Mining Shape Expressions from Positive Examples

Ezio Bartocci*, Jyotirmoy Deshmukh†, Felix Gigler‡, Cristinel Mateis‡, Dejan Ničković‡, Xin Qin†
* TU Wien, Vienna, Austria
† University of Southern California, Los Angeles, CA, USA
‡ AIT Austrian Institute of Technology, Vienna, Austria

***Abstract*—Shape expressions (SEs) is a novel specification language that was recently introduced to express behavioral patterns over real-valued signals observed during the execution of cyber-physical systems. A shape expression is a regular expression composed of arbitrary parameterized shapes such as lines, exponential curves, and sinusoids as atomic symbols with symbolic constraints on the shape parameters. SEs enable a natural and intuitive specification of complex temporal patterns over possibly noisy data. In this paper, we propose a novel method for mining a broad and interesting fragment of SEs from time-series data using a combination of techniques from linear regression, unsupervised clustering and learning finite automata from positive examples. The learned SE for a given dataset provides an explainable and intuitive model of the observed system behavior. We demonstrate the applicability of our approach on two case studies from different application domains and experimentally evaluate the implemented specification mining procedure.**

I. Introduction

From self-driving cars and service robots to the rapidly proliferating Internet-of-Things (IoT) devices [1], cyber-physical systems (CPS) are becoming pervasive in every aspect of our daily life. CPS applications embed computational units with physical entities such as sensors and actuators designed to tightly interact with some physical component or the real-world environment. With the recent strides in artificial intelligence and machine learning, CPS applications are evolving to be tremendously complex systems that can operate (autonomously) in sophisticated and unpredictable environments.

It is common in CPS applications to use many different kinds of sensors and monitors to gather time-series data about various aspects of the application's operation. This includes data about the device's environment, its internal system variables, or physical characteristics of the device (such as power, speed, temperature). A quandary facing many of the application developers is that there is a veritable deluge of gathered data in these systems, and designers are struggling to analyze, utilize and characterize the gathered data. One solution is to consider the vast literature on time-series analysis in the machine learning community. In the context of (unsupervised) learning, most of the work in the ML community focuses on identifying distance metrics on time-series data that enable effective clustering algorithms [2], [3], or identifying features from the data itself [4], [5], [6]. Most ML techniques however suffer from lack of interpretability: for example it is almost impossible to relate the computations performed by successive layers of deep neural networks to the humanly comprehensible reasoning steps. In the last years, several papers [7], [8], [9], [10] have highlighted the need to reconcile ML techniques with symbolic AI such as logic and formal languages to complement and to address this shortcoming. Symbolic representations benefit of their declarative nature. They are re-usable, data-efficient and compositional. They provide an high-level and abstract framework that facilitates generalisation. Furthermore, since they are language-like, they are verifiable and generally closer to the human understanding.

One of the goals of this paper is similar to that of ML methods: to extract information from data. However, the specific usage scenarios that we discuss in this paper require information mining techniques that result in structured artifacts that are interpretable, by the human or by the machine. Consider the problems of mining temporal and logical information about a particular system variable or a physical quantity (i.e. the specification mining problem), the problem of mining the temporal and logical conditions on environment signals that ensures correct system behavior (i.e. the assumption mining problem), or the problem of automatically mining logical patterns from data (i.e. the explainable clustering problem). In each of these applications, explainability has a high value.

Our research hypothesis in this paper is that the recently developed language of *Shape Expressions* (SEs) [11] is an *explainable and interpretable formalism* to perform effective mining from a time-series dataset. SEs essentially allow us to express mean behaviors in the presence of noise. We validate our research hypothesis by developing a new procedure for mining *shape expressions* from (positive examples of) *time-series data*. In the CPS context, each time-series datum is a sequence of time-value pairs encoding system behaviors or a discrete-time trace of the value of a particular system variable (e.g. sensor output, actuator input, state variable, etc.). Given a set of such time-series data, our specification mining procedure performs three main steps:

1) *Segmentation:* this step transforms each time-series datum into a (minimal) sequence of *linear* segments that can optimally approximate the original data within a given error threshold. The algorithm to accomplish this combines linear regression with a recursive procedure to create such a piecewise-linear (PWL) sequence, where for each inferred piece, we learn the parameters: slope, offset and duration.
2) *Abstraction:* this step *clusters* linear segments based on the similarity between their parameter values into some finite number of clusters. For each resulting cluster we assign a unique symbol; thus the symbol representing a cluster conservatively approximates all the line segments in the cluster. This step defines an abstraction function from line segments

Manuscript received April 17, 2020; revised June 17, 2020; accepted July 6, 2020. This article was presented in the International Conference on Embedded Software 2020 and appears as part of the ESWEEK-TCAD special issue.

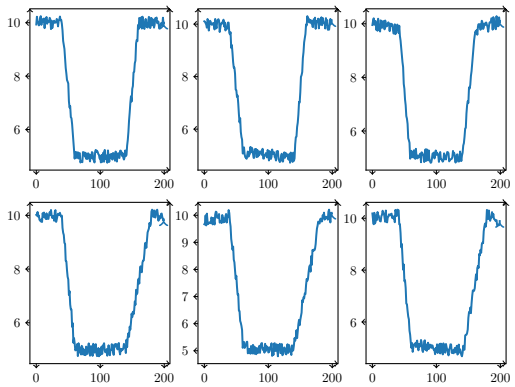


Fig. 1: Illustrative example - set of pulses.

to the finite alphabet of symbols, and thereby allows us to map raw time-series into abstract sequences of symbols (finite words) over this alphabet.

3) *Learning:* this step infers temporal properties from the abstract traces by using a standard algorithm for learning deterministic finite automata (DFA) from positive examples. Finally, we map the DFA to the SE representation.

The resulting shape expression provides a structured model of the observed system behavior that can be explained and interpreted by a human or a machine. The resulting model can group similar and repeating patterns within the behavior. Learning shape expressions from multiple classes of data can facilitate characterizing similarities and differences between different classes. The presented approach is a natural fit for mining specifications from behaviors of system that exhibit piecewise linear behavior but can be also used to approximate nonlinear behavior. Approximating complex nonlinear dynamics with piecewise-linear [12], [13], [14], [15] or multi-affine functions [16] is also a common practise in literature for system identification and for learning hybrid models amenable to formal analysis. We demonstrate the applicability of our approach on two case studies from different application domains and experimentally evaluate the implemented specification mining procedure.

Illustrating example: We use *noisy pulses* shown in Figure 1 to illustrate the various steps in our specification mining approach. Such analog pulses are common in many electronic applications. Distributed Standard Interface (DSI3) is one such example from the automotive domain, where analog pulses are used to encode communication between the micro-controller and sensors in an airbag system-on-chip. The six examples depicted in Figure 1 were synthetically generated – they all appear to visually have similar shape. However we can observe that the segments in the three top pulses have a steeper slope then the corresponding ramping segments in the bottom pulses.

Organization of the paper: In Section II, we discuss the related work. In Section III we provide the necessary background on *shape expressions* while in Section IV, we present our novel approach to infer them from positive examples. Section V presents the experimental evaluation of the proposed method on two case studies. We conclude with results summary and plans for future work in Section VI.

II. Related Work

In the last decade, specification mining [17], [18], [19], [20], [21], [22], [23], [24], [25], [26], [27], [28], [29], [30], [31], [32], [33], [34], [35] has become a very active research field supporting the analysis and the CPS development. Our approach builds on top of *shape expressions* [11], a recently introduced declarative language for specifying and monitoring sophisticated temporal patterns from possibly noisy data. On the contrary, most of the current work in specification mining for CPS [36], [37], [17], [19], [20], [21], [22], [23], [24], [25], [26], [28], [30], [31], [32], [33], [34] centers the development around Signal Temporal Logic [38] (STL) (and its extensions), a temporal logic defined to reason about dense-time mixed-analog signals.

A considerable part of the literature focuses [17], [19], [20], [21], [22], [23], [24] on the problem of learning the optimal parameters starting from a specific template formula (for example invariants [39], [40], [41]). Learning both the structure of the formula and its parameters is more challenging and it has been addressed only recently [25], [26], [21], [28]. The common approach consists in a two-steps heuristics in which first the structure of the formula is inferred and then its parameters are optimised. The structure of the formula can be determined, for example, by using *decision trees* [26], [42], [43], *genetic algorithms* [30], [28], [34], [33] or *SAT-based algorithms* [31]. All the aforementioned methods, with the exception of [33], require both positive and negative examples in the learning phase, while our approach is based only on positive examples.

The work in [33] introduces the notion of *tightness* metric to learn template-driven STL formulas that satisfy as tight as possible a set of user-provided positive examples. In the same paper, the authors show how to combine their approach with a genetic algorithm to infer also the structure of more arbitrary formulas, similarly to [30]. However, genetic algorithms, that are metaheuristics, have the tendency to converge towards local optima rather than the global optimum of the problem. This results in the impossibility to assess whether the learned formula is really the optimal one. Our approach differs from the work in [33] in many ways. First, we consider a different specification language closer to regular expressions than temporal logic. Second, we choose the *mean squared error* as our measure of tightness of the specification with respect to the observed signal. This measure is used to guide the optimal split of signals in basic shapes (in this paper we choose for simplicity lines as our basic shapes). Using standard clustering algorithms, the obtained shapes can be grouped according to their parameters. Each cluster can be expressed as a basic shape expression with constrained parameters that would match the same signals matched by the basic shape expressions in the cluster without incrementing the *mean squared error*. This allows to derive a finite alphabet of symbols (each representing a different shape expression with constrained parameters) and to represent each signal in the training set as a finite sequence of them. Finally, in such setting we show how to use more reliable passive automata learning algorithms [44] rather than metaheuristics (e.g., genetic algorithms) to infer the structure of the expression.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2020.3012240, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems
This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication. The final version of record is available at http://dx.doi.org/10.1109/TCAD.2020.3012240

Our approach does not require to interact with a reactive system and it is indeed orthogonal to active automata learning (AAL) such as $L*$ Angluin's algorithm [45] and its recent developments [46], [47]. AAL is generally employed to learn how to interact with the surrounding environments [48], [49] and it needs to exercise the system in order to infer the relation between the provided input and the observed output. Our approach can be instead applied directly, without the necessity to provide an input.

