Synthesis of Parametric Hybrid Automata
from Time Series

Miriam García Soto[1][0000−0003−2936−5719],
Thomas A. Henzinger[2][0000−0002−2985−7724], and
Christian Schilling[3][0000−0003−3658−1065]

[1] Complutense University of Madrid, Spain
`miriamgs@ucm.es`
[2] IST Austria, Klosterneuburg, Austria
`tah@ist.ac.at`
[3] Aalborg University, Aalborg, Denmark
`christianms@cs.aau.dk`

Abstract. We propose an algorithmic approach for synthesizing linear hybrid automata from time-series data. Unlike existing approaches, our approach provides a whole family of models with the same discrete structure but different dynamics. Each model in the family is guaranteed to capture the input data up to a precision error ε, in the following sense: For each time series, the model contains an execution that is ε-close to the data points. Our construction allows to effectively choose a model from this family with minimal precision error ε. We demonstrate the algorithm's efficiency and its ability to find precise models in two case studies.

Keywords: Synthesis · Hybrid Automata · Time Series.

1   Introduction

Mathematical models are ubiquitous across all sciences [11], from systems biology [23] to epidemiology [43] to cyber-physical systems [25]. The construction of such models is a central challenge in science [38]. One main benefit of *mathematical* models is the clearly defined semantics, which make these models amenable to automatic analysis (such as simulation [9,22] and verification [6,34,1]). Another main benefit that is usually desired is interpretability for high-level reasoning.

Hybrid automata [2,17] are a prominent class of interpretable models with mixed continuous and discrete behavior. They are particularly suitable in biological domains [39,27], where systems typically evolve continuously but are subject to internal and external events, and in cyber-physical domains [21], where physical entities interact with digital devices. In a nutshell, the evolution of a hybrid automaton follows a differential equation associated with one of several locations (or modes), until a discrete event leads to a different location.

In this paper we address the problem of synthesizing a linear hybrid automaton (LHA) [2] from a set of time series. The informal goal of model synthesis is

that the model *captures* the data well. What it means to "capture well" is difficult to formalize. Here we adopt the recent notion of ε-*capturing* from García et al. [13,12], which requires that, for each time series in the input data, the LHA must expose an execution that stays ε-close to all data points (see Fig. 2 for an illustration). In [13,12], the value of ε is fixed in the problem input. Here we consider ε a parameter, which we associate with a *family of parametric models*: LHA whose continuous dynamics are not fixed yet. Each possible fixation of the continuous dynamics corresponds to an instantiated LHA. All instantiated LHA associated with a concrete value of ε have the property that they ε-capture the data. We can then effectively search for an ε-capturing LHA with the minimal value of ε, whose behavior intuitively best resembles the data.

Our algorithm consists of two phases. In the first phase we synthesize the discrete structure of the LHA by fixing the set of locations and mapping data points in the time series to the locations. We propose an algorithm to obtain this mapping based on clustering. In the second phase we construct the parameter space, which is a polyhedron that associates ε to all possible instantiated LHA (i.e., fixations of continuous dynamics) that ε-capture the data. We select a concrete LHA by minimizing the value of ε, for which we solve a linear program.

We evaluate the algorithm in two case studies. In the first case study we investigate the scalability in terms of the different input parameters; we can synthesize a seven-dimensional model with 15 locations from 12,000 data points in 15 minutes, which shows that the algorithm is applicable in practice. In the second case study we use the algorithm to synthesize a model for a biological system (regulation of a cell cycle) in less than half a minute.

Related work. Synthesizing models is known to different communities as *system identification*, *process mining*, or *model learning*. Models that are akin to hybrid automata have been studied extensively in control theory; while the main aim in control theory is to find a controller for a system, which is outside the scope of the present paper, there is still a large body of works on system identification [33,14]. Many of these approaches focus on input-output models, such as autoregressive exogenous (ARX) models and in particular the switched (SARX) [16,32] and piecewise (PWARX) [8,37,30,19,5] versions, and focus on single-input single-output (SISO) systems, but there are also works on multiple-input multiple-output (MIMO) systems [18,42,3]. SARX and PWARX models can be seen as restricted linear hybrid automata where the locations form a state-space partition and the switching behavior is deterministic. This allows to reduce the synthesis problem to a parameter-optimization problem. The second phase of our algorithm also uses a reduction to parameter optimization, but the parameter space is different and our model class is more general.

In computer science, several approaches learn hybrid automata from input-output traces or time series. Similar to our approach, the works in [29,4] first use clustering to learn the discrete structure, but they employ different techniques, such as Angluin's algorithm for learning a finite automaton, and do not provide minimality guarantees for the result. Other approaches construct automata whose discrete structure is acyclic [31] respectively cyclic [15], or a deterministic

model with urgent transitions [24]. The work in [40] exhaustively constructs all possible models for optimizing a cost function, while in our approach the enumeration is only symbolic and we choose a model by solving a linear program, which scales favorably. A recent work shows that timed automata can be effectively learned from traces with a genetic algorithm [41]; learning timed automata has orthogonal challenges: they form a subclass of LHA where all variables are clocks with constant rate 1 and hence no continuous dynamics need to be learned, but the discrete dynamics are more complex than in this work. The work in [44] provides a framework for identifying deterministic models with affine dynamics from input-output traces, while we identify nondeterministic models. Our works in [13,12] proposed the notion of ε-capturing that we adopt here; those works synthesize a model from single traces online, but the algorithms are not scalable for offline usage of realistic dimension and size.

Outline. In Section 2 we fix the terminology. In Section 3 we formalize the synthesis problem and describe our solution on a high level. The low-level descriptions of the two phases of the algorithm follow in Section 4 and Section 5. We evaluate the algorithm in Section 6 and conclude in Section 7.

2 Terminology

Euclidean sets. We write $\mathbf{x}$ for points $(x_1, \ldots, x_n)$ in $\mathbb{R}^n$ and consider the infinity norm $\|\mathbf{x}\| = \max_{x_i} |x_i|$. The *ball* of radius $\varepsilon \in \mathbb{R}_{\geq 0}$ around a point $\mathbf{x} \in \mathbb{R}^n$ is $\mathcal{B}_\varepsilon(\mathbf{x}) = \{\mathbf{y} \in \mathbb{R}^n : \|\mathbf{x} - \mathbf{y}\| \leq \varepsilon\}$. The *$\varepsilon$-bloating* of $\mathcal{X} \subseteq \mathbb{R}^n$ is $\mathcal{X} \oplus \mathcal{B}_\varepsilon(\mathbf{0}) = \{\mathbf{x} + \mathbf{y} : \mathbf{x} \in \mathcal{X}, \|\mathbf{y}\| \leq \varepsilon\}$. A *polyhedron* over $\mathbb{R}^n$ is a finite intersection of *constraints* $\mathbf{a}^T \mathbf{x} \leq b$ where $\mathbf{a} \in \mathbb{R}^n$ and $b \in \mathbb{R}$. Let $\mathbb{P}_n$ be the set of all n-dimensional polyhedra. An *interval* is written $[a, b] = \{x : a \leq x \leq b\} \subseteq \mathbb{R}$.

