on qualitative notion of time. However, for safety-critical real-time systems, real-time or the quantitative notion of time plays a significant role. In the recent work on mining TRE templates from system traces [4], the timing information is considered static and has to be provided explicitly. Specifying accurate time intervals for regular expressions is a challenging task when mining properties for the system is the task at hand. With the motivation to address the problem of mining timing information from system traces, we propose a technique to mine time intervals of timed regular expression (TRE) templates satisfied by a given system's traces.

We extend our work on mining instances of TREs [4] to include extraction of time intervals from their templates. Our algorithm require a TRE template and system traces as input. The algorithms then use the distinct events from the traces to replace the event variables and interval bounds to the interval variables in the TRE templates with actual events and time intervals. The resultant *permutations* of the template are TRE instances. Intuitively, traces are processed against the TRE instances with the help of the timed automaton with a minimum and maximum time interval to evaluate all possible instances.

The algorithm is designed for TRE templates with and without the negation. Although the execution time for mining TRE templates (both with and without negation) is exponential in terms of the number of variables in the TRE templates. There is a constant speed up for TREs without negation over those with negation operator. We also state that our technique is sound i.e., the mined timing intervals by our algorithm satisfies the given threshold of confidence on the provided input traces. We also show that our technique is complete, i.e. our algorithm evaluates all TRE instances to estimate the time intervals which comply on given traces given a pre-specified threshold of confidence.

We evaluate our technique on real-world datasets generated by industry grade applications such as the QNX Neutrino real-time operating system [14].

The key contributions of this paper include:

- An efficient technique for mining time intervals in a Timed Regular Expression (TRE), To our knowledge, this is the first technique for mining time for specifications,
- A feasibility and viability study to show scalability by running the algorithms on industry grade data sets.

III. Background

A. Timed Regular Expressions

Automata theory has been used extensively to handle temporal properties of systems. However, it can handle only the qualitative notion of time. This implies that it captures the sequence of occurrence of events accurately. It lacks to account the time of occurrence between the events. They lack the ability to handle the notion of "real-time". Notion of real-time or more specifically the quantitative notion of time is important for a certain class of applications such as those in real-time systems. Since our focus lies in safety-critical systems, the quantitative notion of time becomes all the more critical. There have been techniques for mining specification that include static timing information [4]. The formalism of Timed Regular Expressions (TRE) was introduced by Asarin et. al. [15], [16].

A TRE template specifies the general structure of the property that we intend to discover. We use the term *event variable* to denote a place holder for an event. For instance, the TRE template $< a.b > [x, y]$ represents *"a is always followed by b within the time interval [x,y]"*, where a and b are *event variables* and where $x \leq y$ are *time interval variables*. We use p to denote the number of event variables and q to denote the time intervals (each interval has start time and end time) present in a TRE template. In the given example p is 2 and q is 1.

The detailed definitions for trace, event, TRE, TRE template, TRE instance, Binding, and Dominant properties can be referred to the following articles [4], [15], [17]. However due to space constraints we have only presented details here that are necessary for understanding the underlying concepts.

A TRE instance corresponds to a TRE template and has an identical TRE structure. Applying a binding to the event variables in a TRE template creates a TRE instance corresponding to that binding. The $binding$ is thus a map used to replace event variables with events from the given alphabet.

We use the idea of Timed automata [18] for implementing our TREs.

In context of mining time intervals, we need to estimate the $dominant$ time intervals applied on the specified TRE. Followed by subsequent filtering of dominant properties based on threshold of support and confidence.

We deploy a ranking module that uses a combination of support and confidence. Mining effective and relevant subset of mined specification is based on the choosing an appropriate thresholds for support and confidence.

It is intended to keep confidence threshold to be at 100% given the large number of mined TRE instances. But in reality, the real-world traces are imperfect and therfore it provides us to lower the threshold. We examine different threshold values for both support and confidence for all feasible properties in estimating dominant properties.

IV. Proposed Approach

A. Workflow

Figure 1 provides a high level overview of the integration of time mining workflow in the framework for mining timed regular expressions proposed by Cutulenco et. al. [4]. This works builds upon our previous work by generalizing our mining framework.

Estimating time intervals in the instances of timed regular expressions is extremely critical to finding valid dominant specifications for the system under consideration. Finding temporal specifications with correct timing allows for developing robust run time monitors for real-time systems. These run time monitors can be used for not only understanding the timed behavior of our system but also for other applications such as anomaly detection.

