

Conservative Hybrid Automata from Development Artifacts

Niklas Metzger
CISPA Helmholtz Center for
Information Security
Saarbrücken, Germany
niklas.metzger@cispa.de

Sanny Schmitt
CISPA Helmholtz Center for
Information Security
Saarbrücken, Germany
sanny.schmitt@stud.uni-saarland.de

Maximilian Schwenger
CISPA Helmholtz Center for
Information Security
Saarbrücken, Germany
maximilian.schwenger@cispa.de

ABSTRACT

The verification of cyber-physical systems operating in a safety-critical environment requires formal system models. The validity of the verification hinges on the precision of the model: possible behavior not captured in the model can result in formally verified, but unsafe systems. Yet, manual construction is delicate and error-prone while automatic construction does not scale for large and complex systems. As a remedy, this paper devises an automatic construction algorithm that utilizes information contained in artifacts of the development process: a runtime monitoring specification and recorded test traces. These artifacts incur no additional cost and provide sufficient information so that the construction process scales well for large systems. The algorithm uses a hybrid approach between a top-down and a bottom-up construction which allows for proving the result *conservative*, while limiting the level of over-approximation.

CCS CONCEPTS

• **Computer systems organization** → **Embedded and cyber-physical systems**; • **Theory of computation** → **Timed and hybrid models**; Formal languages and automata theory.

KEYWORDS

Hybrid Automata, Conservative Construction, Automata Learning

ACM Reference Format:

Niklas Metzger, Sanny Schmitt, and Maximilian Schwenger. 2022. Conservative Hybrid Automata from Development Artifacts. In *Proceedings of Hybrid Systems: Computation and Control (HSCC'22)*. ACM, New York, NY, USA, 13 pages. <https://doi.org/10.1145/nnnnnnn.nnnnnnn>

1 INTRODUCTION

Hybrid systems connect the discrete, logical world of computers with the continuous, unpredictable real world. In the last decades, they became an inevitable part of society by controlling essential infrastructures such as power plants, commercial airplanes, and cars. A mathematical model for such systems are hybrid automata, which combine information regarding their discrete control structure and continuous physical behavior. They can be used in all phases of

development. Before deployment, they allow for analysis and verification of critical properties. During deployment, time-bounded prediction and identification of anomalous or unexpected behavior based on the model is possible. Lastly, recorded mission data enables a post-mortem analysis as well. Albeit indisputably beneficial for the development process, designing a hybrid automaton to properly reflect the semantics of the system is a delicate process. Thus, unsurprisingly, several approaches aim at automatically constructing either hybrid automata or their simpler cousins, timed automata, based on execution traces of the system. These traces are usually a development artifact as they get recorded during test runs. Estimation methods, most prominently machine-learning, yield promising results in terms of reconstructing the correct discrete structure of the automaton and approximating the continuous dynamics. Their great accuracy notwithstanding, the *direction* of the approximation is unclear, resulting in over- and under-approximation. While this suffices for conveying the gist of the system, it does limit its capabilities in terms of safety-critical analyses. For this reason, we propose a construction algorithm for *conservative* hybrid automata, i.e., a guaranteed over-approximation of the original system.

Rather than relying solely on execution traces, the construction employs another development artifact: a formal runtime monitoring specification. This specification is crucial for safety-critical systems as it enables dynamic verification methods — *runtime verification* — in which a dedicated component monitors the behavior of the system during the execution. When the monitor deems this information indicative of a malfunction, it terminates the execution. Note that the constraints imposed by the specification encompass the entire execution and change depending on the state of the system. Hence, to judge the situation accurately, the monitor — and by proxy the specification — needs to keep track of different operational phases.

This comes to show that both the execution traces and the specification manifest expertise acquired during the development process: a) keeping track of operational phases requires the specification to contain information regarding the discrete control structure and b) since the execution traces are recorded test runs, they are expected to satisfy relevant coverage criteria. Hence, not only do they cover the “average”, expected behavior including initialization and termination, they also cover the extreme and corner case behavior. Consequently, both the specification and the traces constitute indispensable resources.

The construction algorithm proposed in this paper first extracts the implicit discrete control structure from the specification, resulting in a strong over-approximation of the system. It then proceeds by a) refining this information based on discrete control signals contained in the execution traces and b) enriching it by analyzing the evolution of samples over time. The resulting automaton

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

HSCC'22, May 4–6, 2022, Milan, Italy

© 2022 Association for Computing Machinery.

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM... \$15.00

<https://doi.org/10.1145/nnnnnnn.nnnnnnn>

is an under-approximation. Hence, the last step of the algorithm merges control modes within the automaton based on discrete evidence found in the traces to finally obtain an over-approximation of the original system. This mixture of a top-down and bottom-up construction results in an automaton that is both a provable over-approximation and retains a high level of precision.

Apart from being conservative, the construction distinguishes itself from existing approaches in two major ways. First, the specification roughly indicates the general discrete structure of the constructed automaton. This alleviates the need to second-guess the structure in its entirety, reducing revisions to local sub-structures. This pushes scalability far beyond L^* -based approaches [5] like Medhat et al. [14] in which significant time is spent to determine the discrete structure. Secondly, the construction reduces the level of over-approximation by merging modes of an under-approximation only if needed. This can result in more fine-grained refinements than when successively widening dynamics until the language of the automaton encompasses every input trace [18].

