Flexible Mining of Prefix Sequences from Time-Series Traces*

António A. Bruto da Costa¹, Goran Frehse², Pallab Dasgupta¹
¹Indian Institute of Technology Kharagpur, ²ENSTA ParisTech

Abstract

Mining temporal assertions from time-series data using information theory to filter real properties from incidental ones is a practically significant challenge. The problem is complex for continuous or hybrid systems because the degrees of influence on a consequent from a timed-sequence of predicates (called its prefix sequence), varies continuously over dense time intervals. We propose a parameterized method that uses interval arithmetic for flexibly learning prefix sequences having influence on a defined consequent over various time scales and predicates over system variables.

Time-series data accounts for a large fraction of the world’s data. The data constitutes a recording of a continuously evolving phenomenon. Most biological and cyber-physical systems produce such data. The study of patterns in time-series data has played a key role in improving our understanding of how these systems behave. Given a trace of a system as time-series data, it is useful to derive succinct human-intelligible descriptions of explanations for observable events in the data as temporal properties that are satisfied by this data. Such properties are useful in the design of systems that admit physical components having behaviours that are characterized experimentally.

Note that, theoretically in the real-time domain, there can be an infinite number of properties satisfied by the data. Trivially, property $\alpha \Rightarrow \beta$ is satisfied by the data if no observation of $\alpha$ exists in it (a support of zero for $\alpha$). Properties with a high support can be used for improving the understanding of system behaviour, to validate specifications, understand gaps in testing, discover previously unknown behaviours and for anomaly detection. Inferring causal relationships from time-series data can be difficult for large data-sets and ill-defined for unknown systems. Most realistic causal relationships exist as timed sequences of events that affect the truth of a target event. For example: "If $v$ is greater than 50kmph and within 5 minutes $x$ is below 50m then within 8 minutes $y$ is more than 67m".

Data-mining for time-series data is a well studied area of the data-sciences. However, most studies have focused on generating summary measures to cluster time-series datasets into natural groupings based on similarity/dissimilarity measures, or use a nominal time-series dataset to identify anomalies in other time-series datasets. Pattern mining for time-series data is also well studied, however such patterns rely on a discretization of the time-series data over which discrete pattern mining algorithms are employed to derive common subsequences among the data-sets. To the best of our understanding, the mining of temporal properties from a single time-series trace (or a set of traces), in the form of cause-effect patterns has not been studied. Such temporal properties are learned, not with the intention of classifying the time-series traces into one class or another, but with the intention of learning explainable formal properties that can be used to better understand how a system behaves when little to no information is available of its internal function.

In an observed time-series trace, given a target event $E$, a prefix sequence is a sequence of observations (as predicates or events) that appear to have an effect on the truth of $E$. An event can have an infinite set of prefix sequences, though not all valid. For a sequence $\alpha$, not observed

*For more information, contact the primary author (antonio.cse.iitkgp@gmail.com).
in the trace, \( \alpha \Rightarrow E \) is trivially true. Hence, as mentioned earlier, \( \alpha \) is not a true prefix sequence since it does not appear in the given data. On the other hand, sequence \( \beta \), observed in the data as a prefix for \( E \), may not be causally related to \( E \), because counter-examples exist in the trace that contain \( \beta \) but are not followed by an observation of \( E \). Moreover, the time delays between events in a prefix sequence can be important. If the delays between any pair of events in the sequence change in any way, this could introduce counter-examples and render the sequence with the changed delays false.

Here, we address the problem of learning causal relationships that exist in time-series data. Given a target observation \( E \) in the data, we compute prefix sequences that are likely causes of \( E \). Our approach is designed to be flexible. It infers causal relationships using an alphabet of predicates over variables in the data. These predicates can either be user-specified or learned using parameter estimation techniques. The methodology does not use strict property templates, and allows flexibility in the property structure with controls allowing the generation of finely to coarsely constrained properties. Furthermore, every generated property must be valid throughout the data. Ranking heuristics are provided to assess the quality of properties. We also present a more natural language for sequences called the Prefix Sequence Inferencing Language (PSI-L), to express learned properties. PSI-L is derived from the popular SystemVerilog Assertion (SVA) language \cite{26}. While the SVA language is designed for the specification of properties over clocked-discrete systems, PSI-L properties describe real-time relationships using the notions of events and predicates over real variables from STL \cite{37}, while adhering to the sequence syntax and semantics of SVA.

In this article, our core contributions are as follows:

- A novel decision tree learning method for learning temporal sequences from time-series data, with fine grained controls to moderate how the decision tree constrains data. Interval arithmetic is used to handle large time-series data-sets.
- A language (PSI-L) for representing properties learned and a method to translate the learned decision tree into PSI-L properties.
- Ranking measures to quantify the quality of the properties learned and implicitly quantify the quality of the data-set.
- Case studies from real and synthetic real system traces.

The article is organized as follows. Section 2 describes a motivating example and Section 3 described the formal language for representing properties mined. Section 4 formally describes the problem we address in this article. Section 5 presents definitions for various structures and metrics used throughout this article. Section 6 describes an algorithm for mining non-temporal (instantaneous) immediate properties. Section 7 we extend the metrics in Section 6.1 to develop an algorithm for mining temporal sequence expressions. Section 8 introduces ranking metrics for PSI properties. Section 9 discusses measures employed to prevent over-fitting using various stopping conditions and pruning methods. Section 11 presents our comments and summarizes the work.

1 Related Work

One of the contributions of this article is in the learning of likely prefixes (explanations as a sequence of interesting events or episodes) for an event in a dense-time real valued signal expressed as a time-series. This form of analysis has characteristics of a variety of standard analyses performed on time-series data but doesn’t cleanly fit into any one of them, nor does a clean reduction seem possible.

This section presents an exploration in the area of data mining from time-series datasets. We describe a number of analysis techniques applicable to time-series datasets and show how our requirement differs from the problems solved in existing literature. We present this as a four-part study broadly partitioned as follows:
1. **Time-Series Data-Mining**: A discussion of work on querying, classifying, prediction and detecting anomalies and patterns from time-series data.

2. **Learning using Templates**: Parameterized property-templates are used to describe requirements that characterize systems. Most work herein uses parameter learning as its basis.

3. **Learning as a Two-Class Classification Problem**: A formal property is learned as a distinguisher between two sets of time-series traces. The learning task involves both the property structure as well as its parameter values.

4. **Learning Allen and Linear Temporal Logic Patterns**: Allen’s before relations are learned from clock-sampled information as sequences of interesting events in the time-series. Additionally, decision trees are used to learn safety and bounded-liveness properties in a subset of LTL from Boolean traces of a Boolean system.

Our comments on current literature and its relation to learning explanations from time-series data are summarized in Section 1.2

1.1 **Time-Series Data-Mining**

Mining from time-series data has been a topic of study for decades. A survey of methods for mining information from such time-series is provided in [19, 41] along with abstractions used to represent large data-sets and the types of analyses performed on these data-sets.

Given time-series $S$ and $G$, and dissimilarity metric $D(S, G)$, existing methods for learning from time-series data can be broadly classified into the following types:

1. **Querying**: From a time series data-base $DB$, given the query $S$, find time-series $G \in DB$, where $D(S, G) < \alpha$, $\alpha$ is a threshold of dissimilarity. The methods suggest using Singular Value Decomposition (SVD), the Discrete Fourier transform (DFT), Discrete Wavelet Transforms (DWT) or Adaptive Piecewise Constant Approximations (APCA) ad the dissimilarity metric and to index the time-series [10, 20].

2. **Clustering**: Find groups of time-series in the data-base $DB$ that are most similar using the dissimilarity metric $D(., .)$. For instance, in [15] time-series recorded from the electrical power-grid are clustered together using Kohonen Maps, while in [29] distance measures are explored for clustering.

3. **Summarization**: Summarize $S$ with an approximation that is representative of $S$. In [27] representative sketches are generated to summarize trends for time-series data. The sketch that is computed is a visual representation of the many time-series in the data-set. Similarly, [43] discusses other visualization methods for visualizing trends in a univariate time-series data set.

4. **Separation Features**: Given time series $S$ and $\hat{S}$, find interesting features that separate the two time-series. In [24] time-series data is analyzed to determine interesting events by partitioning the time-series into piece-wise segments, such that the segments approximate the time-series without being too dissimilar from the original. The points at which the segments join are treated as interesting episodes in the time-series. In [31] the authors suggest discretizing the time-series as a string of time-value pairs and using a sliding window mechanism to identify which sub-strings occur most frequently in the time-series.

5. **Prediction**: Given a time series $S$ over time points $(t_1, ..., t_n)$, predict the behaviour of $S$ as if observed over time points $(t_{n+1}, ..., t_{n+k})$. Prediction relies on the use of various statistical models and techniques and has wide applications. Detailed studies in this area have been done in [9, 25, 42, 8].
6. **Anomaly detection:** The problem of detecting a pattern that deviated from a nominal behaviour is strongly linked to the problem of prediction. Like prediction, it also relies on having a sufficiently accurate model of the time-series to be able to identify deviations \[49, 50, 36\].

7. **Motif Discovery:** A sub-sequence that is observed frequently in a time-series is called a **Motif**. The discovery of Motifs is also used for clustering and other applications. The interested reader may refer to \[19\] for a detailed review of existing literature in this area.

A time-series trace is viewed as a finite series of timed samples and interpolation measures are used to fill gaps where no timed data is present. The studies use summary distance measures to determine the similarity/dissimilarity between two time-series.

### 1.1.1 Learning using Templates

A large repository of work exists on mining parameter values for a template property in parametric STL (PSTL) \[28, 4, 5, 46\] with the aim of optimizing property robustness for the given trace.

The work in \[4\] proposes learning the range of valid parameter values for a PSTL (Parameterized STL) property that a given set of dense-time real valued system traces satisfy. Given a formula in PSTL, \[4\] computes a validity domain for the formula’s time and value parameters such that all traces satisfy the formula given these domains. In \[46\], the authors propose a methodology to compute parameter domains for a property in MTL that a given embedded and hybrid system satisfies. The system is modeled in MATLAB and the authors use their property falsification tool S-TALIRO \[9\] to compute the set of parameter values that robustly satisfy the parameterized MTL property.

In \[28\], the authors propose learning parameter values that satisfy system requirements expressed as template properties in PSTL. They use the framework BREACH \[17\] to compute falsifying traces for a concrete choice of parameter values for the property and iterate until they converge on a combination of parameter values that the given set of system traces satisfy. The work in \[28\] requires system traces along with a model of the system that can be used with their falsification tool.

Most recently, the authors of \[5\] improved on the work in \[4\] by proposing a new methodology to compute the validity domains of parameters of a given PSTL template property by computing bottom-up satisfaction and robustness signals, and by propagating them as a function of time from sub-formulas to formulas.

### 1.1.2 Learning as a Two-Class Classification Problem

While in Section \[1.1.1\] a formula structure was provided as input to the learning task, we now explore studies in which such templates are not-provided, however a syntax is assumed, and property structures and parameter values are learned from traces of black-box hybrid system behaviour.

We focus on the work on Temporal Logic Inference (TLI) in \[33, 7\]. The problem that is solved is the two-class classification problem, wherein the proposed methodologies go further than template learning, by learning a concrete STL-like property, its structure and parameters, with the objective of distinguishing between desirable and undesirable traces.

In \[33\] a property is learned using a directed search over property structures that have a qualitative (language inclusion) and quantitative (robustness \[18\]) partial order among them. The work in \[7\] learns a decision tree that best distinguishes between two trace sets. The property is a mapping of the decision tree into a fragment of STL. Both \[33\] and \[7\] use the robustness measure to optimize the structure and parameters of the property mined.

On the other hand, work in \[6\] learns a discriminator property to distinguish between traces generated by two different processes. The method relies on using a statistical abstraction of the data in the traces, and thereafter uses a two-stage approach for identifying a discriminator.
First, an optimal formula structure is learned from a library of template structures. This is then followed by a tuning of parameters in the formula so as to maximize its discrimination power.

The strategies for mining patterns from system representations can be classified primarily as being analyses for Boolean (digital) or Non-Boolean systems. We consider the space of Non-Boolean systems to include both software and hybrid systems. While software programs are characterized by discrete behaviours, hybrid systems interleave discrete behaviours with continuous behaviours as the continuous evolution of its real-valued variables.

Past work in [48, 47, 21, 22, 16] focus on mining sequences and causal relations for program events. However, the sequences mined do not preserve timing information between the events. Studies in [11, 14, 13, 35] focus on mining cause-effect relations in programs as LTL properties. The methodology in [12] mines timed regular expressions from program traces while in [23], decision trees are used to learn invariants for software programs.

1.1.3 Learning Allen and Linear Temporal Logic Patterns

The tool Goldmine [44] uses decision trees to mine causal relations from clocked traces of Boolean systems as an ordering of events. The assertions mined are in a subset of LTL. To the best of our knowledge the applicability of these techniques is limited to discrete event systems and are not applicable to time-series data over real-time.

The work in [30] mines Allen’s interval relations, specifically event intervals from clocked event traces. The proposed methodology mines \textit{nf}er rules (based on Allen’s Temporal Logic) from learned \textit{before} relations given a set of clocked event traces. The learned sequences are a series of before relations between events in the traces. For instance, from trace data on the NASA’s Mars rover, for a single rover activity command, the miner learned the relation \textit{dispatch before complete}, which indicates that relation that a command is first dispatched before it is completed.