Functions. Given a function $f : A \to B$, let $\mathtt{dom}(f) \subseteq A$ denote its domain. Let $f\!\restriction_D$ denote the restriction of f to set $D \subseteq \mathtt{dom}(f)$. A continuous function $f : [0, T] \to \mathbb{R}^n$ is a *piecewise-linear (PWL) function* with k pieces if there exists a triple $(I, M, \mathbf{x}_0)$ where I is a k-tuple of consecutive time intervals $[t_0, t_1], [t_1, t_2], \ldots, [t_{k-1}, t_k]$ with $[0, T] = \bigcup_{1 \leq i \leq k}[t_{i-1}, t_i]$, M is a k-tuple of slope vectors $\mathbf{m}_i \in \mathbb{R}^n$, and $\mathbf{x}_0 \in \mathbb{R}^n$ is the initial state $f(t_0) = \mathbf{x}_0$, such that each $f\!\restriction_{[t_{i-1}, t_i]}$ is a solution of the differential equation $\dot{\mathbf{x}}(t) = \mathbf{m}_i$, for all $i = 1, \ldots, k$. We refer to the line segments $f\!\restriction_{[t_{i-1}, t_i]}$ as the *pieces* of f. A *time-series* $s : D \to \mathbb{R}^n$ maps time points t from a finite set $D \subseteq \mathbb{R}_{\geq 0}$ to data points $s(t)$. There is a one-to-one correspondence between PWL functions and time series: A PWL function f over $I = ([t_0, t_1], [t_1, t_2], \ldots, [t_{k-1}, t_k])$ induces a time series as the restriction $s = f\!\restriction_D$ to time points $D = \{t_0, t_1, \ldots, t_k\}$, and s induces f as the piecewise-linear interpolation of the data points. Thus we may refer to, e.g., the pieces of a time series. The *distance* between a PWL function f and a time series s with $\mathtt{dom}(f\!\restriction_{\mathtt{dom}(s)}) = \mathtt{dom}(s)$ is $d(f, s) = \max_{t \in \mathtt{dom}(s)} \|f(t) - s(t)\|$.

Linear hybrid automata. An n-dimensional *linear hybrid automaton* (LHA) [2,17] is a tuple $\mathcal{H} = (Loc, E, Flow, Inv, Grd)$, where 1) Loc is the finite set of locations, 2) $E \subseteq Loc \times Loc$ is the transition relation, 3) $Flow : Loc \to \mathbb{R}^n$ is the flow function, 4) $Inv : Loc \to \mathbb{P}_n$ is the invariant function, and 5) $Grd : E \to \mathbb{P}_n$ is the guard function. Our LHA model does not have assignments along the transitions and is also called switched linear system [26]. We also consider partially defined hybrid automata without flows, invariants, or guards assigned. This *discrete structure* $\mathcal{H}_d = (Loc, E)$ only consists of locations and transitions.

The semantics of LHA are described by the set of executions. A *state* of an LHA is a pair $(\ell, \mathbf{x})$ of a location $\ell \in Loc$ and a point $\mathbf{x} \in Inv(\ell)$ in the invariant. An *execution* σ of an LHA evolves continuously according to the flow function in each location. The execution starts in some state $(\ell_1, \mathbf{x}_1)$ and the continuous evolution follows the constant differential equation $\dot{\mathbf{x}} = Flow(\ell_1)$ while satisfying the invariant $Inv(\ell_1)$ for some dwell time $\delta \in \mathbb{R}_{\geq 0}$. The execution can instantaneously switch locations, from a state $(\ell_1, \mathbf{x}_2)$ to another state $(\ell_2, \mathbf{x}_2)$, if there is a transition $(\ell_1, \ell_2) \in E$ and the guard $Grd(\ell_1, \ell_2)$ contains $\mathbf{x}_2$. The projection of an execution σ to the second component is a PWL function, which we denote by σ_π. We use the following compact notation for executions, where $\delta_i \in \mathbb{R}_{\geq 0}$ (for $i \geq 1$) denotes the duration of a dwell action and *jmp* denotes a switch:

$$\sigma \equiv (\ell_1, \mathbf{x}_1) \xrightarrow{\delta_1} (\ell_1, \mathbf{x}_2) \xrightarrow{jmp} (\ell_2, \mathbf{x}_2) \xrightarrow{\delta_2} (\ell_2, \mathbf{x}_3) \xrightarrow{jmp} (\ell_3, \mathbf{x}_3) \; \cdots$$

3   Synthesis of ε-close linear hybrid automata

In this section we formalize the synthesis problem that we address in this paper and give a high-level overview of our approach to solve it. Given a time series, we want to construct an LHA that captures the data up to a given precision. We first formalize the notion of capturing.

Definition 1 (ε-capturing [13]). *Given a time series s and a value $\varepsilon \in \mathbb{R}_{\geq 0}$, we say that an LHA $\mathcal{H}$ ε-captures s if there exists an execution σ of $\mathcal{H}$ such that $d(\sigma_\pi, s) \leq \varepsilon$. We also say that s and σ_π (resp. s and σ) are ε-close.*

Our goal is to construct an LHA that ε-captures several time series.

Problem 1 (ε-close synthesis [13]). Given a finite set of time series $\mathcal{S}$ and a value $\varepsilon \in \mathbb{R}_{\geq 0}$, construct an LHA $\mathcal{H}$ that ε-captures each s in $\mathcal{S}$.

As we observed in [13], it is straightforward to find a solution to the problem even for $\varepsilon = 0$ by simply introducing a fresh location for each piece of the time series. Such a model does not aggregate nor generalize the information in the data and is hence of little use. To obtain a reasonable model, one needs to add another bound to the problem, e.g., by fixing the discrete structure.