We present our time mining algorithms and ranking approach next.

B. Time Mining Algorithm

An intuitive way to evaluate the time intervals for a given TRE template is to partly utilize the existing framework for mining TREs [4]. The framework requires one to provide a TRE template and a set of input trace to mine TREs. We modify the given TRE template by replacing all time interval variables with lower bound to zero and upper bound to a large double. Choice of the upper bound is left to designers or system analysts based on the system under consideration. It is however suggested to set the upper bound sufficiently large than any possible TRE time constraint in the system. The idea is illustrated below with an example.

Example:

In Figure 2, a sample trace is shown, the event alphabet comprises of two events A and B. The label $Time$ denotes the specific time instant of occurrence of each event. Let us assume that the TRE template for which specifications are required to be mined is $\langle 0.1 \rangle [x, y]$. The event variable place holders in the template are 0 and 1 whereas time interval variables are x and y. The algorithm generates all possible permutations by replacing 0 and 1 by the events A and B respectively and vice-versa. The time interval variable lower bound x is set to zero and the upper bound y is set to 20. One can see that fixing the upper bound to 20 allows for mining all TRE instances between $[0, 20]$ and ensures that a frequently occurring pattern will not fall outside the bounds.

Once the bounds for the time intervals are chosen we mine all possible TRE instances with their respective times. A high level description of the steps for mining all possible TRE instances are as follows.

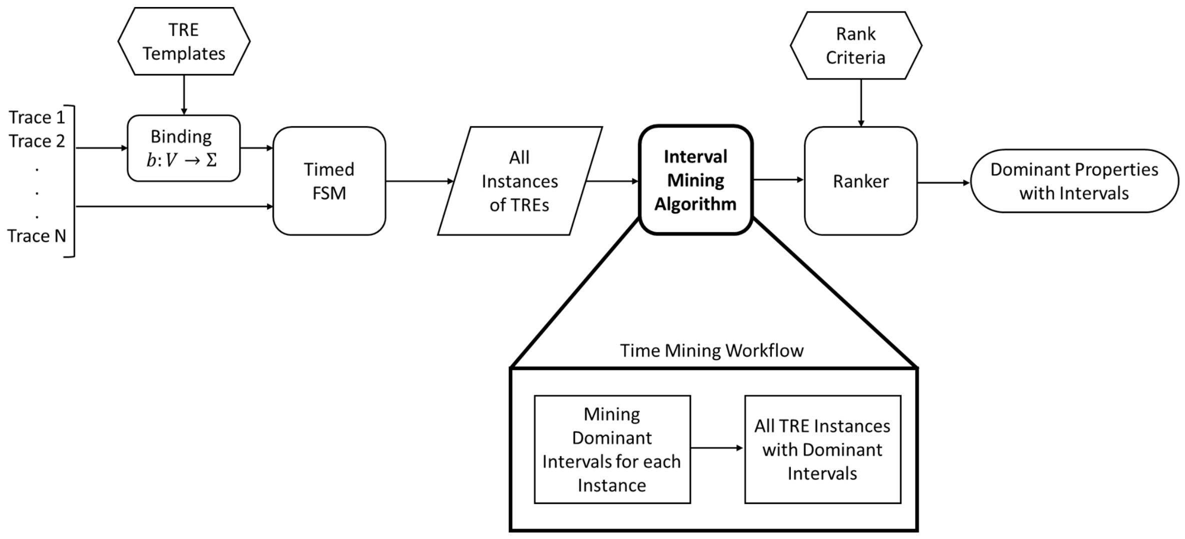
TRE Templates
Rank Criteria
Trace 1
Trace 2
Trace N
Binding $b: V \rightarrow \Sigma$
Timed FSM
All Instances of TREs
Interval Mining Algorithm
Ranker
Dominant Properties with Intervals
Time Mining Workflow
Mining Dominant Intervals for each Instance
All TRE Instances with Dominant Intervals

Fig. 1: Time Interval Estimation for TREs Workflow

```
Time:  1  3  4   5  7  8 11 13 14 17 20
Event: A  B  B   A  A  B  B  A  B  B  A
```

Fig. 2: Sample Trace

- **Representing a trace** Transform the input trace into a representation where each unique event has its corresponding time and event id, *[time, eventid]*.
- **Representing a TRE** Parse the input TRE template and transform it into a timed automaton with time interval derived from the trace as described above.
- **Checking TRE instances over traces** Iterate over the trace and process each time event pair by matching them to the relevant TRE instances.
- **Estimating time intervals for each TRE instance** Iterate over all intervals for each TRE instance. Use statistical learning to identify and fine-tune time intervals in the TRE templates for each instance.

