View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by IST Austria: PubRep (Institute of Science and Technology)

Check for updates

Membership-Based Synthesis of Linear Hybrid Automata

Miriam García Soto(✉), Thomas A. Henzinger, Christian Schilling, and Luka Zeleznik

IST Austria, Klosterneuburg, Austria
{miriam.garciasoto,tah,christian.schilling, luka.zeleznik}@ist.ac.at

Abstract. We present two algorithmic approaches for synthesizing linear hybrid automata from experimental data. Unlike previous approaches, our algorithms work without a template and generate an automaton with nondeterministic guards and invariants, and with an arbitrary number and topology of modes. They thus construct a succinct model from the data and provide formal guarantees. In particular, (1) the generated automaton can reproduce the data up to a specified tolerance and (2) the automaton is tight, given the first guarantee. Our first approach encodes the synthesis problem as a logical formula in the theory of linear arithmetic, which can then be solved by an SMT solver. This approach minimizes the number of modes in the resulting model but is only feasible for limited data sets. To address scalability, we propose a second approach that does not enforce to find a minimal model. The algorithm constructs an initial automaton and then iteratively extends the automaton based on processing new data. Therefore the algorithm is well-suited for online and synthesis-in-the-loop applications. The core of the algorithm is a membership query that checks whether, within the specified tolerance, a given data set can result from the execution of a given automaton. We solve this membership problem for linear hybrid automata by repeated reachability computations. We demonstrate the effectiveness of the algorithm on synthetic data sets and on cardiac-cell measurements.

Keywords: Synthesis · Linear hybrid automaton · Membership

1 Introduction

Natural sciences pursue to understand the mechanisms of real systems and to make this understanding accessible. Achieving these two goals requires observation, analysis, and modeling of the system. Typically, physical components of a

This research was supported in part by the Austrian Science Fund (FWF) under grants S11402-N23 (RiSE/SHiNE) and Z211-N23 (Wittgenstein Award) and the European Union's Horizon 2020 research and innovation programme under the Marie Sklodowska-Curie grant agreement No. 754411.

© The Author(s) 2019

system evolve continuously in real time, while the system may switch among a finite set of discrete states. This applies to cyber-physical systems but also to purely analog systems; e.g., an animal's hunger affects its movement. A proper formalism for modeling such types of systems with mixed discrete-continuous behavior is a hybrid automaton [11]. Unlike black-box models such as neural networks, hybrid automata are easy to interpret by humans. However, designing such models is a time-intensive and error-prone process, usually conducted by an expert who analyzes the experimental data and makes decisions.

In this paper, we propose two automatic approaches for synthesizing a linear hybrid automaton [1] from experimental data. The approaches provide two main properties. The first property is *soundness*, which ensures that the generated model has enough executions: these executions approximate the given data up to a predefined accuracy. The second property is *precision*, which ensures that the generated model does not have too many executions. The behavior of a hybrid automaton is constrained by so-called invariants and guards. *Precision* expresses that the boundaries of these invariants and guards are witnessed by the data, which indicates that the constraints cannot be made tighter. Moreover, the proposed synthesis algorithm is *complete* for a general class of linear hybrid automata, i.e., the algorithm can synthesize any given model from this class.

The first approach reduces the synthesis problem to a satisfiability question for a linear-arithmetic formula. The formula allows us to encode a minimality constraint (namely in the number of so-called modes) on the resulting model. This approach is, however, not scalable, which motivates our second approach. Our second approach follows an iterative model-adaptation scheme. Apart from scalability advantages, this *online* algorithm is thus also well-suited for synthesis-in-the-loop applications.

After constructing an initial model, the second approach iteratively improves and expands the model by considering new experiments. After each iteration, the model will capture all behaviors exhibited in the previous experiments. Given an automaton and new experimental data, the algorithm proceeds as follows. First we ask whether the current automaton already captures the data. We pose this question as a membership query for a piecewise-linear function in the set of executions of the automaton. For the membership query, we present an algorithm based on reachability inside a tube around the function. If the data is not captured, we need to modify the automaton accordingly by adding behavior. We first try to relax the above-mentioned invariants and guards, which we reduce to another membership query. If that query is negative as well, we choose a path in the automaton that closely resembles the given data and then modify the automaton along that path by also adding new discrete structure (called modes and transitions). This modification step is again guided by membership queries to identify the aspects of the model that require improvement and expansion.

As the main contributions, (1) we present an online algorithm for automatic synthesis of linear hybrid automata from data that is *sound*, i.e., guarantees that the generated model approximates the data up to a user-defined threshold, *precise*, i.e., the generated model is tight, and *complete* for a general class of models (2) we solve the membership problem of a piecewise-linear function in a linear hybrid automaton. This is a critical step in our synthesis algorithm

Related Work. The synthesis of hybrid systems was initially studied in control theory under the term *identification*, mainly focused on (discrete-time) switched autoregressive exogenous (SARX) and piecewise-affine autoregressive exogenous (PWARX) models [7,18]. SARX models constitute a subclass of linear hybrid automata with deterministic switching behavior. PWARX models are specific SARX models where the mode invariants form a state-space partition. Fixing the number of modes, the identification problem from input-output data can be solved algebraically by inferring template parameters. However, in contrast to linear hybrid automata, the lack of nondeterminism and the underlying assumption that there is no hidden state (mode) limits the applicability of these models. An algorithm by Bemporad et al. constructs a PWARX model that satisfies a *global* error bound [5]. Ozay presents an algorithm for SARX models where the switching is purely time-triggered [17]. There also exist a few *online* algorithms for the recursive synthesis of PWARX models based on pattern recognition [19] or lifting to a high-dimensional identification problem for ARX models [10,22].

Synthesis is also known as *process mining*, and as *learning models from traces*; the latter refers to approaches based on learning finite-state machines [3] or other machine-learning techniques. More recently, synthesis of hybrid automaton models has gained attention. All existing approaches that we are aware of have structural restrictions of some sort, which we describe below. We synthesize, for the first time, a general class of linear hybrid automata which (1) allows nondeterminism to capture many behaviors by a *concise* representation and (2) provides formal soundness and precision guarantees. The algorithm is also the first *online* synthesis approach for linear hybrid automata.

The general synthesis problem for hybrid automata is hard: for deterministic timed automata (a subclass of linear hybrid automata with globally identical continuous dynamics), one may already require data of exponential length [21]. The approach by Niggemann et al. constructs an automaton with acyclic discrete structure [16], while the approach by Grosu et al., intended to model purely periodic behavior, constructs a cyclic-linear hybrid automaton whose discrete structure consists of a loop [8]. Ly and Lipson use symbolic regression to infer a non-linear hybrid automaton [14]. However, their model neither contains state variables (i.e., the model is purely input-driven, comparable to the SARX model) nor invariants, and the number of modes needs to be fixed in advance. Medhat et al. describe an abstract framework, based on heuristics, to learn linear hybrid automata from input/output traces [15]. They first employ Angluin's algorithm for learning a finite-state machine [3], which serves as the discrete structure of the hybrid automaton, before they decorate the automaton with continuous dynamics. This strict separation inherently makes their approach offline. The work by Summerville et al. based on least-squares regression requires an exhaustive construction of all possible models for later optimizing a cost function over all of them [20]. Lamrani et al. learn a completely deterministic model with urgent transitions using ideas from information theory [12].

2 Preliminaries

Sets. Let $\mathbb{R}$, $\mathbb{R}_{\geqslant 0}$, and $\mathbb{N}$ denote the set of real numbers, non-negative real numbers, and natural numbers, respectively. We write $\mathbf{x}$ for points $(x_1, \ldots, x_n)$ in $\mathbb{R}^n$. Let $\mathtt{cpoly}(n)$ be the set of compact and convex polyhedral sets over $\mathbb{R}^n$. A set $X \in \mathtt{cpoly}(n)$ is characterized by its set of vertices $\mathtt{vert}(X)$. For a set of points Y, $\mathtt{chull}(Y) \in \mathtt{cpoly}(n)$ denotes the convex hull. Given a set $X \in \mathtt{cpoly}(n)$ and $\varepsilon \in \mathbb{R}_{\geqslant 0}$, we define the *$\varepsilon$-bloating* of X as $\lceil X \rceil_\varepsilon := \{\mathbf{x} \in \mathbb{R}^n \mid \exists \mathbf{x}_0 \in X : \|\mathbf{x} - \mathbf{x}_0\| \leqslant \varepsilon\} \in \mathtt{cpoly}(n)$, where $\| \cdot \|$ is the infinity norm. Given an interval $I = [l, u] \in \mathtt{cpoly}(1)$, $\mathtt{lb}(I) = l$ and $\mathtt{ub}(I) = u$ denote its lower and upper bound.