We address this observation in a two-phase algorithm. In the first phase we fix the discrete structure $\mathcal{H}_d$ of the LHA, where we try to reuse the locations for multiple time series (or pieces therein). In the second phase we instantiate the model for the smallest possible value of ε under the given discrete structure.
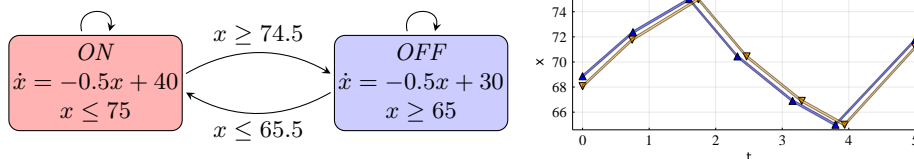
Fig. 1. Left: A hybrid automaton. Right: Two time series (triangle markers) obtained from sampling two executions of the automaton, and induced PWL functions.

Thus in this paper we consider a synthesis problem where we do not fix the value of ε and rather find a sufficiently small value for ε automatically.

Problem 2 (ε-minimal synthesis). Given a finite set of time series $\mathcal{S}$ and a discrete structure $\mathcal{H}_d$, find the minimal value $\varepsilon \in \mathbb{R}_{\geq 0}$ and an instantiation $\mathcal{H}$ of $\mathcal{H}_d$ such that $\mathcal{H}$ ε-captures each s in $\mathcal{S}$.

3.1 Synthesis algorithm

In the next two sections we describe our algorithm to solve the above synthesis problem, but first we give a high-level overview of the algorithm. Our algorithm computes a parametric family of LHA that all ε-capture the given data. The LHA share the same discrete structure but differ in the continuous dynamics. Since ε itself is a parameter of that construction, we can then choose an LHA with a minimal value for ε (which is not necessarily unique) from that family.

Our goal is that the final LHA has an ε-close execution for each time series. To simplify the theoretical presentation, we will use the following conceptual view on our algorithm. Instead of synthesizing an LHA directly, we synthesize ε-close executions. These executions then induce an LHA.

As mentioned, our algorithm proceeds in two phases. In the first phase we fix the discrete structure of the executions (and thus of the resulting LHA). In the second phase we construct the space of continuous dynamics to be assigned to the locations, depending on the value ε. For LHA, this space is a polyhedron, which we call the *flow polyhedron*. We then choose concrete continuous dynamics from the flow polyhedron to instantiate concrete executions (and thus an LHA). We explain each step of the algorithm using the following running example.

Example 1 (running example). We consider two time series in one dimension:

$$
\begin{array}{rccccccc}
t_1 = & (0.00, & 0.76, & 1.59, & 2.32, & 3.15, & 3.79, & 5.00) \\
d_1 = & (68.91, & 72.41, & 75.00, & 70.44, & 66.90, & 65.00, & 71.81) \\
t_2 = & (0.0, & 0.75, & 1.61, & 2.33, & 3.16, & 3.76, & 5.00) \\
d_2 = & (68.16, & 71.85, & 74.70, & 70.22, & 66.75, & 65.00, & 71.92)
\end{array}
$$

We obtained the time series from two random trajectories of a hybrid automaton modeling a simple thermostat controller, all given in Fig. 1. Note that the original continuous dynamics are described by an affine differential equation, which cannot be expressed with an LHA. (We round all numbers to two digits, which explains small inconsistencies over the course of this running example.) ◁

4   Synthesis algorithm, Phase 1: Discrete structure

In this section we describe Phase 1 of the synthesis algorithm. The input is a finite set of time series. The output is a mapping from each piece of the time series (resp. the induced PWL functions) to a *symbolic location* (i.e., a location label). Together with the order of the pieces in the time series, as we explain below, this mapping already fixes the discrete structure $\mathcal{H}_d$ of the LHA.

4.1   Simplification of the time series

In the first step of our algorithm, we preprocess the time series by removing some data points for better stability of the second step (we explain this connection later). Note that for Phase 2 we again use the original time series, so correctness is not affected. The goal is to merge consecutive pieces in the time series with similar slopes, i.e., such that the linear interpolation is a good approximation. In our implementation we use a variant of the Ramer-Douglas-Peucker algorithm [36,7] where we consider time as another dimension. We shortly recall this algorithm but refer to the literature for details. Following a divide-and-conquer scheme, the algorithm starts with only the first and last point of the time series, connects them with a line segment, finds the point $\mathbf{x}$ with the largest distance from the line segment, and, unless this distance is small enough, repeats the process recursively for the corresponding two parts before and after $\mathbf{x}$.

4.2   Assignment of symbolic locations

The goal of the first phase is to determine the discrete structure $\mathcal{H}_d$ of the resulting LHA. For each time series with p pieces we synthesize a corresponding *symbolic execution* of the prospective LHA. These are executions that do not yet contain information about the continuous state, but the discrete state is already determined, i.e., we fix the sequence of visited locations $\ell_1, \ldots, \ell_p$ together with the points in time when the execution switches to a new location. (Here we restrict ourselves to switching in synchrony with the time series.) Thus each symbolic execution consists of a (timed) sequence of symbolic locations. It is easy to see that, by ignoring time, these sequences induce the discrete structure $\mathcal{H}_d$ of an LHA: the set of locations is the union of all locations occurring in the sequences, and there is a transition for each consecutive pair of locations. Formally, for a symbolic execution associated with a time series with p pieces, the discrete structure $\mathcal{H}_d = (Loc, E)$ is given by $Loc = \{\ell_1, \ldots, \ell_p\}$ and $E = \{(\ell_i, \ell_{i+1}) : i = 1, \ldots, p-1\}$, and the generalization to sets of symbolic executions consists of the union of these locations and transitions.

Given a time series with p pieces and a set of symbolic locations $\{\ell_1, \ldots, \ell_\lambda\}$, a symbolic execution as described above is merely a mapping from the pieces to location labels, which we call $\mathcal{M} : \{1, \ldots, p\} \to \{1, \ldots, \lambda\}$. Our algorithm is parametric in the concrete way to obtain this mapping. Typically we are interested in finding an LHA with a small number of locations. Thus the implicit requirement for the mapping is to share locations for multiple pieces.

Algorithm 1 Assignment of a symbolic location to each piece of a set of time series. Line 1 is optional and can be implemented with the identity. Line 2 can be implemented with k-means, which can also provide a good value for $\lambda \ (= k)$ if not specified in the input (as described in Sect. 4.2).

Input: A set of time series $\mathcal{S} = \{s_1, \ldots, s_r\}$ and optionally a number of locations λ
Output: A mapping from the pieces to symbolic locations and a number of locations
 1: $\mathcal{S}' := \mathrm{simplify}(\mathcal{S})$ {see Sect. 4.1}
 2: $\mathcal{M}, \lambda := \mathrm{assign\_location\_labels\_to\_pieces}(\mathcal{S}', \lambda)$ {see Sect. 4.2}
 3: **return** $\mathcal{M}, \lambda$

In our implementation we obtain the mapping using a variant of the k-means clustering algorithm [28]. The input to the clustering algorithm are the slopes of the PWL functions induced by the time series. The k-means algorithm requires to specify upfront the number of clusters k, which corresponds to the number of locations in our setting. If the intended number of locations is already known in advance, this algorithm can be used directly. Otherwise, to find a good value of k automatically, we use a common refinement loop by starting with some value for k (e.g., $k = 1$) and then increasing k until the clustering error (which is defined as the sum of the squared Euclidean distance of each point to its associated cluster center) does not decrease substantially anymore.