The intervals stored while processing the trace are input to our time interval estimation algorithm. The notion of mining relevant and dominant intervals for a TRE is similar to finding the largest subset of time intervals that are most commonly occurring and have minimum variance. We develop an algorithm based on frequency of occurrence of intervals in conjunction with hierarchical clustering.

We look at this problem as a clustering problem. The largest cluster with minimal variance determines the relevant interval. The bounds of the cluster are representative of the interval bounds. Clustering techniques can be classified into two broad categories, partitioning methods (ex. k-means etc.) and hierarchical clustering. We chose hierarchical clustering for two main reasons, 1) it does not require us to pre-specify the number of clusters to be generated and 2) the resulting clusters can be represented as trees or dendrograms which can be visualized easily in some cases.

We propose an algorithmic framework for mining time for a given TRE template and a system trace. Algorithm 1 is useful for preliminary estimation of intervals and uses the empirical distribution of intervals for a given TRE template. Algorithm 2 uses hierarchical clustering for fine-tuning of the estimates from Algorithm 1.

Our approach allows system developers to do preliminary estimates of timing characteristics of property instances. These intervals are further used for determining finer boundaries.

Algorithm 1 presents a detailed description of our approach in the form of a pseudo code.

In Algorithm 1, Σ is the events alphabet and p is the number of event variables in the TRE template. The total number of time intervals in a TRE template is q where each interval ($[\alpha, \beta]$) has lower bound α and upper bound β. A denotes the timed automaton. FS and ES denote the final and error states of the timed automaton respectively. As defined earlier each unique TRE instance is denoted by π. LB and UB are the lower bound and upper bound for the time intervals parametrized on the lower and upper percentile p_l and p_u.

Freedman-Diaconis rule is used to select the size of the bin in a histogram [19]. The Freedman-Diaconis rule is designed to minimize the difference between the area under the empirical probability distribution and the area under the theoretical probability distribution. The bin size is given by the equation, Bin width = $2 \times \frac{IQR(x)}{n^{\frac{1}{3}}}$ where IQR is the inter-quartile range of samples x and n is the number of sample points.

The intervals obtained using Algorithm 1 are preliminary estimates. These provide a generic estimate of the interval ranges. It becomes more challenging if the underlying distribution exhibits complex properties, such as the distribution being multi-modal. To this effect, we extend the results obtained from Algorithm 1 to a hierarchical clustering engine as described in Algorithm 2.

Algorithm 1 Frequency estimation of time intervals in TRE

```
Input: Given a trace with $\Sigma$ unique events and a TRE pattern
    with $p$ event variables and $q$ time intervals
Output: Estimates of each $q$ time intervals in TRE template
    for all $\Sigma^p$ TRE instances
 1: Initialize each of $q$ time interval $[\alpha, \beta]$
 2: $\alpha = 0$ and $\beta$ = max time - $\tau$
 3: Parse TRE and formulate a finite Timed Automaton $\rightarrow$ A
 4: Initialize a $intervalList$ for each interval $q$
 5: Initialize an incidence matrix of size $\Sigma^p$
 6: Initialize the *success* and *reset* counters for all permutations in the matrix
 7: for (each unique event $i$ from the trace) do
 8:     for (each permutation with event $i$ - TRE w/o negation) or (each possible permutation - TRE w/ negation) do
 9:         Update the automaton state;
10:         Update the success & $intervalList$ if A $\rightarrow$ FS;
11:         Update the reset if A $\rightarrow$ ES;
12:     end for
13: end for
14: for (each unique TRE instance $\pi$ of $\Sigma^p$ instances) do
15:     Select all intervals for the TRE instance $\pi$
16:     Estimate bin size for histogram using the Freedman-Diaconis rule
17:     Generate a histogram of intervals for TRE instance $\pi$
18:     Create a density estimate of the histogram
19:     Find $LB$ and $UB$ as $p_l$ and $p_u$ percentiles
20:     Store the interval for $\pi$ with $\alpha$ =LB and $\beta$ =UB
21: end for
```