An empirical evaluation validates three major claims. First, the construction requires few traces to produce decent results. For a fourteen-mode automaton, for example, seven hand-picked or on average 35 random traces suffice for a perfect reconstruction. Secondly, the precision – while not flawless – comes close to the optimal result for adequate input data. Thirdly, the construction algorithm scales extraordinarily well. Even large automata with over 1000 modes can be constructed within mere seconds. All three benefits are the result of relying on development artifacts in form of test traces and a runtime monitoring specification: a readily available resource often left under-utilized.

All in all, the contributions of this paper are:

- A three step construction for conservative hybrid automaton: First, it extracts a discrete over-approximation of the system from a runtime monitoring specification. This approximation is successively enriched with continuous information and refined into an under-approximation by incorporating data from execution traces. Lastly, it merges modes based on discrete evidence found in test traces until obtaining an over-approximation.
- A correctness proof of the construction: under realistic assumptions on the input traces, the constructed automaton subsumes the language of the original system projected onto the behavior exposed through the set of input traces. The projection is required to eliminate parts of the system that are not exposed to the outside such as unreachable modes.
- Experimental results showcasing the quality and scalability of the conservative construction. Even small sets of input traces stemming from random walks allow for precise constructions. Moreover, automata with thousands of modes can be constructed in a matter of seconds.

2 MOTIVATION

The foundation for the conservative construction is a set of traces generated from an unknown system and a runtime monitoring specification thereof. As a running example, consider the automaton depicted in Figure 1a, a simple model for an aircraft. The system starts in a takeoff mode and resides there until reaching cruising

altitude. Here, it can go straight or adjust its course via left and right curves until it attempts a landing. Under windy conditions, the descend-phase is elongated as a precaution.

A specification for the aircraft imposes several constraints depending on the current state of the system. For example, during takeoff, the specification requires the aircraft to accelerate; while traveling it requires a stable altitude; during landing it requires the landing gear to be lowered. An analysis of the specification hence yields a state machine with coarse information on different execution phases as well as conditions on phase-changes. The state machine is depicted in color in Figure 1a, superimposed by the aircraft automaton. As can be seen, the specification does not distinguish between maintaining course or adjusting it; the requirements on the system remain the same. Yet, it contains no indication regarding the continuous behavior of the system.

To fill these gaps, the conservative construction then successively enriches the specification automaton with information extracted from the set of traces. The whole process is illustrated in Figure 1c. It first translates the specification automaton \mathcal{A}^Φ into a hybrid automaton \mathcal{H}_1^+ . For each step of each trace, it adds more modes into the automaton while maintaining the structure provided by the specification. The result, i.e., $\mathcal{H}_{|\Pi|}^+$, is by design overly restrictive: it consists of a single, isolated path for each trace. A merge process based on the discrete behavior of modes remedies this problem and finally produces \mathcal{H}^+ .

The key point behind \mathcal{H}^+ is that it is *conservative*, i.e., under certain assumptions on the specification and traces, \mathcal{H}^+ over-approximates \mathcal{H} . The assumptions are three-fold: the specification needs to a) be a coarse abstraction of the actual system, b) agree with the system on phase changes, and c) the set of traces needs to encompass sufficient information on the discrete behavior. While these assumptions seem strong, they are tailored for the use case at hand such that they are expected to be satisfied naturally. The first and second assumption concern the specification, which is hand-crafted specifically for the system. Hence, the specifier must have had knowledge regarding the abstract control structure, e.g. through which phases the system traverses during a mission. Evidently, the concrete control structure refines the abstract structure for more fine-grained control. This abstract control structure manifests itself in the specification. Here, each mode of the abstract structure imposes a different set of requirements on the system. A violation of such a requirement constitutes a safety-hazard. Hence, engineers need to define precisely when the requirements on the system changes. As a result, the specification needs to contain the same precise criterion for a phase change as the system itself.

Regarding the third criterion, recall that the set of traces is a development artifact that arose from the testing process. Thus, not only do they cover large parts of the system’s regular execution, they represent executions in which the system was purposefully coerced into extreme and corner case behavior. Moreover, the empirical evaluation revealed that even small sets of random walks through the underlying system already produces an adequate trace set. Here, “small” means on average 32 traces for a fourteen state automaton and two traces for a seven state automaton. This reasoning shows why these assumptions, while strong, are realistic when considering carefully engineered, safety-critical system. A