Finally, the problem of piecewise approximation of a signal has received considerable attention in mathematical literature [50], [51], [52], [53]. In particular, Bellman and Roth introduced in [50] a dynamic programming algorithm to compute the curve fitting of a set of data by a set of straight lines. The key idea of their approach is to employ a two-dimensional grid of N points in the abscissa and M points in the ordinate and to search the segmentations of length L, $L = 1, 2, \ldots, N-1$, such that for each L the total error, measured as the sum of the max absolute error of the data with each fitting segment, is minimal among all possible segmentations of length L. For a sufficient large M, their approach approximates our piecewise fit that employs instead regression as the main ingredient. However, our approach does not require the user to provide an extra parameter M. Furthermore, the M points considered in [50] are generally chosen, for practical reasons, over the min and the max value of the signal. This does not necessary produce the best possible local fit, because sometimes a linear segment to fit well the data, needs to start from a value that is bigger or smaller than the max or the min value of the signal respectively. Recent works of Ozay et al. [54], [55]. adapted the dynamic programming algorithm in [50] to infer the linear segments through least squares error directly over the signal and generalise the approach to the segmentation of time-varying affine autoregressive exogenous (ARX) models. Although, our approach to compute the optimal (w.r.t. to the maximum mean square error) split in basic shapes of a signal can be considered as a special case of the aforementioned approaches (see Section IV), the focus of our paper is on mining a specification for a set of signals in a formal language and we use the segmentation as one of the main ingredients.

III. Shape Expressions and Automata

In this section, we define a variant of shape expressions (SE) [11], which are regular expressions where atomic shapes with unknown parameters play the role of atomic primitives. These basic shape primitives are expressions that map parameter and signal variables to a parameterized family of idealized signals, such as lines, sinusoids, exponentials, etc. Here we focus on the class of *linear* SE that considers only lines as atomic shapes. In contrast to SE in [11], we also allow *conjunction* of atomic shapes to facilitate learning of expressions from multi-dimensional data and we also lift the restriction of discrete-time signals. Finally, we choose *mean squared error* (MSE) as our measure of noise.

Let $P = \{p_1, \ldots, p_n\}$ be a set of *parameters*. A parameter valuation v maps parameters $p \in P$ to values $v(p) \in \mathbb{R} \cup \{\bot\}$, where $\bot$ represents the *undefined* value. We use the shortcut $v(P)$ to denote $\{v(p_1), \ldots, v(p_n)\}$. A *constraint* γ over P is a Boolean combination of inequalities over P. We write $v \models \gamma$ when the constraint γ is satisfied by the valuation v. Given $p \in P$ and $p \circ k$ for $\circ \in \{=, <, \leq, >, \geq\}$ and some $k \in \mathbb{R}$, we have that $v(p) = \bot$ implies that $v \not\models p \circ k$. We denote by $\Gamma(P)$ the set of all constraints over P.

Let X be a set of signal variables. A *signal* w over X is a sequence $w = (t_1, v_1) \cdot (t_2, v_2) \cdots (t_n, v_n)$, where $t_i \in \mathbb{Q}_{\geq 0}$ is the *time-stamp* such that for all $1 \leq i < n$, $t_i < t_{i+1}$ and v_i is a valuation in $V(X)$. We denote by $|w| = n$ the *size* of w. Given two signals $w = (t_1, v_1) \cdot (t_2, v_2) \cdots (t_m, v_m)$ and $w' = (t'_1, v'_1) \cdot (t'_2, v'_2) \cdots (t'_n, v'_n)$, their *concatenation* $w \cdot w'$ is defined if $t_m = t'_1$ and $v_n = v'_1$ and corresponds to the sequence $(t_1, v_1) \cdots (t_m, v_m) \cdot (t'_2, v'_2) \cdots (t'_n, v'_n)$. Given a signal w of size n and $i, j \in \mathbb{N}$ such that $1 \leq i \leq j \leq n$, we denote by $w[i:j]$ the sequence $(t_i, v_i) \cdots (t_j, v_j)$. The *duration* d of a signal is defined as $d = t_n - t_1$. We allow signals of null duration $d = 0$, which results in the unique signal with the empty time domain[1].

A. Shape Expressions

We now provide the syntax and semantics of linear SE defined over the set X of signals and the set P of parameter variables. The set P contains all the parameters (slope, relative offset and duration) of the linear segments defined by the shape expression. The *atomic shape* $\mathsf{lin}_x(a, b, \underline{l})$ is the expression that maps parameter variables $\{a, b, \underline{l}\} \subseteq P$, where a is the *slope*, b the *relative offset*[2] and $\underline{l}$ the *duration*, and the signal variable $x \in X$ to a parameterized representation of an idealized line signal, defined as follows:

$$\begin{aligned}\mathsf{lin}_x(a, b, \underline{l}) \quad \equiv \quad & \{w[i:j] \mid \exists v. t_j - t_i = v(\underline{l}) \wedge \forall k \in [i, j], \\ & w(x, t_k) = v(a) t_k + (v(b) - v(a) t_i)\}.\end{aligned}$$

Definition III.1 (Linear SE syntax). *A linear shape expression φ is given by the grammar*

$$\begin{aligned}\psi \quad &::= \quad \mathsf{lin}_x(a, b, \underline{l}) \mid \psi_1 \cap \psi_2 \\ \varphi \quad &::= \quad \epsilon \mid \psi \mid \varphi_1 \cup \varphi_2 \mid \varphi_1 \cdot \varphi_2 \mid \varphi^* \mid \varphi : \gamma\end{aligned}$$

where $x \in X$, $\{a, b, \underline{l}\} \subseteq P$, and $\gamma \in \Gamma(P)$, where P is the set of all (slope, relative offset, duration) parameters of the atomic line shapes defined in the expression.

The symbol ϵ denotes the *empty word*, the operators $\cup$, $\cap$, $\cdot$ and $*$ denote the classical (extended) regular expression *union*, *intersection*, *concatenation* and *Kleene star* respectively, while $\varphi : \gamma$ says that φ is *constrained* by γ. We write φ^i as an abbreviation of $\varphi \cdots \varphi$ (i times). We denote by $\mathbb{L}_X(P)$ the set of expressions of the form $\mathsf{lin}_x(P')$ for $x \in X$ and $P' \subseteq P$. The set of shape expressions over P and X is denoted $\Phi(P, X)$.

The semantics of shape expressions is given as a relation between signals and parameter valuations, which we call a *language*. We associate with every shape expression a *noisy language* $\mathcal{L}_\nu$ for some noise tolerance threshold $\nu \geq 0$, capturing the ν-approximate meaning of the expression. The *exact*

[1]The signal with the empty time domain is equivalent to the empty word in the classical language theory.

[2]The relative offset of a line segment is the value of the line at the beginning of the segment.

language $\mathcal{L}$ capturing the precise meaning of the expression is obtained by setting ν to zero. We formally define the noisy language as follows.

Definition III.2 (SE noisy language)**.** *Let $\nu \in \mathbb{R}_{\geq 0}$ be a noise tolerance threshold. The noisy language $\mathcal{L}_\nu$ of a shape expression is defined as follows:*

$$\begin{aligned}
\mathcal{L}_\nu(a,b,\underline{l}) &= \{(w,v) \mid v(\underline{l}) = t_{|w|} - t_1, \\
&\qquad \mu(w(x), \mathit{lin}_x(v(a), v(b), v(\underline{l})) \leq \nu\} \\
\mathcal{L}_\nu(\psi_1 \cap \psi_2) &= \mathcal{L}_\nu(\psi_1) \cap \mathcal{L}_\nu(\psi_2) \\
\mathcal{L}_\nu(\epsilon) &= \{(w,v) \mid |w| = 0\} \\
\mathcal{L}_\nu(\varphi_1 \cdot \varphi_2) &= \{(w_1 \cdot w_2, v) \mid (w_1, v) \in \mathcal{L}_\nu(\varphi_1) \\
&\qquad \textit{and } (w_2, v) \in \mathcal{L}_\nu(\varphi_2)\} \\
\mathcal{L}_\nu(\varphi_1 \cup \varphi_2) &= \mathcal{L}_\nu(\varphi_1) \cup \mathcal{L}_\nu(\varphi_2) \\
\mathcal{L}_\nu(\varphi^*) &= \bigcup_{i=0}^{\infty} \mathcal{L}_\nu(\varphi^i) \\
\mathcal{L}_\nu(\varphi : \gamma) &= \{(w,v) \mid (w,v) \in \mathcal{L}_\nu(\varphi) \textit{ and } v \models \gamma\}
\end{aligned}$$

The noisy SE language is defined as the set of all signal/parameter valuation pairs, such that the distance of the signal from the ideal shape signal defined by the shape expression and instantiated by the parameter valuation is smaller or equal than the noise threshold, according to some metric μ. In this paper, we use the *mean squred error* as our metric. Let $w = (t_1, v_1) \cdots (t_n, v_n)$ and $\hat{w} = (t_1, \hat{v}_n) \cdots (t_n, \hat{v}_n)$ be two signals over X of size n, where w represents the observed signal and $\hat{w}$ a prediction model. The mean squared error $\mathrm{MSE}(w, \hat{w})$ of w relative to $\hat{w}$ is a statistical measure of how well the predictions of a (regression) model approximates the observations, and is defined as follows:

$$\mathrm{MSE}(w, \hat{w}) = \frac{1}{n \times |X|} \sum_{x \in X} \sum_{i=1}^{n} (v_i(x) - \hat{v}_i(x))^2$$

B. Shape Automata

We now define *shape automata*, which act as recognizers for shape expressions. They are akin to finite state automata in which edges are labeled by shape expressions with unknown parameters, and parameter constraints.

Definition III.3 (Shape automata)**.** *A shape automaton is a tuple $\langle P, X, Q, \Delta, S, F \rangle$, where (1) P is the set of* parameters*, (2) X is the set of real-valued* signal variables*, (3) Q is the set of control* locations*, (4) $\Delta \subseteq Q \times \mathcal{P}(\Sigma_X(P)) \times \Gamma(P) \times Q$ is the set of* edges*, (5) $S \subseteq Q$ is the set of* starting *locations, and (6) $F \subseteq Q$ is the set of* final *locations.*

Shape expressions and shape automata are equivalent and inter-translatable.