The k-means algorithm is sensitive to the initial choice of the cluster centers. The preprocessing step proposed in Sect. 4.1 increases the stability in this regard. As initial candidates for the cluster centers we choose the first k slopes induced by the simplified time series. This choice results in candidates that are sufficiently different in practice and thus k-means yields more robust clusters.

We summarize the main steps of Phase 1 in Algorithm 1.

Example 2 (cont'd). The input to the clustering algorithm are the slope values of the two time series. In the table below we list the clustering cost for different numbers of clusters k, together with the relative improvement compared to $k-1$:

clusters (k)	1	2	3	4	5	6	7	8
cost	259.76	17.07	11.80	2.46	0.78	0.09	0.04	0.01
rel. [%]	–	0.93	0.31	0.79	0.68	0.89	0.60	0.61

The table suggests that good values for k are 2, 4, or 6. To obtain a small model, here we settle for $k = 2$ locations. The associated (one-dimensional) cluster centers (representing slopes) are 4.53 and -4.46. For both time series, the assigned clusters are $(1, 1, 2, 2, 2, 1)$, corresponding to the symbolic location ℓ_1 for the pieces 1, 2, 6 and symbolic location ℓ_2 for the other three pieces. ◁

5   Synthesis algorithm, Phase 2: Continuous dynamics

In this section we describe Phase 2 of the synthesis algorithm. The input is a finite set of time series together with a discrete structure $\mathcal{H}_d$ obtained in Phase 1,

which is represented by the mapping $\mathcal{M}$ assigning a symbolic location to each piece of the time series. The output is an LHA $\mathcal{H}$ and a value for ε such that $\mathcal{H}$ ε-captures the time series. As mentioned before, we describe how to obtain an ε-close *corresponding execution* for each time series.

5.1   Construction of the flow polyhedron

In the first step, we construct the flow polyhedron P, which represents the set of all possible continuous dynamics such that the corresponding executions are ε-close to the time series. Here ε itself is a dimension of P. For technical reasons, we construct a new flow polyhedron for each time series.

Assume that we have n-dimensional data in the form of r time series and we want to synthesize an LHA with λ locations. Say that we consider a time series with p pieces. Then P is a polyhedron with $\lambda n + rn + 1$ dimensions. The first λn dimensions represent the location slopes. The next rn dimensions represent the coordinates of the initial states $\mathbf{x}_0^{(j)}$ of the j-th execution. (These $\mathbf{x}_0^{(j)}$ are auxiliary dimensions which we are not interested in.) The last dimension is ε.

Next we describe the constraints of P. These constraints express that the distance between the time series and the execution is less than ε (and thus the execution ε-captures the time series). We need to express the symbolic value of the execution, $\mathbf{x}_k$, at each time point t_k of the time series. Let $\mathbf{q}_k$ be the k-th data point of the time series, starting at $k = 0$. For each data point we have $2n$ constraints (i.e., $2n(p + 1)$ constraints in total) to express the requirement $\mathbf{x}_k \in \mathcal{B}_\varepsilon(\mathbf{q}_k)$. In $n = 1$ dimension, for each k we express the requirement with the two constraints $x_k - \varepsilon \leq q_k$ and $x_k + \varepsilon \geq q_k$. In $n > 1$ dimensions we have such constraints in each dimension.

It remains to explain how to express the term x_k. For $k = 0$ we represent $\mathbf{x}_0$ with the dedicated variables $x_0^{(\cdot)}$. For $k > 0$ we rewrite x_k using the following identity: $x_k = x_0 + \sum_{j=1}^{k}(t_j - t_{j-1})m^{(j)}$. The time points t_j are known constants and the $m^{(j)}$ are the slope variables for the j-th piece (recall that we have associated the pieces with locations in advance).

Below we formalize the flow polyhedron for $r = 1$ time series.

Definition 2. *Given a time series s with p pieces and an associated mapping* $\mathcal{M} : \{1, \ldots, p\} \to \{1, \ldots, \lambda\}$, *the* flow polyhedron P_s *is defined as*

$$\{(\mathbf{m}_1, \ldots, \mathbf{m}_\lambda, \mathbf{x}_0, \varepsilon) \in \mathbb{R}^{\lambda n + n} \times \mathbb{R}_{\geq 0} \mid \mathbf{x}_0 \in \mathcal{B}_\varepsilon(s(t_0)),$$
$$\mathbf{x}_0 + (t_1 - t_0)\mathbf{m}_{\mathcal{M}(1)} \in \mathcal{B}_\varepsilon(s(t_1)),$$
$$\mathbf{x}_0 + (t_1 - t_0)\mathbf{m}_{\mathcal{M}(1)} + (t_2 - t_1)\mathbf{m}_{\mathcal{M}(2)} \in \mathcal{B}_\varepsilon(s(t_2)),$$
$$\vdots$$
$$\mathbf{x}_0 + (t_1 - t_0)\mathbf{m}_{\mathcal{M}(1)} + \ldots + (t_p - t_{p-1})\mathbf{m}_{\mathcal{M}(p)} \in \mathcal{B}_\varepsilon(s(t_p))\}.$$

Example 3 (cont'd). Our example has $n = 1$ dimension, $\lambda = 2$ locations, and $r = 2$ time series. The flow polyhedron consists of five variables $(m_1, m_2, x_0^{(1)}, x_0^{(2)}, \varepsilon)$.