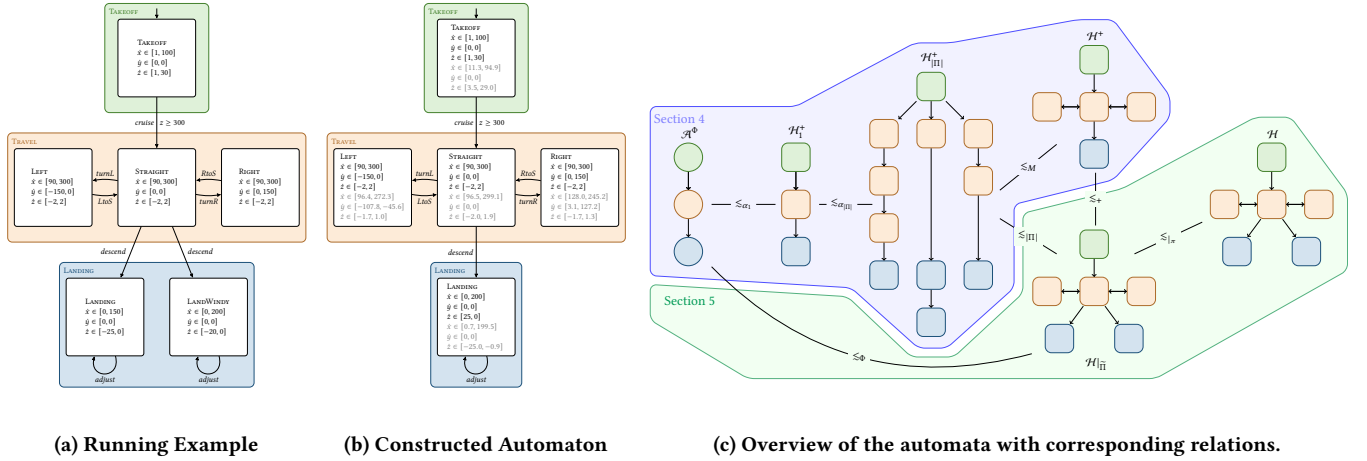


Figure 1: The specification automaton for an aircraft superimposed by its hybrid automaton in Figure 1a and the result of the construction from three hand-picked traces (black dynamics) and ten random traces (gray dynamics) in Figure 1b. An overview of the automata is displayed in Figure 1c, where the notation $\mathcal{H}_1 \lesssim \mathcal{H}_2$ indicates that \mathcal{H}_2 simulates \mathcal{H}_1 .

formal description of the assumptions and the proof that \mathcal{H}^+ is conservative can be found in Section 5.

The empirical evaluation of the approach in Section 6 allows for validating three claims. a) The construction requires a *low volume of input traces* — especially when compared to machine-learning approaches. For the running example, the construction requires as little as three traces of length eight. b) It *scales linearly* for increasing dimension and *quadratically* in the number of traces and size of the original/constructed system. Constructing a three-dimensional automaton with 2^{10} modes based on a specification with nine states and 512 traces requires less than a second. c) Though over-approximating, the constructed automaton has a *high level of precision*. For the aircraft, the construction is identical to the original apart from a merge of the two terminal modes.

3 PRELIMINARIES

Definition 3.1 (Interval). $\mathcal{I}^n = \mathbb{R}^{2n}$ denotes an n -dimensional rectangle where $\mathcal{I}^1 = \mathcal{I}$ is an interval over \mathbb{R} . The multiplication of a k - and an ℓ -dimensional rectangle yields a $(k + \ell)$ -dimensional rectangle. Addition and multiplication of intervals with scalars are geometric translation and scaling, respectively.

Definition 3.2 (Convex Hull). Let S be a convex set. The *convex hull* of two convex sets $A \subseteq S$ and $B \subseteq S$ is the minimal convex set covering both A and B . Further, let \mathcal{S} be a non-empty set of convex sets. $\text{Conv}(\mathcal{X})$ computes the convex set covering all elements of \mathcal{X} by applying the convex hull iteratively in arbitrary order. $\text{Conv}(\mathcal{X})$ yields the identity for singleton inputs.

$$\text{chull}(A, B) = \bigcup_{p_a \in A} \bigcup_{p_b \in B} [p_a, p_b]$$

$$\text{Conv}(Y \cup \mathcal{X}) = \text{chull}(Y, \text{Conv}(\mathcal{X}))$$

3.1 Hybrid Automata

An n -dimensional *Multi-Rectangular Hybrid Automaton* \mathcal{H} is a 6-tuple $(M, \Lambda, \text{flow}, E, \gamma, s_I)$ over \mathbb{R}^n . M denotes the finite set of discrete control modes. A *state* $s \in M \times \mathbb{R}^n$ of the automaton consists of a discrete mode and a continuous state. Here, $s_I = (\mu_I, x_I) \in M \times \mathbb{R}^n$ is the *initial state* and Λ is the finite set of *action labels*. $\text{flow}: M \rightarrow \mathcal{I}^n$ defines the *dynamics* of modes. When entering one, a random value is drawn from the respective interval for each dimension. $E \subseteq M \times \Lambda \times M$ is the finite set of edges containing discrete, labeled *transitions* between two modes. Lastly, $\gamma: E \rightarrow \mathcal{I}^n$ assigns guard conditions to edges. A transition can only be taken in a state if its continuous component lies within the rectangle. Figure 1a shows the easy-to-grasp visual representation of a hybrid automaton where each white rectangle constitutes a mode. The text inside it defines the dynamics as differential equations, edge labels state guard conditions.