Obtaining accurate timing characteristics are critical to obtaining specifications that depict temporal behavior of dynamical systems. Therefore, fine-tuning of crude intervals is desirable. We introduce a hierarchical clustering approach that attempts to formulate largest possible clusters while minimizing variance within the clusters. The approach results in a hierarchical cluster tree that is visualized in form of a dendrogram. We use a dynamic tree cutting algorithm [20] to identify the clusters in the tree. The min and max values of each cluster represent the lower and the upper bounds of the time interval. In Algorithm 2, the matrix C computes the dissimilarity between the intervals obtained for TRE instance π within the interval $[\alpha,\beta]$. I and M denotes list of clusters and clusters to be merged respectively in the specified range. Q_π is the list of fine tuned interval ranges for π.

$$d_{ij} = d(\{X_i\}, \{X_j\}) = \|X_i - X_j\|^2 \quad (1)$$

We use Ward's minimum variance method [21] for hierarchical clustering of our data points. Ward's method uses squared Euclidean distance as initial measure of cluster separation shown in Equation 1, where d_{ij} is the distance between two intervals and $\{X_i\}$ and $\{X_j\}$. The overall method is implemented using a well-known class of recursive LanceWilliams algorithm [22].

Finally, the dominant instances for the mined TRE instances along with their time intervals are obtained using Algorithm 3. The algorithm evaluates the dominance of TRE

Algorithm 2 Interval Fine Tuning Algorithm for TRE

```
Input: Set of mined TRE instances $\pi$, their $q$ time interval
    bounds $\alpha$ and $\beta$, and sets of time intervals for each bound $[\alpha,\beta]$
Output: Estimates of each $q$ time intervals in the TRE template
    for all TRE instances
 1: for each TRE instance $\pi \in \Pi$ do
 2:     Evaluate distance matrix $C\,\forall q, \alpha \leq q \leq \beta$      }
 3:     Init. clusters $I\,\forall q, \alpha \leq q \leq \beta$                 }
 4:     Init. clusters to be merged $M\,\forall q, \alpha \leq q \leq \beta$   }
 5:     while While clusters of size unity in $I$ do                            }
 6:         Merge clusters $c_i, c_j$ with $minC$                               } Ward's
 7:         Append $c_i, c_j$ to list $M$                                       } Method
 8:         Update $C$ based on new clusters                                    }
 9:         Update cluster list $I$                                             }
10:     end while                                                               }
11:     Generate a dendrogram based on the cluster merging sequence in $M$
12:     *Dynamically* determine dominant clusters from dendrogram
13:     Return list $Q_\pi$ of min and max of each of the clusters for $q$ time intervals for $\pi$
14: end for
```

instances based on the frequency of occurrence of each TRE instance and the preliminary estimates of time interval for each TRE instance obtained from Algorithm 1. In Algorithm 3, S and C are the evaluated values of support and confidence. S_{th} and Con_{th} are the desired threshold values for support and confidence.

Algorithm 3 TRE Ranking algorithm

```
Input: Set of mined TRE instances $\pi \in \Pi$, their list of $q$
    time interval sets as $Q_\pi$ from Algorithm 2, and their
    frequency, desired support and confidence thresholds as
    $S_{th}$ and $Con_{th}$
Output: Set of dominant TRE properties with their $q$ time
    intervals
 1: Evaluate Support - S
 2: Evaluate Confidence - Con
 3: Select TREs instance where $S \geq S_{th}$ and $C \geq Con_{th}$
 4: Return selected TRE instances and their estimated time intervals
```

V. Discussions

We will examine the following important characteristics of the proposed time interval mining algorithm for temporal property templates: soundness, optimality and scalability.

A. *Soundness*

We argue that our approach is sound, it means that the time intervals reported by our algorithms actually satisfies the given criteria of support and confidence. The algorithm continuously monitors the success and reset rates for each TRE instance in the results matrix. The rates are direct results of execution of the acceptor automaton for the TRE on a trace and are used for calculation of the confidence value for the TRE instance.

Given that the reported rates and the derived confidence value are valid, it follows that the derived support value is valid as well. Therefore, any mined specification and its time interval that our algorithms report satisfies the confidence and support constraints.

VI. Experimental Setup

The implementation is done using a combination of R and C++ and is made available as a R package [1]. We use the R package 'Rcpp'[2] for integration of R and C++ and make use of C++ internally for better performance. Ragel[3] is a framework we used to synthesize Timed Automata for TREs.