Here m_1 and m_2 represent the slopes of the two locations, $x_0^{(1)}$ and $x_0^{(2)}$ represent the initial state of the first resp. second execution, and ε represents the allowed distance between the time series and the executions. Below we show the 14 constraints for the first execution:

$$
\begin{array}{rl|rl}
& x_0^{(1)} - \varepsilon \le 68.91 & & -x_0^{(1)} - \varepsilon \le -68.91 \\
0.76m_1 & + x_0^{(1)} - \varepsilon \le 72.41 & -0.76m_1 & -x_0^{(1)} - \varepsilon \le -72.41 \\
1.59m_1 & + x_0^{(1)} - \varepsilon \le 75.00 & -1.59m_1 & -x_0^{(1)} - \varepsilon \le -75.00 \\
1.59m_1 + 0.72m_2 & + x_0^{(1)} - \varepsilon \le 70.44 & -1.59m_1 - 0.72m_2 & -x_0^{(1)} - \varepsilon \le -70.44 \\
1.59m_1 + 1.55m_2 & + x_0^{(1)} - \varepsilon \le 66.90 & -1.59m_1 - 1.55m_2 & -x_0^{(1)} - \varepsilon \le -66.90 \\
1.59m_1 + 2.20m_2 & + x_0^{(1)} - \varepsilon \le 65.00 & -1.59m_1 - 2.20m_2 & -x_0^{(1)} - \varepsilon \le -65.00 \\
2.80m_1 + 2.20m_2 & + x_0^{(1)} - \varepsilon \le 71.81 & -2.80m_1 - 2.20m_2 & -x_0^{(1)} - \varepsilon \le -71.81 \triangleleft
\end{array}
$$

Note that, for multiple time series, each flow polyhedron only constrains n dimensions of the rn dimensions reserved for the initial states $x_0^{(\cdot)}$. The need for the separate dimensions will become clear when we aggregate the different flow polyhedra in the next step. Any feasible point inside the polyhedron P represents a concrete execution in an LHA that ε-captures the time series. We formalize this statement after defining the corresponding LHA in the next step.

5.2   The common solution space

In the first phase we implicitly fixed the discrete evolution of the executions, which also induced the discrete structure of the LHA we want to synthesize. In the previous step we obtained the flow polyhedra P_s, one for each time series s. In the next steps we combine these results to obtain concrete executions by assigning the continuous states. The concrete executions also induce the final LHA, i.e., we assign continuous dynamics, invariants, and guards.

Since we want to obtain one LHA to ε-capture *all* time series, we need to find compatible values for the dynamics and ε. For that purpose we can just intersect all flow polyhedra. Let $P_{\mathcal{H}} = \bigcap_{s \in \mathcal{S}} P_s$ be the polyhedron resulting from this intersection. Note that, since we used disjoint dimensions for the $\mathbf{x}_0^{(\cdot)}$ for different executions, the initial states are not shared in $P_{\mathcal{H}}$. (We note that intersecting polyhedra in constraint representation is a constant-time operation.)

5.3   Choice of minimizing parameters

Now we have to choose *any* feasible point $\mathbf{p}$ in $P_{\mathcal{H}}$. We argue that the most interesting points are those that minimize ε, since they correspond to executions that are closest to the original data. (In applications where further constraints should be considered, other choices are possible.) Minimizing a polyhedron in the dimension of ε means to solve the corresponding linear program with objective function ε, which is efficient in practice. We remark that $P_{\mathcal{H}}$ is bounded in the dimension of ε from below by 0, so this minimization always returns a proper solution $\mathbf{p} = (\mathbf{m}_1, \ldots, \mathbf{m}_\lambda, \mathbf{x}_0^{(1)}, \ldots, \mathbf{x}_0^{(r)}, \varepsilon)$. The point $\mathbf{p}$ contains a number for

---

Algorithm 2 Synthesis algorithm.

Input: A set of time series $\mathcal{S} = \{s_1, \ldots, s_r\}$, a number of locations λ, and a mapping from the pieces of each time series to symbolic locations $\mathcal{M}$
Output: An LHA $\mathcal{H}$ and a minimal value ε such that $\mathcal{H}$ ε-captures all elements of $\mathcal{S}$
 1: **for** $s \in \mathcal{S}$ **do**
 2:     $P_s :=$ flow_polyhedron$(s, \mathcal{M}, \lambda)$ {see Sect. 5.1}
 3: **end for**
 4: $P_{\mathcal{H}} := \bigcap_{s \in \mathcal{S}}^{r} P_s$ {see Sect. 5.2}
 5: *slopes*, $\varepsilon :=$ choose_minimizing_point$(P_{\mathcal{H}})$ {see Sect. 5.3}
 6: $\mathcal{H} :=$ construct_automaton$(\mathcal{S}, \mathcal{M}, slopes, \varepsilon)$ {see Sect. 5.4}
 7: **return** $\mathcal{H}, \varepsilon$

---

each dimension. The first λn numbers are the slope values for the locations, in the order they have been specified. The next rn numbers are the values of $\mathbf{x}_0^{(\cdot)}$ for the different executions (note again that we do not need these numbers). The last number is the corresponding value for ε.

5.4 Construction of the final LHA

Next we describe, for a given time series s_i over time instants $t_0, t_1, \ldots, t_p$, the execution that is induced by the above point $\mathbf{p}$. Let $\mathbf{m}_1, \ldots, \mathbf{m}_\lambda$ be the slopes taken from the point and $\mathcal{M}$ be the mapping from the pieces of s_i to the associated location (e.g., $\ell_{\mathcal{M}(1)}$ is the location for the first piece, with slope $\mathbf{m}_{\mathcal{M}(1)}$) obtained in Algorithm 1. The execution is a PWL function whose pieces have the same duration as the pieces of s_i. As defined before, the execution starts at $\mathbf{x}_0 = \mathbf{x}_0^{(i)}$ and the end point of the k-th piece is $\mathbf{x}_k = \mathbf{x}_0 + \sum_{j=1}^{p}(t_j - t_{j-1})\mathbf{m}_{\mathcal{M}(j)}$.

$$
\begin{aligned}
(\ell_{\mathcal{M}(1)}, \mathbf{x}_0) &\xrightarrow{t_1 - t_0} (\ell_{\mathcal{M}(1)}, \mathbf{x}_0 + (t_1 - t_0)\mathbf{m}_{\mathcal{M}(1)}) \\
&\xrightarrow{jmp} (\ell_{\mathcal{M}(2)}, \mathbf{x}_0 + (t_1 - t_0)\mathbf{m}_{\mathcal{M}(1)}) \\
&\xrightarrow{t_2 - t_1} (\ell_{\mathcal{M}(2)}, \mathbf{x}_0 + (t_1 - t_0)\mathbf{m}_{\mathcal{M}(1)} + (t_2 - t_1)\mathbf{m}_{\mathcal{M}(2)}) \\
&\ \ \vdots \\
&\xrightarrow{t_p - t_{p-1}} (\ell_{\mathcal{M}(p)}, \mathbf{x}_0 + \sum_{j=1}^{p}(t_j - t_{j-1})\mathbf{m}_{\mathcal{M}(j)})
\end{aligned}
$$