Semantics. Hybrid automata allow for two kinds of transitions: control mode changes according to E and delays according to the *flow* of the current mode during which the system state evolves continuously. More formally, the semantics of a multi-rectangular hybrid automaton \mathcal{H} are defined based on valid *omniscient traces* through \mathcal{H} . An omniscient trace $\tilde{\pi} \in \mathbb{R}^n \times \mathbb{R}_{\geq 0} \times \mathbb{R}^n \times (E \times \mathbb{R}_{\geq 0} \times \mathbb{R}^n)^k$ with $\tilde{\pi} = x_0, \delta_0, x_1, e_1, \delta_1, \dots, x_k, e_k, \delta_k, x_{k+1}$ is valid for an automaton $(\tilde{\pi} \triangleright \mathcal{H})$ iff: a) the trace starts in the initial state of \mathcal{H} , i.e., $x_I = x_0$. b) the first discrete transition starts in the initial mode, so $e_1 = (\mu_I, \lambda, \mu)$ for some λ and μ . c) all guards are satisfied: $\forall 1 \leq i \leq k: x_i \in \gamma(e_i)$. d) all delay transitions are valid, i.e., for $0 \leq i \leq k$ and $e_{i+1} = (\mu_s, \lambda, \mu_t)$, the state changes according to the flow: $x_{i+1} \in (x_i + \delta_i \cdot \text{flow}(\mu_s))$. The language of an automaton is the set of valid traces: $\mathcal{L}(\mathcal{H}) = \{\tilde{\pi} \mid \tilde{\pi} \triangleright \mathcal{H}\}$.

Definition 3.3 (Observable Traces). An observable trace π is an omniscient trace stripped of its information regarding source and target modes: $\pi \in \mathbb{R}^n \times \mathbb{R}_{\geq 0} \times \mathbb{R}^n \times (\Lambda \times \mathbb{R}_{\geq 0} \times \mathbb{R}^n)^k$.

For the remainder of the paper, unless stated otherwise, a trace refers to an observable trace and we only consider finite languages.

Notation. Any automaton with decoration such as \mathcal{H}^+ will be implicitly destructured into its components with the same decoration, e.g. M^+ denotes the modes of \mathcal{H}^+ . $(x)_k$ denotes the k th component of the n -dimensional vector x for $0 < k \leq n$. For a finite set A , $|A|$ denotes the cardinality of A . The length $|\pi|$ of a trace $\pi \in \mathbb{R}^n \times \mathbb{R}_{\geq 0} \times \mathbb{R}^n \times (\Lambda \times \mathbb{R}_{\geq 0} \times \mathbb{R}^n)^k$ is the number of timed transitions occurring in it, i.e., $k + 1$. A trace of length $k + 1$ is implicitly destructured into the following components: $\pi = x_0^\pi, \delta_0^\pi, x_1^\pi, e_1^\pi, \delta_1^\pi, \dots, x_k^\pi, e_k^\pi, \delta_k^\pi, x_{k+1}^\pi$. Moreover, mode μ is a member of an omniscient trace if it reaches the mode at least once: $\mu \in \pi \iff \exists i: e_i^\pi = (\mu_1, \lambda, \mu_2)$ with $\mu \in \{\mu_1, \mu_2\}$. A *step* of a trace is the combination of a delay and a discrete transition. Further, let Π be a sequence of traces in arbitrary order. Then, π_i denotes the i th entry of the sequence with $i \leq |\Pi|$.

Bisimulation. Discrete bisimulation on two (hybrid) automata is defined conventionally by disregarding any continuous behavior or behavior not shared among the automata.

Definition 3.4 (Discrete Bisimulation). Two modes of μ_1, μ_2 of two automata $\mathcal{H}_1, \mathcal{H}_2$ are discretely bisimilar $\mu_1 \approx \mu_2$ iff for all transition labels $\lambda \in \Lambda_1 \cap \Lambda_2$:

$$(\mu_1, \lambda, \mu'_1) \in E_1 \implies \exists \mu'_2: (\mu_2, \lambda, \mu'_2) \in E_2 \wedge \mu'_1 \approx \mu'_2 \text{ and}$$

$$(\mu_2, \lambda, \mu'_2) \in E_2 \implies \exists \mu'_1: (\mu_1, \lambda, \mu'_1) \in E_1 \wedge \mu'_2 \approx \mu'_1$$

Further, the two automata are discretely bisimilar $\mathcal{H}_1 \approx \mathcal{H}_2$ iff $\mu_1^1 \approx \mu_1^2$.

4 CONSTRUCTING CONSERVATIVE AUTOMATA

The construction proceeds in three steps: First, it extracts information from the specification to obtain a finite state machine \mathcal{A}^Φ and a table mapping discrete control mode changes to conditions for undergoing such a change. The automaton is a coarse abstraction of the underlying system. The second step transforms it into a hybrid automaton \mathcal{H}^+ and iteratively refines it by extracting information regarding the continuous behavior from the input traces. By design, the refinement overshoots its goals, resulting in an abstraction that is too fine. As a remedy, the third step merges parts of the automaton to construct a conservative automaton. The alternation in coarseness has the effect that the resulting automaton is an over-approximation without being overly permissive.

4.1 Extracting Discrete Information from the Specification

The requirements on the system change depending on its current state. For example, during the landing of an airplane, the landing gear must be lowered whereas it is required to be retracted when on traveling altitude. Hence, the specification needs to keep track of relevant parts of the system state to impose the proper restrictions. This process of keeping track induces an abstract state machine that lacks any information on the continuous dynamics since the monitor solely relies on external input data such as sensor readings. Each abstract state may summarize several concrete modes of

the actual system. In the plane example, the requirements on the abstract mode “in full flight” apply to both the control mode “no wind” and “tail wind” even though they have different continuous dynamics. By assumption, the contrary is false: a change of requirements on the system is always accompanied by a change in concrete modes. Intuitively, a change of requirements is strongly linked to an action or reaction of the system: approaching a geofence imposes new constraints and hence prompts a reactionary change of course to satisfy them. A formalization of these assumptions follows in Section 5.1.