Functions and Sequences. Given a function f, let $\mathtt{dom}(f)$ resp. $\mathtt{img}(f)$ denote its domain resp. image. Let $f\lfloor_A$ denote the restriction of f to domain $A \subseteq \mathtt{dom}(f)$. We define a *distance* between functions f and g with the same domain and codomain by $d(f, g) := \max_{t \in \mathtt{dom}(f)} \|f(t) - g(t)\|$. A *sequence* of length m is a function $s : D \to A$ over an ordered finite domain $D = \{i_1, \ldots, i_m\} \subseteq \mathbb{N}$ and a set A, and we write $\mathtt{len}(s)$ to denote the length of s. A sequence s is also represented by enumerating its elements, as in $s(i_1), \ldots, s(i_m)$.

Affine and Piecewise-Linear Functions. An *affine piece* is a function $p : I \to \mathbb{R}^n$ over an interval $I = [t_0, t_1] \subseteq \mathbb{R}$ defined as $p(t) = \mathbf{a}t + \mathbf{b}$ where $\mathbf{a}, \mathbf{b} \in \mathbb{R}^n$. Given an affine piece p, $\mathtt{init}(p)$ denotes the start point $p(t_0)$, $\mathtt{end}(p)$ denotes the end point $p(t_1)$, and $\mathtt{slope}(p)$ denotes the slope $\mathbf{a}$. We call two affine pieces p and p' *adjacent* if $\mathtt{end}(p) = \mathtt{init}(p')$ and $\mathtt{ub}(\mathtt{dom}(p)) = \mathtt{lb}(\mathtt{dom}(p'))$. For $m \in \mathbb{N}$, an *m-piecewise-linear (m-PWL) function* $f : I \to \mathbb{R}^n$ over interval $I = [0, \mathsf{T}] \subseteq \mathbb{R}$ consists of m affine pieces $p_1, \ldots, p_m$, such that $I = \cup_{1 \leqslant j \leqslant m} \mathtt{dom}(p_j)$, $f(t) = p_j(t)$ for $t \in \mathtt{dom}(p_j)$, and for every $1 < j \leqslant m$ we have $\mathtt{end}(p_{j-1}) = \mathtt{init}(p_j)$. We show a 3-PWL function in Fig. 1 on the left. Let $\mathtt{pieces}(f)$ denote the set of affine pieces of f. We refer to f and the sequence $p_1, \ldots, p_m$ interchangeably and write "PWL function" if m is clear from the context. A *kink* of a PWL function is the point between two adjacent pieces. Given a PWL function $f : I \to \mathbb{R}^n$ and a value $\varepsilon \in \mathbb{R}_{\geqslant 0}$, the *$\varepsilon$-tube* of f is the function $\mathtt{tube}_{f,\varepsilon} : I \to \mathtt{cpoly}(n)$ such that $\mathtt{tube}_{f,\varepsilon}(t) = \lceil f(t) \rceil_\varepsilon$.

Graphs. A *graph* is a pair (V, E) of a finite set V and a relation $E \subseteq V \times V$. A *path* π in (V, E) is a sequence $v_1, \ldots, v_m$ with $(v_{j-1}, v_j) \in E$ for $1 < j \leqslant m$.

Hybrid Automata. We consider a particular class of hybrid automata [1,11].

Definition 1. *A n-dimensional* linear hybrid automaton (LHA) *is a tuple $\mathcal{H} = (Q, E, X, Flow, Inv, Grd)$, where (1) Q is a finite set of modes, (2) $E \subseteq Q \times Q$ is a transition relation, (3) $X = \mathbb{R}^n$ is the continuous state-space, (4) $Flow : Q \to \mathbb{R}^n$ is the flow function, (5) $Inv : Q \to \mathtt{cpoly}(n)$ is the invariant function, and (6) $Grd : E \to \mathtt{cpoly}(n)$ is the guard function*

We sometimes annotate the elements of LHA $\mathcal{H}$ by a subscript, as in $Q_\mathcal{H}$ for the set of modes. We refer to $(Q_\mathcal{H}, E_\mathcal{H})$ as the *graph of* LHA $\mathcal{H}$.

An LHA evolves continuously according to the flow function in each mode. The behavior starts in some mode $q \in Q$ and some continuous state $\mathbf{x} \in Inv(q)$.

For every mode $q \in Q$, the continuous evolution follows the differential equation $\dot{\mathbf{x}} = \mathit{Flow}(q)$ while satisfying the invariant $\mathit{Inv}(q)$. The behavior can switch from one mode q_1 to another mode q_2 if there is a transition $(q_1, q_2) \in E$ and the guard $\mathit{Grd}((q_1, q_2))$ is satisfied. During a switch, the continuous state does not change. This type of system is sometimes called a switched linear hybrid system [13].

Definition 2. *Given an n-dimensional* LHA $\mathcal{H} = (Q, E, X, \mathit{Flow}, \mathit{Inv}, \mathit{Grd})$, *an* execution σ *is a triple* $\sigma = (\mathcal{I}, \gamma, \delta)$, *where* $\mathcal{I}$ *is a sequence of consecutive intervals* $[t_0, t_1], [t_1, t_2], \dots, [t_{m-1}, t_m]$ *with* $[\![\mathcal{I}]\!] = \cup_{0 \leqslant j < m}[t_j, t_{j+1}]$, *and* $\gamma : [\![\mathcal{I}]\!] \to \mathbb{R}^n$ *and* $\delta : \{1, \dots, m\} \to Q$ *are functions with the following restrictions:*

- *for all* $1 \leqslant j < m$, $\gamma(t) \in \mathit{Inv}(\delta(j))$ *for* $t \in \mathcal{I}(j)$ *and* $\dot{\gamma}(t') = \mathit{Flow}(\delta(j))$ *for all* t' *in the interior of* $\mathcal{I}(j)$, *i.e.,* $\gamma\lfloor_{\mathcal{I}(j)}$ *is an affine function satisfying the invariant and following the flow, and*
- *for all* $1 \leqslant j < m$, $(\delta(j), \delta(j+1)) \in E$ *and* $\gamma(t) \in \mathit{Grd}((\delta(j), \delta(j+1)))$ *where* $t = \mathtt{ub}(\mathcal{I}(j))$, *i.e., if a transition is taken, then the guard is satisfied.*

We denote the set of all executions of $\mathcal{H}$ by $\mathtt{exec}(\mathcal{H})$. Given an LHA $\mathcal{H}$, we say that an execution σ *follows a path* π in $\mathcal{H}$, that is, in the graph $(Q_{\mathcal{H}}, E_{\mathcal{H}})$, denoted as $\sigma \overset{\mathcal{H}}{\rightsquigarrow} \pi$, if $\mathtt{len}(\mathcal{I}) = \mathtt{len}(\pi)$ and $\delta(j) = \pi(j)$ for every $0 \leqslant j < \mathtt{len}(\mathcal{I})$.

From Time-series Data to PWL *Functions.* Experimental data typically comes as *time series*, i.e., data is only available at sampled points in time. A time series is a sampling $s : D \to \mathbb{R}^n$ over a finite time domain $D \subseteq [0, \mathsf{T}]$. Since the LHA model features piecewise-linear executions, we focus on piecewise-linear approximation of the data. PWL functions can approximate any continuous behavior with arbitrary precision. There are different yet valid choices for approximating data. For a single time series, linear interpolation gives a perfect fit, but contains many kinks; other algorithms minimize the number of kinks for a given error bound [6,9]. One can preprocess multiple time series into a single PWL function using, e.g., linear regression. In this paper, we leave the choice of abstraction open and assume that the input is given as PWL functions.

3 Synthesis of Linear Hybrid Automata

In this section, we specify the synthesis problem, consider two different specifications, synchronous and asynchronous, and present the automated approach for solving the synchronous problem. The overall goal is to synthesize a linear hybrid automaton from a set of PWL functions such that the automaton *captures* the behavior described by each of the PWL functions up to a bound ε.