We have not yet described the invariants and guards of the resulting LHA. We say that a data point in the time series is associated with a location if the preceding or the succeeding piece in the time series is assigned that location in the mapping from Algorithm 1. Similarly, a data point is associated with the transition (ℓ_i, ℓ_j) if the preceding piece is associated with ℓ_i and the succeeding piece is associated with ℓ_j. A sufficient condition for our construction to be correct is: define the invariant of each location as the ε-bloated convex hull around all data points associated with it, and define the guard of each transition as the ε-bloated union around all data points associated with it. In our implementation
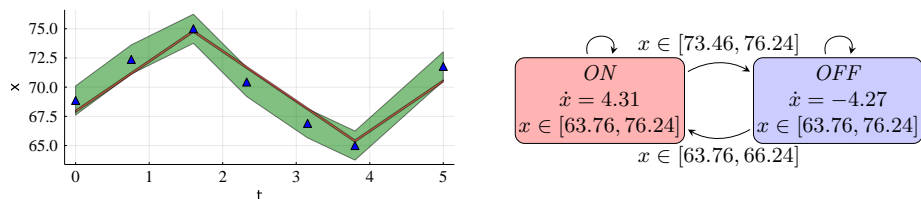
Fig. 2. Left: The first time series (triangle markers) inside an ε-tube (green) and the corresponding induced execution (red), for two locations. Right: The synthesized LHA.

we use the ε-bloated interval hull in both cases. That is, we take the smallest box around all data points as defined above and then extend the box in each direction by ε. We summarize the main steps of Phase 2 in Algorithm 2.

Example 4 (cont'd). We intersect the two flow polyhedra and minimize the resulting polyhedron in the dimension of ε to receive the following point: $m_1 = 4.31, m_2 = -4.27, x_0^{(1)} = 67.90, x_0^{(2)} = 67.63, \varepsilon = 1.24$. Thus we have synthesized the following execution for the first time series: $(\ell_1, 67.90) \xrightarrow{0.76} (\ell_1, 71.18) \xrightarrow{jmp} (\ell_1, 71.18) \xrightarrow{0.84} (\ell_1, 74.80) \xrightarrow{jmp} (\ell_2, 74.80) \xrightarrow{0.72} (\ell_2, 71.72) \xrightarrow{jmp} (\ell_2, 71.72) \xrightarrow{0.83} (\ell_2, 68.18) \xrightarrow{jmp} (\ell_2, 68.18) \xrightarrow{0.64} (\ell_2, 65.44) \xrightarrow{jmp} (\ell_2, 65.44) \xrightarrow{1.21} (\ell_1, 70.66)$. The execution and the final LHA are depicted in Fig. 2.   ◁

5.5   Correctness

We show that the algorithm produces an LHA that ε-captures the given data.

Lemma 1. *For every time series s that is input to Algorithm 2, the induced execution ε-captures s, where ε is obtained in Line 5.*

Proof. The constraints of the flow polyhedron P_s corresponding to s enforce that the induced execution is ε-close to all data points of s. This even holds for *any* point in P_s. Since the concrete choice of the point in Line 5 is taken from $P_{\mathcal{H}}$, which is a subset of P_s, the claim follows.

Theorem 1. *The LHA $\mathcal{H}$ synthesized in Algorithm 2 ε-captures all time series, where ε is obtained in Line 5. Furthermore, Algorithm 2 solves Problem 2 in polynomial time.*

Proof. Lemma 1 ensures that the induced executions ε-capture the time series. It remains to show that these induced executions belong to $\mathcal{H}$. This holds by construction of $\mathcal{H}$; we only sketch the main arguments. Each execution follows the slopes of the associated locations. For each location switch there exists a transition in $\mathcal{H}$. The executions always stay in ε-proximity to the data points, and hence they stay inside the invariants at all times. Similarly, since the executions change the location at time points of the data, the guards are satisfied. The solution to Problem 2 follows from the minimization of ε in Line 5. For the polynomial complexity, observe that the flow polyhedron's size is polynomial in the input and that the minimization can be implemented polynomially [20].
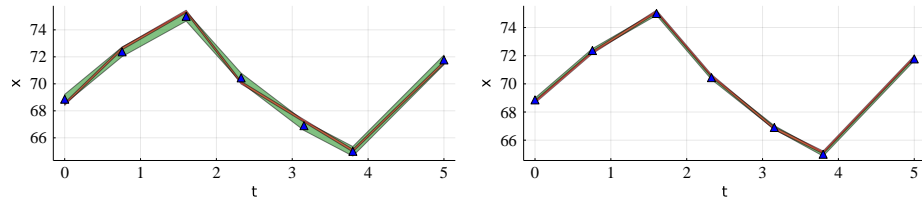
Fig. 3. The first time series (triangle markers) from Fig. 2 inside other ε-tubes (green) and the corresponding induced executions (red). Left: The result obtained for four locations ($\varepsilon = 0.38$). Right: The result obtained for six locations ($\varepsilon = 0.15$).

We remark that the number of locations λ and the sequence of locations obtained from Algorithm 1 influence the quality of the LHA resp. the size of ε but not the validity of the theorem (correctness of Algorithm 2). If these inputs are unsuitably chosen, the algorithm just returns a larger value for ε.

Example 5 (cont'd). Fig. 3 shows the synthesized executions and corresponding values of ε for the first time series with $\lambda = 4$ and $\lambda = 6$ locations.   ◁

6   Evaluation

In this section we describe our implementation and present experimental results. Our implementation in the Julia programming language is available at https://github.com/HySynth/HySynthParametric. To generate time series, we implemented a simulator of hybrid automata based on the ODE toolbox Differential-Equations.jl [35]. For polyhedral computations we use LazySets.jl [10].

We evaluate our algorithm in two case studies. In the first case study we investigate the scalability. In the second case study we synthesize an LHA model on data obtained from a model of a biological system. We note that all experiments are fully automatic with no human involved in the annotation or modeling.

Scalability. In the first case study we measure the scalability of the algorithm in four different input dimensions: the data dimension n, the number of time series r, the number of data points per time series p, and the number of locations in the final automaton λ. Here we do not use the preprocessing from Section 4.1 for better comparability between different runs. The majority ($> 90\%$) of the run time is spent in solving the linear program (Line 5 in Algorithm 2).

To obtain the time series, we instantiate a parametric version of the thermostat model (our running example) with n independent thermostats running in parallel. We obtain r random simulations of time duration $T = 40$, which are represented as time series, and then choose the first p data points from them. Since we fix λ, we pass it to Algorithm 1, which then skips the refinement procedure for k-means clustering in Line 2 and directly uses λ clusters.