Theorem III.1 (SE $\Leftrightarrow$ SA [11])**.** *For any shape expression φ there exists a shape automaton $\mathcal{A}_\varphi$ such that $\mathcal{L}_\nu(\mathcal{A}_\varphi) = \mathcal{L}_\nu(\varphi)$ for all $\nu \geq 0$. For any shape automaton $\mathcal{A}$ there exists a shape expression $\varphi_\mathcal{A}$ such that $\mathcal{L}_\nu(\varphi_\mathcal{A}) = \mathcal{L}_\nu(\mathcal{A})$ for all $\nu \geq 0$.*

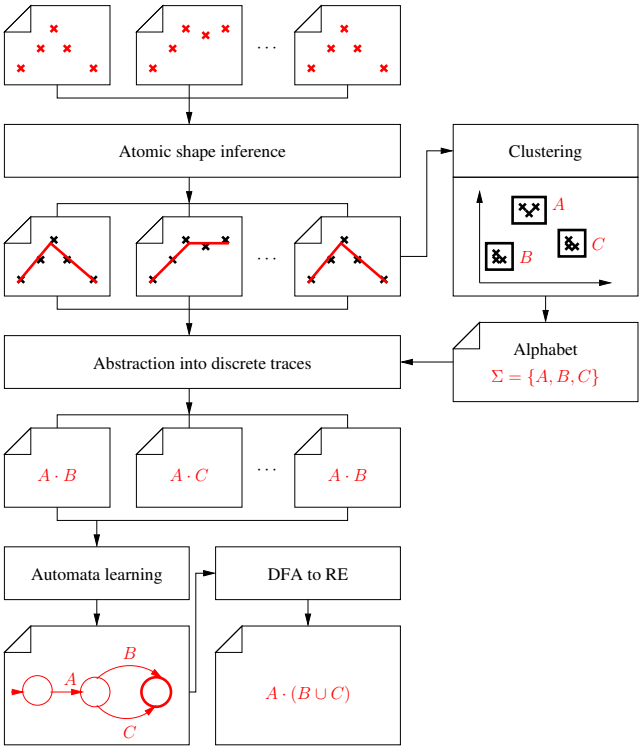


Fig. 2: Learning shape expressions from examples - an overview.

IV. Learning Shape Expressions from Examples

In this section, we present the procedure for mining shape expressions from positive examples. The high-level overview of the approach is depicted in Figure 2. The procedure starts by approximating each time series into a sequence of linear segments. Each individual dimension x in a segment is fully characterized by three parameters: (1) slope a_x, (2) (relative) offset b_x and (3) duration d_x. In the next step, the procedure collects all the inferred segments from all examples, and clusters them according to a_x, b_x and d_x using the k-means clustering method, where k is dynamically determined. We use the clustering outcomes to generate a finite alphabet Σ of size k, where each letter in Σ corresponds to a specific cluster. In the next step, we map each line segment approximation to its associated letter and hence obtain a set of finite traces over that alphabet. We finally use a passive automata learning algorithm to learn a deterministic finite automaton (DFA) from the set of words, which we translate to a shape expression. In the remainder of the section, we present in detail each individual step of the procedure.

A. Approximating Time Series with Sequences of Linear Segments

Let $w = (t_1, v_1) \cdots (t_n, v_n)$ be a signal of size n and ϵ an error threshold. In this section, we present a dynamic programming-based method that finds the piecewise-linear approximation of w that minimizes its number of linear segments while ensuring that each segment is approximated with MSE bounded by ϵ.

The idea of using dynamic programming (DP) to compute a piecewise-linear approximation of a signal is not new. Bellman

and Roth introduced a DP-based algorithm decades ago [50]. They showed that DP offers a simple and direct approach to determine the linear segmentation on a predefined grid which best approximates a given signal. This idea has been recently (i) adapted to fit the linear segments directly on the signal rather than selecting them from a predefined grid, and (ii) generalized to the problem of segmentation of time-varying affine autoregressive exogenous (ARX) models of the form:

$$y_t = \sum_{i=1}^{n_a} a_t^i y_{t-i} + \sum_{i=1}^{n_c} c_t^i u_{t-i} + k_t + \eta_t \quad (1)$$

where u, y and η denote the input, output and noise, respectively, and $t \in [t_0, N]$ with $t_0 = max(n_a, n_c)$ [54], [55].

The method we use in this work is a special case of the approach proposed in [54], [55] in which we set $n_a = 0$, $n_c = 1$ and $u_{t-1} = t$ in equation (1) and select the mean squared error, i.e. the ℓ_2-norm with square, as fitting error function. Moreover, we take the maximum among the fitting errors of the single segments, i.e. the ℓ_∞-norm, instead of their sum to define the overall fitting error of a segmentation. This will slightly change the recursion equation in the DP setting but the approach remains basically the same as in [54].

In the following, we sketch an algorithm which efficiently solves the problem of finding the minimum number of switches with bounded fitting error as introduced in [54] under the above mentioned settings.

We first compute the linear regression and its associated MSE for every segment of w. Let $P = \{a_x, b_x \mid x \in X\}$ be slope and relative offset parameters for each dimension $x \in X$. Given a segment $u = w[i:j]$, $1 \leq i < j \leq |w|$, we denote by $\mathsf{lr}(u)$ the *linear regression* of u, defined as

$$\mathsf{lr}(u) = \text{argmin}_{v \in V(P)} \text{MSE}(u, \hat{u}_v),$$

where $\hat{u}_v$ is the linear approximation of u with respect to v, i.e., we have $\hat{u}_v = (t_i, \hat{v}_i) \cdots (t_j, \hat{v}_j)$ such that $\hat{v}_k(x) = v(a_x)t_k + (v(b_x) - v(a_x)t_k)$, for all $i \leq k \leq j$ and $x \in X$.

We store the linear regression and its associated MSE for every segment of w in arrays $\mathfrak{p}$ and $\mathfrak{e}$, respectively. In particular, we have that $\mathfrak{p}[i][j] = v^\star$ and $\mathfrak{e}[i][j] = \text{MSE}(u, \hat{u}_{v^\star})$, where $u = w[i:j]$ and $v^\star = \mathsf{lr}(u)$. For simplicity, we will denote by $\text{MSE}^{\mathsf{lr}}(u)$ the MSE of a segment u relative to its optimal linear approximation $\hat{u}_{v^\star}$ computed by $\mathsf{lr}(u)$, i.e. $\text{MSE}^{\mathsf{lr}}(u) = \text{MSE}(u, \hat{u}_{v^\star})$.

A *split* τ of w is a sequence $\tau = \{s_1, s_2, \ldots, s_k, s_{k+1}\}$ of indices such that $1 \leq k < n$, $s_1 = 1$ and $s_{k+1} = n$. We denote by $|\tau| = k + 1$ the size of τ. Note that τ induces over w exactly k adjacent segments $w[s_i : s_{i+1}]$, $i \in [1, k]$. If $k = 1$, $\tau = \{1, n\}$ induces exactly one segment $w[1:n]$ over w which is the whole signal w.

We now lift the definitions of linear regression and MSE to the piecewise-linear case, driven by the split τ. We define the *τ-linear regression*, denoted by $\mathsf{lr}_\tau(w)$, as the sequence $\mathsf{lr}_\tau(w) = \{v_1^\star, \ldots, v_k^\star\}$ of parameter valuations, where for all $i \in [1, k]$, $v_i^\star = \mathsf{lr}(w[s_i : s_{i+1}])$. We define the error of the τ-linear regression, denoted by $\mathcal{E}_w(\tau)$, as the dominant MSE among all $\text{MSE}^{\mathsf{lr}}(w[s_i : s_{i+1}])$, $i \in [1, k]$:

$$\mathcal{E}_w(\tau) = \max_{i \in [1,k]} \text{MSE}^{\mathsf{lr}}(w[s_i : s_{i+1}])$$

We say that τ ϵ-approximates w, denoted by $\tau \sim_\epsilon w$, if $\mathcal{E}_w(\tau) \leq \epsilon$.

Given another split τ' of w, we say that τ and τ' are ϵ-equivalent, denoted by $\tau =_\epsilon \tau'$ if $|\tau| = |\tau'|$ and $\mathcal{E}_w(\tau) = \mathcal{E}_w(\tau')$. We say that τ ϵ-refines τ', denoted by $\tau <_\epsilon \tau'$ if $\tau \sim_\epsilon w$, $\tau' \sim_\epsilon w$ and either $|\tau| < |\tau'|$ (first criteria) or $|\tau| = |\tau'|$ and $\mathcal{E}_w(\tau) < \mathcal{E}_w(\tau')$ (second criteria). Moreover, we say that τ is *ϵ-optimal* for w if $\tau \sim_\epsilon w$ and $\tau \leq_\epsilon \tau'$ for all τ' of w such that $\tau' \sim_\epsilon w$.

We are now ready to define our method split, presented in Algorithm 1, that finds an optimal τ-split of w with respect to the threshold ϵ_{max}. The procedure is implemented as a dynamic programming algorithm that recursively finds optimal splits for sub-segments of w. The algorithm maintains the arrays $\mathfrak{s}$ and $\mathfrak{e}$ that in each cell $\mathfrak{s}[i][j]$ (originally initialized to the empty sequence $\{\}$) and $\mathfrak{e}[i][j]$ (originally initialized to $\text{MSE}^{\mathsf{lr}}(w[i:j])$) stores the ϵ_{max}-optimal split τ^* of $w[i:j]$ and $\mathcal{E}_w(\tau^*)$, respectively. The procedure[3] is initially invoked with $\mathsf{split}(w)$.

The inductive step $\mathsf{split}(w[i,j])$ works as follows. The algorithm first checks whether an optimal split for $w[i:j]$ has been already computed and stored in $\mathfrak{s}[i][j]$. If yes, the available result is returned (lines $2-3$). If not, the procedure needs to compute the optimal split. There are two possibilities. If $w[i:j]$ can be linearly approximated with regression MSE smaller than or equal to ϵ_{max}, then no further split is needed (lines $5-6$). Otherwise, $w[i:j]$ must be split in smaller segments (lines $7-13$). We first initialize τ^* to the set of all indices of $w[i:j]$ and ϵ^* to the corresponding $\mathcal{E}_w(\tau^*)$ which is 0 since all segments induced by this τ^* over $w[i:j]$ consist of just 2 data points (line 8). We then generate all promising ϵ_{max}-optimal split candidates of $w[i:j]$ (lines $9-12$) and we update τ^* and ϵ^* when the current candidate ϵ_{max}-refines the current τ^* (lines $13-14$). That is, for all $k \in [i+1, j-1]$, the algorithm recursively applies $\mathsf{split}(i,k)$ to compute an optimal split τ_L of the prefix segment $w[i:k]$ and loads from $\mathfrak{e}[i][k]$ its $\mathcal{E}_w(\tau_L)$ into ϵ_L. Moreover, the algorithm approximates the remaining suffix segment $w[k:j]$ with just one line segment given by the linear regression of $w[k:j]$ and loads in ϵ_R its MSE^{lr} available in $\mathfrak{e}[i][k]$ (line 10). It then builds a new ϵ_{max}-optimal split candidate τ by joining τ_L with τ_R and computing its $\mathcal{E}_w(\tau)$ denoted by ϵ as the maximum between ϵ_L and ϵ_R (line 11). If $\tau \sim_{\epsilon_{max}} w[i:j]$, i.e. $\epsilon \leq \epsilon_{max}$, and $\tau <_{\epsilon_{max}} \tau^*$ (line 12), then τ^* is updated with the new candidate τ (line 13). After k iterations, τ^* and ϵ^* contain the first encountered ϵ_{max}-optimal split of $w[i:j]$ and its corresponding $\mathcal{E}_w(\tau^*)$, respectively. Finally, τ^* and ϵ^* are cached to $\mathfrak{s}[i][j]$ and $\mathfrak{e}[i][j]$ (line 14) and the procedure returns τ^*.