1.2 Our Comments

Note that our aim is to mine explanations of an \textit{interesting} event \( E \) (called the \textit{target}), where the event \( E \) is known from domain knowledge of the system. The explanation is expected to be mined as a formal sequence of events in the data that appear as a prefix of \( E \). We evaluate existing literature in this regard as follows:

- From studies on data-mining for patterns or summaries from time series, the types of patterns mined are over a known alphabet and use standard pattern matching algorithms that are tailored to work with discretized time-series data. The mined patterns are not of the cause-effect type, and do not derive explanations for known events, but are mined as being a summary representation of the data. A summary is mined as strings of ordered events or described pictorially using representative summary graphs. It does not explicitly contain any timing information, and require further analysis by an engineer.

To the best of our understanding, the literature on data-mining does not address the problem of deriving explanations for known events using a formal structure.

- The work on template based learning requires a template in a logic language such as MTL or STL to be provided. Here the form of the explanation for an event in the system is expected to be known, while parameters may be learned.

- The aim of the two-class classification problem [33, 7, 6] is to distinguish between two sets of traces. A representative logic formula is learned to for a set of desirable traces. The formula is constrained so as to not satisfy any of the traces in the set of undesirable traces. However, while deriving explanations for a specific event that is considered \textit{interesting}, such an event could occur in multiple places in a single trace. One could argue that sub-traces containing the occurrence of the event could be labeled as one class, while those where the event does not occur could be labeled as another class. However it is not clear how such
Figure 1: Traffic Surveillance data-set. Vehicle paths are shown in blue with demarcations for areas of activity.

a division could be made, given the infinite number of ways in which the time-series could be split.

- The work on mining Allen intervals and LTL specifications are interesting but cannot be used to mine explanations over dense-time real-valued signals.

We aim to learn sequences that appear as prefixes of a known target in the set of given traces. Given time-series trace as a dense-time real valued signal, and a known observable event $E$ in the trace, we wish to compute prefix sequences that when observed appear to explain the observation $E$. In Section 10 we propose an algorithm that uses decision trees to mine such prefixes.

2 Mining Explanations: Why does it happen?

We start with a motivating example. Consider the traffic map shown in Figure 1. The figure is representative of an area containing five major areas of activity; namely a residential complex, schools, a mall, eateries, and an industrial estate. Vehicles are tagged with GPS devices to monitor their movements. The location and velocity of all vehicles is recorded. In addition to learning patterns that describe various routes that a vehicle follows, or answering queries such as which roads to people most use to drive to the eateries, we also wish to learn about safety critical issues such as, are there any crashes, and if so where and when do they most often occur.

Prior to verification, engineers do not always have a complete specification of the system being designed, nor is it easy to determine the cause of failures or bugs when they occur. The problem we wish to address is to mine as many explanation patterns from the traces so as to uncover potential reasons that could improve an engineers understanding on why (or when) some event occurs.

The following PSI-L formula describes one explanation for a vehicle crash:

```
22<=x<=24 && 15<=y<=20 ##[0:0.368797] v>100
##[2.20079:2.3] !(route==1)|=> crash
```

The formula reads as "If the car is in the region $x \in [22:24]$ and $y \in [15:20]$ and if within the following $0h22m8s$ the velocity is above $100$kmph and thereafter in the next $2h12m$
to 2h18m if the route is not Route-1 a crash occurs°. This is an example of a property mined from the data depicted in Figure [1]. Observe that the formula describes a sequence of events and very finely grained time delays between the events. Mined patterns are always in the form of such sequences. The events may be mined or provided as inputs using domain knowledge. Time-delays are computed from the decision tree generated.

3 Explaining using Sequences: Prefix Sequence Inferencing Language

We express explanations in the form of a sequence of events or episodes (predicates over real-valued variables). In addition to an ordering between events, the timing between adjacent events is key. An explanation for a target event is observed as a prefix to the target. The language used for describing an explanation for an event is therefore called the Prefix Sequence Inferencing language (PSI-L).

3.1 PSI-L Syntax

The language used to describe prefix sequences inferred from time-series data has the following general syntax:

\[ S \Rightarrow E \text{ or } S \Rightarrow \tau_0 E \]

where, \( S \) is a prefix sequence, also known as a sequence expression, of the form \( s_n \tau_n s_{n-1} \tau_{n-1} \ldots \tau_1 s_0 \). A delay \( \tau_i \) is a time interval of the form \([a:b] , a \leq b \) and each \( s_i \) is a Boolean expression of PORVs and events. The length of the sequence expression is \( n \) (having \( n \) non-temporal sub-expressions). The predicate or event \( E \) in a PSI-L formula is assumed to be known. It is in the context of \( E \) that \( S \) is learned. \( E \) is called the target of the PSI-L property. The notation \( S_i^j \) is used to denote the expression \( s_j \tau_j \ldots \tau_{i+1} s_i \), \( 0 \leq i \leq j \). Hence \( S \equiv S_0^n \).

3.2 PSI-L Semantics for Traces

For variable set \( V \), the set \( \mathbb{D} = \mathbb{R}^{\geq 0} \times \mathbb{R}^{|V|} \) is the domain of valuations of timestamps and variables in \( V \). A data point is a tuple \((t, \eta) \in \mathbb{D}, t \in \mathbb{R}^{\geq 0} \) and \( \eta \in \mathbb{R}^{|V|} \). For variable \( x \in V \), the value of \( x \) in the data point \((t, \eta) \) is given as \( \eta[x] \). Boolean and real-valued variables are treated the same in view of the implementation. A Boolean value at a data point is either 1 for true or 0 for false, and \( \{0,1\} \subset \mathbb{R} \).

Definition 1. Hybrid System Trace: A trace \( \mathcal{T} \) of a hybrid system is an ordered list of tuples \((t_1, \eta_1), (t_2, \eta_2), (t_3, \eta_3), \ldots (t_d, \eta_d), \forall i \in \mathbb{N}_{d-1} t_i < t_{i+1} \). The length of \( \mathcal{T} \), the number of tuples in \( \mathcal{T} \), expressed as \(|\mathcal{T}|\), is \( d \). The temporal length of \( \mathcal{T} \), denoted \(|\mathcal{T}|\), is \( t_d - t_1 \).

\( \mathcal{T}(i) \) denotes the \( i \)th data point \((t_i, \eta_i)\) in trace \( \mathcal{T} \).

A sub-trace \( \mathcal{T}_i^j \) of \( \mathcal{T} \) is defined as the ordered list \((t_i, \eta_i), (t_{i+1}, \eta_{i+1}), \ldots (t_j, \eta_j) \mid i,j \in \mathbb{N}_d \) and \( i \leq j \).

Definition 2. Match of a Sequence Expression and a PSI-L formula: The sub-trace \( \mathcal{T}_i^j \), \( i \leq j \), of \( \mathcal{T} \) models the sequence expression \( S_1^m := s_m \tau_{m-1} s_{m-1} \tau_{m-2} \ldots \tau_1 s_1 \), denoted \( \mathcal{T}_i^j \models S_1^m \) iff:

- \( \eta_i = s_m \),
- \( \exists_{j \geq k \geq i} \mathcal{T}_k^j \models S_1^{m-1} \) and \( t_k - t_i \in \tau_{m-1} \).

An PSI-L formula can match at zero or more data points in \( \mathcal{T} \). A PSI-L property \( s_n \tau_n s_{n-1} \tau_{n-1} \ldots \tau_1 s_0 \Rightarrow \tau_0 E \) matches in \( \mathcal{T} \) at \( \mathcal{T}(j) \) iff, \( \exists_{i \leq j \leq k} \mathcal{T}_i^j \models S_0^m \) and \( t_k - t_j \in \tau_n \). The sub-trace \( \mathcal{T}_i^k \) is then a witness to the PSI-L property in trace \( \mathcal{T} \).
4 Problem Definition

Given a trace $T$, and an observation, the target, given as a PORV or an event $E$, we wish to find a set of PSI-L formulas that are valid throughout trace $T$.

We assume a bound $n \in \mathbb{N}$, $n \geq 0$, on the length of the prefix. We also take as input a resolution $k \in \mathbb{R}$ as a maximum delay between sub-expression in a prefix sequence, that is, it is assumed that every $\tau_i \in [0 : k], 1 \leq i \leq n$. Individual delay bounds are refined once a concrete PSI-L property is learned.

The user may provide a set of known predicates which form the predicate alphabet $\mathbb{P}$ used by the mining algorithm. The set $\mathbb{P}$ may also be extracted before hand using existing techniques [24] to learn interesting events in the trace. Parameterized predicates may also be learned using parameter optimization techniques, however this is not the focus of this article. We assume, for now, for ease of understanding, that the set $\mathbb{P}$ is given.

5 PSI-Arithmetic and Decision Making Metrics

Given a data-set, a decision tree is a tree structure of nodes representing queries about the data. The children of a query node are labeled with responses to the queries, with each child containing only the data satisfying the labels along the path from it to the root of the tree.

The process of constructing a decision tree is aimed at using query nodes to split the data-set to reduce the disorder in the data-sets at its child nodes. The leaf nodes in a decision tree are terminal nodes at which the data is homogenous. All other nodes are non-terminal nodes.

In this article we focus on binary decision trees. The query at each query node is a predicate from $\mathbb{P}$, and the children are labeled with either true or false. The error (chaos in the data) at a node is viewed from the perspective of the given target $E$. While many studies on decision trees exist, in this article we use terminologies for statistical terms as given in References [44, 2]. While the intuitive meaning for these terms remains the same as in standard literature, we redefine some terms to be consistent with the semantics of PSI-L properties.

At each query node, a predicate $P$ is chosen from $\mathbb{P}$ such that it minimizes the error in the resulting child nodes, that is the data at the child nodes is more homogenous than the parent node with respect to the truth of $E$. The process of node splitting continues until a leaf node in which $E$ is either valid or unsatisfiable is reached. Along with other constraints discussed in Section 9 we use a bound on the depth of the decision tree as the principal stopping criteria for the search along a path in the tree.

The labels along the path from the root to a node, together form a set of constraints on the data-set present at the node.

A summary of the methodology for mining prefix sequences is depicted in Figure 2. Initially, a predicate or event $E$ is presented as the target for which prefix sequences that appear to cause $E$ are to be mined. The set of traces is initially Booleanized (abstracted as a set of intervals) using the predicate alphabet $\mathbb{P}$ and $E$ (which may also form part of $\mathbb{P}$).

We use interval arithmetic to represent dense-time and handle time arithmetically, instead of as a series of samples, making the methodology robust to variations in the mechanism used for sampling the data. This also allows us to parameterize time delays between sequenced event, and compute the trade-offs involved while varying the temporal positions of the events.

Figure 2: Prefix Sequence Property Mining Workflow
Definition 3. Interval Set of a predicate \( P \) for trace \( T \): The Interval Set of a predicate \( P \) for trace \( T \), \( \mathcal{I}_T(P) \), is the set of all non-overlapping maximal time intervals, \([a,b)\); \( a < b \), in \( T \) where \( P \) is true. The interval \([t_i : t_j)\) is in \( \mathcal{I}_T(P) \) if \( \forall k \geq j - i \), \( T(k) = P \). The length of the interval set \( \mathcal{I}_T(P) \), denoted \( |\mathcal{I}_T(P)| \), is defined as, \( |\mathcal{I}_T(P)| = \sum_{i=0}^{n-1} |\mathcal{I}_T(P)(b-a)\) \( \) \( \)

The trace \( T \) is translated into a Truth Set, the set of all labeled interval sets for predicates in \( P \).

**Assumption:** It is assumed that the trace is the result of a sufficiently accurate sampling of the process under observation with respect to the choice of predicates in \( P \). This is possible to achieve during a simulation of a mixed-signal circuit. For more information, the interested reader may refer to Ref. [38].

Definition 4. Truth Set for trace \( T \) and Predicate Set \( P \): The Truth Set for \( P \), \( \mathcal{I}_T(P) \), is the set of all Interval Sets for the trace \( T \) of all predicates \( P \in \mathbb{P} \). \( \mathcal{I}_T = \{\mathcal{I}_T(P) | P \in \mathbb{P}\} \)

Definition 5. Constraint Set: A constraint set \( C \) is a set of constraints at a node in the decision tree. Each constraint is a pair of predicate and its position in the prefix-sequence, \((P,i)\), where \( P \in \mathbb{P} \), \( i \in [0 \ldots n] \), \( n \in \mathbb{N} \). The set of constraints in \( C \) can be expressed as a partial prefix in PSL-L. In the prefix-sequence \( s_n \tau_n s_{n-1} \tau_{n-1} \ldots \tau_1 s_0 \), the sub-expression \( s_i \), \( 0 \leq i \leq n \), is formed by the predicates \( P \) in the bucket \( \mathcal{B}_i(C) \). The prefix sequence formed from the constraint set \( C \) is given as \( \mathcal{S}_C \).