We consider the following combination of parameters: $n \in \{1, 3, 5, 7\}$, $r \in \{1, 20, 40, 60\}$, $p \in \{50, 100, 150, 200\}$, and $\lambda \in \{1, 5, 10, 15\}$. To examine the
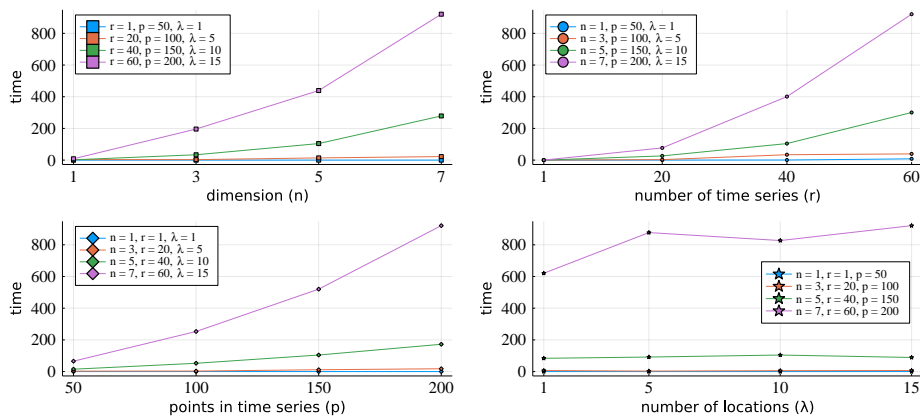
Fig. 4. Scalability in four different algorithm parameters. Each parameter varies between four values. In each of the four plots we vary one parameter and fix the remaining three. Each plot shows four graphs with indices $i = 1, \ldots, 4$, where for graph i we fix the parameters to their i-th value (which are also given in the legend).

scalability in these four dimensions, we fix three parameters and plot the run time for varying only one of the parameters in Figure 4.

From the results we observe that the input parameters n and r have the main influence on the complexity of the problem (the corresponding graphs have the steepest growth). The parameter p is less influential, and the parameter λ has almost no influence (the corresponding graphs barely grow and are not even monotonic). While λ influences the dimension of the flow polyhedron P, the different constraints are weakly coupled in these additional dimensions and thus the linear program is not substantially harder to solve.

In practice, when the data comes from experiments, the problem dimension n is fixed, and so is p if the data points are obtained from periodic measurements of fixed duration. Increasing r corresponds to additional experimental runs. The parameter λ can be freely chosen, but since a major benefit of hybrid automata is that they are interpretable models, we argue that λ should not be too large. Hence we believe that the algorithm is efficient enough to be used for real applications. We substantiate this claim in the next case study.

Regulation of a cell cycle. We consider the hybrid-automaton model of the regulation of a mammalian cell cycle from [39]. The cell cycle is modeled in nine phases. The model has one location for each phase, affine differential equations ($\dot{\mathbf{x}} = A\mathbf{x} + \mathbf{b}$), and assignments associated with some transitions. There are three main dimensions (CycA, CycB, and CycE), one secondary dimension for the mass of the cell, and time as auxiliary dimension for time-triggered transitions.

We run our synthesis algorithm on 20 time series obtained from random simulations of the model proposed in [39]. In total these time series consist of 3,557 data points. Before passing them to the algorithm, we project out the time variable. Hence our model cannot reason about time-dependent behavior. We used
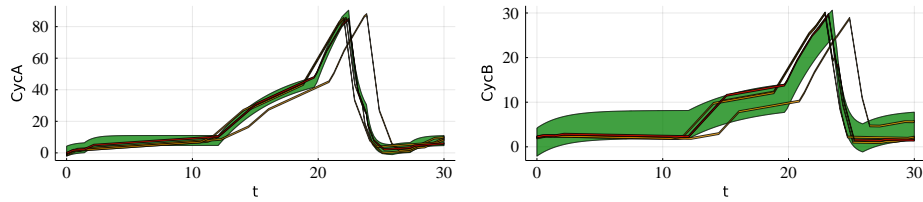
Fig. 5. The first two variables of the cell-cycle regulation with the ε-tube induced by the first time series ($\varepsilon = 3.15$, green), the corresponding induced execution (red), and three random simulations of the synthesized model (orange).

the refinement process for choosing the number of locations (λ) automatically. After 26 seconds we obtain an LHA with nine locations and a precision value $\varepsilon = 3.15$. In Figure 5 we show the ε-tube around the first time series together with three random simulations of the synthesized LHA. The ε-tube looks reasonably tight for the CycA dimension but wider for the CycB dimension, which is because the value of ε is the same in all dimensions, but the plot scales differ.

7   Conclusion

We have presented a synthesis algorithm to obtain a linear hybrid automaton from a set of time series. The algorithm uses two independent phases. In the first phase it constructs the discrete structure of the automaton. In the second phase it constructs the parameter space of all possible solutions and then selects an automaton by solving a linear program. The automaton is guaranteed to contain executions that are ε-close to the time series, where ε is minimal for the discrete structure chosen in the first phase. The algorithm is polynomial and scales to thousands of data points, but it also works with scarce data.

We see several directions for future work. The choice of the discrete structure in the first phase is important. We have proposed a heuristic implementation based on clustering that does not take the number of transitions into account. Reducing that number can remove unwanted behavior in the resulting model.

By minimizing ε we only minimize the maximum deviation of the executions from the data points. One can encourage the solver to find executions that stay close to the data points (the middle of the ε-tube in the plots). This can be encoded in the linear program by associating a cost to the sum of the deviation.

A more challenging extension is to use other classes of dynamics such as affine differential equations. The (exponential) solutions for such systems still have a closed form. Thus, instead of a linear program, we can solve a general optimization problem as in [12]. The difficult part is how to select the appropriate symbolic dynamics for the different parts of the time series.

Finally, in this paper we have only considered the automatic aspects of the algorithm. However, we believe that truly useful modeling ultimately requires interaction with a human in the loop. The separation of concerns – first finding a suitable discrete structure and formulating a parametric solution for finding

suitable continuous dynamics – allows scientists to incorporate domain knowledge, e.g., by adding further modeling constraints beyond ε-capturing. A key question is how to refine the model if the results are not accepted.