The extraction of the abstract automaton varies depending on the specification language. This paper uses the RTLOLA [9, 17] monitoring framework since it was designed for and integrated into safety-critical cyber-physical systems [1, 6, 8]. An RTLOLA specification consists of input streams representing data the monitor receives from the system, output streams and triggers. With output streams the specifier declares how to process input data with the goal of analyzing the state of the system. Lastly, triggers define conditions on output streams. The satisfaction of a trigger condition prompts the monitor to emit a message to the system, informing it about a violation of a safety requirement or the detection of a phase change.

A specification can keep track of the current set of requirements imposed on the system by using an output stream μ^Φ . The value of μ^Φ indicates in which abstract state the system is. Assume there are two abstract states μ_1^Φ and μ_2^Φ , and a state transition occurs under some condition φ . Then, the μ^Φ stream has the following shape:

| **output** $\mu^\Phi := \text{if } \mu^\Phi = \mu_1^\Phi \wedge \varphi \text{ then } \mu_2^\Phi \text{ else last}(\mu^\Phi)$

Here, $\text{last}(\mu^\Phi)$ provides the last value of the μ^Φ . For more possible abstract states and transitions, the conditional statement can be extended accordingly. In addition, a state change is accompanied by a respective trigger:

| **trigger** $\text{last}(\mu^\Phi) = \mu_1^\Phi \wedge \mu^\Phi = \mu_2^\Phi \wedge \varphi \text{ "}\mu_1^\Phi \rightarrow \mu_2^\Phi \text{ with } \lambda\text{"}$

The trigger checks for a change in μ^Φ from μ_1^Φ to μ_2^Φ and emits this information coupled with the name λ of the respective transition.

An analysis of the output stream and trigger declarations yields two artifacts:

Definition 4.1 (Specification Automaton). Given a specification Φ , $\mathcal{A}^\Phi = (V^\Phi, E^\Phi, v_1^\Phi)$ is the *abstract specification automaton* of Φ and $\Gamma^\Phi = (V^\Phi \times \lambda \times V^\Phi) \rightarrow \mathcal{I}^n$ is the *guard condition table* of Φ .

In the construction of \mathcal{A}^Φ , V^Φ and v_1^Φ are the domain and initial value of μ^Φ , respectively. Then, for each trigger as the one stated before, E^Φ contains the edge $(\mu_1^\Phi, \lambda, \mu_2^\Phi)$ and $\Gamma^\Phi(e) = \varphi$. Note that the following assumes \mathcal{A}^Φ to be free of unreachable states and related edges. This is the case in sensible specifications and can easily be enforced by pruning the respective parts of the graph.

4.2 Extracting Continuous Information from Traces

While the specification provides information about the system’s *discrete* structure, the traces reveal how the *continuous* state of the system evolves over time. They also reveal mode changes within a single abstract state. This information allows for transforming \mathcal{A}^Φ into a more fine-grained automaton with annotated dynamics in

each mode. For this, the transformation iteratively constructs an automaton \mathcal{H}^+ , processing each position of all traces in separation. This requires to keep track of two maps: a concrete mode-map $\psi: \Pi \rightarrow M$ that maps each trace to the mode of the constructed automaton in which it currently resides, and an abstract mode-map $\alpha: \mu \rightarrow V^\Phi$ mapping each concrete mode to an abstract one in the specification automaton \mathcal{A}^Φ .

Definition 4.2 (Construction Initialization). The construction starts with a quasi-empty hybrid automaton \mathcal{H}_1^+ that is structurally similar to \mathcal{A}^Φ , a concrete mode-map ψ_1 and an abstract mode-map α_1 defined as:

$$\begin{aligned} M_1 &= \{\mu_I\} & \Lambda_1 &= \emptyset & E_1 &= \emptyset & \gamma_1(e) &= \mathbb{1} & \psi_1(\pi) &= \mu_I \\ \alpha_1(\mu_I) &= v_I^\Phi & \text{flow}_1(\mu) &= \prod_{\pi_i \in \Pi} \text{solve}(x_0^{\pi_i}, x_1^{\pi_i}, \delta_0^\pi) \end{aligned}$$

Here, $\mathbb{1}$ denotes the neutral element with respect to the multiplication of intervals. Moreover, the solve-function computes the singular interval representing the linear dynamics exhibited by a delay transition:

$$\text{solve}(x, x', \delta) = [(x' - x)\delta^{-1}, (x' - x)\delta^{-1}]$$

Thus, \mathcal{H}_1^+ already incorporates the information of each trace regarding their first delay transition.

After the initialization, the procedure successively incorporates information contained in further positions of the traces.

Definition 4.3 (Construction Step). Given the automaton \mathcal{H}_k^+ , concrete mode-map ψ_k , and abstract mode-map α_k from the previous construction step. Consider the k -th step of each input trace, i.e., $x_k^{\pi_i}$, $\lambda_k^{\pi_i}$, $\delta_k^{\pi_i}$, and $x_{k+1}^{\pi_i}$ for all $0 < i \leq |\Pi|$. The k th step of the construction produces \mathcal{H}_{k+1}^+ , ψ_{k+1} , and α_{k+1} .