The reader may be reminded that a prefix sequence is a sequence of the form \( s_n \tau_n s_{n-1} \tau_{n-1} \ldots \tau_1 s_0 \), wherein our algorithm initially assumes that \( n \) is known and the maximum size of each \( \tau_i \), \( 0 \leq i < n \), is also known to be \( k \). Unless otherwise computed, it is assumed that each \( \tau_i \) is the interval \([0 : k] \). During the learning process the interval for \( \tau_i \) is unknown and is computed post-facto, that is once a decision tree is learned. Each position \( s_i \) in the sequence is a placeholder for a Boolean expression of events and PORVs, and may be empty. Initially all placeholders are empty and the learning algorithm to be introduced later will decide which predicates or events are placed in the various sequence locations. It is possible for some placeholders to remain empty in a sequence expression. In such cases, adjacent delay expressions merge to produce larger temporal delays.

Definition 6. Prefix-Bucket: For a constraint set \( C \), the prefix-bucket at position \( i \in \mathbb{N} \), given as \( \mathcal{B}_i(C) \), is the set of all pairs \((P,i) \in C \). The set of all buckets for a constraint set \( C \) is written as \( \mathcal{B}(C) \) or simply \( \mathcal{B} \) if constraint set \( C \) is known from context.

The terms prefix-bucket and bucket are used interchangeably in the following text and mean the same. When the constraint set \( C \) is known, we use the notation \( \mathcal{B}_i \) to mean \( \mathcal{B}_i(C) \).

For a constraint set \( C \), the interval set for bucket \( \mathcal{B}_i \), given as \( \mathcal{I}_T(\mathcal{B}_i) \), is the set of truth intervals where the constraints in \( C \) are all true.

The learning algorithm must place predicate and event constraints into various buckets. As mentioned earlier some buckets may remain empty, resulting in the delays in the sequence that appear before and after it to merge.

Definition 7. Interval Work-Set: An interval work-set \( \mathcal{W}_0 \) is a set \( \{\mathcal{I}_{\mathcal{B}_0},\ldots,\mathcal{I}_{\mathcal{B}_n}\} \) of labeled truth intervals for buckets \( \mathcal{B} = \{\mathcal{B}_0,\ldots,\mathcal{B}_n\} \) of constraint set \( C \). Different combinations of bucket truth intervals, produce a unique interval work-set. Given the set \( \mathcal{W}_0 \), the set \( \mathcal{W}_k = \{\mathcal{I}_{\mathcal{B}_j} | i \leq j \leq k\} \).

For trace \( T \), the set of all work sets that can be derived from \( \mathcal{I}_T(\mathcal{B}_1) \), \( 0 \leq i \leq n \), is given as \( \mathcal{W}_C \) or simply \( \mathcal{W} \) when the context \( C \) is known.

Sequence expression matches must be determined for each combination of sub-sequence expression truth intervals, that is for each unique interval work-set. We achieve this by computing the Forward Influence and Backward Influence for a sequence expression \( S \). In the following example we present a brief illustration and provide the intuition behind the use of the forward and backward influences, respectively defined in Definitions [S] and [9].

**Example 1.** Consider the sequence expression \( \mathcal{B}_2 \#\#[1 : 4], \mathcal{B}_1 \#\#[2 : 8], \mathcal{B}_0 \), and sequence expression truth interval sets \( \mathcal{I}_T(\mathcal{B}_2) = \{[2 : 4]\}, \mathcal{I}_T(\mathcal{B}_1) = \{[3 : 5], [7 : 9]\} \) and \( \mathcal{I}_T(\mathcal{B}_0) = \{[4 : \)
There are $1 \times 2 \times 2 = 4$ interval work-sets.

The computation of forward influence using Definition 8, for each possible interval work-set is described in the form of a tree in Figure 3. An interval work-set is the set of truth intervals of buckets encountered along a path from the root to a leaf node in the tree. The tree is rooted at a node corresponding to the truth interval $[2 : 4]$ for $B_2$. Level $i$ in the tree corresponds to the computation of $F(S, W^i_0)$. For each interval set, the backward influence is computed, using Definition 9. The computation begins with the leaves of the tree in Figure 3, and proceeds backwards through the sequence expression to determine the intervals corresponding to each match, indicated as a bottom-up computation in Figure 4.

Intuitively, for a sequence expression $s_k d_k \cdots d_2 s_1 d_1 s_0$, the forward influence from position $k$ to position 1, $F(S, W^i_k)$, computes the largest interval in $I_{B_1}$ that corresponds to a match computed for the sequence of intervals $I_{B_k}, \ldots, I_{B_1}$. In a sequence expression computing the forward influence is not sufficient to identify the sequence of intervals attributing to a match. We explain this using Figure 4. Observe the second column in Figure 4 corresponding to the interval work-set $W^0_0 = \{[2 : 4], [3 : 5], [12 : 19]\}$. From Figure 3, the forward influence computes the influence intervals to be $[2 : 4], [3 : 5], [12 : 13]$. The interval $[3 : 5]$ corresponds to the forward influence match up to $B_1$. The truth interval of $B_0$ under consideration for this match is the interval $[12 : 19]$. Of the truth interval $[3 : 5]$ of $B_1$, observe that $[3 : 4] \oplus [2 : 8] = [5 : 12], [5 : 12] \cap [12 : 19] = \emptyset$, which does not fall within the truth interval $[12 : 19]$ of $B_0$, and thus truth interval $[3 : 4]$ cannot contribute to a match. On the other hand, $[4 : 5] \oplus [2 : 8] = [6 : 13]$, and $[6 : 13] \cap [12 : 19] = [12 : 13]$. Therefore, of the interval $[3 : 5]$ only $[4 : 5]$ contributes to a match.

**Definition 8. Forward Influence $F(S, W^0_0)$:** The forward influence for a prefix sequence expression $S = s_n \tau_n s_{n-1} \cdots \tau_1 s_0$, given the interval work-set $W^0_0 = \{I_{B_0}, \ldots, I_{B_n}\}$, is an interval, recursively defined as follows:
For $S$, the set of intervals in $F(S, W^n_0)$ are called the end-match intervals of $S$.

**Definition 9. Backward Influence** $B(S, W^n_0, i)$: The backward influence, $B(S, W^n_0, i)$, $i \in [0, n]$, for a sequence expression $S = s_n \tau_n s_{n-1} \ldots \tau_1 s_0$, given the interval work-set $W^n_1 = \{I_{B_0}, I_{B_1}, \ldots, I_{B_n}\}$, is an interval defined as follows:

\[
B(S, W^n_0, 0) = F(S, W^n_0)
\]
\[
B(S, W^n_0, i) = (B(S, W^n_0, i - 1) \ominus d) \cap F(S, W^n_1), \quad 0 < i \leq n
\]

For $S$, the set of intervals in $B(S, W^n_0, n)$ are called the begin-match intervals of $S$.

We use the shorthand notation $F^n_0$ to represent $F(S, W^n_0)$, and $B^n_0$ to represent $B(S, W^n_0, i)$, for a given prefix sequence expression $S$ having $n$ buckets, and interval work-set $W^n_0$.

**Proposition 1.** For a potential infinite continuum of prefix sequence expression matches associated with an interval work-set, tight delay intervals between sub-expressions of the sequence can be computed using the backward influence.

It should be remembered that the prefix sequence expression $S = s_n \tau_n s_{n-1} \ldots \tau_1 s_0$ is constructed from the constraint set $C$. The delay terms between sub-expressions in $S$ are multiples of the resolution $k$ (as described in Section 4). Hence, we use $S$ and $C$ interchangeably in the text, wherever convenient.

**Definition 10. Influence Set** The influence set for a sequence expression for constraint set $C$, $S_C = s_n d_n s_{n-1} \ldots d_1 s_0$, given interval sets for each bucket, $I_{\tau}(B_i)$, $0 \leq i \leq n$, is defined as the union of end-match intervals over all possible work sets $W^n_0 \in W$, defined as follows: $I_{\tau}(S_C) = \bigcup_{W^n_0 \in W} F(S_C, W^n_0)$

**Definition 11. Length of a Truth Set:** The length of a Truth Set $|I_{\tau}(S)|$, under constraint set $C$, represented by $|I_{\tau}(S)|$, is defined as the length of the influence set, given as $|I_{\tau}(S_C)|$.

Properties over dense real-time may match over a continuum of time points. We use time intervals to represent truth points for predicates (Definition 3) and sets of predicates (Definition 4). A prefix sequence is built from a set predicate constraints (Definition 5), each predicate having a fixed position in the prefix sequence. Multiple predicates sharing the same position in the sequence form a prefix-bucket. A predicate can be true over multiple disjoint time intervals. All predicates in the same bucket are conjuncted together, and hence a bucket of predicates may be true over a set of intervals. Choosing combination of truth intervals, one truth interval from each bucket position, forms an interval work-set (Definition 7). For the prefix-sequence and a given work-set, the forward influence (Definition 8) provides a mechanism for computing the end time-points associated with the match of the prefix-sequence, while the backward influence (Definition 9) provides a mechanism for computing the begin time-points associated with the matching end time-points of the forward influence. The set of all end time-points for all work-sets of a prefix-sequence form a set of intervals where the prefix-sequence has influence, the influence set (Definition 10). The choice of constraints $C$ limits the length of the truth set (Definition 11) and determines the decisions made for mining additional constraints.

## 6 Decision Trees for Immediate Relations

The semantics of sequence expressions allow for both, immediate causality and future causality to be asserted. Immediate causality, expressed as $S \Rightarrow E$, is observed when the truth of the sequence $S$ at time $t$ causes the consequent $T$ to be true at time $t$. Future causality, expressed by the assertion $S \Rightarrow \##[a:b] E$ relates the truth of the sequence $S$ at time $t$ with the truth of $E$ at time $t' \in [t + a, t + b]$.

We first describe the decision making metrics we use for mining immediate relations, and thereafter present an algorithm that uses these metrics to construct a decision tree.
6.1 Metrics for Decision-Making

At each node the statistical measures of Mean and Error are used to evaluate the node. Standard decision tree algorithms use measures of Entropy or Gini-Index \[2\] to measure the disorder and chaos in the data.

For immediate relations of the form \( A \Rightarrow B \), where \( A \) and \( B \) are Boolean expressions, we use the standard Shannon Entropy as a measure of disorder, at a node, evaluated over time intervals. We use Information gain to evaluate decisions at internal nodes of the decision tree. We redefine the measure of entropy and information gain later in Section 5, adapting them for prefix-sequences. In this section, we use Error and Gain to respectively refer to Shannon Entropy and Information Gain evaluated over sets of truth intervals.

6.1.1 Mean

For countable Boolean data points, the data can be represented as a table with statistical counts for a class \( x \) being viewed in terms of the number of rows of the table in which \( x \) is labeled true. In a table of \( K \) rows, where \( J \) is the quantum of rows in the trace that are witness to \( x \)’s truth, i.e. state \( x \) is \( \top \), the arithmetic mean for \( x \) is \( \frac{J}{K} \). For traces describing real-time, since time is real-valued a table is no longer a viable form of representation for the data. We use time intervals that represent the truth of a predicate to deal with dense time. We adapt the definition of arithmetic mean as found in standard texts \[39\] for handling intervals of truth.

**Definition 12.** Mean\(_{\mathcal{T}}(E)\): For the target class \( E \), the proportion of time in trace \( \mathcal{T} \) that \( E \) is true in the trace constrained by \( \mathcal{C} \):

\[
\text{Mean}_{\mathcal{T}}(E, \mathcal{C}) = \frac{\mid \mathcal{I}(E) \cap \mathcal{I}(\mathcal{S}_E) \mid}{\mid \mathcal{I}(\mathcal{S}_E) \mid}
\]

The mean represents the conditional probability of \( E \) being true under the influence of the constraints in \( \mathcal{C} \). For convenience, we use \( \mu_{\mathcal{T}}(E) \) to refer to \( \text{Mean}_{\mathcal{T}}(E, \mathcal{C}) \).

6.1.2 Error

For a target \( E \), the error is a measure of entropy in the data with regards to the classes \( E \) and \( \neg E \). It is a measure of how well the set of constraints \( \mathcal{C} \) explain \( E \). An error value of zero indicates that there is no disagreement in the class \( (E \text{ or } \neg E) \) under the constraint set \( \mathcal{C} \).

**Definition 13.** Error\(_{\mathcal{T}}(E, \mathcal{C})\): For the target class \( E \), the error for the trace \( \mathcal{T} \) constrained by \( \mathcal{C} \) is defined as follows:

\[
\text{Error}_{\mathcal{T}}(E, \mathcal{C}) = -\mu_{\mathcal{T}}(E) \times \log_2(\mu_{\mathcal{T}}(E)) - \mu_{\mathcal{T}}(\neg E) \times \log_2(\mu_{\mathcal{T}}(\neg E))
\]

For convenience, we use \( \epsilon_{\mathcal{T}}(E) \) to refer to \( \text{Error}_{\mathcal{T}}(E, \mathcal{C}) \).

The choice of query at a query node is key to constructing a decision tree representing good quality prefix sequences. A query is a predicate chosen from the predicate set \( \mathcal{P} \). The utility of a query is measured by the reduction of error it would bring if used to split the truth set at a query node. This utility metric is called the gain. While many gain metrics exist \[2\], we find Information Gain to work best for our two class application. Furthermore, Information Gain works well when there are two classes. While, in Section 6 we use the definitions presented here, in Section 7 we redefine information gain to cope with issues arising from the nature of the problem of mining temporal sequences. The standard definition of Information Gain proposed in Ref. \[10\] adapted to sets of intervals is given below.