Example IV.1. *Figure 3 depicts the optimal τ-split of the raw pulse time series resulting from the application of the split procedure.*

Next, we show that our split procedure computes an ϵ_{max}-optimal segmentation of w in time quadratic to the size of the trace.

Theorem IV.1. *Let w be a signal, $\epsilon_{\max}$ a threshold and $\tau = \mathsf{split}(w)$. We have that τ is an $\epsilon_{\max}$-optimal split of w.*

[3]To simplify the presentation, we assume that ϵ_{max}, $\mathfrak{p}$, $\mathfrak{e}$ and $\mathfrak{s}$ are global variables which are initialized accordingly before $\mathsf{split}(w)$ is invoked.

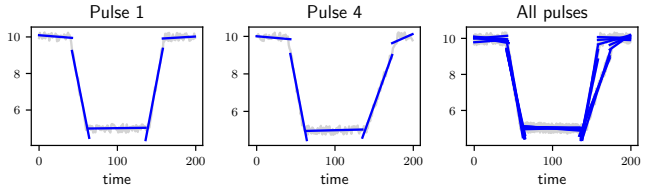


Fig. 3: Example - inferring linear segments from pulses.

Algorithm 1: split - optimal splitting of a signal segment.

Input : w - signal of the form $(t_i, v_i) \cdots (t_j, v_j)$
Output: τ^* - an ϵ_{max}-optimal split of w

Function *split*(w)
  if $\mathfrak{s}[i][j] \neq \{\}$ **then**
    $\tau^* \leftarrow \mathfrak{s}[i][j]$
  else
    if $\mathfrak{e}[i][j] \leq \epsilon_{\max}$ **then**
      $\tau^* \leftarrow \{i, j\}; \epsilon^* \leftarrow \mathfrak{e}[i][j]$
    else
      $\tau^* \leftarrow \{i, i+1, \ldots, j\}; \epsilon^* \leftarrow 0$
      for $k \in \{i+1, ..., j-1\}$ **do**
        $(\tau_L, \epsilon_L) \leftarrow (\mathsf{split}(w[i,k]), \mathfrak{e}[i][k])$
        $(\tau_R, \epsilon_R) \leftarrow (\{k, j\}, \mathfrak{e}[k][j])$
        $\tau \leftarrow \tau_L \cup \tau_R; \epsilon \leftarrow max(\epsilon_L, \epsilon_R)$
        if $\epsilon \leq \epsilon_{\max}$ *and* $(|\tau| < |\tau^*|$ *or* $(|\tau| = |\tau^*|$ *and* $\epsilon < \epsilon^*))$ **then**
          $\tau^* \leftarrow \tau; \epsilon^* \leftarrow \epsilon$
    $\mathfrak{s}[i][j] \leftarrow \tau^*; \mathfrak{e}[i][j] \leftarrow \epsilon^*$
  return τ^*

Proof Sketch. Assume that split* is a procedure that computes an ϵ_{max}-optimal split of a signal. Consider an arbitrary signal w and let $w = w_1 \cdot w_2 \cdots w_n$ be an optimal partition $\tau^* = \mathsf{split}^*(w)$ of w in n segments with $\mathcal{E}_w(\tau^*) = \max_{1 \leq i \leq n} \mathrm{MSE}^{\mathsf{lr}}(w_i)$. We prove that $\tau = \mathsf{split}(w)$ splits w in exactly n segments with $\mathcal{E}_w(\tau) = \mathcal{E}_w(\tau^*)$. We first prove by induction on the number of segments in the optimal solution that for every $1 \leq i \leq n$, $\tau_i = \mathsf{split}(w_1 \cdots w_i)$ splits $w_1 \cdots w_i$ in at most i segments with $\mathcal{E}_w(\tau_i) \leq \max_{1 \leq j \leq i} \mathrm{MSE}^{\mathsf{lr}}(w_j)$. Thus $\mathsf{split}(w_1 \cdots w_n)$ splits w in at most n segments with $\mathcal{E}_w(\tau) \leq \mathcal{E}_w(\tau^*)$. By assumption that $w = w_1 \cdot w_2 \cdots w_n$ is an optimal split, it follows that $\mathsf{split}(w)$ splits w in exactly n segments with $\mathcal{E}_w(\tau) = \mathcal{E}_w(\tau^*)$. Base case $i = 1$: by assumption, we have that $\mathrm{MSE}^{\mathsf{lr}}(w_1) \leq \epsilon_{max}$, hence by the definition of Algorithm 1 $\tau_1 = \mathsf{split}(w_1)$ does not further split w_1 and $\mathcal{E}_{w_1}(\tau_1) = \mathrm{MSE}^{\mathsf{lr}}(w_1)$. For $1 < i < n$, assume by inductive hypothesis that $\tau_i = \mathsf{split}(w_1 \cdots w_i)$ splits the $w_1 \cdots w_i$ signal in i segments with $\mathcal{E}_{w_1 \cdots w_i}(\tau_i) \leq \max_{1 \leq j \leq i} \mathrm{MSE}^{\mathsf{lr}}(w_j)$. We now show that $\tau_{i+1} = \mathsf{split}(w_1 \cdots w_i \cdot w_{i+1})$ splits $w_1 \cdots w_i \cdot w_{i+1}$ in $i+1$ segments with $\mathcal{E}_{w_1 \cdots w_{i+1}}(\tau_{i+1}) \leq \max_{1 \leq j \leq i+1} \mathrm{MSE}^{\mathsf{lr}}(w_j)$. By assumption, $\mathrm{MSE}^{\mathsf{lr}}(w_{i+1}) \leq \epsilon_{max}$. It follows that the partition of split of $w_1 \cdots w_{i+1}$ into split of $w_1 \cdots w_i$ and split of w_{i+1} is a valid candidate, according to Algorithm 1. By assumption, we have that split splits $w_1 \cdots w_i$ into i segments and split splits w_{i+1} into a single segment. In addition, we have that $\mathcal{E}_{w_1 \cdots w_{i+1}}(\tau_{i+1}) = \max(\mathcal{E}_{w_1 \cdots w_i}(\tau_i), \mathrm{MSE}^{\mathsf{lr}}(w_{i+1})$ by definition of split. By assumption, we have that $\mathcal{E}_{w_1 \cdots w_i}(\tau_i) \leq \max_{1 \leq j \leq i} \mathrm{MSE}^{\mathsf{lr}}(w_j)$. It follows that $\mathcal{E}_{w_1 \cdots w_i}(\tau_{i+1}) \leq \max_{1 \leq j \leq i+1} \mathrm{MSE}^{\mathsf{lr}}(w_j)$. Since there may be more optimal split candidates, it follows that split splits $w_1 \cdots w_{i+1}$ into at most $i+1$ segments with error bounded by $\max_{1 \leq j \leq i+1} \mathrm{MSE}^{\mathsf{lr}}(w_j)$. ■

The linear segmentation of w of size $|w| = n$ requires computing the linear regressions for all $w[i:j]$, $1 \leq i < j \leq n$, which makes $\frac{n^2-n}{2}$ linear regressions in total. Lemma IV.1 shows that this can be done in $\mathcal{O}(n^2)$ time by employing the incremental linear regression.

Lemma IV.1. *Let w be a signal of size n. Then the linear regressions for all $w[i:j]$, $1 \leq i < j \leq n$, are computed in $\mathcal{O}(n^2)$ time.*

Proof Sketch. The linear regressions for all $w[i;j]$, $j \in [i+1, n]$, can be obtained as by-products by incrementally computing the linear regression of $w[i:n]$. Thus, we only need to incrementally compute the linear regressions for $n-1$ segments comprising $n, n-1, \ldots 2$ data points, respectively. Since the incremental simple linear regression of a segment comprising k data points is computed in $\mathcal{O}(k)$ time, it follows that the $n-1$ incremental linear regressions are computed in $\mathcal{O}(n + (n-1) + \ldots + 2) = \mathcal{O}(n^2)$ time. ■

Proposition IV.1. *Let w be a signal of size n and $\epsilon_{\max}$ a threshold. Then split(w) is computed in $\mathcal{O}(n^2)$ time.*

Proof Sketch. By construction, split(w) invokes itself recursively to first solve the optimization sub-problems for the sub-segments of w in the following order: split$(w[1:2])$, split$(w[1:3])$, ..., split$(w[1:n-1])$. That is, for each k, $2 \leq k \leq n-1$, split$(w[1:k])$ is invoked exactly $n-k$ times. Thus, we can optimize the split algorithm by caching the result of split$(w[1:k])$ when it is invoked the first time and retrieving it from the cache in constant time at all successive invocations. Computing split$(w[1:k])$ when it is invoked the first time can be done in $\mathcal{O}(k)$ time since it requires the access of $k-2$ cached results and the computation of a finite number of other operations which is upper bounded by a constant. The remaining $n-k-1$ invocations of split$(w[1:k])$, each one requiring the access in constant time of the cached result, are computed in $\mathcal{O}(n-k)$ time. It follows that split(w) can be computed in $\mathcal{O}(\sum_{k=2}^{n-1} n) = \mathcal{O}(n^2)$ time. ■

From Lemma IV.1 and Proposition IV.1 it follows that, given a signal w of size n and a threshold ϵ_{max}, an ϵ_{max}-optimal split of w is computed in $\mathcal{O}(n^2)$ time. Indeed, we first need to compute the linear regressions for all $w[i:j]$, $1 \leq i < j \leq n$, which is done in $\mathcal{O}(n^2)$ time according to Lemma IV.1, and subsequently run split(w), which is also executed in $\mathcal{O}(n^2)$ time according to Proposition IV.1. Note that the time complexity $\mathcal{O}(n^2)$ includes the worst case scenario which occurs when split(w) performs the maximum possible number of recursion invocations in order to compute an ϵ_{max}-optimal split, i.e. the ϵ_{max}-optimal split has $n-1$ segments. This is a polynomial order lower than the complexity from [54],

[55] applied to our simple linear regression setting. This optimization is possible by employing the incremental linear regression which allows to produce the results of $\frac{n^2-n}{2}$ simple linear regressions by computing only $\mathcal{O}(n)$ simple linear regressions, each one in $\mathcal{O}(n)$ time.