For brevity, given $e = (\mu_1, \lambda, \mu_2)$, let $\alpha_k(e) = (\alpha_k(\mu_1), \lambda, \alpha_k(\mu_2))$. Moreover, let $\Phi(\pi[.k], e)$ be true iff the edge e of the specification automaton was derived from a trigger for which the monitor reports a violation for the trace π up to the k th step. Lastly, $\mu_{i,k}$ are fresh modes.

$$M_{k+1} = M_k \cup \bigcup_i \{\mu_{i,k}\} \quad \Lambda_{k+1} = \Lambda_k \cup \bigcup_i \{\lambda_k^{\pi_i}\}$$

$$E_{k+1} = E_k \cup \bigcup_i \{(\psi_k(\pi_i), \lambda_k^{\pi_i}, \mu_{i,k})\} \quad \psi_{k+1}(\pi_i) = \mu_{i,k}$$

$$\gamma_{k+1}(e) = \begin{cases} \gamma_k(e) & \text{if } e \in E_k \\ \Gamma^\Phi(\alpha_{k+1}(e)) & \text{otherwise} \end{cases}$$

$$\text{flow}_{k+1}(\mu) = \begin{cases} \text{solve}(x_k^{\pi_i}, x_{k+1}^{\pi_i}, \delta_k^{\pi_i}) & \text{if } \mu = \mu_{i,k} \\ \text{flow}_k(\mu) & \text{otherwise} \end{cases}$$

$$\alpha_{k+1}(\mu) = \begin{cases} \mu^\alpha & \text{if } \exists \pi: \Phi(\pi[.k], (\alpha_k(\psi_k(\pi)), \lambda_k^\pi, \mu^\alpha)) \\ \alpha_k(\mu) & \text{if } \mu \in M_k \\ \alpha_k(\psi_k(\pi_i)) & \text{if } \mu = \mu_{i,k} \end{cases}$$

Intuitively, for each position of each trace the construction (i) adds a new mode with the dynamics exhibited by the delay transition, (ii) adds a new edge from μ to μ' for the discrete transition, and (iii) updates the mode maps accordingly. The latter means that if the transition was accompanied by a step in \mathcal{A}^Φ , α maps the μ' to the respective abstract mode and looks up the guard from the

specification. Otherwise, it maps μ' to the same abstract state as μ with a vacuous guard indicating a lack of information.

4.3 Merging Modes

Evidently, following the procedure yields an automaton with at most $|\pi| \cdot |\Pi|$ modes arranged as a tree as can be seen in Figure 1c. It transformed the overly coarse specification automaton into an overly fine hybrid automaton. To find the sweet spot between both extremes, the next construction step merges modes within an abstract state provided they are sufficiently similar. Suppose some relation $\sim_{\exists \lambda}$ captures this notion of similarity. Then, intuitively, the construction deems any two modes $\mu \not\sim_{\exists \lambda} \mu'$ sufficiently *dissimilar* such that they must represent different modes in the original system. For this, let \sim_α denote the relation induced by α for a constructed hybrid automaton, $\mu_1 \sim_\alpha \mu_2$ indicates that both modes refine the same abstract state, i.e., $\alpha(\mu_1) = \alpha(\mu_2)$.

Definition 4.4 (Action Similarity). For a constructed hybrid automaton \mathcal{H}^+ , two modes $\mu_1, \mu_2 \in M^+$ are action-similar if they share some discrete characteristics and reside in the same abstract state of the specification. Assume there are some modes $\mu'_1, \mu'_2 \in M^+$ and action $\lambda \in \Lambda^+$ a

$$\begin{aligned} \mu_1 \sim_{\exists \lambda} \mu_2 &\iff \alpha(\mu_1) = \alpha(\mu_2) \wedge \\ &(\{\mu'_1, \lambda, \mu_1\}, \{\mu'_2, \lambda, \mu_2\}) \subseteq E^+ \vee \\ &(\{\mu_1, \lambda, \mu'_1\}, \{\mu_2, \lambda, \mu'_2\}) \subseteq E^+ \end{aligned}$$

Note that by construction \sim_α is coarser than $\sim_{\exists \lambda}$.

Terminal modes need further attention: consider the automaton in Figure 1a. There are two identical traces in the language of the automaton starting in TAKEOFF and traversing STRAIGHT, but ending in different LANDING modes. Based on these traces the construction cannot distinguish the two terminal modes, since the difference in modes is unobservable. In fact, there is no finite set of traces for which they can be distinguished with certainty. This forces the construction to merge them as can be seen in Figure 1b.

Definition 4.5 (Terminal and Merge Similarity). Two terminal modes are *terminal-similar* iff they reside in the same abstract state.

$$\mu_1 \sim_\perp \mu_2 \iff \text{outdeg}(\mu_1) = \text{outdeg}(\mu_2) = 0 \wedge \alpha(\mu_1) = \alpha(\mu_2)$$

Two modes are *merge-similar* iff they are either action-similar or terminal-similar: $\sim_M = \sim_{\exists \lambda} \cup \sim_\perp$

Merge Operation. The merge operation now minimizes the automaton with respect to \sim_M by building the quotient automaton. Formally, a merge operates on an equivalence relation \approx over the set of modes where each equivalence class $\zeta \subseteq M$ will be replaced by a single representative $\llbracket \zeta \rrbracket_\approx$. By slight abuse of notation let $\llbracket \mu \rrbracket_\approx = \llbracket \zeta \rrbracket_\approx$ for $\mu \in \zeta$. Moreover, if context permits, the subscript may be omitted. The representative conserves the language of each mode contained in ζ by retaining discrete transitions and computing the convex hull for its continuous components.