**Definition 14.** Gain: The gain (improvement in error) of choosing \( P \in \mathcal{P} \) to add to constraint set \( \mathcal{C} \), at a node having error \( \epsilon \) is as follows:

\[
\text{Gain} = \epsilon - \frac{\mid \mathcal{I}(\mathcal{S}_{\mathcal{C} \cup \{P\}}) \mid}{\mid \mathcal{I}(\mathcal{S}_\mathcal{C}) \mid} \times \epsilon_{\mathcal{T}}(\mathcal{S}_{\mathcal{C} \cup \{P\}}(E)) - \frac{\mid \mathcal{I}(\mathcal{S}_{\mathcal{C} \cup \{\neg P\}}) \mid}{\mid \mathcal{I}(\mathcal{S}_\mathcal{C}) \mid} \times \epsilon_{\mathcal{T}}(\mathcal{S}_{\mathcal{C} \cup \{\neg P\}}(E))
\]

In Definition 14, we assume predicate \( P \) is always to be placed in bucket \( \mathcal{B}_0 \). We deal with an extended Definition of gain in Section 7 when dealing with the algorithm for mining prefixes.
Algorithm 1: Miner: Mining Immediate Assertions from Traces

Input: Truth Set $I_T(P)$ for trace $T$, Predicate List $P$, Target $E$, Constraint Set $C$.
Output: Prefix Set $A$.

1. if stoppingCondition($I_T(P)$, $P$, $E$, $C$) then;
2. return;
3. $m \leftarrow \mu_C(E); e \leftarrow \epsilon_C(E);$
4. if $e = 0$ then
5. if $m = 0$ then $A \leftarrow A \cup \{C \Rightarrow \neg E\};$
6. else $A \leftarrow A \cup \{C \Rightarrow E\};$
7. return;
8. end
9. $P_{best} \leftarrow \phi; g_{best} \leftarrow -\infty;$
10. forall $P \in P$ do
11. $g \leftarrow e - \epsilon_{C \cup \{P\}}(E) - \epsilon_{C \cup \{\neg P\}}(E);$
12. if $g > g_{best}$ then $P_{best} \leftarrow P; g_{best} \leftarrow g ;$
13. end
14. Miner($I_T(P)$, $P \setminus \{P_{best}\}, C \cup \{P_{best}\});$
15. Miner($I_T(P)$, $P \setminus \{P_{best}\}, C \cup \{\neg P_{best}\});$

6.2 A Miner for Immediate Relations

We first describe the algorithm for mining immediate causality where the sequence expression is a single Boolean expression of predicates and events, and extend this algorithm to mine concurrent assertions over sequence expressions of arbitrary temporal length.

The input to the miner is the truth set for predicates in the predicate alphabet $P$. Algorithm 1 learns a decision tree to characterize a target event or predicate $E$. At every query node of the tree, the measures described in Section 6.1 are used to add a predicate $P$, that maximizes the gain, to the constraint set $C$. The constraint set $C$ at a node is the set of predicate truth choices made along the path following the parent links up to the root. The algorithm only mines immediate causal relations.

Example 2. Consider a run of Algorithm 1 on the truth set shown in Table 1 with $P_3$ as the target predicate. The truth set is obtained by Booleanizing the signals $x$ and $y$ in Figure 5 with predicates $P_1 \equiv x \geq 0.9 \times V_r$, $P_2 \equiv y \geq 0.1 \times V_r$ and $P_3 \equiv z \geq x + y$, where $V_r$ is a constant. The temporal length of the trace $T$ is 13.

| List     | Truth Time Intervals                           |
|----------|-----------------------------------------------|
| $I_T(P_1)$ | $\{[2 : 4), [7 : 8), [9 : 9.5), [10 : 13)\}$ |
| $I_T(P_2)$ | $\{[0 : 1), [5 : 5.5), [9.8 : 9.9)\}$         |
| $I_T(P_3)$ | $\{[1 : 5), [6 : 9.5), [10 : 13)\}$          |

Table 1: Interval Set $I_T(P)$.

The decision tree produced by Algorithm 1 is shown in Figure 6.

At the root node (line 1 of Algorithm 1) the $\langle$mean, error$\rangle$ tuple is calculated for predicate $P_3$ to be $(0.81, 0.21)$. At line 5, for each predicate, the gain is computed. When computing gain, the algorithm must choose between $P_1$ and $P_2$ to branch on.

Choosing $P_1$ for the partitioning step, causes the interval sets to be constrained as follows:
Choosing $P_1$ for the partitioning, the $(\mu, \epsilon)$ are $(1, 0)$ when $P_1$ is true and $(0.61, 0.28)$ when false. The gain computed for the choice of $P_1$ is $0.21 - 0.28 = -0.07$. Similarly choosing $P_2$ for the partitioning step, $(\mu, \epsilon)$ are $(0, 0)$ when $P_2$ is true and $(0.92, 0.11)$

When choosing $P_1$ for the partitioning, the $(\mu, \epsilon)$ are $(1, 0)$ when $P_1$ is true and $(0.61, 0.28)$ when false. The gain computed for the choice of $P_1$ is $0.21 - 0.28 = -0.07$. Similarly choosing $P_2$ for the partitioning step, $(\mu, \epsilon)$ are $(0, 0)$ when $P_2$ is true and $(0.92, 0.11)$.

Choosing $P_1$, the $(\mu, \epsilon)$ are $(1, 0)$ when $P_1$ is true and $(0.61, 0.28)$ when false. The gain computed for the choice of $P_1$ is $0.21 - 0.28 = -0.07$. Similarly choosing $P_2$ for the partitioning step, $(\mu, \epsilon)$ are $(0, 0)$ when $P_2$ is true and $(0.92, 0.11)$

Choosing $P_2$ for the partition produces the following constrained lists:

Choosing $P_2 = \top$

Choosing $P_2 = \bot$

When choosing $P_1$ for the partitioning, the $(\mu, \epsilon)$ are $(1, 0)$ when $P_1$ is true and $(0.61, 0.28)$ when false. The gain computed for the choice of $P_1$ is $0.21 - 0.28 = -0.07$. Similarly choosing $P_2$ for the partitioning step, $(\mu, \epsilon)$ are $(0, 0)$ when $P_2$ is true and $(0.92, 0.11)$.

Figure 5: Truth waveforms for predicates $P_1 \equiv x \geq 0.9 \times V_r$, $P_2 \equiv y \geq 0.1 \times V_r$ and $P_3 \equiv z \geq x + y$, where $V_r$ is a constant.

Figure 6: Decision tree generated for Interval Sets of Table 1 and target $P_3$. 
when $P_2$ is false, resulting in a gain of $0.21 - 0.11 = 0.1$. The gain from choosing $P_2$ is larger (indicating a better correlation with $P_3$), hence, at line 6 in Algorithm 2 the predicate chosen in $P_2$. Thereafter, $P_2$ is added to the set of constraints $C$ in recursive calls to the Miner. The decision tree then branches on $P_2$. In one recursive call $P_{best}$ in added to $C$, while in the other $\neg P_{best}$ is added.

Every path in the decision tree to a leaf node represents a set of constraints under which the target $P_3$’s truth is homogenous (either exclusively true or exclusively false). A leaf node in the decision tree is a node at which the error is zero, indicating a 100% confidence for the assertion generated therein. In the decision tree of Figure 7 the child node corresponding to the constraint set $C = \{P_2\}$ results in a zero error node indicating that an assertion is generated here. Since the mean at the node is zero, it indicates a correlation with the negation of the target $P_3$. Therefore the assertion generated at this node would be $P_2 \Rightarrow \neg P_3$.

At the child node where $P_2$ is false, since $P_1$ is the only predicate left for splitting, in a manner similar to the one discuss earlier, the data is split on the truth of $P_1$, as follows:

Choosing

$$I_{T_{\neg P_2}}(P_1) = \{[2 : 4), [7 : 8), [9 : 9.5), [10 : 13)]\}$$

$P_1 = \top$

$$I_{T_{\neg P_2, (P_3)}(P_1)} = \{[2 : 4), [7 : 8), [9 : 9.5), [10 : 13)]\}$$

Choosing

$$I_{T_{\neg P_2} \neg P_3(P_1)} = \{[1 : 2), [5.5 : 7), [8 : 9), [9.9 : 10)]\}$$

$P_1 = \bot$

$$I_{T_{\neg P_2} \neg P_3(P_3)} = \{[1 : 2), [6 : 7), [8 : 9]}$$

The values of $\langle \text{mean, error} \rangle$ under the constraints $\{\neg P_2, P_1\}$ and $\{\neg P_2, \neg P_1\}$ are respectively, $(1, 0)$ and $(0.83, 0.28)$. In the decision tree, on splitting, one child node has a non-zero error, indicating that the constraints at that node are still inconsistent with the target $P_3$. Therefore at this stage, since we have no further predicates to refine the dataset, the algorithm terminates. The other child node with the assignment $\{P_1, \neg P_2\}$ results in a zero error node, with a mean of one, asserting a correlation with the positive occurrence of the consequent $P_3$. Therefore the assertion generated at this node is $(P_1 \land \neg P_2) \Rightarrow P_3$.

\[ \square \]

7 Generalizing Prefixes to Sequence Expressions

Applying traditional decision tree learning on the vanilla truth-set would yield only immediate properties as described in Section 6. The methodology needs to be appropriately adapted to mine sequence expressions relating events over time with the target $E$. To mine such prefix sequences, $E$’s truth must be tested with the truth of other predicates over past time points. This would allow us to compute the influence a predicate has on the truth of $E$ over time. This is achieved by using pseudo-targets that allow us to evaluate constraints that have past influence on time-points of truth for the target.

Standard measures used for decision making in Section 6 are not suitable for evaluating the goodness of a decision involving pseudo-targets. The classification task we deal with is aimed at classifying time points describing behaviours that explain when the target is true or when it is false. While this is similar to a standard two class classification task, it is not. Due to the non-deterministic semantics of prefix-sequences, the two classes may share end-match time points, requiring an adapted definition of Shannon Entropy. Similarly, the decision made at each node of the decision tree splits the data-set in a manner that allows sharing of time-points between the branches of the split, thus requiring a special handling of gain. We therefore introduce variations of these measures enabling the best greedy decision be taken. The proposed measures are evaluated using a correlation-coverage metric to measure the proportion of the target’s truth covered by the mined prefixes.
Figure 7: Truths of Predicates=\{P, Q, R, E\} and Pseudo-Targets (n=3, k=0.4) for E. The horizontal red bands indicate where the predicate is false, while the green bands indicate when the predicate is true.

7.1 Pseudo-Targets for Sequence Expressions

Observe the truth set of predicates P and E in Figure 7. The horizontal green bands indicate time intervals where the predicate is true, while the red bands indicate intervals where the predicate is false. It is clear that the truths of P and E do not align. Our intention is to learn a temporal relation between P and E (if such a relation exists). When learning relations in the presence of an alphabet of predicates, it is expensive to examine the relationships between every pair of predicates in the alphabet and their relationship in turn with the target. Additionally it is important to have domain knowledge about the system to determine the quantum of time in the past of E that a predicate would be expected to have an influence on it, if at all. In order to scale such an analysis to large predicate alphabets we use pseudo-targets. Pseudo-targets allow us to compute summary statistical measures that give clear indications of the existence or absence of such relations.

Definition 15. Pseudo-Target: A pseudo-target is an artificially created target computed by stretching the truth of the targets interval set back in time by a multiple of the delay resolution k. The target E stretched back in time by an amount \(i \times k\) is denoted as \(E^i\). The interval list for the pseudo-target \(E^i\) in trace \(T\) is computed as follows:

\[
I_T(E^i) = I_T(E) \ominus [0 : i \times k] \tag{1}
\]

and,

\[
I_T(P) \ominus [a : b] = \bigcup_{I \in I_T(P)} I \ominus [a : b] \tag{2}
\]

where, the \(\ominus\) represents the Minkowski difference between intervals: \([\alpha : \beta] \ominus [a : b] = [\alpha - b : \beta - a]\).

The resolution and the length \(n\) of the sequence expression are meta-parameters of the algorithm that is built. The number of pseudo-targets generated is \(n\).