Definition 3 (Soundness). *Given a* PWL *function* f *and a value* $\varepsilon \in \mathbb{R}_{\geqslant 0}$, *we say that an* LHA $\mathcal{H}$ ε-captures f *if there exists an execution* $\sigma = (\mathcal{I}, \gamma, \delta)$ *in* $\mathtt{exec}(\mathcal{H})$ *with* $d(f, \gamma) \leqslant \varepsilon$.

The value ε quantifies the acceptable deviation of an execution's continuous function γ from the PWL function f. For $\varepsilon = 0$, γ must precisely follow f. A straightforward formulation of the problem we want to solve is the following.

Problem 1 (Synthesis). Given a finite set of PWL functions $\mathcal{F}$ and $\varepsilon \in \mathbb{R}_{\geqslant 0}$, construct an LHA $\mathcal{H}$ that ε-captures every function $f \in \mathcal{F}$.

Observe that this problem is not well-posed, as it can be satisfied by an automaton that exhibits an excessive amount of behavior. Hence our second goal for the synthesis algorithm is to ensure constraints on the automaton's size. We start with the synthesis of an LHA with minimal number of modes.

3.1 Synchronous Switching Specification

For now, we require that the executions in the LHA switch *synchronously* with the given PWL functions. Under this assumption, we tackle a refinement of Problem 1:

Problem 2 (Synchronous synthesis). Given a finite set of PWL functions $\mathcal{F}$ and a value $\varepsilon \in \mathbb{R}_{\geqslant 0}$, construct an LHA $\mathcal{H}$ that ε-captures every function $f \in \mathcal{F}$ synchronously, and furthermore require that $\mathcal{H}$ has the minimal number of modes.

In the following, we present an algorithm to solve Problem 2. The idea is, given a PWL function f, to synthesize an execution σ that is ε-close to f. Recall that the continuous function γ of an execution is essentially just another PWL function. Any LHA that contains the execution σ has to comprise a mode for each different slope in γ. Thus a minimal number of modes can be achieved by minimizing the number of different slopes in γ. By fixing a number of different slopes, we encode the existence of γ as a logical formula $\phi_{f,\varepsilon}$, which will be satisfiable if and only if there exists a suitable function γ.

Let m be the number of affine pieces $p_1, \ldots, p_m$ in f with $\texttt{dom}(p_j) = [t_{j-1}, t_j]$ for $1 \leqslant j \leqslant m$. We refer to the time instants t_j as the switching times of f, and to $\mathbf{x}_j = f(t_j)$ as the switching points of f. Fixing a number $\ell \in \mathbb{N}$, we want to construct a PWL function γ_ℓ, consisting of m affine pieces $p'_1, \ldots, p'_m$ with ℓ different slopes, with the same switching times as in f, with switching points $\mathbf{y}_0, \ldots, \mathbf{y}_m$ ε-close to those in f (which is necessary and sufficient for $d(f, \gamma_\ell) \leqslant \varepsilon$), and with unknown slopes $\mathbf{b}_1 = \texttt{slope}(p'_1), \ldots, \mathbf{b}_m = \texttt{slope}(p'_m)$. We define the logical formula

$$\phi_{f,\varepsilon}(\ell) := \bigwedge_{j=1}^{m} \mathbf{y}_j = \mathbf{y}_{j-1} + \mathbf{b}_j(t_j - t_{j-1}) \wedge \bigwedge_{j=0}^{m} \mathbf{y}_j \in \lceil \mathbf{x}_j \rceil_\varepsilon \wedge \bigwedge_{j=1}^{m} \bigvee_{k=1}^{\ell} \mathbf{b}_j = \mathbf{c}_k,$$

which is satisfiable if and only if there exists a suitable PWL function γ_ℓ. For lifting to a set of functions $\mathcal{F}$, we define the formula $\phi_{\mathcal{F},\varepsilon}(\ell) := \bigwedge_{f \in \mathcal{F}} \phi_{f,\varepsilon}(\ell)$. These formulae fall into the theory of linear arithmetic and can be effectively solved by an SMT solver. Now, we can state the following results.

Lemma 1. *Let $\mathcal{F}$ be a finite set of* PWL *functions and $\varepsilon \in \mathbb{R}_{\geqslant 0}$. If $\phi_{\mathcal{F},\varepsilon}(\ell)$ is satisfiable for some integer value ℓ, then there exists a set of* PWL *functions $\mathcal{F}'$ such that $|\mathcal{F}'| = |\mathcal{F}|$, each function in $\mathcal{F}$ is ε-close to some function in $\mathcal{F}'$, and the number of distinct slopes in $\mathcal{F}'$ does not exceed ℓ.*

The set $\mathcal{F}'$ can be extracted from a satisfying assignment. We define a hybrid automaton with minimal number of locations 0-capturing a given PWL function.

Definition 4 (Canonical automaton). *Let f be an n-PWL function. The* canonical automaton of f *is $\mathcal{H}_f := (Q, E, \mathbb{R}^n, Flow, Inv, Grd)$ with*

- $Q = \{q_{\mathbf{a}} \mid \exists p \in \textit{pieces}(f) : \texttt{slope}(p) = \mathbf{a}\}$,
- $E = \{(q_{\mathbf{a}}, q_{\mathbf{a}'}) \mid \exists p, p' \in \textit{pieces}(f)\, \textit{adjacent} : \texttt{slope}(p) = \mathbf{a}, \texttt{slope}(p') = \mathbf{a}'\}$,
- $Flow(q_{\mathbf{a}}) = \mathbf{a}$,
- $Inv(q_{\mathbf{a}}) = \texttt{chull}(\{\texttt{img}(p) \mid p \in \textit{pieces}(f) : \texttt{slope}(p) = \mathbf{a}\})$, *and*
- $Grd((q_{\mathbf{a}}, q_{\mathbf{a}'})) = \texttt{chull}(\{\texttt{end}(p) \mid \exists p, p' \in \textit{pieces}(f)\, \textit{adjacent} : \texttt{slope}(p) = \mathbf{a}, \texttt{slope}(p') = \mathbf{a}'\})$.

Lemma 2. *Given a* PWL *function f, the canonical automaton $\mathcal{H}_f$ 0-captures f, and every* LHA *that 0-captures f has at least as many modes as $\mathcal{H}_f$.*

Definition 5 (Merging). *Given two hybrid automata $\mathcal{H}_i = (Q_i, E_i, X, Flow_i, Inv_i, Grd_i)$, $i = 1, 2$ with $Q_1 \cap Q_2 = \emptyset$, let $Q_{\mathbf{a}} = Q_{\mathbf{a}}^{\mathcal{H}_1} \cup Q_{\mathbf{a}}^{\mathcal{H}_2}$ be the locations with flow equal to $\mathbf{a}$. We define the* merging of $\mathcal{H}_1$ *and* $\mathcal{H}_2$ *as $\mathcal{H}_1 \sqcup \mathcal{H}_2 := (Q, E, X, Flow, Inv, Grd)$ with $Q = \{q_{\mathbf{a}} \mid \mathbf{a} \in \mathbb{R}^n, Q_{\mathbf{a}} \neq \emptyset\}$, $E = \{(q_{\mathbf{a}}, q_{\mathbf{a}'}) \mid \exists (q, q') \in E_1 \cup E_2, q \in Q_{\mathbf{a}}, q \in Q'_{\mathbf{a}}\}$, $Flow(q_{\mathbf{a}}) = \mathbf{a}$, $Inv(q_{\mathbf{a}}) = \texttt{chull}(\{Inv_i(q) \mid q \in Q_{\mathbf{a}}, i = 1, 2\})$, and $Grd((q_{\mathbf{a}}, q_{\mathbf{a}'})) = \texttt{chull}(\{Grd_i((q, q')) \mid (q, q') \in E_i, q \in Q_{\mathbf{a}}, q' \in Q_{\mathbf{a}'}, i = 1, 2\})$.*

Theorem 1. *Given a finite set of* PWL *functions $\mathcal{F}$ and a value $\varepsilon \in \mathbb{R}_{\geqslant 0}$, let ℓ be the smallest integer such that $\phi_{\mathcal{F},\varepsilon}(\ell)$ is satisfiable and let $\mathcal{F}'$ be a set of* PWL *functions corresponding to a satisfying assignment. Then, the merging of canonical automata $\sqcup_{f \in \mathcal{F}'} \mathcal{H}_f$ solves Problem 2.*

The above synthesis algorithm works well with short and low-dimensional PWL functions but does not scale to realistic problem sizes due to the heavy use of disjunctions. We next address scalability with a new online algorithm.