Approximating Noisy Non-Linear Data with Sequences of Linear Segments: The segmentation algorithm presented in this section is particularly effective in approximating behavior of piecewise-linear systems. For this class of systems, it is natural to associate the MSE resulting from the segmentation algorithm to the presence of noise in the observed data. The same segmentation procedure can also be used to approximate non-linear behaviors. It should be noted that the MSE resulting from the application of our segmentation method to non-linear data does not only come from the noise in data, but also from the linear approximation of non-linearities. There is a trade-off between having an accurate approximation that requires multiple linear segments and the risk of over-fitting the data. In addition, the non-linearity approximation error can reduce the robustness of the segmentation procedure to noise.

We illustrate the challenge of segmenting noisy non-linear data with an *exponential decay* example to which we add *normally distributed noise*. Figure 4 shows the effect of varying the maximum noise threshold ν and the standard deviation σ of the normally distributed noise with mean 0 to the outcome of the segmentation procedure. We can observe that the procedure consistently approximates the signal with two segments for $\sigma \in [0, 0.03]$ and $\nu \in [0.005, 0.01]$. By lowering ν to 0.002, the less noisy signals are still approximated with two segments. However, the signal with noise that has $\sigma = 0.03$ requires a third approximation segment. Similarly, less noisy signals are approximated with three segments when ν equals to 0.001, while the noisiest signal with $\sigma = 0.03$ requires a four-segment approximation. We can see that the level of noise and non-linearity in data can affect the outcome of the segmentation procedure.

B. Abstracting Sequences of Linear Segments to Finite Traces over Finite Alphabets

Clustering is an unsupervised learning method for grouping data points with similar properties. Given a set of data points U and a finite alphabet Σ, clustering procedures aim at finding an appropriate labeling function $\lambda : U \to \Sigma$ that maps data points $u \in U$ to letters in Σ. Clustering algorithms aim at grouping data points with similar features, while ensuring highly dissimilar features between different clusters.

Let $P = \{a_x, b_x, d_x \mid x \in X\}$ be the set of parameters, where a_x is the slope, b_x the relative offset and d_x the duration of a segment defined over $x \in X$. A valuation $v(P)$ assigns real values to these parameters, and $V(P)$ denotes the set of all valuations over P. We note that valuations in $V(P)$ provide unique characterizations of linear (multi-dimensional) segments. Given a set $U \subseteq V(P)$ of parameter valuations obtained in Section IV-A, our objective is to group and label similar segments.

We use the well-known *k-means clustering* method [56] to find our labeling function λ. This procedure aims to partition $|U|$ valuations into k clusters in which each parameter valuation $v \in U$ belongs to the cluster with the nearest mean (called *centroid*), serving as a representative of the cluster. Implicit objective function in k-means measures the *within cluster sum of squares* (WCSS), which is the sum of distances of data points from their cluster centroids.

First, the procedure initializes k centroids. The common implementation of this step consists in chosing random values for centroids. In the second phase, the algorithm repeats the following actions until it converges: (1) it calculates the distance between all valuations and centroids, assigning each valuation to the cluster represented by its closest centroid, and (2) it updates centroid positions by finding the average values of data points that are part of the cluster.

We use the popular *elbow method* to find the optimal number k of clusters for U. It is typically used as a visual method, consisting of plotting the value of the WCSS produced by different values of k and finding the "elbow" in the plot. We automate this step by defining a threshold c and stopping when the difference between WCSS for k and $k-1$ clusters falls below c.

The k-means clustering algorithm maps U into an alphabet Σ of size $|\Sigma| = k$. In the next step, we provide a meaning to each letter $a \in \Sigma$. Intuitively, for each cluster, we define the *tightest* box that includes all of its data points (parameter valuations) and represent it symbolically as a set of constraints. Let $\lambda^{-1} : \Sigma \to 2^U$ denote the *inverse* labelling function. For each $A \in \Sigma$, we associate a constrained atomic shape expression $\psi_A = \bigwedge_{x \in X} \mathsf{lin}_x(a_x, b_x, d_x) : a_x \in [a_x^1, a_x^2] \wedge b_x \in [b_x^1, b_x^2] \wedge d_x \in [d_x^1, d_x^2]$, where for all $p \in P$, $p_1 = \min_{v \in \lambda^{-1}(A)} v(p)$ and $p_2 = \max_{v \in \lambda^{-1}(A)} v(p)$.

Example IV.2. *Figure 5 (top-left) shows all the pulse linear segment parameters projected on the slope and duration valuations. Figure 5 (top-right) depicts the result of applying k-means clustering to these segments and for different values of k, while Figure 5 (bottom-left) shows the parameters grouped into $k = 5$ clusters, the optimal number of clusters according to the automated elbow method. Figure 5 (bottom-right) illustrates the bounding box that over-approximates all the data points in the cluster labelled by A. We associate to the letters of the inferred alphabet the following symbolic expressions:*

$$\begin{array}{lcl} A & : & \mathit{lin}(a_1, b_1, d_1) : a_1 \in [-0.261, -0.197] \wedge \\ & & b_1 \in [9.06, 9.70] \wedge d_1 \in [18, 24] \\ B & : & \mathit{lin}(a_2, b_2, d_2) : a_2 \in [-0.007, 0.031] \wedge \\ & & b_2 \in [9.23, 10.09] \wedge d_2 \in [26, 44] \\ C & : & \mathit{lin}(a_3, b_3, d_3) : a_3 \in [-0.001, 0.001] \wedge \\ & & b_3 \in [4.96, 5.12] \wedge d_3 \in [69, 74] \\ D & : & \mathit{lin}(a_4, b_4, d_4) : a_4 \in [0.110, 0.116] \wedge \\ & & b_4 \in [4.43, 4.60] \wedge d_4 \in [38, 39] \\ E & : & \mathit{lin}(a_5, b_5, d_5) : a_5 \in [0.199, 0.263] \wedge \\ & & b_5 \in [4.35, 4.45] \wedge d_5 \in [20, 22] \end{array} \quad (2)$$

C. Inferring Expressions from Finite Traces

In the previous section, we showed how we can partition the set of linear segments to a few equivalence classes through discretization of their parameter space. We also showed how we can associate each equivalence class with a symbol. Let the set of such symbols be denoted by Σ. Essentially, through the methods in previous sections, we have mapped each timed trace into a string in $\Sigma^\star$. In this section, we show how we can learn a *deterministic finite automaton* (DFA)

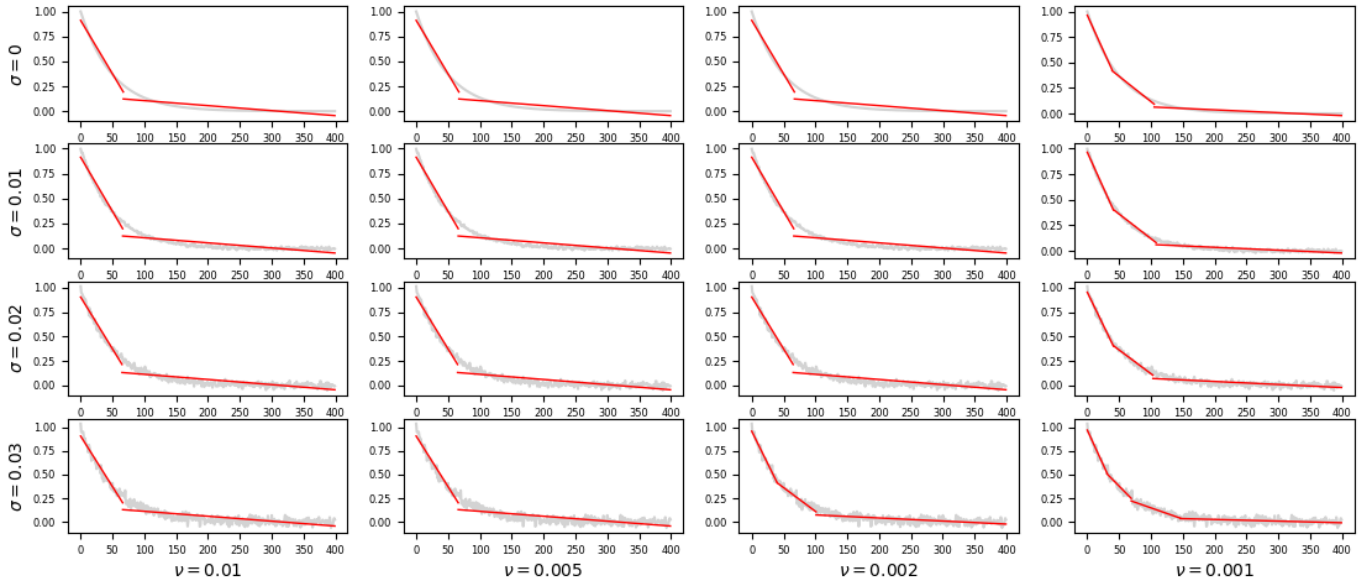


Fig. 4: Approximating exponential decay with linear segments – robustness to noise in data.

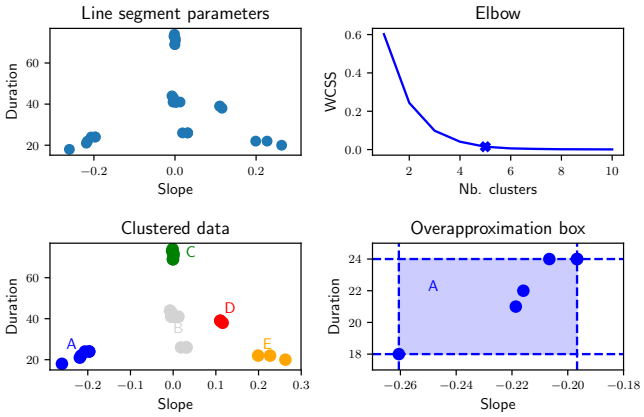
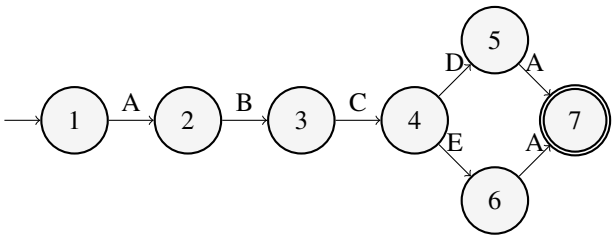


Fig. 5: Example - clustering.



$$\varphi = A \cdot B \cdot C \cdot (D \cup E) \cdot A \tag{3}$$

Fig. 6: Inferred automaton and expression.

that accepts every string corresponding to all the positive examples in our dataset. For this purpose, we use an off-the-shelf DFA learning algorithm called RPNI (Reduced Positive and Negative Inference) [44].