For a prefix sequence with at-most \(n\) sub-expressions and a delay resolution \(k\), we compute truth set, \(\hat{I}_T(P)\) from \(I_T(P)\) as follows:

\[
\hat{I}_T(P) = I_T(P) \cup \left( \bigcup_{1 \leq i \leq n} I_T(E^i) \right) \tag{3}
\]
Example 3. In Figure 7, \( n = 3 \) and \( k = 0.4 \), the truth intervals of three pseudo-targets of predicate \( E \) are shown. A pseudo-target \( E^2 \) is computed according to Equation 7. For instance, given \( \mathcal{I}_r(E) = \{ [5 : 8.3], [11.8 : 13], [18 : 20] \} \) and \( \mathcal{I}_r(\neg E) = \{ [0 : 5], [8.3 : 11.8], [13 : 18] \} \), \( \mathcal{I}_r(E^2) \) and \( \mathcal{I}_r(\neg E^2) \) are computed to be as follows:

\[
\mathcal{I}_r(E^2) = \{ [5 : 8.3], [11.8 : 13], [18 : 20] \} \cup [0 : 0.8] \\
= \{ [4.2 : 8.3], [11 : 13], [17.2 : 20] \}
\]

\[
\mathcal{I}_r(\neg E^2) = \{ [0 : 5], [8.3 : 11.8], [13 : 18] \} \cup [0 : 0.8] \\
= \{ [0 : 5], [7.5 : 11.8], [12.2 : 18] \}
\]

Observe that in Figure 7, the true and false intervals for pseudo-targets are not complementary. We wish to mine prefixes to explain both \( E \) and \( \neg E \). Hence while generating pseudo-targets, Equation 3 is also used to generate the pseudo-truth intervals for when \( E \) is false. Due to the non-deterministic match semantics of PSI-L, described in Section 3.2, the stretched portions of the targets intervals for its true state and false state overlap. This overlap does create confusion in computing the metrics for decision making at query nodes of the decision tree. We describe this in the following section.

7.2 Effect of Pseudo-Targets on Decision Making

In the building of a traditional decision tree (as in Algorithm 1), at each decision node, statistical measures, \( \text{Mean}_{\mathcal{T}_c}(E) \) and \( \text{Error}_{\mathcal{T}_c}(E) \), are consistently computed with respect to a single target, in this case \( E \).

Here, our aim is to build temporal sequences of Boolean formulas that explain the target’s truth. At each decision node, we must decide which predicate best reduces the error in the resulting split, while simultaneously choosing a temporal position for the predicate in the \( n \)-length prefix sequence. We achieve the later by choosing to test a predicate with each pseudo-target, to identify which pseudo-target (and therefore which position), given a possibly non-empty partial prefix, is most correlated with the predicate under test. The choice of predicate and position that gives the best correlation is then chosen. We must still decide what measure is best to determine correlation. In the rest of this section, we describe two methods for computing error, the challenges involved, and demonstrate why one is superior to the other.

At each query node of the decision tree, with constraint set \( C \), for target \( E \), we wish to carry out the following broad steps:

1. Compute \( \text{Mean}_{\mathcal{T}_c}(\hat{E}) \) and \( \text{Error}_{\mathcal{T}_c}(\hat{E}) \).
2. For each \( \langle P, i \rangle \), where \( P \in \mathbb{P}, 1 \leq i \leq n \), and \( \langle P, i \rangle \notin C \), \( \langle \neg P, i \rangle \notin C \):
   (a) \( C_1 = C \cup \langle P, i \rangle \), \( C_0 = C \cup \langle \neg P, i \rangle \).
   (b) Compute \( \text{Mean}_{\mathcal{T}_c}(\hat{E}) \), \( \text{Error}_{\mathcal{T}_c}(\hat{E}) \), \( \text{Mean}_{\mathcal{T}_c^0}(\hat{E}) \), \( \text{Error}_{\mathcal{T}_c^0}(\hat{E}) \).
   (c) Compute the gain on splitting the current node on \( \langle P, i \rangle \).
3. Report the arguments \( \langle P^*, i^* \rangle \) that contribute the best gain from Step 2.

In the core steps of the above procedure, namely Step 1 and Step 2, it is yet unclear how the statistical measures are to be computed. The definition of \( \text{Mean}_{\mathcal{T}_c}(\hat{E}) \) and \( \text{Error}_{\mathcal{T}_c}(E) \) in Section 6.1 assume that the true and false interval lists of \( E \) are compliments of each other, however, for a pseudo-target \( E \) this is not true, hence the metrics cannot be directly applied. Furthermore, even though in each iteration of Step 2 a choice of \( \langle P, i \rangle \) is made, it is unclear with respect to which pseudo-target the measures must be computed. We first resolve the later and then address the computation of the statistical measures.

Note that in Step 2 (a), for a predicate \( P \), once a pseudo-target position is determined, the split considers \( P \) being true in one node and \( P \) being false in the other, while the temporal position for \( P \) and \( \neg P \) remains the same for both child nodes of the split.
### 7.2.1 Choosing the Pseudo-Target for Mean and Error

The reader may recall that an assertion in PSI-L has the following syntax:

\[ S \Rightarrow E \quad \text{or} \quad S \Rightarrow \tau_0 \ E \]

where, the prefix sequence \( S \) of length \( n \) is of the form \( s_n \ s_{n-1} \ \ldots \ s_0 \). In the prefix sequence each \( s_i \) is a bucket at position \( i \), a Boolean expression of PORVs and events. For a prefix sequence of maximum length \( n \), at a query node with constraint set \( C \), some bucket positions may be empty (interpreted as the Boolean expression \( \text{true} \)).

In the case when \( S \Rightarrow E \), the last sub-expression is \( s_0 \). The alternate syntax \( S \Rightarrow \tau_0 \ E \) is developed to take into account the cases wherein the last sub-expression is \( s_i \), \( i > 0 \). In such cases, the forward match of the sequence (Definition 8) must be further stretched by \( i \) delay intervals to match with \( E \). These delay intervals are coalesced into \( \tau_0 \).

**Example 4.** Consider evaluating the constraint set \( C = \{(Q, 3), (P, 2)\} \). \( \mathcal{B}_0(C) = \{(Q, 3)\} \), \( \mathcal{B}_1(C) = \{} \) and \( \mathcal{B}_0(C) = \{\} \). \( \mathcal{T}_r(B_3) = \{[0.9 : 4.2], [6.3 : 11.4], [14.5 : 18.5]\} \), and \( \mathcal{T}_r(B_2) = \{[4.3 : 6], [9.4 : 12.2], [13.2 : 14], [17.3 : 20]\} \). The partial prefix sequence that results from \( C \) is \( S \equiv Q \# \langle 0:0.4 \rangle P \).

According to the semantics defined in Section 3.3, a match of \( S \) implies that \( E \) is true after two delay intervals. This then asserts \( Q \# \langle 0:0.4 \rangle P \Rightarrow \# \langle 0:0.8 \rangle E \). In this case therefore, evaluating \( C \) requires using pseudo-target \( E^2 \) as shown in Figure 7.

**Proposition 2.** The relevant pseudo-target for constraint set \( C \) is the smallest position non-empty bucket \( B_i \), \( 0 \leq i \leq n \) for \( C \).

### 7.2.2 Adapting Mean and Error for Pseudo-Targets

The measure of Error in Definition 13 assumes that classes are independent. Standard decision tree algorithms assume this to be true since there is usually no data point in the data set belongs to more than one class. However, for a pseudo-target this is not the case, because, the true and false states of the pseudo-target overlap due to the non-deterministic manner in which the intervals are stretched. Hence a traditional error computation which assumes independence between classes would misrepresent the relationships that exist.

Furthermore, at each decision node, we make two decisions, first deciding which predicate to pick, and second deciding which temporal position (pseudo-target) gives the best gain for the chosen predicate. Pseudo-targets are only representations of the learning objective for learning correct temporal positions for a predicate. Hence, we do not treat all pseudo-targets as part of the set of classes (Otherwise, for \( n \) pseudo-targets there would then be \( 2 \times (n + 1) \) classes).

In Example 4 we present the computation of mean and error using the definitions given in Section 6.1 to motivate the choices for adapting the traditional definitions of Information Gain.

**Example 5.** Consider the constraint set \( C = \{(Q, 3), (P, 2)\} \) from Example 4 and the constraint set \( \tilde{C} = \{(H, 3), (P, 2)\} \), where the truth of predicate \( H \) is shown in Figure 8. Consider computing \( \text{Mean}_T(E^2) \) and \( \text{Error}_T(E^2) \) for \( C \) and \( \tilde{C} \), according to Section 6.1.

We first compute the Influence Set, according to Definition 10.

- \( \mathcal{S}_C = Q \# \langle 0:0.4 \rangle P \). Therefore, \( \mathcal{I}(\mathcal{S}_C) = \{[4.3 : 4.6], [9.4 : 11.8], [17.3 : 18.9]\} \). \( |\mathcal{I}(\mathcal{S}_C)| = (4.6 - 4.3) + (11.8 - 9.4) + (18.9 - 17.3) = 4.3 \).

- \( \mathcal{S}_\tilde{C} = H \# \langle 0:0.4 \rangle P \). Therefore, \( \mathcal{I}(\mathcal{S}_\tilde{C}) = \{[4.3 : 4.6], [9.4 : 11.8]\} \). \( |\mathcal{I}(\mathcal{S}_\tilde{C})| = (4.6 - 4.3) + (11.8 - 9.4) = 2.7 \).

18
Figure 8: Predicates=\{P,Q,H,E\} and Pseudo-Targets (n=3, k=0.4) for E. The horizontal red bands indicate where the predicate is false, while the green and turquoise blue bands indicate when a predicate is true. C = \{\{Q,3\}, \{P,2\}\} and \(\hat{C}\) = \{\{H,3\}, \{P,2\}\} are constraint sets; while \(F(S_C, W_2^3)\) and \(F(S_{\hat{C}}, W_2^3)\) are their respective end match influence intervals.

The predicate H is similar in truth to Q, however unlike Q is false in the interval [14.5 : 18.5].

\[
\begin{align*}
\mu_{T_c}(E^2) &= \frac{|I_{T_c}(E^2) \cap I(S_C)|}{|I(S_C)|} \\
&= \frac{(0.3 + 0.8 + 1.6)}{(4.3)} = \frac{(2.7)}{(4.3)} \\
&= 0.6279
\end{align*}
\]

\[
\begin{align*}
\epsilon_{T_c}(E^2) &= - (-0.12690) - (-0.08064) \\
&= 0.04654
\end{align*}
\]

\[
\begin{align*}
\mu_{T_c}(\neg E^2) &= \frac{|I_{T_c}(\neg E^2) \cap I(S_C)|}{|I(S_C)|} \\
&= \frac{(0.3 + 2.4 + 0.7)}{(4.3)} = \frac{(3.4)}{(4.3)} \\
&= 0.7907
\end{align*}
\]

\[
\begin{align*}
\mu_{T_c}(E^2) &= \frac{|I_{T_c}(E^2) \cap I(S_{\hat{C}})|}{|I(S_{\hat{C}})|} \\
&= \frac{(0.3 + 0.8)}{(2.7)} = \frac{(1.1)}{(2.7)} \\
&= 0.4074
\end{align*}
\]

\[
\begin{align*}
\epsilon_{T_c}(E^2) &= - (-0.15888) - (0) \\
&= 0.15888
\end{align*}
\]

However, observe that the partial prefix \(S_C\) has a non-zero error when related to \(E^2\) (indicated in red for \(F(S_C, W_2^3)\) in the figure). On the other hand, although the computation above indicates a non-zero error when considering \(\hat{C}\), in the figure for the partial prefix \(S_{\hat{C}}\), visually one observes no error (indicated by the blue band overlayed on \(\neg E^2\)), indicating the assertion \(H \# \# [0:0.4] P \not\Rightarrow # # [0:0.8] \neg E\).

From Example 5 it is clear that using the measures of mean and error from Section 6.1, it is possible to miss potential relations that may exist in the data presented in Figure 8. This is primarily due to the fact that the measures ignore the overlap of the truth intervals of a pseudo-target’s true and false state.

**Definition 16. JointError** \(\mu_{T_c}(E^2)\): For the target class \(E^2\), the joint error for the trace \(T\) constrained by \(C\) is defined as follows:

\[
J E_{T_c}(E^2) = \epsilon_{T_c}(E^2) + \mu_{T_c}(E^2 \land \neg E^2) \times \log_2(\mu_{T_c}(E^2 \land \neg E^2))
\]
While computing the Joint Error, the term \( \mu_{\mathcal{T}_c}(E^i \land \neg E^j) \times \log_2(\mu_{\mathcal{T}_c}(E^i \land \neg E^j)) \) represents the entropy in the region of the overlap. By adding this term, we effectively ensure that the entropy from the overlap is considered only once when computing the Joint Error.

**Example 6.** In Example 3 on computing the Joint Error for the two constraint sets \( \mathcal{C} \) and \( \hat{\mathcal{C}} \) we get the following:

\[
\mu_{\mathcal{T}_c}(E^2 \land \neg E^1) = \frac{|\mathcal{T}_{\mathcal{T}_c}(E^2 \land \neg E^1) \cap \mathcal{T}(\mathcal{S}_c)|}{|\mathcal{T}(\mathcal{S}_c)|} = \frac{(0.3 + 0.8 + 0.7)}{(4.3)} = 0.4186
\]

\[
J E_{\mathcal{T}_c}(E^1) = 0.20754 + (-0.15831) = 0.04923
\]

\[
\mu_{\mathcal{T}_c}(E^2 \land \neg E^2) = \frac{|\mathcal{T}_{\mathcal{T}_c}(E^2 \land \neg E^2) \cap \mathcal{T}(\mathcal{S}_c)|}{|\mathcal{T}(\mathcal{S}_c)|} = \frac{(0.3 + 0.8)}{(2.7)} = 0.4074
\]

\[
J E_{\mathcal{T}_c}(E^1) = 0.15888 + (-0.15888) = 0.0
\]

The use of Joint Error thus correctly depicts the associations present in the data under the constraints being considered.