3.2 Asynchronous Switching Specification

We now change the requirement from the previous subsection (minimality in the models' discrete structure) to tightness in the model's state-space constraints. Intuitively, for every vertex $\mathbf{v}$ of an invariant or guard in $\mathcal{H}$ there should be some witness data $f \in \mathcal{F}$ that is close to $\mathbf{v}$ (at some point in time).

Definition 6 (Precision). *Given an* LHA *$\mathcal{H} = (Q, E, X, Flow, Inv, Grd)$, let $\textit{vert}(\mathcal{H})$ denote the union of the vertices of the invariants and guards:*

$$\textit{vert}(\mathcal{H}) = \bigcup_{q \in Q} \textit{vert}(Inv(q)) \cup \bigcup_{e \in E} \textit{vert}(Grd(e))$$

Given a set of PWL *functions $\mathcal{F}$ and a value $\varepsilon \in \mathbb{R}_{\geqslant 0}$, we say that $\mathcal{H}$ is ε* -precise *(with respect to $\mathcal{F}$) if the following holds:*

$$\forall \boldsymbol{v} \in \textit{vert}(\mathcal{H})\ \exists f \in \mathcal{F}\ \exists t \in \texttt{dom}(f) : \|\boldsymbol{v} - f(t)\| \leqslant \varepsilon.$$

The restriction to the vertices is reasonable because all sets are compact convex polyhedra. Note that ε-capturing compares functions to the automaton's executions, while ε-precision compares functions to the automaton's state-space.

We also relax the limitation to synchronously switching executions. Instead, we allow *asynchronous* switching, characterized as follows: for every function f ε-captured by $\mathcal{H}$, there exists an execution $\sigma \in \texttt{exec}(\mathcal{H})$ with the same number of switches as there are kinks in f, i.e., $\texttt{len}(\mathcal{I}) = |\texttt{pieces}(f)|$, and where the j-th switch in the execution should take place during the time period between the kinks $j-1$ and $j+1$. We close this section with the new problem statement (a refinement of Problem 1), and present a solution in the next section.

Problem 3 (Asynchronous synthesis). Given a finite set of PWL functions $\mathcal{F}$ and a value $\varepsilon \in \mathbb{R}_{\geqslant 0}$, construct an ε-precise LHA $\mathcal{H}$ that ε-captures every function $f \in \mathcal{F}$ asynchronously.

4 Membership-based Synthesis Approach

In this section, we present an algorithm for solving Problem 3. The core of the algorithm is a reachability computation for providing the polyhedral regions where executions of an LHA that are ε-close to a given PWL function f are allowed to switch. More precisely, given a path π and the ε-tube of f, the algorithm iteratively constructs the set inside the ε-tube where an execution following π can switch, without escaping from the tube. These reachable set are, in general, computed with respect to a starting compact convex polyhedron P, a pair of adjacent affine pieces p and p', and a pair of modes q and q' along π.

Definition 7. *Given an* LHA $\mathcal{H} = (Q, E, X, \mathit{Flow}, \mathit{Inv}, \mathit{Grd})$ *and a value* $\varepsilon \in \mathbb{R}_{\geqslant 0}$, *a* reachable switching set $\mathit{switch}_{\mathcal{H}}(P, p, p', q, q')$ *from a set* P *with respect to two adjacent affine pieces* p, p' *and a path* $\pi := q, q'$ *in* $\mathcal{H}$ *is defined as*

$$\{\mathbf{x} \in \mathit{Grd}((q, q')) \mid \exists \sigma = (\mathcal{I}, \gamma, \delta) \in \texttt{exec}(\mathcal{H}) : \sigma \overset{\mathcal{H}}{\rightsquigarrow} \pi, \texttt{dom}(\gamma) = \texttt{dom}(p) \cup \texttt{dom}(p'),$$
$$\gamma(0) \in P, \gamma(t) \in \textit{tube}_{p,\varepsilon}(t) \cup \textit{tube}_{p',\varepsilon}(t), \text{ and } \mathbf{x} = \gamma(\texttt{ub}(\mathcal{I}(0)))\}.$$

Inductive Reachable Switching Computation. Given an LHA $\mathcal{H}$, an m-PWL function $f = p_1, \dots, p_m$, a value $\varepsilon \in \mathbb{R}_{\geqslant 0}$ and a path $\pi = q_1, \dots, q_m$ in the graph $(Q_{\mathcal{H}}, E_{\mathcal{H}})$, we compute the reachable switching set P_j^π for every $0 \leqslant j \leqslant m$:

- $P_0^\pi := \mathit{Inv}_{\mathcal{H}}(q_1) \cap \texttt{tube}_{f,\varepsilon}(0)$,
- $P_j^\pi := \mathit{switch}_{\mathcal{H}}(P_{j-1}^\pi, p_{j-1}, p_j, q_{j-1}, q_j)$ for $1 < j < m$, and
- $P_m^\pi := \{\mathbf{x} \in \mathit{Inv}(q_m) \mid \exists \sigma = (\mathcal{I}, \gamma, \delta) \in \texttt{exec}(\mathcal{H}) : \sigma \overset{\mathcal{H}}{\rightsquigarrow} q_m, \gamma(0) \in P_{m-1}^\pi,$
  $\texttt{dom}(\gamma) = \texttt{dom}(p_m),\ \gamma(t) \in \texttt{tube}_{p_m,\varepsilon}(t)$ and $\mathbf{x} = \gamma(\texttt{ub}(\mathcal{I}(m)))\}$.

We denote the set of all reachable switching sets P_j^π by $\mathcal{P}^\pi$. We are now ready to present the complete synthesis algorithm.

Algorithm 1. Synthesis

```
Input: A set of PWL functions $\mathcal{F} = \{f_0, \ldots, f_N\}$ and a value $\varepsilon \in \mathbb{R}_{\geqslant 0}$
Output: A linear hybrid automaton $\mathcal{H}$ that solves Problem 3
 1: $\mathcal{H} :=$ InitLha$(f_0, \varepsilon)$        ▷ construct initial model for $\varepsilon$-capturing $f_0$
 2: for $f \in \mathcal{F} \setminus \{f_0\}$ do
 3:     $(ans, \pi) :=$ Membership$(f, \mathcal{H}, \varepsilon)$
 4:     if not $ans$ then
 5:         $\overline{\mathcal{H}} :=$ RelaxAll$(\mathcal{H}, f, \varepsilon)$        ▷ relax model constraints entirely
 6:         $(ans, \pi) :=$ Membership$(f, \overline{\mathcal{H}}, \varepsilon)$
 7:         if $ans$ then
 8:             $\mathcal{H} :=$ RelaxPath$(\mathcal{H}, f, \varepsilon, \pi)$   ▷ relax model constraints for $\varepsilon$-capturing $f$
 9:         else
10:             $\mathcal{H} :=$ Adapt$(\mathcal{H}, f, \varepsilon, \pi)$        ▷ adapt model for $\varepsilon$-capturing $f$
11: return $\mathcal{H}$
```

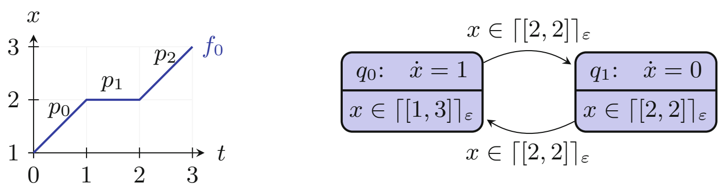
4.1 Membership-based Synthesis Algorithm

The synthesis algorithm outlined in Algorithm 1 computes an LHA $\mathcal{H}$ solving Problem 3 for a given finite set of PWL functions $\mathcal{F}$ and a value $\varepsilon \in \mathbb{R}_{\geqslant 0}$. The algorithm initially infers an LHA $\mathcal{H}$ that ε-captures the first function f_0 of $\mathcal{F}$ in an ε-precise manner in line 1. The remaining PWL functions are handled in an iterative loop. For each PWL function f, the algorithm performs a membership query, where it checks if f is ε-captured by the LHA $\mathcal{H}$ in line 3. If the query results in a positive answer ($ans = True$), nothing needs to be done. Otherwise, the query returns a path π and the LHA $\mathcal{H}$ needs to be modified. The modification of the automaton $\mathcal{H}$ is performed in two attempts. The first attempt, in line 5, temporarily increases invariants and guards of $\mathcal{H}$. If such a modification is sufficient to let the membership query succeed, the modifications are made permanent in line 8. Otherwise, in the second attempt the algorithm adds new modes and/or transitions to $\mathcal{H}$ along the path π. Below we describe every procedure of Algorithm 1 in detail.