Given a set S of example strings, the first step in RPNI is to construct the *prefix tree acceptor* (PTA) from the given examples. The PTA $\mathcal{A}$ is described as a tuple $(Q, \Sigma, q_\lambda, \delta, \mathbb{F}_A)$, where Q is the finite set of states, q_λ is the initial state, $\delta \subseteq Q \times \Sigma \to Q$ is the transition function, and $\mathbb{F}_A$ is the set of accepting states. The set Q is essentially the prefix-closure of S, and for every state $q_u \in Q$, and every $a \in \Sigma$, $\delta(q_u, a) = q_{ua}$. Finally, if $u \in S$, $q_u \in \mathbb{F}_A$.

While a PTA constructed using the above rules is a DFA accepting the given set of examples, it is typically not the minimally consistent DFA. The algorithm RPNI is a heuristic to minimize the DFA. A common idea in passive learning of DFAs from examples is to label the states of the PTA as RED (which is initially just the initial state corresponding to the empty string), and BLUE (initially the prefix-closure of the set S). The RPNI algorithm then repeatedly selects a BLUE state and seeks to merge it with a RED state, while ensuring that the merged automaton is compatible with the given set of examples.

Example IV.3. *We use the RPNI algorithm to learn an automaton from a set of finite traces obtained from abstracting the segmented pulses and map it to its associated shape expression. Figure 6 depicts the inferred automaton and the shape expression. We can observe that the variability in the slope of the before-last segment is translated into the choice of taking either the transition labelled by D or E from the state 4, or equivalently into the union operation in the shape expression. The procedure is thus effectively able to distinguish between two different classes of pulses in unsupervised fashion.*

The combination of the specification structure that uses regular expressions with parameterized linear segments as expression atoms facilitates understating and explaining the mined shape. The explainability of the shape can be further enhanced by visualizing segments that satisfy the specification. Figure 7 visualizes the pulse specification with a set of synthetically generated examples that match the shape (blue). The figure also shows the training examples (light blue) to

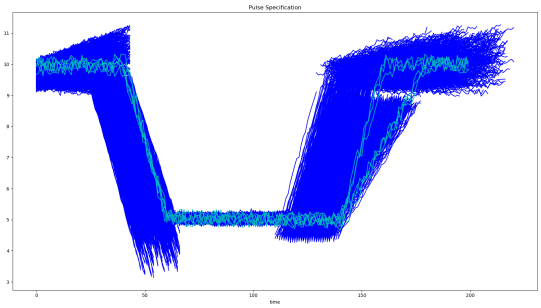


Fig. 7: Visualizing the mined specification.

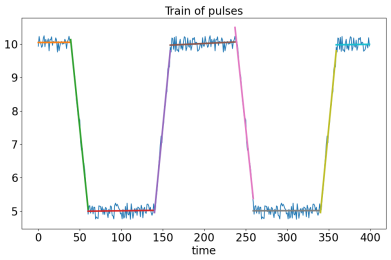


Fig. 8: Train of pulses and its segmentation.

illustrate the generalization done by the specification.

We finally also illustrate how our procedure generalizes the repeated shapes by learning repetitions in the form of a Kleene start. The result of using both individual pulse shapes of the first class, together with a pulse train signal shown in Figure 8 results in learning the specification $(A^* \cdot B \cdot C \cdot D)^+$

D. Discussion

The specification mining procedure, presented in this section, adopts several *design choices* that we motivate but we also discuss potential alternatives:

- *Linear segments:* the restriction to linear atomic shapes has a twofold motivation: (1) linear approximations are easy to understand and hence facilitate explainability of inferred specifications, and (2) linear approximations admit efficient regression algorithms. For some applications, linear segments may not be sufficient to faithfully capture a shape. In addition, linear approximation of non-linear behaviors may not be sufficiently robust to noise, as discussed in the las paragraph of Section IV-A. Our approach can be generalized to richer atomic shapes at the expense of computational efficiency.
- *Duration as a parameter:* we chose to treat segment duration as a parameter that we use in the process of mapping raw data to a finite trace. In practice, two segments with similar slopes and relative offsets, but different durations my be mapped to two different letters of the inferred finite alphabet. An alternative approach would consist in treating the duration as a special parameter. We could use the slope and the relative offset to map sequences of linear segments to finite words, and then use durations to extend finite words to *timed words*. We could use methods such as timed k-trail [57] to learn a *timed automaton* from the set of timed traces.
- *Semantics of a Letter in the Alphabet:* we map each letter in the finite alphabet to a box constraint over parameters that include all the segments in the cluster that characterizes that letter. There are several other approaches that could be used: (1) associating a more sophisticated over-approximation, such as a zonotope, of the segments in the cluster, or (2) defining the cluster centroid as the representative of the cluster and increase the noise tolerance based on the properties of the cluster. To simplify the presentation, we also use the minimal bounded box that includes all the points in the cluster. This means that the over-approximation is not robust for the extreme points in the cluster. This problem can be addressed by bloating the bounding box by an amount defined by the user.
- *Clustering and Learning Algorithms:* we use k-means clustering and RPNI as off-the-shelf algorithms to do clustering and passive learning from positive examples. These algorithms can be replaced with other procedures that implement the same function in a different manner. For instance, we can use silhouette clustering [58] instead of k-means clustering.

V. Experimental Evaluation

In this section, we analyze both computational and qualitative aspects of our specification mining procedure and demonstrate its usability on two case studies. All the experiments were performed on a Razer computer with the Intel Core i7 4.1 GHz processor and 16GB RAM.

A. Experimental Results

We use our illustrative train of pulses to study computational and qualitative aspects of our specification mining procedure. We first analyse the scalability of our approach with respect to both the size and the number of the training examples. We summarize the results in Table I, where t_s, t_c, t_l and t_{total} denote segmentation, clustering, automata learning and total time. We can first observe that the experimental results follow the theoretical quadratic growth in the size of the signals and linear growth in the number of signals. We can also see that the segmentation part dominates the computation. This result is expected given that the segmentation is used to map a usually large number of raw data samples into a relatively small number of segments. The computation time for automata learning is negligible with respect to other parts of the procedure. This is mainly due to the fact that there are often equivalent abstract (finite) traces that are fed to the automata learning algorithm.

Next, we analyse the sensitivity of our approach to the choice of the maximum error threshold ϵ_{max}. Table II depicts shape expressions inferred from the same set of training examples (see Figure 1) under different maximum error thresholds and clustering termination criteria. As expected, the maximum error threshold can significantly influence the segmentation part of the algorithm and have an impact on the learned specification. It is interesting to observe that the size of the threshold does not monotonically affect the size of the specification.

B. Mining Patterns in ECG Data

In this case study, we consider ECG signals from the PhysioBank database [59], which contains 549 records from 290 subjects (209 male and 81 female, aged from 17 to 87). Each

# traces	$\|w\|$	$t_s(s)$	$t_c(s)$	$t_l(s)$	$t_{total}(s)$
1	10	0.0001	0.1020	0.0006	0.1027
1	100	0.0105	0.1451	0.0146	0.1698
1	1000	0.7733	0.2982	0.3081	1.3796
10	10	0.0010	0.1109	0.0006	0.1124
10	100	0.0909	0.2162	0.0172	0.3242
10	1000	7.2220	1.6809	0.3838	9.2867
100	10	0.0101	0.1626	0.0007	0.1734
100	100	0.9508	0.9909	0.0313	1.9730
100	1000	72.7604	16.1123	0.4196	89.2922
1000	10	0.1012	0.5912	0.0007	0.6930
1000	100	9.8420	8.7352	0.0208	18.5980
1000	1000	722.1752	149.0021	0.3944	871.5717

TABLE I: Computational cost of the specification mining algorithm.

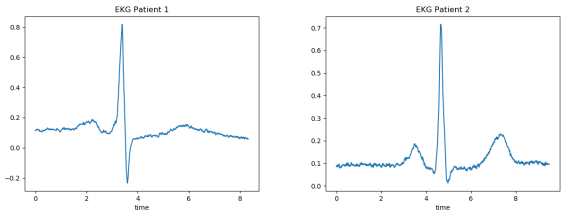
ϵ_{max}	φ	# Clusters
0.05	$A \cdot B \cdot C \cdot (D \cup E) \cdot A$	5
0.1	$A \cdot B \cdot C \cdot (D \cup E) \cdot A$	5
0.5	$F \cdot (G \cdot (H \cup I) \cup J \cdot I)$	5
1	$K \cdot L$	2
5	$(M \cdot M) \cup (N \cdot O)$	3
10	$P \cup Q$	2

TABLE II: Sensitivity of specification mining to the maximum error threshold.

record includes 15 simultaneously measured signals, digitized at 1,000 samples per second, with 16-bit resolution over a range of $\pm 16.384mV$. The diagnostic classes for the subjects participating in the recordings include cardio-vascular diseases such as myocardial infarction, cardiomyopathy, dysrhythmia and myocardial hypertrophy.

In this experiment, we considered two sets of examples. The first set of examples comes from a male 70 year old patient with myocardial infarction, an anterior acute infarction and an antero-septal former infarction. The patient had hyperlipoproteinemial as an additional diagnosis, was not a smoker at the time of the admission and had two coronary vessels involved in the disease. The second set of examples comes from a male 52 year old patient with myocardial infarction with antero-septal acute infarction and no former infarctions. The patient had gastritis and rheumatoid arthritis as additional diagnoses, was not a smoker at the time of the admission and had one coronary vessel involved in the disease. Figure 9 depicts two heart beats from patient 1 and patient 2, respectively.

Each dataset contains 125 heart beat measurements. We use both datasets to infer a common four-valued alphabet shown in Equation 4. The letters A and B represent short and long almost-constant segments, respectively, while C and D represent short segments with high negative and positive slopes, respectively.



(a) Patient 1 (b) Patient 2

Fig. 9: Example of a heart beat from two patients.