Similar to the challenges posed by the overlapping of classes, when considering a predicate and temporal position for splitting a node, the two nodes that result from such a split can have overlaps in the data points they represent.

**Example 7.** Consider the split of the data-set resulting from adding \( H \) to the constraint set \( \{(P, 2)\} \) to obtain \( \mathcal{C}_0 = \{\neg H, 3\}, \{P, 2\} \) and \( \hat{\mathcal{C}}_1 = \{H, 3\}, \{P, 2\} \). Due to the non-deterministic semantics of the temporal operator \( [\# [a:b] \), the forward influence of \( \hat{\mathcal{C}}_0 \) and \( \hat{\mathcal{C}}_1 \) overlap.

\[
F(\mathcal{S}_{C_0}, \mathcal{W}_2) = \{[4.3 : 6], [11.4 : 12.2], [13.2 : 14], [17.3 : 20]\}
\]

\[
F(\mathcal{S}_{C_1}, \mathcal{W}_2) = \{[4.3 : 4.6], [9.4 : 11.8]\}
\]

The two sets overlap at all time-points in the intervals [4.3 : 4.6] and [11.4 : 11.8]. Due to these overlaps, the sum of the weights in the definition of Gain in Definition 14,

\[
\frac{|F(\mathcal{S}_{C_0}, \mathcal{W}_2)|}{|F(\mathcal{S}_{\{(P, 2)\}})|} + \frac{|F(\mathcal{S}_{C_1}, \mathcal{W}_2)|}{|F(\mathcal{S}_{\{(P, 2)\}})|} > 1.
\]

Furthermore, the influence set of intervals for the constraint set \( \{(P, 2)\} \) is \( \mathcal{I}(\mathcal{S}_{\{(P, 2)\}}) = \{[4.3 : 6], [9.4 : 12.2], [13.2 : 14], [17.3 : 20]\}. Since the temporal position of \( H \) is earlier than that of \( P \), \( F(\mathcal{S}_{C_0}, \mathcal{W}_2) \cup F(\mathcal{S}_{C_1}, \mathcal{W}_2) = \mathcal{I}(\mathcal{S}_{\{(P, 2)\}}) \), however this need not always be the case. If \( H \) is placed later than \( P \) in the sequence, the forward influence list can change substantially, since it would now be contained in the interval list of \( H \).

Due to both, potential overlapping data-points between the child nodes of a split and the data-points after the split being potentially different from those of the parent node, a revised definition of Gain is defined.

**Definition 17. Joint Gain:** The gain (improvement in error) of choosing to split on \( P \in \mathcal{P} \), at bucket position \( b \), given the existing constraint set \( \mathcal{C} \), at a node having error \( \epsilon \) as follows:

\[
\text{JointGain}(\epsilon, \mathcal{C}, P, b) = \epsilon - \alpha(\mathcal{C}, P, b) \times \epsilon_{\mathcal{T}_{\mathcal{C}\cup\{(P,b)\}}}(E) - \alpha(\mathcal{C}, \neg P, b) \times \epsilon_{\mathcal{T}_{\mathcal{C}\cup\{\neg(P,b)\}}}(E)
\]

where,

\[
\alpha(\mathcal{C}, P, b) = \frac{|F(\mathcal{S}_{\mathcal{C}\cup\{(P,b)\}}, \mathcal{W}_i^j)|}{|F(\mathcal{S}_{\mathcal{C}\cup\{(P,b)\}}, \mathcal{W}_i^j)| + |F(\mathcal{S}_{\mathcal{C}\cup\{\neg(P,b)\}}, \mathcal{W}_i^j)|}
\]

and \( i \) and \( j \) are respectively the smallest and largest non-empty bucket indexes in \( \mathcal{C}\cup\{(P,b)\} \).

**7.3 A Miner for Prefix Sequences**

The prefix sequence inference mining algorithm is presented as Algorithm 2. The length of the sequence \( n \), and the delay resolution \( k \) are meta-parameters of the algorithm. Every choice of \( n \) and \( k \) yields a different instance of the algorithm. The choice of values of the meta-parameters
must come from the domain. The algorithm learns a decision tree for PSI properties for the truth set $\hat{I}_T(\mathcal{P})$ for a choice of $n$ and $k$.

In the algorithm $\beta_{E\ell}(\cdot)$ is used as shorthand for $J_E\tau_\ell(\cdot)$ as given in Definition 16. Line 2 tests the current node for termination. One of the criteria for terminating is that the node is homogenous with respect to $E$, that is the error at the node is zero. Other stopping conditions are described later in Section 9.

In Line 3 the smallest non-empty bucket is computed from the constraint set $C$, and is used in Line 7 to compute the error for $C$. The loop at Line 5 iterates over every pseudo-target position, while Line 6 chooses a predicate from the predicate alphabet $\mathcal{P}$. Any predicate and pseudo-target position combination already present in $C$ are ignored in Line 6. The Joint Gain for the choice of predicate and pseudo-target is computed in Line 7 and the best gain, and its associated arguments are determined in Line 8. The computation of Gain uses the smallest non-empty bucket index for the choice of pseudo-target, with respect to which the Entropy and Gain are computed. Once a best predicate and position in the prefix is decided, lines 9 and 10 branch on the new constraint sets.

### 7.4 From Decision Trees to PSI-L Formulae

The nodes in the decision tree at which the error is zero are the leaf nodes of the tree and have homogenous data with respect to the truth of $E$. We call these nodes PSI nodes and the labels along the path from the a PSI node to the root (the constraint set $C$) form a PSI property template. A template consists of predicates and their relative sequence position from the target. Concretizing the PSI template involves computing the relative positions of predicates from each other. We do this by grouping predicates that fall in the same relative temporal position into a bucket, and then compute tight time delays that separate buckets in order of their temporal distance from the target $E$. The computation of tight separating intervals between buckets assumes an any-match semantic.

Given buckets $B_0, B_1, \ldots, B_n$, the assertion constructed has one of the following forms:

$$B_n \tau_n B_{n-1} \tau_{n-1} \ldots \tau_1 B_0 \Rightarrow T$$
when $B_0 \neq \phi$

$$B_n \tau_n B_{n-1} \tau_{n-1} \ldots \tau_2 B_1 \tau_1 \Rightarrow T$$
otherwise

We wish to compute tight intervals, $\tau_i$, $1 \leq i \leq n$.

We define a widening operation for a set of intervals $\mathcal{I}$ as follows:

$$W(\mathcal{I}) = [\min_{I \in \mathcal{I}} l(I) : \max_{I \in \mathcal{I}} r(I)]$$

At a PSI node, the set of constraints $C$ is known. We use the notation $B_i$, for bucket $i$, to denote the set of predicates having influence on the target with a step size of $[0 : i \times k]$. The interval set for $B_i$ is given as follows:
\[
\mathcal{I}_{\mathcal{T}_E}(B_i) = \{ I \mid I = \bigcap_{P_j \in B_i} I_j \text{ for some } I_j \in \mathcal{I}_{\mathcal{T}_E}(P_j) \}
\]

We compute a tight delay separation interval, \(Sep_{\mathcal{C}}(B_i, E)\), for \(i > 0\) of \(B_i\) from \(E\) as follows:

\[
Sep_{\mathcal{C}}(B_i, T) = \mathcal{W}(I = ((I_{B_i} \oplus [0 : i \times k] \cap I_T) \ominus I_{B_i}) \cap [0 : i \times k])
\]

for \(I_{B_i} \in \mathcal{I}_{\mathcal{T}_E}(B_i)\) and \(I_T \in \mathcal{I}_{\mathcal{T}}(T)\)

The separation \(\tau_i\) between \(B_i\) and \(B_{i-1}\) is computed as follows:

\[
\tau_i = \begin{cases} 
Sep_{\mathcal{C}}(B_i, T) \ominus Sep_{\mathcal{C}}(B_{i-1}, T) & , 1 \leq i \leq n, B_0 \neq \phi \\
Sep_{\mathcal{C}}(B_i, T) \ominus Sep_{\mathcal{C}}(B_{i-1}, T) & , 2 \leq i \leq n, B_0 = \phi \\
Sep_{\mathcal{C}}(B_1, T) & , i = 1
\end{cases}
\]

8 Ranking Mined PSI-L Properties

The decision tree learned by Algorithm 2 can compute several prefixes. It is important to rank these in terms of those that are likely to be causal relations and those that are not. Furthermore, it is also important to understand how the mined prefixes are related to the trace.

We measure the goodness of a PSI property \(S \Rightarrow E^i\), where \(i\) is the smallest non-empty bucket index in \(S\), using heuristic metrics of Support, and Correlation. We also measure trace covered for the set of PSI properties generated.

Definition 18. Support: For a property \(S \Rightarrow \hat{E}\), the quantum of time for which \(S\) is true in the trace is the support of \(S \Rightarrow E^i\).

\[
Support(S \Rightarrow E^i) = \frac{||\mathcal{I}(S)||}{||\mathcal{T}||}
\]

where \(\mathcal{I}(S)\) is the influence interval list for the sequence \(S\) computed according to Definitions 16 and 8, while \(||\mathcal{T}||\) is the length of the trace given in Definition 4.

A high support for PSI property \(S \Rightarrow \hat{E}\) is indicative of \(S\) being frequently true in the trace.

Definition 19. Correlation: For the assertion \(S \Rightarrow E^i\), correlation indicates how much of \(S\)’s truth is associated with \(E\), that is the quantum of the consequent, \(\varphi\)’s truth, that the antecedent \(S\) contributes to.

\[
Correlation(S \Rightarrow E^i) = \frac{||\mathcal{I}(S) \oplus [0 : i \times k] \cap \mathcal{I}(E)||}{||\mathcal{I}(E)||}
\]

Definition 20. Trace Coverage: Trace coverage quantifies the fraction of the trace that is explained by the properties generated by the miner.

Given a trace \(\mathcal{T}\) of length \(L\), and the mined property set \(\mathcal{A}\), the coverage interval list of \(T\) by \(\mathcal{A}\), denoted \(Cov(\mathcal{T}, \mathcal{A})\), is computed as follows:

\[
Cov(\mathcal{T}, \mathcal{A}) = \bigcup_{(S \Rightarrow E^i) \in \mathcal{A}} (\mathcal{I}(S) \oplus [0 : i \times k]) \cap \mathcal{I}(\hat{E})
\]

where \(\hat{E} \in \{E, \neg E\}\). The percentage of coverage is then given by \(\frac{|Cov(\mathcal{T}, \mathcal{A})|}{L} \times 100\).

9 Stopping Conditions, Over-fitting and Pruning

9.1 Stopping Criteria

While building the decision tree, for prefixes, there are two conditions that we employ to terminate the growth of the tree.

1. Purity of Node: When the constraint set \(C\) completely determines the truth of the target \(E\), the node is 100% pure and further growth is terminated. A node with constraint set \(C\), and minimum bucket position \(b\), is considered pure if the best error at the node is zero, that is \(\beta_{\mathcal{T}_E}(E^b) = 0\).

2. Depth Constraints: It is also possible to define a depth threshold, \(\alpha_d\), and stop the tree from growing if the length of the current exploration path crosses \(\alpha_d\).
9.2 Over-fitting

Decision trees are known to suffer from problems of over-fitting. This is exceptionally problematic with data that is discrete. In this case, when dealing with dense real-time data, over-fitting is the result of generating prefixes that have low temporal support.

To prevent over-fitting, the following two measures may be employed:

1. **Using a support threshold:** A threshold $\alpha_s$ is defined to indicate the minimum support below which a prefix is being over-fit to the data. If a node with constraint set $C$ has a support below $\alpha_s$, further splitting of the node is terminated.

2. **Using a correlation threshold:** A threshold $\alpha_c$ is defined to indicate the minimum correlation below which a prefix is being over-fit to the target. If a node with constraint set $C$ has a correlation below $\alpha_c$ with its associated target $\tilde{E}$, further splitting of the node is terminated.

When multiple time-series are used for training, the prefixes learned from one can be used to perform pruning of the decision tree learned from another if the prefix sequence infer opposing target truths. Beyond this, for a single time-series pruning is performed a-priori during the growth of the tree, using the stopping conditions.

The parameters $\alpha_d$, $\alpha_s$ and $\alpha_c$ are treated as meta-parameters of the decision tree learning algorithm.

10 Mining in the Value Domain

In the earlier sections we assumed that the set of predicates that form the sequence are known. The algorithm introduced therein is tasked to chose the predicates that, when sequenced in a specific way, best explains the target predicate (or event).

Here, we relax the assumption of a known predicate alphabet, and allow situations where the predicate alphabet is partially available or is empty. The definition of Joint Gain, which is designed to be sensitive to dependencies between the child nodes resulting from a data-set split. The Joint Gain, from choosing attributes (predicate and time) for splitting a node, varies widely with the constraints already chosen along the path from the node to the root (as given in Definition 17). These variations are dependent on the time-series; the interdependencies between variables recorded and existing temporal dependencies in the data.