Initialization. The procedure InitLha(f, ε) constructs an initial LHA $\mathcal{H}$ that ε-captures an m-PWL function f. Observe that by Lemma 2 the canonical automaton $\mathcal{H}_f$ 0-captures (and hence ε-captures) the function f. In order to allow similar dynamical behaviors in a given LHA $\mathcal{H}$, the procedure InitLha(f, ε) ε-bloats both invariant and guards polyhedra. The procedure InitLha(f, ε) outputs the ε-bloated canonical automaton $\mathcal{H}_f^\varepsilon$ and is illustrated in Fig. 1.

Definition 8. *Given an* LHA $\mathcal{H} = (Q, E, X, Flow, Inv, Grd)$, *we define the* ε -bloated LHA of $\mathcal{H}$ *as* $\mathcal{H}^\varepsilon = (Q, E, X, Flow, Inv^\varepsilon, Grd^\varepsilon)$ *where* $Inv^\varepsilon(q) = \lceil Inv(q) \rceil_\varepsilon$ *for every* $q \in Q$ *and* $Grd^\varepsilon(e) = \lceil Grd(e) \rceil_\varepsilon$ *for every* $e \in E$.

Lemma 3. *Given a* PWL *function* f *and* $\varepsilon \in \mathbb{R}_{\geqslant 0}$, $\mathcal{H}_f^\varepsilon$ ε*-captures* f.



x
3
2
1
0 1 2 3
t
p_0
p_1
p_2
f_0
$x \in \lceil[2,2]\rceil_\varepsilon$
q_0: $\dot{x} = 1$
$x \in \lceil[1,3]\rceil_\varepsilon$
q_1: $\dot{x} = 0$
$x \in \lceil[2,2]\rceil_\varepsilon$
$x \in \lceil[2,2]\rceil_\varepsilon$

Fig. 1. Example describing the procedure InitLha(f, ε) for a 3-pwl function $f = f_0$ (depicted on the left). The function f_0 consists of three pieces p_0, p_1, p_2 with slopes $1, 0, 1$, respectively. The lha on the right is constructed as follows. Mode q_0 corresponds to pieces p_0 and p_2; the invariant is the ε-bloating of interval $[1, 3]$ (which is the convex hull of every start and end point in both pieces). Likewise, mode q_1 corresponds to piece p_1. Transitions and their guards correspond to the kinks of f_0 at $t = 1$ and $t = 2$.

Membership. The procedure Membership$(f, \mathcal{H}, \varepsilon)$ checks whether there exists an *asynchronous* execution $\sigma = (\mathcal{I}, \gamma, \delta)$ in $\mathcal{H}$ such that $d(f, \gamma) \leqslant \varepsilon$ holds. Let us introduce the required notions to formalize the membership problem.

Definition 9. *An execution $\sigma = (\mathcal{I}, \gamma, \delta)$ of an* lha *$\mathcal{H}$ is* consistent *with an m-*pwl *function f, described by the affine pieces $p_1, \ldots, p_m$, if $\texttt{len}(\mathcal{I}) = m$, $[\![\mathcal{I}]\!] = \texttt{dom}(f)$, and $\texttt{ub}(\mathcal{I}(j)) \in \texttt{dom}(p_j) \cup \texttt{dom}(p_{j+1})$ for every $1 \leqslant j < m$.*

Problem 4 (Membership). Given an m-pwl function f, an lha $\mathcal{H}$, and a value $\varepsilon \in \mathbb{R}_{\geqslant 0}$, decide if there exists an execution $\sigma = (\mathcal{I}, \gamma, \delta)$ in $\texttt{exec}(\mathcal{H})$ that is consistent with f and such that $d(f, \gamma) \leqslant \varepsilon$ holds.

The procedure Membership$(f, \mathcal{H}, \varepsilon)$ solves Problem 4 by computing the reachable switching sets for every path π of length m in $\mathcal{H}$ until finding a path π where every reachable switching set P_j^π for $0 \leqslant j \leqslant m$ is nonempty. Upon finding a path π satisfying the previous constraints, Membership$(f, \mathcal{H}, \varepsilon)$ returns *True* as answer, together with the path π. If there does not exist such a path π, it returns *False* as answer. We show an example in Fig. 2(a). We remark that, for a fixed path, Problem 4 is a timestamp-generation problem [2] with the restriction to time intervals for switching and the ε-tube as solution corridor.

Lemma 4. *Let $\mathcal{H}$ be an* lha *and f be an m-*pwl *function. Then there exists a path π of length m in $\mathcal{H}$ such that the final reachable switching set P_m^π is not empty if and only if there exists an execution σ in $\texttt{exec}(\mathcal{H})$ solving Problem 4.*

Relaxation. If Membership$(f, \mathcal{H}, \varepsilon)$ returns *False*, RelaxAll$(\mathcal{H}, f, \varepsilon)$ constructs an automaton $\overline{\mathcal{H}}$ that is equivalent to $\mathcal{H}$ except that its invariants and guards are enlarged to allow additional executions inside the $\texttt{tube}_{f,\varepsilon}$. Then, the algorithm computes Membership$(f, \overline{\mathcal{H}}, \varepsilon)$. If the answer is *False* again, the algorithm proceeds to the adaptation procedure in line 10. Otherwise (if the answer is *True*), we obtain a path π in $\overline{\mathcal{H}}$. Then the algorithm executes the procedure RelaxPath$(\mathcal{H}, f, \varepsilon, \pi)$, which extends the constraints of invariants and guards

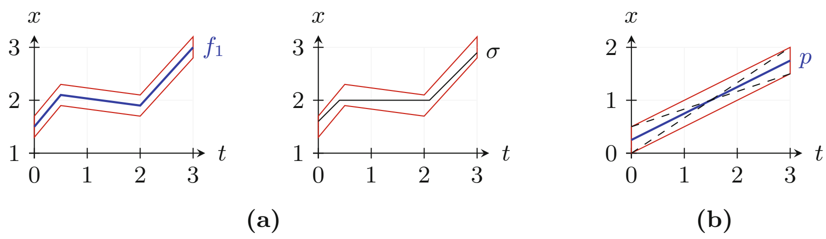


Fig. 2. (a) Example describing the procedure MEMBERSHIP$(f, \mathcal{H}, \varepsilon)$. On the left we depict a 3-PWL function f_1 and its ε-tube. On the right we show a possible execution in the LHA from Fig. 1. **(b)** Given an affine piece p, we say that another piece has a *similar* slope if it does not leave the tube. In the figure, we show the minimal and the maximal allowed slopes by dashed segments.

in $\mathcal{H}$ for the modes in π by taking the convex hull with the corresponding reachable switching sets $P_j^\pi \in \mathcal{P}^\pi$. The relaxation procedure applied on the running example is shown in Fig. 3.

Adaptation. If both the membership query and the relaxation procedure fail, the procedure ADAPT$(\mathcal{H}, f, \varepsilon, \pi)$ modifies the LHA $\mathcal{H}$ for ε-capturing f. Conceptually, we construct a new path π', based on some path π, and modify $\mathcal{H}$ accordingly such that the graph of $\mathcal{H}$ contains π'. Recalling Lemma 4, we need to ensure that every reachable switching set in $\mathcal{P}^{\pi'}$ is nonempty. We construct π' by trying to preserve the modes in path π. If this is not possible, we try to replace them by existing modes in the LHA $\mathcal{H}$ whenever possible, potentially adding new transitions. The last option is to create new modes. Finally, we extend the LHA $\mathcal{H}$ by adding the new transitions and/or modes determined by the new path π'.