Acknowledgements

References

1. Althoff, M., Frehse, G., Girard, A.: Set propagation techniques for reachability analysis. Annual Review of Control, Robotics, and Autonomous Systems **4** (2020)
2. Alur, R., Courcoubetis, C., Henzinger, T.A., Ho, P.: Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In: Hybrid Systems. vol. 736, pp. 209–229 (1992)
3. Bako, L., Vidal, R.: Algebraic identification of MIMO SARX models. In: HSCC. vol. 4981, pp. 43–57 (2008)
4. Bartocci, E., Deshmukh, J., Gigler, F., Mateis, C., Nickovic, D., Qin, X.: Mining shape expressions from positive examples. Trans. Comput. Aided Des. Integr. Circuits Syst. **39**(11), 3809–3820 (2020)
5. Bemporad, A., Garulli, A., Paoletti, S., Vicino, A.: A bounded-error approach to piecewise affine system identification. Trans. Automat. Contr. **50**(10) (2005)
6. Clarke, E.M., Henzinger, T.A., Veith, H., Bloem, R. (eds.): Handbook of Model Checking. Springer (2018)
7. Douglas, D.H., Peucker, T.K.: Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. Cartographica **10**(2) (1973)
8. Ferrari-Trecate, G., Muselli, M.: Single-linkage clustering for optimal classification in piecewise affine regression. In: ADHS. vol. 36, pp. 33–38. Elsevier (2003)
9. Fishwick, P.A.: Handbook of dynamic system modeling. CRC Press (2007)
10. Forets, M., Schilling, C.: LazySets.jl: Scalable symbolic-numeric set computations. Proceedings of the JuliaCon Conferences **1**(1), 11 (2021)
11. Frigg, R., Hartmann, S.: Models in science. In: The Stanford Encyclopedia of Philosophy. Metaphysics Research Lab, Stanford University (2020)
12. García Soto, M., Henzinger, T.A., Schilling, C.: Synthesis of hybrid automata with affine dynamics from time-series data. In: HSCC. pp. 2:1–2:11 (2021)
13. García Soto, M., Henzinger, T.A., Schilling, C., Zeleznik, L.: Membership-based synthesis of linear hybrid automata. In: CAV. vol. 11561, pp. 297–314 (2019)
14. Garulli, A., Paoletti, S., Vicino, A.: A survey on switched and piecewise affine system identification. IFAC Proceedings Volumes **45**(16), 344–355 (2012)
15. Grosu, R., Mitra, S., Ye, P., Entcheva, E., Ramakrishnan, I.V., Smolka, S.A.: Learning cycle-linear hybrid automata for excitable cells. In: HSCC. vol. 4416, pp. 245–258 (2007)
16. Hashambhoy, Y., Vidal, R.: Recursive identification of switched ARX models with unknown number of models and unknown orders. In: CDC. pp. 6115–6121 (2005)
17. Henzinger, T.A.: The Theory of Hybrid Automata, pp. 265–292. Springer (2000)

18. Huang, K., Wagner, A., Ma, Y.: Identification of hybrid linear time-invariant systems via subspace embedding and segmentation (SES). In: CDC (2004)
19. Juloski, A.L., Weiland, S., Heemels, W.P.M.H.: A Bayesian approach to identification of hybrid systems. Trans. Autom. Control. **50**(10), 1520–1533 (2005)
20. Khachiyan, L.G.: A polynomial algorithm in linear programming. In: Doklady Akademii Nauk. vol. 244, pp. 1093–1096. Russian Academy of Sciences (1979)
21. Khaitan, S.K., McCalley, J.D.: Design techniques and applications of cyberphysical systems: A survey. IEEE Syst. J. **9**(2), 350–365 (2015)
22. Klee, H., Raimondi, A.: Simulation of dynamic systems with Matlab and Simulink. J. Artif. Soc. Soc. Simul. **11**(2) (2008)
23. Klipp, E., Liebermeister, W., Wierling, C., Kowald, A.: Systems biology: a textbook. John Wiley & Sons (2016)
24. Lamrani, I., Banerjee, A., Gupta, S.K.S.: HyMn: Mining linear hybrid automata from input output traces of cyber-physical systems. In: ICPS. pp. 264–269 (2018)
25. Lee, E.A., Seshia, S.A.: Introduction to embedded systems: A cyber-physical systems approach. Mit Press (2017)
26. Liberzon, D.: Switching in Systems and Control. Birkhäuser Boston (2003)
27. Liu, L., Bockmayr, A.: Formalizing metabolic-regulatory networks by hybrid automata. Acta Biotheoretica **68**(1), 73—85 (2020)
28. Lloyd, S.P.: Least squares quantization in PCM. Trans. Inf. Theory **28**(2) (1982)
29. Medhat, R., Ramesh, S., Bonakdarpour, B., Fischmeister, S.: A framework for mining hybrid automata from input/output traces. In: EMSOFT (2015)
30. Nakada, H., Takaba, K., Katayama, T.: Identification of piecewise affine systems based on statistical clustering technique. Autom. **41**(5), 905–913 (2005)
31. Niggemann, O., Stein, B., Vodencarevic, A., Maier, A., Kleine Büning, H.: Learning behavior models for hybrid timed systems. In: AAAI (2012)
32. Ozay, N.: An exact and efficient algorithm for segmentation of ARX models. In: ACC. pp. 38–41 (2016)
33. Paoletti, S., Juloski, A.L., Ferrari-Trecate, G., Vidal, R.: Identification of hybrid systems: A tutorial. Eur. J. Control **13**(2-3), 242–260 (2007)
34. Platzer, A.: Logical Foundations of Cyber-Physical Systems. Springer (2018)
35. Rackauckas, C., Nie, Q.: DifferentialEquations.jl - a performant and feature-rich ecosystem for solving differential equations in Julia. JORS **5**(1) (2017)
36. Ramer, U.: An iterative procedure for the polygonal approximation of plane curves. Comput. Graph. Image Process. **1**(3), 244–256 (1972)
37. Roll, J., Bemporad, A., Ljung, L.: Identification of piecewise affine systems via mixed-integer programming. Autom. **40**(1), 37–50 (2004)
38. Silvert, W.: Modelling as a discipline. Int. J. General Systems **30**(3) (2001)
39. Singhania, R., Sramkoski, R.M., Jacobberger, J.W., Tyson, J.J.: A hybrid model of mammalian cell cycle regulation. PLoS Comput. Biol. **7**(2) (2011)
40. Summerville, A., Osborn, J.C., Mateas, M.: CHARDA: causal hybrid automata recovery via dynamic analysis. In: IJCAI. pp. 2800–2806 (2017)
41. Tappler, M., Aichernig, B.K., Larsen, K.G., Lorber, F.: Time to learn - learning timed automata from tests. In: FORMATS. vol. 11750, pp. 216–235 (2019)
42. Verdult, V., Verhaegen, M.: Subspace identification of piecewise linear systems. In: CDC. pp. 3838–3843 (2004)
43. Vynnycky, E., White, R.: An introduction to infectious disease modelling. OUP Oxford (2010)
44. Yang, X., Beg, O.A., Kenigsberg, M., Johnson, T.T.: A framework for identification and validation of affine hybrid automata from input-output traces. Trans. Cyber-Phys. Syst. **6**(2) (2022)