We provide further insights into the Gain function (we use $\text{Gain}$ and Joint Gain to mean the same), how it varies with time and choices of predicate. Our analysis of the Gain function is also verified experimentally. We show how optimization techniques may be used to learn predicates and their temporal position, so as to maximize the Gain. In our work, we use simulated annealing to explore the terrain of the Gain function during the decision tree learning process, to mine prefix explanations for the target when little to no domain knowledge is available. The aim is to find arguments that maximize the Gain, that is to pick a predicate and its position in time relative to the target such that the combination provides the best Gain for the split.

Sub-section 10.1 provides a detailed analysis of the shape of the Gain function as the Gain varies with predicates and their position relative to the target. In Sub-section 10.2 we present an algorithm for learning predicates using Simulated Annealing.

10.1 The shape of the Gain Function

The outcome of the learning algorithm (explanations as PSI-L sequences), is strongly dependent on the method used to compute the Gain for a split. We observe the general shape of the Gain function and use this understanding to develop an integrated approach that learns new predicates and timing relations at each decision node aimed at improving the quality of the prefix sequences learned.
10.1.1 Variations of Gain with Temporal Positions

Given an existing constraint set \( C \) (possibly empty - at the root of the tree), a candidate predicate \( P \), and its temporal position \( i \) in the sequence, the Gain is dependent on the resulting pseudo-target association.

**Theorem 1.** The Gain, of placing predicate \( P \) in bucket index \( i \), monotonically increases with an increase in \( i \).

**Proof.** The pseudo-target association is dependent on the smallest index among the non-empty buckets in the constraint lists resulting from the split. The split results in two nodes, one with the constraint list \( C \cup \{ (P, i) \} \) and the other with the constraint list \( C \cup \{ (\neg P, i) \} \). The smallest non-empty bucket is therefore the minimum of \( i \) and the index of the smallest index non-empty bucket in \( C \). Let the index of the smallest index non-empty bucket in \( C \) be \( b \). Let \( \hat{b} \) be the lesser of \( i \) and \( b \). The length of interval list for the pseudo-target, \( |I(E^{\hat{b}})| \), becomes larger with larger values of \( \hat{b} \) (From Definition 15).

Initially, at the root of the decision tree, the constraint set, \( C \), is empty. Hence, as \( \hat{b} \) increases, a larger fraction of the truth of \( P \) would be covered by the pseudo-target \( E^{\hat{b}} \). This would cause the quantum of counter-examples for both true and false states of \( P \) and its association with \( E \) to reduce, leading to a reduced entropy, and therefore an increase in Gain. Hence as \( \hat{b} \) increases, the Gain would monotonically increase remain stagnant at a plateau. This is depicted in Figure 9.

When the constraint set, \( C \), is non-empty, i.e. there is at least one element \( \langle Q, j \rangle \in C \), the following cases arise:

1. \([b > i]\) (Figure 10): The end-match of the sequence \( C \cup \{ P, i \} \) is computed by adding \([0 : (b - i) \times k]\) to the end-match of \( C \). While maintaining \( i < b \), as \( i \) increases, i.e. \( i \) is a bucket further from the target, but closer to the end-match of \( C \) (depicted in Figure 10), the length of the end-match of the resulting constraint-set \( C \cup \{ P, i \} \), monotonically decreases. For larger differences between \( b \) and \( i \) (smaller values of \( i \)), the end-match is wider, hence the potential for counter-examples (with respect to the target) is higher. Therefore, as \( i \) increases, the entropy monotonically decreases.

2. \([b \leq i]\) (Figure 11): For constraint-set \( C \), either bucket \( i \), \( \langle B_i \rangle \), is empty or non-empty. When \( P \) is placed in bucket \( i \geq b \), \( B_i' = B_i \cup \{ P, i \} \). We have the following two cases:

   (a) \( B_i \) is non-empty: On adding \( P \) to bucket \( i \), let The interval list of \( P \) is intersected with the interval list of \( B_i \) and therefore the resulting bucket has the interval list \( I(B_i') \subseteq I(B_i) \).

   (b) \( B_i \) is empty: Let \( h \) be the smallest bucket index, \( h > i \), such that \( B_h \neq \phi \), and let \( j \) be the largest bucket index, \( j < i \), such that \( B_j \neq \phi \). If such a index \( h \) does not exist,
then $i$ is the largest index non-empty bucket in the prefix, while the case that $j$ does not exist is not possible under the present case ($b \leq i$).

If $B_h \neq \phi$, then the forward influence of $B_h$ on $B_j$ is computed as follows:

$$\Theta = (I(B_h) \oplus [0: (h - j) \times k]) \cap I(B_j)$$

However, on adding $P$ in bucket $i$, $h > i > j$, the forward influence of $B_h$ on $B_j$ is computed as follows:

$$\Omega = ((I(B_h) \oplus [0: (h - i) \times k]) \cap I(B_i)) \oplus [0: (i - j) \times k]) \cap I(B_j)$$

Hence, $\Omega \subseteq \Theta$. The potentially reduced forward influence on $B_j$ similarly propagates toward reducing the end-match for the sequence having $P$ in bucket $i$. A reduced end-match has lesser potential for entropy, and therefore yields a higher gain, or leaves the gain unchanged.

### 10.1.2 Variations of Gain with Predicate parameters

In this work, we assume a finite set of parameterized predicate templates. These predicate templates are of the form $x \preceq c$, where $\preceq \in \{<,>,=\}$. Coupled with the Boolean operator for negation, all relational predicates may be formed.

**Proposition 3.** The containment property of inequalities implies an equivalent containment property for interval lists; that is, the following holds:

$$(P_1 \rightarrow P_2) \Rightarrow (I(P_1) \subseteq I(P_2)) \text{ and similarly,}$$

$$(P_1 \rightarrow P_2) \Rightarrow (\neg P_2 \rightarrow \neg P_1) \Rightarrow (I(\neg P_2) \subseteq I(\neg P_1))$$

For instance, for predicates $P_1 \equiv (x < 10)$ and $P_2 \equiv (x < 20)$, $I(P_1) \subseteq I(P_2)$. Given a relational operator $\preceq$, a partial ordering therefore exists between the predicates that can be generated using the template associated with $\preceq$.

**Theorem 2.** For a predicate template $x \preceq c$, for variable $x$, as $c$ varies monotonically, making the predicate weaker, the variation in gain is non-monotonic.

![Diagram](image10.png)

**Figure 10:** Relative Position of Predicate $P$ with respect to the end-match interval list for a constraint set, with the end-match represented as $S$. The index of the minimum index non-empty bucket, $b$, is 8. The index of the bucket where $P$ may be placed in the sequence is $i$.

![Diagram](image11.png)

**Figure 11:** Relative Position of Predicate $P$ with respect to the end-match interval list for a constraint set, with the end-match represented as $S$. The index of the minimum index non-empty bucket, $b$, is 8. The index of the bucket where $P$ may be placed in the sequence is $i$. 

25
Figure 12: Interval Lists of Predicates $P_1$, $P_2$, $P_3$, and Target $E$. Here, $P_1 \rightarrow P_2 \rightarrow P_3$, $\mathcal{I}(P_1) \subseteq \mathcal{I}(P_2) \subseteq \mathcal{I}(P_3)$. The index of the bucket where a predicate may be placed in the sequence is $i$.

Proof. As the constant parameter $c$, for predicate $x \bowtie c$, varies making the predicate weaker (moving from $P_1$ to $P_2$, for instance, in Figure 12), the interval list for the predicate being true becomes wider, while the interval list for the predicate being false shrinks. The entropy contributions redistribute between the two states of the predicate. This results in unpredictable variations in the Gain.

The entropy contributed by the truth state of a predicate ($P$ or $\neg P$), in general, depends on the truth distribution of the target with respect to the truth of the predicate. As a predicate’s parameter is varied, making it weaker, the negation of the predicate becomes stronger. As more time points are added to the weaker predicate, the entropy it contributes depends on the proportion of time of the target’s truth that overlaps with the truth of the now weaker predicate. If this quantity is disproportionate to the widening of the predicate’s truth intervals, then the entropy contributed would be larger.

For a predicate being made stronger, a similar situation arises. If the time intervals of the target’s truth states do not proportionately reduce, as the truth intervals for the predicate shrink, the entropy contributed by the stronger predicate would increase.

Observe the truth of predicates $E$ and $P_1$ in Figure 12. Observe that $P_2$ and $P_3$ are weaker than $P_1$. For $P_2$, the weakness disproportionately adds time points for $\neg E$ to the time points where $P_2$ is true, while for $\neg P_2$, time points for $\neg E$ reduces, resulting in an increase in the entropy for the choice of $P_2$ in temporal position zero from 0.48 for $P_1$ to 0.49 for $P_2$. However, for $P_3$, the weakness yields an increase in the overlap for both $E$ and $\neg E$, with more of an overlap for $\neg E$, which in turn reduces the entropy from 0.48 for $P_1$ to 0.46 for $P_3$.

Since the variation of Gain with predicate parameters depends solely on the data-set and the relationships therein, we use optimization techniques over the space of predicate parameters to compute the predicate instance, and its temporal (bucket) position, that together yield the best Gain.

10.2 Mining Predicates within the Decision Tree

Simulated Annealing is a combinatorial optimization search technique introduced in the early ’80s [32, 45] and was inspired by the simulation of the annealing of solids; hence the name. The physical process of annealing involves using a heat bath heated to high temperatures, at which particles of the solid rearrange themselves haphazardly forming a liquid. The temperature of the heat bath is gradually reduced, so as to allow the particles of the solid to arrange themselves in the minimal energy state of a corresponding lattice. This outcome is likely only if the initial temperature of the heat bath is sufficiently high and the cooling takes place at a rate that is sufficiently slow. The interested reader is referred to [34] for a detailed study of the algorithm of Simulated Annealing used to solve combinatorial optimization problems, along with a proof of asymptotic convergence.

In our study, we use Simulated Annealing to determine the attributes on which to split the decision tree at each node; the predicate and its position in time relative to the target as arguments that maximize the Gain. We describe the important elements and terms used in the algorithm below.
1. **Configuration:** The configuration represents the state that we wish to perturb, with the aim of bringing it to a low energy level. At each decision tree node, the configuration is the constraint set $C$ and a choice of a candidate predicate $P$ in a temporal position in the past relative to the target. A perturbation is a change made the parameters of $P$ and to the temporal position.

2. **Generation Mechanism:** A generation mechanism is required so that given a configuration $C$, a new configuration $\hat{C}$ may be created by perturbing the predicate parameter and temporal position. In the algorithm, the value of the control parameter (temperature) is used to define the size of the perturbation.

3. **Configuration Neighbourhood:** The amount of perturbation is controlled by the control parameter (temperature), and defined the neighbourhood of a configuration. For a time-series signal variable $x$ having a domain of values in the range $[x_{\min}, x_{\max}]$ as observed in the time-series, for a control (temperature) range of $[t_{\min}, t_{\max}]$, the neighbourhood for a parameter value $c$ in predicate $x \bowtie \triangleleft c$ is computed as follows:

$$\delta = \frac{(t - t_{\min})}{(t_{\max} - t_{\min})}$$

$$c_{\text{left}} = c - (c - x_{\min}) \times \delta$$

$$c_{\text{right}} = c + (x_{\max} - c) \times \delta$$

4. **Cost function:** The Joint Gain, in Definition 17, is used as the cost function. In our use, we aim to maximize the cost.

5. **Control Parameter/Temperature Schedule:** The temperature schedule determines how the temperature changes in every iteration of the Annealing Algorithm, that is the rate of change of temperature with time. While many alternatives are possible, including a linear schedule, a geometric schedule [32], and Dynamic schedules [1], we use a linear schedule, as this gives us promising results. We also allow provisions for using other temperature schedules.

The Simulated Annealing algorithm is presented in Algorithm 3. The algorithm computes the best predicate (one with the highest Gain) that may be placed in a specific bucket index (temporal) position relative to the target. The time-series contains timestamped valuations for variables of the system being recorded. For each variable in the time-series, and each relational operator, a simulated annealing is used to compute the best predicate.

In the algorithm, Line 2 computes the entropy for the present node. In line 5 the function $\text{getRange}(x, T)$ computes the domain of the variable $x$ in the trace $T$. The function $\text{schedule}(i)$ in Line 10 returns the value of the control parameter (temperature) for the given value of $i$. A number of schedules may be used to compute the control parameter value over time, some of which are detailed in [34]. In line 20 the function $\text{random}(a,b)$ returns a random real number between $a$ and $b$, $a,b \in \mathbb{R}$.

The Prefix Sequence mining algorithm uses Algorithm 3 to learn predicates at each decision tree node.

11 **Summary**

The quality of a formal specification relies on the domain expert’s anticipation of the potential bad behaviours (safety properties) and expected good behaviours (liveness properties). A significant volume of bugs detected late in the design cycle are attributed to the fact that the domain expert did not think of the causal relations in the relevant scenarios. An assertion miner can identify those causal relationships within a system which did not occur to the domain
**ALGORITHM 3: learnPredicate**: Learn a predicate of the form $x \bowtie c$, and its temporal position to maximize Gain, for n-length, k-resolution Prefix Sequences.

**Input**: Variable list $V$, Time-series $T$, Set of relational operators $O$, Current Configuration $C$, Target predicate $E$, Target truth list $I(E)$, Bucket Index $sp$, schedule - a mapping of time to a control parameter value.