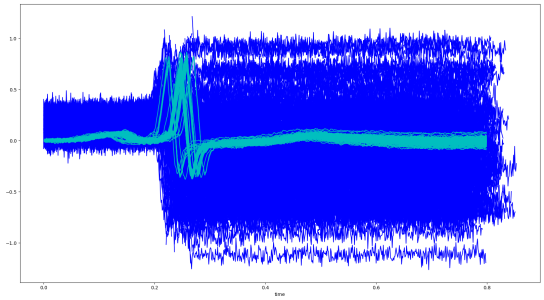


Fig. 10: Visualization of the ECG specification for patient one.

$$
\begin{array}{lcl}
A & : & \mathsf{lin}(a_1, b_1, d_1) : a_1 \in [-1.1, 1.4] \wedge \\
 & & b_1 \in [-1.8, 2.4] \wedge d_1 \in [0.04, 0.29] \\
B & : & \mathsf{lin}(a_2, b_2, d_2) : a_2 \in [-0.45, 0.14] \wedge \\
 & & b_2 \in [-1.9, 0.4] \wedge d_2 \in [0.32, 0.55] \\
C & : & \mathsf{lin}(a_3, b_3, d_3) : a_3 \in [-69, -29] \wedge \\
 & & b_3 \in [-158, -63] \wedge d_3 \in [0.015, 0.025] \\
D & : & \mathsf{lin}(a_4, b_4, d_4) : a_4 \in [5.3, 44] \wedge \\
 & & b_4 \in [1.3, 92] \wedge d_4 \in [0.008, 0.03]
\end{array}
\tag{4}
$$

We now set the maximum error threshold $\epsilon_{max} = 0.001$ and learn a a separate specification for each patient, as shown in Equation 5.

$$
\begin{array}{l}
\varphi_1 = (A^* \cdot D \cdot C \cdot D \cdot B) \\
\varphi_2 = (A^* \cdot D \cdot C \cdot A \cdot (A \cup B))
\end{array}
\tag{5}
$$

The two specifications capture the approximated heart-beat behavior of the two patients – a relatively constant behavior (A^*), followed by a peak ($D \cdot C \cdot D$ for patient 1 and $D \cdot C$ for patient two), followed by another relatively constant behavior (B for patient 1 and $A \cdot (A \vee B)$ for patient 2. We can observe that at this level of abstraction, the two patients share similar constant prefix, followed by a peak. We note that the second increase in the slope in the peak of patient two is small and is not captured with this level of abstraction. We can observe that our method captures both similarities and differences between the heartbeats of two patients. The two inferred specifications can also be explained in terms of behaviors that they accept. Figure 10 shows such a visualization where the behaviors generated from φ_2 are shown in blue and the input data is shown in cyan.

We also measured the robustness of the segmentation procedure with respect to the noise in the data. We found that for the maximum threshold error ν of 0.001 used in this case study, patient 1 and 2 data would tolerate in the worst case an additional $6e-5$ (6% of ν) and $3e-5$ (3% of ν) MSE

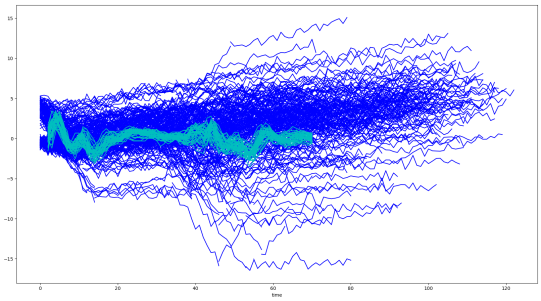


Fig. 11: Visualization of the robot motion specification.

per segment, respectively, without changing the number of generated segments.

C. Mining robot motion patterns

In [5], the authors describe a time-series dataset obtained from a SONY AIBO robot – a small dog-shaped robot mounted with a tri-axial accelerometer. In the experimental setting, the robot walks on two different surfaces: carpet and cement. Cemented floors being harder than carpet offer more reactive force to the robot, resulting in clear and sharp changes in acceleration of the robot. The time-series data corresponds to the X-axis readings of the robot labeled according to the surface on which it walks (i.e. cement or carpet). Each datum had 70 time steps, and we analyzed a total of 6 traces[4].

In this case study, we explore the machine interpretability of shape expressions. We learn a specification from 30 behaviors of robot moving on the cement surface. We set the maximum threshold error ν to 0.5 and we infer the specification over the finite alphabet shown in Equation 6.

$$\begin{array}{lcl} A & : & \mathsf{lin}(a_1,b_1,d_1) : a_1 \in [-0.78,-0.08] \wedge \\ & & b_1 \in [1.9,22] \wedge d_1 \in [4,15] \\ B & : & \mathsf{lin}(a_2,b_2,d_2) : a_2 \in [-0.004,0.08] \wedge \\ & & b_2 \in [-2,0.2] \wedge d_2 \in [36,53] \\ C & : & \mathsf{lin}(a_3,b_3,d_3) : a_3 \in [0.002,0.12] \wedge \\ & & b_3 \in [-7.1,-0.2] \wedge d_3 \in [21,32] \\ D & : & \mathsf{lin}(a_4,b_4,d_4) : a_4 \in [-0.2,0.2] \wedge \\ & & b_4 \in [0.3,2.1] \wedge d_4 \in [4,15] \\ E & : & \mathsf{lin}(a_5,b_5,d_5) : a_5 \in [0.2,0.8] \wedge \\ & & b_5 \in [-12,0.24] \wedge d_5 \in [2,10] \end{array} \quad (6)$$

We infer the specification $\varphi_3 = ((A \cup D \cup E)^* \cdot (C \cdot (A \cup E)^* \cup B) \cdot C$ and we show in Figure 11 some behaviors allowed by the specification.

The mined specification can be used to detect whether the robot is on the cement surface. Such information could be conceivably used by a supervisory controller for the robot that may make different control decisions based on the surface on which the robot walks. Figure 12 shows that the behavior of the robot waking on the carpet and on the cement are very similar. Specification φ_3 is not able to discriminate between the two surfaces due to generalization resulting from our approach doing passive learning from positive examples only. This type of problem can be addressed by using active learning and introducing negative examples that would help discriminating between the two classes of behaviors.

[4]The actual data was acquired from the UCR time-series repository [60].

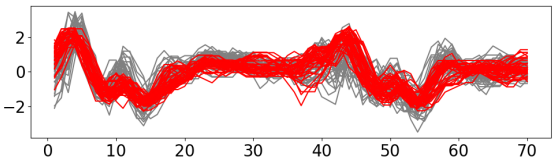


Fig. 12: Robot data - walking on cement (grey) and carpet (red).

VI. Conclusions and Future Work

We introduced a novel procedure for mining linear shape expressions from time series. It combines segmentation of raw data, clustering, abstraction and passive automata learning. We believe that the presented approach enables understanding and explaining data and facilitates discovering interesting patterns in time series. We implemented the algorithm in a prototype and applied it to two case studies from medical and robotic domains.

We plan to explore the applicability of specification mining in explainability of black-box models, testing and anomaly detection. In this paper, we restricted ourselves to linear shape expressions. We plan to extend our methodology to non-linear shapes and study the trade-off between the computational cost and the additional explainability of the specification. We investigated a passive learning approach from positive examples. We plan to also study active learning of shape expressions and learning from both positive and negative examples. The quality of the resulting specification depends on the maximum error threshold and the number of clusters used to abstract the traces. These two parameters are currently manually set and require domain knowledge. We will investigate criteria for automatically choose these parameters. In our paper, we have shown that we can use our approach to partition similar shapes into different classes, for instance to separate normal from anomalous heart beats. However, our approach does not guarantee that the intersection between the two mined specifications is empty. We plan to study refinement-based techniques to mine specifications that accurately characterize specific classes of shapes.

Acknowledgements: The authors acknowledge the support of the IKT der Zukunft Austrian FFG project ADVANCED (nr. 874044).

References

[1] D. Ratasich, F. Khalid, F. Geissler, R. Grosu, M. Shafique, and E. Bartocci, "A roadmap toward the resilient internet of things for cyber-physical systems," *IEEE Access*, vol. 7, pp. 13 260–13 283, 2019.

[2] E. J. Keogh and M. J. Pazzani, "Scaling up dynamic time warping for datamining applications," in *Proc. of KDD*, 2000, pp. 285–289.

[3] ——, "An enhanced representation of time series which allows fast and accurate classification, clustering and relevance feedback," in *Proc. of KDD*, vol. 98, 1998, pp. 239–243.

[4] L. Ye and E. Keogh, "Time series shapelets: a new primitive for data mining," in *Proc. of KDD*. ACM, 2009, pp. 947–956.

[5] A. Mueen, E. Keogh, and N. Young, "Logical-shapelets: an expressive primitive for time series classification," in *Proc. of KDD*. ACM, 2011, pp. 1154–1162.

[6] J. Lines, L. M. Davis, J. Hills, and A. Bagnall, "A shapelet transform for time series classification," in *Proc. of KDD*. ACM, 2012, pp. 289–297.

[7] M. Garnelo and M. Shanahan, "Reconciling deep learning with symbolic artificial intelligence: representing objects and relations," *Current Opinion in Behavioral Sciences*, vol. 29, pp. 17–23, 2019, sI: 29: Artificial Intelligence (2019).

[8] B. M. Lake, T. D. Ullman, J. B. Tenenbaum, and S. J. Gershman, "Building machines that learn and think like people," *Behavioral and Brain Sciences*, vol. 40, p. e253, 2017.

[9] A. Verma, V. Murali, R. Singh, P. Kohli, and S. Chaudhuri, "Programmatically interpretable reinforcement learning," in *Proc. of ICML*, ser. Proceedings of Machine Learning Research, vol. 80, 2018, pp. 5052–5061.

[10] A. Santoro, F. Hill, D. G. T. Barrett, A. S. Morcos, and T. P. Lillicrap, "Measuring abstract reasoning in neural networks," in *Proc. of ICML*, ser. Proceedings of Machine Learning Research, vol. 80, 2018, pp. 4477–4486.

[11] D. Nickovic, X. Qin, T. Ferrère, C. Mateis, and J. V. Deshmukh, "Shape expressions for specifying and extracting signal features," in *Proc. of RV*, ser. LNCS, vol. 11757, 2019, pp. 292–309.

[12] T. Dang and R. Testylier, "Hybridization domain construction using curvature estimation," in *Proc. of HSCC*. ACM, 2011, pp. 123–132.

[13] T. Dang, O. Maler, and R. Testylier, "Accurate hybridization of nonlinear systems," in *Proc. of HSCC*. ACM, 2010, pp. 11–20.