We developed TRE property pattern variants [4] shown next. By using these TRE variants as TRE templates, we mined TRE instances and their time intervals from real-world system traces: QNX tracelogger. We used different thresholds for *support* and *confidence* to uncover interesting TRE instances. For the experiments, we used a single eight core machine equipped with the Intel i7-3820 CPU at 3.60 GHz and 31.4 Gb of RAM. The machine runs Ubuntu 14.04 LTS 64 bit.

A. TRE Templates

The following four TRE templates are used for the evaluation of both real and synthetic traces.

T-1(response): $(\hat{\ } P)^*.((\langle P.(\hat{\ } S)^*.S\rangle[\alpha,\beta]).(\hat{\ } P)^*$

T-2(alternating): $(\hat{\ } P,S)^*.((\langle P.(\hat{\ } P,S)^*.S\rangle[\alpha,\beta]).(\hat{\ } P,S)^*$

T-3(multi-effect): $(\hat{\ } P,S)^*.((\langle P.(\hat{\ } P,S)^*.S\rangle[\alpha,\beta]).(\hat{\ } P)^*$

T-4(multi-cause): $(\hat{\ } P,S)^*.((\langle P.(\hat{\ } S)^*.S\rangle[\alpha,\beta]).(\hat{\ } P,S)^*$

VII. Experimental Results

A. Performance Evaluation using Real QNX Traces

The QNX real time operating system (RTOS) is used in many safety critical systems, such as medical devices, nuclear monitoring systems, vehicles, etc. The QNX RTOS has a very advanced logging facility, *tracelogger*. Tracelogger facilitates detailed tracing of the kernel and user process activity on any system. More specifically, it can log interrupt activity, various states of processes and threads, communication within the system, kernel calls, custom user events, and much more. The logged events give a detailed view into the behavior of the system, but due to the large quantity of the produced information are often difficult to make use of by developers and system designers. These traces are thus a perfect resource for dynamic mining of system properties, specifications and time intervals for TREs.

For the evaluation we used a set of traces collected from an operational hexacopter loaded with the QNX RTOS and a user control process. Of interest in this portion of the evaluation are the time intervals and thresholds for support and confidence that would produce a minimal dominant set of specifications and timing constraints that can be assessed by developers or system designers.

The hexacopter trace used for the evaluation contained 1 million events, with 205 distinct events. We used the four TRE templates, T-1 to T-4, for the evaluation. The table report the time intervals estimated by our algorithm given varying values of threshold for each of the TRE template.

The results are better understood using an a single instance from our TRE instances for the template T-1. The intervals obtained during Algorithm 1 for all instances of the template T-1 were stored and a high-level estimate of the time interval was obtained for each instance. For example, we took the instance where `P:INT_ENTR` and `S:INT_EXIT`. The total number of instances found in the trace were 92,941. There are 31,126 occurrences of the instance we are considering. We perform frequency analysis using histogram on the obtained intervals as shown in Figure 3. We estimate the interval range which consists of 80% of the intervals for this instance. It provides us with a preliminary range of time interval for ths given instance.

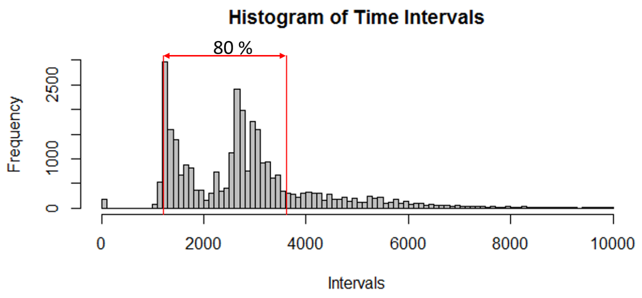

Fig. 3: Histogram of time intervals for an instance of TRE template T-1

The intervals within the 80% range are selected and subjected to Algorithm 2 for fine-tuning of the interval. The results of hierarchical clustering are shown using a *dendrogram*. The dendrogram for our instance is shown in Figure 4. Sometimes dendrograms are not easy to read if the dataset size is huge. The dynamic cut algorithm finds the clusters by cutting the tree. There are three clusters highlighted in the tree by rectangles. The interval ranges for each interval is shown as [α_1 =1209,β_1 =2084], [α_2 =2125,β_2 =2795], and [α_3 =2833,β_3 =3625]. The individual counts of the instance for each time intervals are also shown.

The outcome of the Algorithm 2 shortlists three intervals with their rankings based on the number of instance counts. These can then be analyzed by system designers for better understanding of the temporal characteristics.