Definition 4.6 (Merge Automaton). Merging an automaton \mathcal{H} with respect to an equivalence relation \approx yields an automaton $\mathcal{H} \downarrow_\approx$ where all elements of an equivalence class get merged into a single element retaining its language.

Algorithm 1 Construct Conservative Hybrid Automaton

Require: Specification Φ , Traces Π

- 1: Extract $\mathcal{A}^\Phi, \Gamma^\Phi$ from Φ ▷ Definition 4.1
- 2: Construct $\mathcal{H}_1^+, \psi_1, \alpha_1$ from Π and Γ^Φ ▷ Definition 4.2
- 3: **for** k from 1 to $|\pi|$ **do** ▷ Definition 4.3
- 4: Update to $\mathcal{H}_k^+, \psi_k, \alpha_k$
- 5: **end for**
- 6: Compute the action-similarity $\sim_{\exists\lambda}$ ▷ Definition 4.4
- 7: Compute the merge-similarity \sim_M ▷ Definition 4.5
- 8: Compute $\mathcal{H}^+ = \mathcal{H}_{|\Pi|}^+ \downarrow_{\sim_M}$ ▷ Definition 4.6

$$\begin{aligned}
M \downarrow_{\approx} &= \{ \llbracket \mu \rrbracket \mid \mu \in M \} & s_I \downarrow_{\approx} &= (\llbracket \mu_I \rrbracket , x_I) \\
flow \downarrow_{\approx} (\llbracket \zeta \rrbracket) &= Conv(\bigcup_{\mu \in \zeta} \{ flow(\mu) \}) & \psi \downarrow_{\approx} (\pi) &= \llbracket \psi(\pi) \rrbracket \\
E \downarrow_{\approx} &= \{ (\llbracket \mu_1 \rrbracket , \lambda , \llbracket \mu_2 \rrbracket) \mid (\mu_1, \lambda, \mu_2) \in E \} & \alpha \downarrow_{\approx} (\llbracket \mu \rrbracket) &= \alpha(\mu) \\
\Lambda \downarrow_{\approx} &= \{ \lambda \mid \exists e \in E \downarrow_{\approx} : e = (\llbracket \zeta_1 \rrbracket , \lambda , \llbracket \zeta_2 \rrbracket) \} \\
\gamma \downarrow_{\approx} ((\zeta_1 , \lambda , \zeta_2)) &= Conv(\{ (\mu_1, \lambda, \mu_2) \mid \mu_1 \in \zeta_1 \wedge \mu_2 \in \zeta_2 \})
\end{aligned}$$

4.4 Putting it Together: Construction Algorithm

The overall construction algorithm now proceeds as outlined in Algorithm 1: First, the procedure extracts information from the specification, constructs the initial automaton and refines it successively by iterating over the traces. After processing all traces completely, the procedure computes and applies the merges with respect to action- and terminal-similarity.

Time Complexity. The construction process consists of three phases: extraction, construction and merging. Recall that the dimensionality, i.e., the number of continuous state variables is n . The first phase scales linearly in the size of the specification $O(|\Phi|)$. The second phase construct an automaton with a single mode per step of any trace. Its size and the running time of the construction scales linearly with the number and length of traces as it creates a new mode per step of any trace. It is also linear in the dimension since the dynamics of each dimension have to be computed separately per mode. Hence, the complexity is in $O(n \cdot |\pi| \cdot |\Pi|)$. Lastly, the complexity of the last phase depends on the complexity of a single merge, which is linear in the dimension, and the number of merges. The latter is quadratic in the size of $\mathcal{H}_{|\Pi|}^+$, which in turn is linear in the number and length of traces: $O(n \cdot |\mathcal{H}_{|\Pi|}^+|^2) = O(n \cdot |\pi|^2 \cdot |\Pi|^2)$. This, however, only describes the worst case. The procedure compares each mode against each other with respect to \sim_M . In the best case, all elements of an equivalence class are identified successively by chance. In this case, the process is quadratic in the number of equivalence classes: $\Omega(n \cdot |\sim_M|^2)$. Here, $|\sim_M|$ denotes the number of equivalence classes induced by \sim_M with $|\mathcal{A}^\Phi| \leq |\sim_M| = |M^+| \leq |M_{|\Pi|}^+|$. In conclusion, the overall asymptotic running time is dominated by the merge procedure: $O(n \cdot |\pi|^2 \cdot |\Pi|^2)$

5 CORRECTNESS OF CONSTRUCTION

The validation of the construction requires a proof that the constructed automaton is — under certain assumptions on the input — indeed conservative. For this, a key criterion is that the automaton over-approximates the discrete and continuous behavior of the original system when projected down to the parts that contributed to the inputs. Evidently, if the original system encompasses parts that were neither reflected in the specification, nor traversed in the input traces, the constructed automaton cannot reconstruct it.

Hence, this section first formalizes requirements on the input data. Then, a definition of projection automata enables proving that the constructed automaton subsumes the language of the projected original system.

5.1 Requirements on Input Data

The construction of the conservative automaton relies on the quality of the input traces and specification. Hence, they need to satisfy three criteria: (i) the specification must be an abstraction of the real system, (ii) its trigger conditions must be at least as restrictive as the respective conditions on mode changes, and (iii) the trace set needs to traverse every control mode of the system sufficiently often to capture the discrete behavior.