In more detail, given an LHA $\mathcal{H}$, an m-PWL function f and a path $\pi = q_1, \ldots, q_m$ in $\mathcal{H}$, we start with path $\pi' = \pi$. Then, the adaptation procedure checks whether there is an empty reachable switching set in $\mathcal{P}^{\pi'}$. Every time we detect emptiness of the set $P_j^{\pi'}$ for some $0 \leqslant j \leqslant m$, a mode in the path π' is replaced in order to make $P_j^{\pi'}$ nonempty. We first try to replace the mode q_{j+1} if it exists. If $P_j^{\pi'}$ is still empty or q_{j+1} does not exist, we repeat the replacement for q_j, q_{i-1}, and so on, until $P_j^{\pi'}$ finally becomes nonempty.

For the replacement of the j-th mode q in the path π' we follow two strategies. The first strategy is to replace the mode q by an existing mode $q' \neq q$ in $\mathcal{H}$ such that $Flow_{\mathcal{H}}(q')$ is *similar* to $\texttt{slope}(p_j)$. Formally, let T be the duration of piece p_j. $Flow_{\mathcal{H}}(q')$ is similar to $\texttt{slope}(p_j)$ if $\|\texttt{init}(p_j) + \mathsf{T} \cdot Flow_{\mathcal{H}}(q') - \texttt{end}(p_j)\| \leqslant 2\varepsilon$. See Fig. 2(b) for an example. If the first strategy fails, the second strategy is to create a new mode q^* with flow $\texttt{newflow}(q^*) = \texttt{slope}(p_j)$ for replacement in π'. We denote the set of existing modes similar to some mode q in π by $\texttt{sim}(\pi')$, and the set of new modes q^* by $\texttt{new}(\pi')$. Once the path π' is constructed, the adaptation of the LHA $\mathcal{H}$ is performed with respect to π'. Figure 4 exemplifies the adaptation of the LHA in Fig. 1.

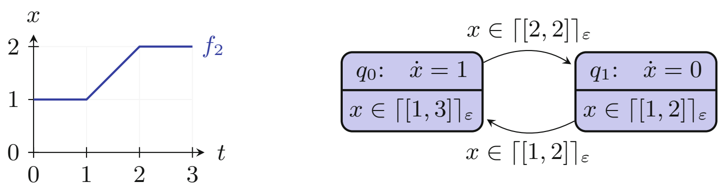

x
2
1
0
f_2
0
1
2
3
t
$x \in \lceil[2,2]\rceil_\varepsilon$
q_0: $\dot{x} = 1$
$x \in \lceil[1,3]\rceil_\varepsilon$
q_1: $\dot{x} = 0$
$x \in \lceil[1,2]\rceil_\varepsilon$
$x \in \lceil[1,2]\rceil_\varepsilon$

Fig. 3. Example describing the procedure RELAXPATH$(\mathcal{H}, f, \varepsilon, \pi)$ for $\mathcal{H}$ given in Fig. 1, $f = f_2$ (depicted on the left), and path $\pi = q_1, q_0, q_1$. The algorithm increases the invariant of mode q_1 by computing the convex hull of the old invariant $\lceil[2,2]\rceil_\varepsilon$ and the set $\lceil[1,1]\rceil_\varepsilon$. Analogously, the guard of the transition (q_1, q_0) is increased.

Definition 10. *The* adaptation *of the* LHA $\mathcal{H} = (Q, E, X, \mathit{Flow}, \mathit{Inv}, \mathit{Grd})$ *with respect to an* m-PWL *function* f *with affine pieces* $p_1, \ldots, p_m$ *and a path* $\pi = q_1, \ldots, q_m$ *is the* LHA $\mathcal{H}' = (Q', E', X, \mathit{Flow}', \mathit{Inv}', \mathit{Grd}')$ *defined as:*

- $Q' := Q \cup \mathtt{new}(\pi')$,
- $E' := E \cup \{(q_j, q_{j+1}) \mid 1 \leqslant j < m\}$,
- $\mathit{Flow}'(q) := \begin{cases} \mathtt{newflow}(q) & \textit{if } q \in \mathtt{new}(\pi'), \\ \mathit{Flow}(q) & \textit{otherwise}, \end{cases}$
- $\mathit{Inv}'(q) := \begin{cases} \mathtt{chull}(\bigcup_{q=q_j, q\neq q_1} P^{\pi'}_{j-1} \cup \bigcup_{q=q_j} P^{\pi'}_{j}) & \textit{if } q \in \mathtt{new}(\pi'), \\ \mathtt{chull}(\mathit{Inv}(q) \cup \bigcup_{q=q_j, q\neq q_1} P^{\pi'}_{j-1} \cup \bigcup_{q=q_j} P^{\pi'}_{j}) & \textit{if } q \in \mathtt{sim}(\pi'), \\ \mathit{Inv}(q) & \textit{otherwise}, \end{cases}$
- $\mathit{Grd}'((q, q')) := \begin{cases} \mathtt{chull}(\bigcup_{q=q_j, q'=q_{j+1}} P^{\pi'}_{j}) & \textit{if } q \in \mathtt{new}(\pi') \\ & \textit{or } q' \in \mathtt{new}(\pi'), \\ \mathtt{chull}(\mathit{Grd}((q, q')) \cup \bigcup_{q=q_j, q'=q_{j+1}} P^{\pi'}_{j}) & \textit{if } q \in \mathtt{sim}(\pi') \\ & \textit{or } q' \in \mathtt{sim}(\pi'), \\ \mathit{Grd}((q, q')) & \textit{otherwise.} \end{cases}$

If there is no path of length m in the graph of $\mathcal{H}$, we choose a shorter path π in $\mathcal{H}$ of length m' for the adaptation procedure. Then, for every position $j \geqslant m'$, we define the reachable switching set P^{π}_{j} as an empty set and proceed as usual.

4.2 Discussion

The construction of the initial LHA (line 1 in Algorithm 1) can be modified to *clustering* pieces with *similar* slopes. This can help reducing the number of modes in the initial automaton, but does not guarantee that the first PWL function f_0 is ε-captured. To fix this, f_0 can be included in the loop of Algorithm 1.

Algorithm 1 follows a *local* repair strategy, based on a single PWL function. Thanks to this, the algorithm can be used in an online setting where new data arrives after the algorithm has started. However, the resulting model is influenced

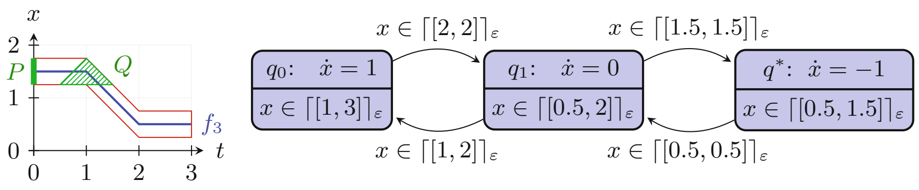

x
t
P
Q
f_3
0
1
2
3
q_0: $\dot{x} = 1$
$x \in \lceil[1,3]\rceil_\varepsilon$
q_1: $\dot{x} = 0$
$x \in \lceil[0.5,2]\rceil_\varepsilon$
q^*: $\dot{x} = -1$
$x \in \lceil[0.5,1.5]\rceil_\varepsilon$
$x \in \lceil[2,2]\rceil_\varepsilon$
$x \in \lceil[1,2]\rceil_\varepsilon$
$x \in \lceil[1.5,1.5]\rceil_\varepsilon$
$x \in \lceil[0.5,0.5]\rceil_\varepsilon$

Fig. 4. Example describing the procedure ADAPT($\mathcal{H}, f, \pi, \varepsilon$) for the LHA $\mathcal{H}$ in Fig. 1 with respect to the 3-PWL function $f = f_3$ and the path $\pi = q_1, q_0, q_1$ and $\varepsilon = 0.25$. The initial reachable switching set P_0^π is the projection of the set P on state x. Considering the flows in q_1 and q_0, the next reachable switching set P_1^π is the projection of the set Q on state x. Observe that from Q, using the flow of q_1, the reachable switching set P_2^π is empty. We thus add a new mode q^* and obtain the new path $\pi' = q_1, q^*, q_1$.

by the order in which the algorithm processes the functions $f \in \mathcal{F}$. In the simple case that $\mathcal{F}$ only contains affine functions with the same slope, all models resulting from different processing orders will consist of a single mode with the same flow, and the invariant bounds differ by at most ε. Furthermore, for a precision value $\varepsilon = 0$, the result is always order-independent.