Once we have filtered the instances and obtained their time intervals, subsequently we find the dominant instances with their time intervals using ranking module as expressed in Algorithm 3.

In the Table I, Table II, Table III, and Table IV, we present the observation of the most dominant instance counts for TRE

[1]https://bitbucket.org/a_narayan/tre-mining-time/
[2]http://www.rcpp.org/
[3]http://www.colm.net/open-source/ragel/

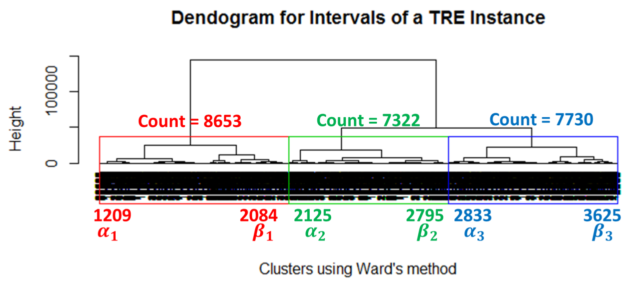

Fig. 4: Dendrogram of time intervals for selected range of the TRE instance

TABLE I: Dominant Instance counts and Intervals for TRE T-1 for various threshold of confidence

Confidence	Instance Counts	Interval
0.9	1278	[1245, 2245]
0.8	2344	[1351, 3426]
0.7	3859	[1487, 4650]
0.6	4794	[1511, 3925]
0.5	6473	[1711, 3450]

TABLE II: Dominant Instance counts and Intervals for TRE T-2 for various threshold of confidence

Confidence	Instance Counts	Interval
0.9	750	[843, 1567]
0.8	933	[1334, 2973]
0.7	1256	[1247, 2095]
0.6	1326	[1234, 3134]
0.5	1945	[1721, 2341]

TABLE III: Dominant Instance counts and Intervals for TRE T-3 for various threshold of confidence

Confidence	Instance Counts	Interval
0.9	245	[765, 1943]
0.8	454	[4985, 5520]
0.7	543	[1295, 2694]
0.6	698	[2670, 3985]
0.5	746	[1367, 3254]

patterns T-1, T-2, T-3, and T-4 with varying confidence levels, support of 1, and top time interval for each of those instances.

It is evident from the results that the number of instance counts of the most dominant property increases with reduction in the confidence level. Reduction in confidence allows for relaxation in the filtering of mined properties. The time interval shows the interval corresponding to the most dominant property mined.

TABLE IV: Dominant Instance counts and Intervals for TRE T-4 for various threshold of confidence

Confidence	Instance Counts	Interval
0.9	184	[987, 1864]
0.8	256	[1276, 4598]
0.7	452	[3575, 4830]
0.6	561	[2351, 3476]
0.5	845	[478, 1678]

VIII. Conclusions

This paper presents a novel approach for mining time intervals in TREs with and without negation in embedded system traces. We presented a set of experiments to demonstrate that the algorithm is scalable, robust, and sound. We validate our framework on industrial strength safety-critical real-time applications traces from a running hexacopter with QNX real-time operating system.

We believe that our framework is generally applicable and is especially useful for constructing more advanced analysis tools that require TRE specification mining. In future, we propose to develop an interactive query language framework for estimating both the time intervals and dominant specifications from system traces.

References

[1] C. Lemieux, D. Park, and I. Beschastnikh, "General LTL Specification Mining," in *Proceedings of the 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, ser. ASE '15. Washington, DC, USA: IEEE Computer Society, 2015, pp. 81–92. [Online]. Available: http://dx.doi.org/10.1109/ASE.2015.71

[2] M. Christodorescu, S. Jha, and C. Kruegel, "Mining specifications of malicious behavior," in *Proceedings of the 1st India software engineering conference*. ACM, 2008, pp. 5–14.

[3] Z. Kincaid and A. Podelski, "Automated Program Verification," in *Language and Automata Theory and Applications: 9th International Conference, LATA 2015, Nice, France, March 2-6, 2015, Proceedings*, vol. 8977. Springer, 2015, p. 25.