[14] E. Asarin, T. Dang, and A. Girard, "Hybridization methods for the analysis of nonlinear systems," *Acta Informatica*, vol. 43, no. 7, pp. 451–476, 2007.

[15] R. Grosu, S. Mitra, P. Ye, E. Entcheva, I. V. Ramakrishnan, and S. A. Smolka, "Learning cycle-linear hybrid automata for excitable cells," in *Proc. of HSCC*, ser. LNCS, vol. 4416. Springer, 2007, pp. 245–258.

[16] R. Grosu, G. Batt, F. Fenton, J. Glimm, C. Le Guernic, S. A. Smolka, and E. Bartocci, "From cardiac cells to genetic regulatory networks," in *Proc. of CAV*, ser. LNCS, vol. 6806. Springer, 2011, pp. 396–411.

[17] Z. Xu and A. A. Julius, "Census signal temporal logic inference for multiagent group behavior analysis," *IEEE Transactions on Automation Science and Engineering*, vol. 15, no. 1, pp. 264–277, 2018.

[18] C. Ackermann, R. Cleaveland, S. Huang, A. Ray, C. P. Shelton, and E. Latronico, "Automatic requirement extraction from test cases," in *Proc. of RV 2010*, 2010, pp. 1–15.

[19] E. Asarin, A. Donzé, O. Maler, and D. Nickovic, "Parametric identification of temporal properties," in *Proc. of RV*, ser. LNCS, vol. 7186. Springer, 2012, pp. 147–160.

[20] B. Hoxha, A. Dokhanchi, and G. E. Fainekos, "Mining parametric temporal logic properties in model-based design for cyber-physical systems," *STTT*, vol. 20, no. 1, pp. 79–93, 2018.

[21] E. Bartocci, L. Bortolussi, and G. Sanguinetti, "Data-driven statistical learning of temporal logic properties," in *Proc. of FORMATS*, 2014, pp. 23–37.

[22] X. Jin, A. Donzé, J. V. Deshmukh, and S. A. Seshia, "Mining requirements from closed-loop control models," *IEEE TCAD*, vol. 34, no. 11, pp. 1704–1717, 2015.

[23] L. V. Nguyen, J. Kapinski, X. Jin, J. V. Deshmukh, K. Butts, and T. T. Johnson, "Abnormal Data Classification Using Time-Frequency Temporal Logic," in *Proc. of HSCC*. ACM, 2017, pp. 237–242.

[24] J. Zhou, R. Ramanathan, W.-F. Wong, and P. S. Thiagarajan, "Automated property synthesis of ODEs based bio-pathways models," in *Proc. of CMSB 2017*, 2017, pp. 265–282.

[25] Z. Kong, A. Jones, and C. Belta, "Temporal Logics for Learning and Detection of Anomalous Behavior," *IEEE Trans. Aut. Control*, vol. 62, no. 3, pp. 1210–1222, Mar. 2017.

[26] G. Bombara, C.-I. Vasile, F. Penedo, H. Yasuoka, and C. Belta, "A Decision Tree Approach to Data Classification Using Signal Temporal Logic," in *Proc. of HSCC*, 2016, pp. 1–10.

[27] G. Bombara and C. Belta, "Signal clustering using temporal logics," in *RV*, 2017, pp. 121–137.

[28] S. Bufo, E. Bartocci, G. Sanguinetti, M. Borelli, U. Lucangelo, and L. Bortolussi, "Temporal logic based monitoring of assisted ventilation in intensive care patients," in *Proc. of ISoLA*, 2014, pp. 391–403.

[29] R. Medhat, S. Ramesh, B. Bonakdarpour, and S. Fischmeister, "A framework for mining hybrid automata from input/output traces," in *Proc. of EMSOFT*. IEEE, 2015, pp. 177–186.

[30] L. Nenzi, S. Silvetti, E. Bartocci, and L. Bortolussi, "A robust genetic algorithm for learning temporal specifications from data," in *Proc. of QEST 2018*, ser. LNCS, vol. 11024. Springer, 2018, pp. 323–338.

[31] D. Neider and I. Gavran, "Learning linear temporal properties," in *Proc. of FMCAD*. IEEE, 2018, pp. 1–10.

[32] P. Kyriakis, J. V. Deshmukh, and P. Bogdan, "Specification mining and robust design under uncertainty: A stochastic temporal logic approach," *ACM TECS*, vol. 18, no. 5s, pp. 96:1–96:21, 2019.

[33] S. Jha, A. Tiwari, S. A. Seshia, T. Sahai, and N. Shankar, "TeLEx: learning signal temporal logic from positive examples using tightness metric," *Formal Methods in System Design*, vol. 54, no. 3, pp. 364–387, 2019.

[34] J. Lamp, S. Silvetti, M. Breton, L. Nenzi, and L. Feng, "A logic-based learning approach to explore diabetes patient behaviors," in *Proc. of CMSB*, ser. LNCS, vol. 11773. Springer, 2019, pp. 188–206.

[35] M. G. Soto, T. A. Henzinger, C. Schilling, and L. Zeleznik, "Membership-based synthesis of linear hybrid automata," in *Proc. of CAV*, ser. LNCS, vol. 11561. Springer, 2019, pp. 297–314.

[36] G. Bombara and C. Belta, "Online learning of temporal logic formulae for signal classification," in *Proc. of ECC*, 2018, pp. 2057–2062.

[37] P. Vaidyanathan, R. Ivison, G. Bombara, N. A. DeLateur, R. Weiss, D. Densmore, and C. Belta, "Grid-based temporal logic inference," in *Proc. of CDC*. IEEE, 2017, pp. 5354–5359.

[38] O. Maler and D. Nickovic, "Monitoring properties of analog and mixed-signal circuits," *STTT*, vol. 15, no. 3, pp. 247–268, 2013.

[39] M. D. Ernst, J. H. Perkins, P. J. Guo, S. McCamant, C. Pacheco, M. S. Tschantz, and C. Xiao, "The Daikon system for dynamic detection of likely invariants," *Sci. Comput. Program.*, vol. 69, no. 1–3, p. 35–45, 2007.

[40] E. Bartocci, N. Manjunath, L. Mariani, C. Mateis, and D. Nickovic, "Automatic failure explanation in CPS models," in *Proc. of SEFM*, ser. LNCS, vol. 11724. Springer, 2019, pp. 69–86.

[41] E. Bartocci, N. Manjunath, L. Mariani, C. Mateis, D. Ničković, and F. Pastore, "CPSDebug: A tool for explanation of failures in cyber-physical systems," in *Proc. of ISSTA*. New York, NY, USA: Association for Computing Machinery, 2020, pp. 569–572.

[42] E. Bartocci, E. A. Gol, I. Haghighi, and C. Belta, "A formal methods approach to pattern recognition and synthesis in reaction diffusion networks," *IEEE Trans. Control. Netw. Syst.*, vol. 5, no. 1, pp. 308–320, 2018.

[43] R. Grosu, S. A. Smolka, F. Corradini, A. Wasilewska, E. Entcheva, and E. Bartocci, "Learning and detecting emergent behavior in networks of cardiac myocytes," *Commun. ACM*, vol. 52, no. 3, pp. 97–105, 2009.

[44] C. de la Higuera, *Grammatical Inference: Learning Automata and Grammars*. Cambridge University Press, 2010.

[45] D. Angluin, "Learning regular sets from queries and counterexamples," *Inf. Comput.*, vol. 75, no. 2, pp. 87–106, 1987.

[46] M. Isberner, F. Howar, and B. Steffen, "The TTT algorithm: A redundancy-free approach to active automata learning," in *Proc. of RV*, ser. LNCS, vol. 8734, 2014, pp. 307–322.

[47] B. Steffen, F. Howar, and M. Isberner, "Active automata learning: From DFAs to interface programs and beyond," in *Proc. of ICGI*, ser. JMLR Proceedings, vol. 21. JMLR.org, 2012, pp. 195–209.

[48] Y. Chen, J. Tumova, A. Ulusoy, and C. Belta, "Temporal logic robot control based on automata learning of environmental dynamics," *The International Journal of Robotics Research*, vol. 32, no. 5, pp. 547–565, 2013.

[49] J. Fu, H. G. Tanner, J. Heinz, and J. Chandlee, "Adaptive symbolic control for finite-state transition systems with grammatical inference," *IEEE Trans. Automat. Contr.*, vol. 59, no. 2, pp. 505–511, 2014.

[50] R. Bellman and R. Roth, "Curve fitting by segmented straight lines," *Journal of the American Statistical Association*, vol. 64, no. 327, pp. 1079–1084, 1969.

[51] T. Pavlidis and S. L. Horowitz, "Segmentation of plane curves," *IEEE Trans. Comput.*, vol. 23, no. 8, pp. 860–870, 1974.

[52] T. Pavlidis, "Linguistic analysis of waveforms," in *Computer and Information Sciences*, ser. SEN Report Series Software Engineering, J. T. TOU, Ed. Elsevier, 1971, vol. 2, pp. 203–225.

[53] U. Ramer, "An iterative procedure for the polygonal approximation of plane curves," *Computer Graphics and Image Processing*, vol. 1, no. 3, pp. 244–256, 1972.

[54] N. Ozay, "An exact and efficient algorithm for segmentation of ARX models," in *Proc. of ACC*, 2016, pp. 38–41.

[55] G. Chou, N. Ozay, and D. Berenson, "Incremental segmentation of ARX models," *IFAC-PapersOnLine*, vol. 51, no. 15, pp. 587–592, 2018, 18th IFAC Symposium on System Identification SYSID 2018.

[56] J. MacQueen *et al.*, "Some methods for classification and analysis of multivariate observations," in *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, vol. 1, no. 14. Oakland, CA, USA, 1967, pp. 281–297.

[57] F. Pastore, D. Micucci, and L. Mariani, "Timed k-tail: Automatic inference of timed automata," in *Proc. of ICST 2017*. IEEE Computer Society, 2017, pp. 401–411.

[58] P. J. Rousseeuw, "Silhouettes: a graphical aid to the interpretation and validation of cluster analysis," *Journal of computational and applied mathematics*, vol. 20, pp. 53–65, 1987.

[59] A. e. a. Goldberger, "Physiobank, physiotoolkit, and physionet: components of a new research resource for complex physiologic signals," *Circulation*, vol. 101, no. 23, pp. e215–e220, 2000.

[60] H. A. e. a. Dau, "The ucr time series classification archive," October 2018, https://www.cs.ucr.edu/∼eamonn/time_series_data_2018/.