We now discuss the restrictions of the models we obtain from Algorithm 1. We did not include a set of initial states in our presentation, but the generalization is straightforward. Our transitions do not include assignments, which would make executions discontinuous. The usual assumption in many application domains, e.g., life sciences, is that the underlying system is continuous, so having assignments would not be desirable. In the setting where the input is given as time-series data, discrete events would typically be approximated by steep slopes in the PWL function. In the setting where the input is given as discontinuous PWL functions f, in order to ε-capture f, one would generally require that the automaton switches synchronously with f (cf. Sect. 3.1), instead of asynchronous switching as in our algorithm. Under this additional assumption, we can pose the procedures MEMBERSHIP and RELAXPATH as a single linear program (similar to formula $\phi_{f,\varepsilon}$). This linear program can also be used to identify assignments.

The continuous dynamics of our models are defined by constant differential equations. As mentioned before, this class generally suffices to approximate an arbitrary continuous function (by increasing the number of modes). An extension of our approach to use polyhedral differential *inclusions* (also called linear envelopes) is by merging modes of "similar" dynamics. This may, however, lead to the dilemma that several modes are equally similar.

4.3 Theoretical Properties of the Membership-based Synthesis

The following theorem asserts that Algorithm 1 solves Problem 3.

Theorem 2 (Soundness and precision). *Given a finite set of* PWL *functions* $\mathcal{F}$ *and a value* $\varepsilon \in \mathbb{R}_{\geqslant 0}$*, let* $\mathcal{H}$ *be an automaton resulting from* Synthesis$(\mathcal{F}, \varepsilon)$*. Then* $\mathcal{H}$ *both* ε*-captures all functions in* $\mathcal{F}$ *and is* ε*-precise with respect to* $\mathcal{F}$*.*

Algorithm 1 satisfies a completeness property in the following sense. For every model $\mathcal{H}$ from a certain class we can find a set $\mathcal{F}$ of PWL functions and a value ε such that Synthesis$(\mathcal{F}, \varepsilon)$ results in $\mathcal{H}$. Before we can characterize the class of models, we first need to introduce some terminology.

Definition 11. *Let* $q \in Q$ *be a mode with invariant* $X = Inv(q)$ *and flow* $Flow(q)$*. We call a continuous state* $\mathbf{x}_2 \in X$ forward reachable *in* q *if there is a continuous state* $\mathbf{x}_1 \in X$ *such that* $\mathbf{x}_2$ *is reachable from* $\mathbf{x}_1$ *by just letting time pass, i.e.,* $\exists t > 0 : \mathbf{x}_2 = \mathbf{x}_1 + Flow(q) \cdot t$*. Analogously, we call state* $\mathbf{x}_2 \in X$ backward reachable *in* q *if there is a state* $\mathbf{x}_1 \in X$ *such that* $\mathbf{x}_2$ *is reachable from* $\mathbf{x}_1$*. A continuous state is* dead *in* q *if it is neither forward reachable nor backward reachable in* q*.*

We characterize the class of automata $\mathcal{H} = (Q, E, X, Flow, Inv, Grd)$ for which the algorithm is complete by considering the following assumptions: (1) no invariant contains a dead continuous state. Furthermore, if $e = (q_1, q_2)$ is a transition, then all continuous states in the guard $Grd(e)$ are forward reachable in q_1 and backward reachable in q_2, and (2) no two modes have the same slope □

Roughly speaking, Assumption (1) asserts that, after every switch, an execution can stay in the new mode for a positive amount of time.

Theorem 3 (Completeness). *Given an* LHA $\mathcal{H}$ *satisfying Assumptions (1) and (2), there exist* PWL *functions* $\mathcal{F}$ *such that* Synthesis$(\mathcal{F}, 0)$ *results in* $\mathcal{H}$*.*

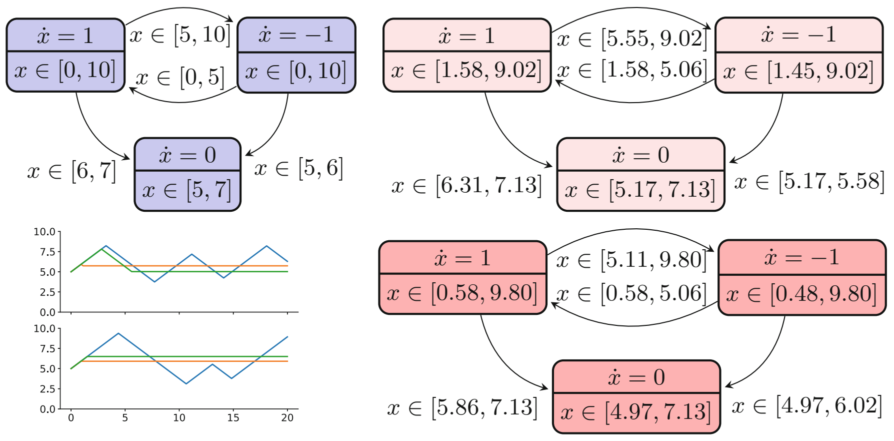
5 Experimental Results

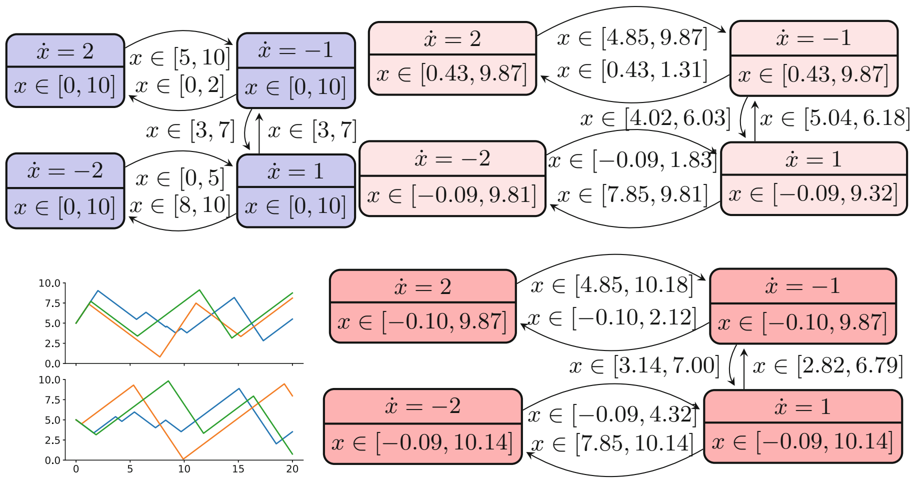
In this section, we present the experiments used to evaluate our algorithm. The algorithm was implemented in Python and relies on the standard scientific computation packages. For the computations involving polyhedra we used the `pplpy` wrapper to the Parma Polyhedra Library [4].

Case Study: Online Synthesis. We evaluate the precision of our algorithm by collecting data from the executions of existing linear hybrid automata. For each given automaton, we randomly sample ten executions and pass them to our algorithm, which then constructs a new model. After that, we run our algorithm with another 90 executions, but we reuse the intermediate model, thus demonstrating the online feature of the algorithm. We show the different models for two handcrafted examples in Table 1. We tried both sampling from random states and from a fixed state. The examples show the latter case, which makes sampling the complete state-space and thus learning a precise model harder.

The first example contains a sink with two incoming transitions, which requires at least two simulations to observe both transitions. Consequently, the algorithm had to make use of the *adaptation* step at least once to add one of the

Table 1. Synthesis results for two automaton models. The original model is shown in blue. The synthesis result after 10 iterations is shown in bright red, and after another 90 iterations in dark red. On the bottom left we show three sample executions starting from the same point (top: original model, bottom: synthesized model after 100 iterations). We used $\varepsilon = 0.2$ in all cases. Numbers are rounded to two places.





transitions. In the second example, some parts of the state-space are explored less frequently by the sampled executions. Hence the first model obtained after ten iterations does not represent all behavior of the original model yet. After the additional 90 iterations, the remaining parts of the state space have been visited, which is reflected in the precise bounds of the resulting model. In the table, we also show three sample executions from both the original and the final synthesized automaton to illustrate the similarity in the dynamical behavior.