[4] G. Cutulenco, Y. Joshi, A. Narayan, and S. Fischmeister, "Mining timed regular expressions from system traces," in *Proceedings of the 5th International Workshop on Software Mining*, ser. SoftwareMining 2016. New York, NY, USA: ACM, 2016, pp. 3–10. [Online]. Available: http://doi.acm.org/10.1145/2975961.2975962

[5] G. Ammons, R. Bodík, and J. R. Larus, "Mining Specifications," in *Conference Record of POPL 2002: The 29th SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Portland, OR, USA, January 16-18, 2002*, J. Launchbury and J. C. Mitchell, Eds. ACM, 2002, pp. 4–16. [Online]. Available: http://doi.acm.org/10.1145/503272.503275

[6] M. Bonato, G. Di Guglielmo, M. Fujita, F. Fummi, and G. Pravadelli, "Dynamic Property Mining for Embedded Software," in *Proceedings of the Eighth IEEE/ACM/IFIP International Conference on Hardware/-Software Codesign and System Synthesis*. ACM, 2012, pp. 187–196.

[7] M. D. Ernst, J. H. Perkins, P. J. Guo, S. McCamant, C. Pacheco, M. S. Tschantz, and C. Xiao, "The Daikon system for dynamic detection of likely invariants," *Science of Computer Programming*, vol. 69, no. 1, pp. 35–45, 2007.

[8] M. Boshernitsan, R. Doong, and A. Savoia, "From Daikon to Agitator: lessons and challenges in building a commercial tool for developer testing," in *Proceedings of the 2006 international symposium on Software testing and analysis*. ACM, 2006, pp. 169–180.

[9] S. Hangal and M. S. Lam, "Tracking down software bugs using automatic anomaly detection," in *Proceedings of the 24th international conference on Software engineering*. ACM, 2002, pp. 291–301.

[10] S. Hangal, N. Chandra, S. Narayanan, and S. Chakravorty, "IODINE: a tool to automatically infer dynamic invariants for hardware designs," in *Proceedings of the 42nd annual Design Automation Conference*. ACM, 2005, pp. 775–778.

[11] 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: ACM, 2004, pp. 23–28. [Online]. Available: http://doi.acm.org/10.1145/996821.996832

[12] J. Yang, D. Evans, D. Bhardwaj, T. Bhat, and M. Das, "Perracotta: mining temporal API rules from imperfect traces," in *Proceedings of the 28th international conference on Software engineering*. ACM, 2006, pp. 282–291.

[13] J. Yang and D. Evans, "Automatically inferring temporal properties for program evolution," in *Software Reliability Engineering, 2004. ISSRE 2004. 15th International Symposium on*. IEEE, 2004, pp. 340–351.

[14] "QNX Neutrino RTOS," http://www.qnx.com/products/neutrino-rtos/neutrino-rtos.html.

[15] O. M. Eugene Asarin, Paul Caspi, "Timed Regular Expressions."

[16] R. Alur, P. Černỳ, P. Madhusudan, and W. Nam, "Synthesis of interface specifications for Java classes," *ACM SIGPLAN Notices*, vol. 40, no. 1, pp. 98–109, 2005.

[17] C. Lemieux, D. Park, and I. Beschastnikh, "General LTL Specification Mining," in *Automated Software Engineering (ASE), 2015 30th IEEE/ACM International Conference on*. IEEE, 2015, pp. 81–92.

[18] R. Alur and D. L. Dill, "A Theory of Timed Automata," *Theor. Comput. Sci.*, vol. 126, no. 2, pp. 183–235, Apr. 1994. [Online]. Available: http://dx.doi.org/10.1016/0304-3975(94)90010-8

[19] D. Freedman and P. Diaconis, "On the histogram as a density estimator:l2 theory," *Zeitschrift für Wahrscheinlichkeitstheorie und Verwandte Gebiete*, vol. 57, no. 4, pp. 453–476, Dec 1981. [Online]. Available: https://doi.org/10.1007/BF01025868

[20] P. Langfelder, B. Zhang, and S. Horvath, "Defining clusters from a hierarchical cluster tree: the dynamic tree cut package for r," *Bioinformatics*, vol. 24, no. 5, pp. 719–720, 2007.

[21] J. H. W. Jr., "Hierarchical grouping to optimize an objective function," *Journal of the American Statistical Association*, vol. 58, no. 301, pp. 236–244, 1963. [Online]. Available: http://www.tandfonline.com/doi/abs/10.1080/01621459.1963.10500845

[22] W. H. Day and H. Edelsbrunner, "Efficient algorithms for agglomerative hierarchical clustering methods," *Journal of classification*, vol. 1, no. 1, pp. 7–24, 1984.