**Output**: Predicate $x \bowtie c$.

1. $b \leftarrow$ Smallest non-empty bucket position in $C$;
2. $\hat{\beta} \leftarrow \beta_T(E^b)$;
3. $P_{next} \leftarrow \phi$; $profitP_{next} \leftarrow -\infty$;
4. for $x \in V$ and $\bowtie \in O$ do
5. $[x_{min}, x_{max}] = \text{getDomain}(x, T)$;
6. $c \leftarrow \frac{x_{min} + x_{max}}{2}$;
7. $P \leftarrow "x \bowtie c"$;
8. $P'' \leftarrow \phi$; $profitP'' \leftarrow \infty$;
9. for $i \leftarrow 1$ to $\infty$ do
10. $t \leftarrow \text{schedule}(i)$;
11. if $t == 0$ then $b$;
12. reak from this loop;
13. $profitP' \leftarrow -\infty$; $P' \leftarrow \phi$;
14. foreach $Q \leftarrow \text{neighbour}(P)$ do
15. $profit \leftarrow \text{JointGain}(\hat{\beta}, C, Q, sp)$;
16. if $profit > profitP'$ then
17. $profitP' \leftarrow profit$;
18. $P' \leftarrow Q$;
19. if $profitP' > profitP''$ then $P'' \leftarrow P'$; $profitP'' \leftarrow profitP'$;
20. else if $(c - profitP') > \text{random}(0, 1)$ then $P'' \leftarrow P'$; $profitP'' \leftarrow profitP'$;
21. if $profitP'' > profitP_{next}$ then $P_{next} \leftarrow P''$; $profitP_{next} \leftarrow profitP''$;
22. return $P_{next}$;

expert. This is relevant in various signal domains; including cyber-physical systems, medical, smart-grids, transportation, etc.

Given a set of predicates over the variables of the hybrid system, the nk-PSI-Miner presented in this article is able to mine cause $\Rightarrow$ effect time patterns from Booleanized traces of hybrid systems. The causes are in the form of sequences of Boolean expressions over the given predicates separated in time from one another, that when combined in that sequence are likely to be the cause of a given target predicate. The predicates themselves can come from domain knowledge or be mined using techniques involving simulated annealing using the measure of gain as the optimization function. We discuss methods for measuring the goodness of the prefixes mined and mechanisms for decision tree pruning and preventing over-fitting using these measures.

A mechanism for mining predicates in-the-loop is also provided that takes into account timing constraints from the prefix sequence. We provide insights into the shape of the Gain function for temporal sequences and present a methodology for learning predicates that maximize gain for a given bucket position. For a candidate predicate, the Gain increases monotonically with an increase in the index of the bucket in which the predicate is placed. However, for a given bucket index, the Gain varies with the choice of parameter for a predicate template in unpredictable ways, indicating a deep correlation between the choice of parameter and the target’s truth, which in turn depends on the relations that exist in the time-series. This is precisely the challenge we address here, that is, learning the potential causal relations and dependencies that exist in the data.

Our intention is to extend this article with results from a variety of domains.
References

[1] Aarts, E., Korst, J., and Michiels, W. Simulated Annealing. Springer US, Boston, MA, 2005, pp. 187–210. (Cited on page 27)

[2] Aggarwal, C. C. Data Mining: The Textbook. Springer Publishing Company, Incorporated, 2015. (Cited on pages 8 and 12)

[3] Annpureddy, Y., Liu, C., Fainekos, G., and Sankaranarayanan, S. S-TaLiRo: A Tool for Temporal Logic Falsification for Hybrid Systems. In Tools and Algorithms for the Construction and Analysis of Systems (Berlin, Heidelberg, 2011), P. A. Abdulla and K. R. M. Leino, Eds., Springer Berlin Heidelberg, pp. 254–257. (Cited on page 4)

[4] Asarin, E., et al. Parametric identification of temporal properties. In Proc. of the 2nd International Conference on Runtime Verification (2012), pp. 147–160. (Cited on page 4)

[5] Bakhirkin, A., Ferrère, T., and Maler, O. Efficient parametric identification for stl. In HSCC (2018), HSCC ’18, ACM, pp. 177–186. (Cited on page 4)

[6] Bartocci, E., Bortolussi, L., and Sanginetti, G. Data-driven statistical learning of temporal logic properties. In FORMATS (2014). (Cited on pages 3 and 4)

[7] Bombara, G., et al. A decision tree approach to data classification using signal temporal logic. In HSCC (2016), pp. 1–10. (Cited on pages 4 and 5)

[8] Brockwell, P. J., and Davis, R. A. Time Series: Theory and Methods. Springer-Verlag, Berlin, Heidelberg, 1986. (Cited on page 3)

[9] Brockwell, P. J., and Davis, R. A. Introduction to Time Series and Forecasting, 2nd ed. Springer, Mar. 2002. (Cited on page 3)

[10] Chakrabarti, K., Keogh, E., Mehrotra, S., and Pazzani, M. Locally adaptive dimensionality reduction for indexing large time series databases. ACM Trans. Database Syst. 27, 2 (June 2002), 188–228. (Cited on page 3)

[11] Chang, P. H., and Wang, L. C. Automatic assertion extraction via sequential data mining of simulation traces. In ASP-DAC (Jan 2010), pp. 607–612. (Cited on page 3)

[12] Cutulenco, G., et al. Mining timed regular expressions from system traces. In Proc. of ASE (2016), pp. 3–10. (Cited on page 5)

[13] Danese, A., Filini, F., and Pravadelli, G. A time-window based approach for dynamic assertions mining on control signals. In Proc. of VLSI-SoC (2015), pp. 246–251. (Cited on page 5)

[14] Danese, A., Ghasempouri, T., and Pravadelli, G. Automatic extraction of assertions from execution traces of behavioural models. In Proc. of DATE (2015), pp. 67–72. (Cited on page 5)

[15] Debregeas, A., and Hebrail, G. Interactive interpretation of kohonen maps applied to curves. In Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining (1998), KDD’98, AAAI Press, pp. 179–183. (Cited on page 3)

[16] DeOrio, A., et al. Inferno: Streamlining verification with inferred semantics. IEEE TCAD 28, 5 (May 2009), 728–741. (Cited on page 5)

[17] Donzé, A. Breach, a toolbox for verification and parameter synthesis of hybrid systems. In Computer Aided Verification (Berlin, Heidelberg, 2010), T. Touili, B. Cook, and P. Jackson, Eds., Springer Berlin Heidelberg, pp. 167–170. (Cited on page 4)
[18] Donzé, A., and Maler, O. Robust satisfaction of temporal logic over real-valued signals. In Proc. of FORMATS (2010), pp. 92–106. (Cited on page [4])

[19] Esling, P., and Agon, C. Time-series data mining. ACM Comput. Surv. 45, 1 (Dec. 2012), 12:1–12:34. (Cited on pages [3] and [4])

[20] Faloutsos, C., Ranganathan, M., and Manolopoulos, Y. Fast subsequence matching in time-series databases. In Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data (1994), SIGMOD ’94, ACM, pp. 419–429. (Cited on page [3])

[21] Gabel, M., and Su, Z. Javert: Fully automatic mining of general temporal properties from dynamic traces. In Proc. of SIGSOFT (2008), ACM, pp. 339–349. (Cited on page [3])

[22] Gabel, M., and Su, Z. Symbolic mining of temporal specifications. In Proc. of ICSE (2008), ACM, pp. 51–60. (Cited on page [5])

[23] Garg, P., et al. Learning invariants using decision trees and implication counterexamples. In Proc. of POPL (2016). (Cited on page [5])

[24] Guralnik, V., and Srivastava, J. Event detection from time series data. In Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (New York, NY, USA, 1999), KDD ’99, ACM, pp. 33–42. (Cited on pages [3] and [8])

[25] Harris, R., and Sollis, R. Applied Time Series Modelling and Forecasting John Wiley. 01 2003. (Cited on page [3])

[26] IEEE. 1800-2012 - IEEE Standard for SystemVerilog—Unified Hardware Design, Specification, and Verification Language. http://standards.ieee.org/findstds/standard/1800-2012.html. (Cited on page [3])

[27] Indyk, P., Koudas, N., and Muthukrishnan, S. Identifying representative trends in massive time series data sets using sketches. In Proceedings of the 26th International Conference on Very Large Data Bases (San Francisco, CA, USA, 2000), VLDB ’00, Morgan Kaufmann Publishers Inc., pp. 363–372. (Cited on page [3])

[28] Jin, X., Donze, A., Deshmukh, J. V., and Seshia, S. A. Mining requirements from closed-loop control models. IEEE TCAD 34, 11 (Nov 2015), 1704–1717. (Cited on page [4])

[29] Kalpakis, K., Gada, D., and Puttagunta, V. Distance measures for effective clustering of arima time-series. In Proceedings of the 2001 IEEE International Conference on Data Mining (Washington, DC, USA, 2001), ICDM ’01, IEEE Computer Society, pp. 273–280. (Cited on page [3])

[30] Kaufmann, S., and Fischmeister, S. Mining temporal intervals from real-time system traces. In Proc. of 6th International Workshop on Software Mining (Nov 2017), pp. 1–8. (Cited on page [5])

[31] Keogh, E., Lonardi, S., and Chiu, B. Y.-c. Finding surprising patterns in a time series database in linear time and space. In Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (New York, NY, USA, 2002), KDD ’02, ACM, pp. 550–556. (Cited on page [3])

[32] Kirkpatrick, S., Gelatt Jr., C. D., and Vecchi, M. P. Optimization by simulated annealing. In IBM Computer Science/Engineering Technology Report. IBM, 1982. (Cited on pages [26] and [27])

[33] Kong, Z., et al. Temporal logic inference for classification and prediction from data. In HSCC (2014). (Cited on pages [4] and [5])
[34] LAARHOVEN, P. J. M., AND AARTS, E. H. L., Eds. Simulated Annealing: Theory and Applications. Kluwer Academic Publishers, Norwell, MA, USA, 1987. (Cited on pages 26 and 27)

[35] LEMIEUX, C., PARK, D., AND BESCHASTNIKH, I. General ltl specification mining (t). In Proc. of ASE (2015). (Cited on page 3)

[36] MA, J., AND PERKINS, S. Online novelty detection on temporal sequences. In Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (New York, NY, USA, 2003), KDD ’03, ACM, pp. 613–618. (Cited on page 4)

[37] MALER, O., AND NICKOVIC, D. Monitoring temporal properties of continuous signals. In Proceedings of Formal Modeling and Analysis of Timed Systems (FORMATS-FTRTFT). Volume 3253 of LNCS (2004), Springer, pp. 152–166. (Cited on page 3)

[38] MA, J., AND PERKINS, S. Online novelty detection on temporal sequences. In Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (New York, NY, USA, 2003), KDD ’03, ACM, pp. 613–618. (Cited on page 4)

[39] OTT, R. L., AND LONGNECKER, M. T. Introduction to Statistical Methods and Data Analysis (with CD-ROM). Duxbury Press, Boston, MA, USA, 2006. (Cited on page 12)

[40] QUINLAN, J. R. Induction of decision trees. Mach. Learn. 1, 1 (Mar. 1986), 81–106. (Cited on page 73)

[41] RALANAMAHATANA, C. A., ET AL. Mining Time Series Data. Springer US, 2005, pp. 1069–1103. (Cited on page 3)

[42] TSAY, R. Analysis of financial time series, 2. ed ed. Wiley series in probability and statistics. Wiley-Interscience, Hoboken, NJ, 2005. (Cited on page 3)

[43] VAN WIJK, J. J., AND VAN SELOW, E. R. Cluster and calendar based visualization of time series data. In Proceedings of the 1999 IEEE Symposium on Information Visualization (Washington, DC, USA, 1999), INFOVIS ’99, IEEE Computer Society, pp. 4–. (Cited on page 3)

[44] VASUDEVAN, S., ET AL. GoldMine: Automatic assertion generation using data mining and static analysis. In Proc. of DATE (March 2010), pp. 626–629. (Cited on pages 3 and 8)

[45] ČERNÝ, V. Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. J. Optim. Theory Appl. 45, 1 (Jan. 1985), 41–51. (Cited on page 26)

[46] YANG, H., HOXHA, B., AND FAINEKOS, G. Querying parametric temporal logic properties on embedded systems. In Prof. of ICTSS (2012). (Cited on page 4)

[47] YANG, J., ET AL. Perracotta: Mining temporal api rules from imperfect traces. In Proc. of ICSE (2006), ACM, pp. 282–291. (Cited on page 3)

[48] YANG, J., AND EVANS, D. Dynamically inferring temporal properties. In Proc. of PASTE (2004), ACM, pp. 23–28. (Cited on page 3)

[49] YPMA, A., YPMA, E., AND DUIN, R. P. Novelty detection using self-organizing maps. In Proc. of ICONIP’97 (1997), Springer, pp. 1322–1325. (Cited on page 4)

[50] ZHONG, S., AND KHOSHOFTAAR, T. M. Clustering-based network intrusion detection. International Journal of Reliability, Quality and Safety Engineering 14, 2 (2007), 169–187. (Cited on page 4)