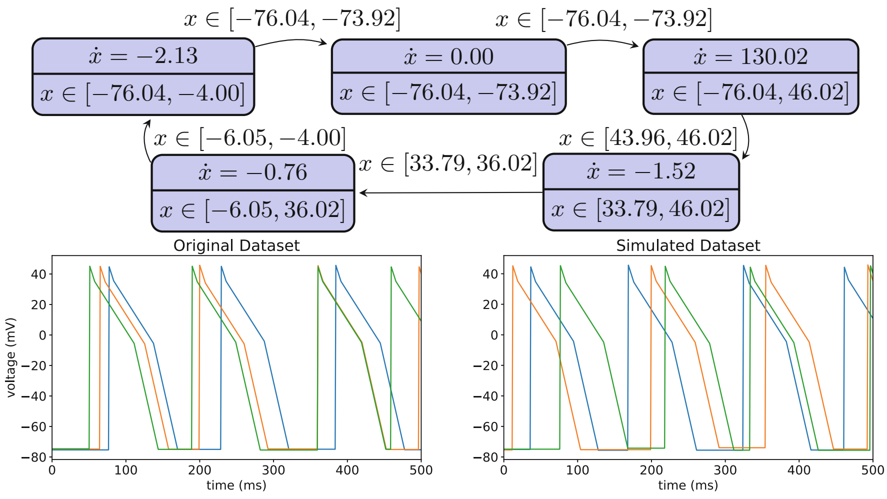


Fig. 5. Results for the cell model. Top: synthesized model using our algorithm. Bottom: three input traces (left) and random simulations of the synthesized model (right).

Case Study: Cell Model. For our case study we synthesize a hybrid automaton from voltage traces of excitable cells. Excitable cells are an important class of cells comprising neurons, cardiac cells, and other muscle cells. The main property of excitable cells is that they exhibit electrical activity which in the case of neurons enables signal transmission and in the case of muscle cells allows them to contract. The excitation signal usually follows distinct dynamics called action potential. Grosu et al. construct a *cyclic-linear hybrid automaton* from action-potential traces of cardiac cells [8]. In their model they identify six modes, two of which exhibit the same dynamics and are just used to model an input signal.

Our algorithm successfully synthesizes a model, depicted in Fig. 5, consisting of five modes that roughly match the normal phases of an action potential. We evaluate the quality of the synthesized model by simulating random executions and visually comparing to the original data (see the bottom of Fig. 5).

6 Conclusion

In this paper we have presented two fully automatic approaches to synthesize a linear hybrid automaton from data. As key features, the synthesized automaton captures the data up to a user-defined bound and is tight. Moreover, the online feature of the membership-based approach allows to combine the approach with alternative synthesis techniques, e.g., for constructing initial models.

A future line of work is to design a methodology for identification of weak generalizations in the model, and use them for driving the experiments and, in consequence, adjusting the model. We would first synthesize a model as before, but then identify the aspects of the model that are least substantiated by the data (e.g., areas in the state space or specific sequences in the executions). Then we would query the system for data about those aspects, and repair the model accordingly. As another line of work, we plan to extend the approach to go from dynamics defined by piecewise-constant differential equations toward linear envelopes. Our approach can be seen as a generalization, to LHA, of Angluin's algorithm for constructing a finite-state machine from finite traces [3], and we plan to pursue this connection further.

References

1. Alur, R., Courcoubetis, C., Henzinger, T.A., Ho, P.-H.: Hybrid automata: an algorithmic approach to the specification and verification of hybrid systems. In: Grossman, R.L., Nerode, A., Ravn, A.P., Rischel, H. (eds.) HS 1991-1992. LNCS, vol. 736, pp. 209–229. Springer, Heidelberg (1993). https://doi.org/10.1007/3-540-57318-6_30
2. Alur, R., Kurshan, R.P., Viswanathan, M.: Membership questions for timed and hybrid automata. In: RTSS, pp. 254–263. IEEE Computer Society (1998). https://doi.org/10.1109/REAL.1998.739751
3. Angluin, D.: Learning regular sets from queries and counterexamples. Inf. Comput. **75**(2), 87–106 (1987). https://doi.org/10.1016/0890-5401(87)90052-6
4. Bagnara, R., Hill, P.M., Zaffanella, E.: The Parma Polyhedra Library: toward a complete set of numerical abstractions for the analysis and verification of hardware and software systems. Sci. Comput. Program. **72**(1–2), 3–21 (2008). https://doi.org/10.1016/j.scico.2007.08.001
5. Bemporad, A., Garulli, A., Paoletti, S., Vicino, A.: A bounded-error approach to piecewise affine system identification. IEEE Trans. Autom. Control **50**(10), 1567–1580 (2005). https://doi.org/10.1109/TAC.2005.856667
6. 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), 112–122 (1973)
7. Garulli, A., Paoletti, S., Vicino, A.: A survey on switched and piecewise affine system identification. IFAC Proc. Vol. **45**(16), 344–355 (2012). https://doi.org/10.3182/20120711-3-BE-2027.00332
8. Grosu, R., Mitra, S., Ye, P., Entcheva, E., Ramakrishnan, I.V., Smolka, S.A.: Learning cycle-linear hybrid automata for excitable cells. In: Bemporad, A., Bicchi, A., Buttazzo, G. (eds.) HSCC 2007. LNCS, vol. 4416, pp. 245–258. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-71493-4_21
9. Hakimi, S.L., Schmeichel, E.F.: Fitting polygonal functions to a set of points in the plane. CVGIP Graph. Model. Image Process. **53**(2), 132–136 (1991). https://doi.org/10.1016/1049-9652(91)90056-P
10. Hashambhoy, Y., Vidal, R.: Recursive identification of switched ARX models with unknown number of models and unknown orders. In: CDC, pp. 6115–6121 (2005). https://doi.org/10.1109/CDC.2005.1583140
11. Henzinger, T.A.: The theory of hybrid automata. In: Inan, M.K., Kurshan, R.P. (eds.) Verification of Digital and Hybrid Systems. NATO ASI Series (Series F: Computer and Systems Sciences), vol. 170, pp. 265–292. Springer, Berlin, Heidelberg (2000). https://doi.org/10.1007/978-3-642-59615-5_13

12. 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. IEEE (2018). https://doi.org/10.1109/ICPHYS.2018.8387670
13. Liberzon, D.: Switching in Systems and Control. Birkhäuser, Boston (2003). https://doi.org/10.1007/978-1-4612-0017-8
14. Ly, D.L., Lipson, H.: Learning symbolic representations of hybrid dynamical systems. JMLR **13**, 3585–3618 (2012). http://dl.acm.org/citation.cfm?id=2503356
15. Medhat, R., Ramesh, S., Bonakdarpour, B., Fischmeister, S.: A framework for mining hybrid automata from input/output traces. In: EMSOFT, pp. 177–186. IEEE (2015). https://doi.org/10.1109/EMSOFT.2015.7318273
16. Niggemann, O., Stein, B., Vodencarevic, A., Maier, A., Kleine Büning, H.: Learning behavior models for hybrid timed systems. In: AAAI. AAAI Press (2012). http://www.aaai.org/ocs/index.php/AAAI/AAAI12/paper/view/4993
17. Ozay, N.: An exact and efficient algorithm for segmentation of ARX models. In: ACC, pp. 38–41. IEEE (2016). https://doi.org/10.1109/ACC.2016.7524888
18. 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). https://doi.org/10.3166/ejc.13.242-260
19. Skeppstedt, A., Lennart, L., Millnert, M.: Construction of composite models from observed data. Int. J. Control **55**(1), 141–152 (1992). https://doi.org/10.1080/00207179208934230
20. Summerville, A., Osborn, J.C., Mateas, M.: CHARDA: causal hybrid automata recovery via dynamic analysis. In: IJCAI, pp. 2800–2806. ijcai.org (2017). https://doi.org/10.24963/ijcai.2017/390
21. Verwer, S.: Efficient identification of timed automata: theory and practice. Ph.D. thesis, Delft University of Technology, Netherlands (2010). http://resolver.tudelft.nl/uuid:61d9f199-7b01-45be-a6ed-04498113a212
22. Vidal, R., Anderson, B.D.O.: Recursive identification of switched ARX hybrid models: exponential convergence and persistence of excitation. In: CDC, vol. 1, pp. 32–37 (2004). https://doi.org/10.1109/CDC.2004.1428602

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.