Definition 5.1 (Adequacy of Input Data). A specification Φ and trace set Π are *adequate* for a hybrid automaton \mathcal{H} iff they satisfy three criteria.

- (1) The specification induces a coarser automaton \mathcal{A}^Φ than the original, i.e., $\exists \sim_\Phi : \mathcal{H} \downarrow_{\sim_\Phi} \approx \mathcal{A}^\Phi$.
- (2) Assume $V^\Phi = \{ \llbracket \zeta \rrbracket \mid \zeta \in \mathcal{P} \}$. For any discrete transition that is both in \mathcal{H} and \mathcal{A}^Φ , the specification contains a mode change condition that is at least as permissive as the guard of the respective transition in \mathcal{H} . Formally, let $\mu_1^\Phi \not\sim_\Phi \mu_2^\Phi$ and $\mu_1^\Phi \approx \mu_1 \wedge \mu_2^\Phi \approx \mu_2$. Then, for all $(\mu_1^\Phi, \lambda, \mu_2^\Phi) \in E^\Phi$ and $e = (\llbracket \mu_1 \rrbracket_{\sim_\Phi} , \lambda , \llbracket \mu_2 \rrbracket_{\sim_\Phi}) \in E \downarrow_{\sim_\Phi}$:

$$\gamma(\mu, \lambda, \mu) \implies \Gamma^\Phi(e)$$

- (3) For every mode μ in \mathcal{H} , let $a_\mu = \text{indeg}(\mu) + \text{outdeg}(\mu)$ be the number of input and output actions of μ in Π . The trace set needs to contain more than $a_\mu(a_\mu - 1)/2$ traversals through μ . Here, a trace π traverses through a mode if its omniscient counterpart $\tilde{\pi}$ contains two subsequent edges first ending and then starting from μ .

$$\forall \mu \in M_{\mathcal{H}} : | \{ (e_i^{\tilde{\pi}}, e_{i+1}^{\tilde{\pi}}) \mid \exists \tilde{\pi} \in \tilde{\Pi}, i < |\tilde{\pi}|, \lambda, \lambda' \in \Lambda_{\mathcal{H}} :$$

$$(e_i^{\tilde{\pi}}, e_{i+1}^{\tilde{\pi}}) = ((\mu_1, \lambda, \mu), (\mu, \lambda', \mu_2)) \} | > \frac{(a_\mu)(a_\mu - 1)}{2}$$

Evidently, these criteria depend on the original hybrid automaton, which seemingly contradicts the premise of this paper regarding the unavailability of such a model. Nevertheless, the criteria are designed in a way that they are either satisfied naturally or can be satisfied without access to all formal details of the system.

First, consider 5.1.1 and 5.1.2. These criteria restrict the specification, which was hand-crafted for the underlying system. Here, a reasonable specification summarizes control modes that are subject to the same requirements; at the same time, the specification needs to capture changes in the abstract state precisely to impose the

correct sub-specification on the system. Thus, even without perfect knowledge of the inner workings and dynamics of the system, the first two criteria can be ensured. Consider the third criterion, which is concerned with the trace set. A thorough testing process demands that all discrete paths¹ through the system are tested at least once. Moreover, the system has a fixed control interface, represented by Λ . As a result, it is reasonable to assume that the number of times each control mode is traversed during the development exceeds the threshold required by Criterion 5.1.3. This again does not rely on knowledge about the exact mode structure nor dynamics of the underlying system.

The exact threshold for the third criterion seems arbitrary but is anchored in graph theory, the impact of which can be seen in the next lemma.

LEMMA 5.2 (TRACE CONNECTIVITY). *Let Φ and Π be adequate for \mathcal{H} . For any mode μ in \mathcal{H} with incoming edge label λ_i and outgoing edge label λ_o , there is a mode μ' in \mathcal{H}^+ with the very same edge labels and $\alpha(\mu) = \alpha(\mu')$.*

PROOF. By reduction on the graph connectivity problem. Let $\mathcal{G}(\mu, \Pi) = (V, E)$ be a graph where V is the set of labels of incoming or outgoing edges of μ in \mathcal{H} . E connects two action labels λ, λ' if there is a trace in Π that reaches μ via λ and leaves via λ' . The solution of the graph connectivity problem states that λ and λ' are necessarily connected if $|E|$ exceeds $|V| \cdot (|V| - 1)/2$. This threshold corresponds to Criterion 5.1.3. Recall that $\sim_{\exists\lambda}$ relates all modes with at least one common incoming or outgoing edge label. Thus, since $\sim_{\exists\lambda} \subseteq \sim_M$, all respective modes are merged in \mathcal{H}^+ and by Definition 4.6, the resulting mode $\llbracket \mu \rrbracket_{\sim_M}$ retains these transitions. Lastly, \sim_M refines \sim_Φ , hence $\alpha(\mu) = \alpha(\llbracket \mu \rrbracket_{\sim_M})$, which concludes the proof. \square

5.2 Projection Automata

The assessment of the quality of the reconstruction depends on the projection of the original system onto the set of traces. This first requires a definition of projections on automata.

Definition 5.3 (Projection Automata). The projection of an automaton \mathcal{H} down to a set of omniscient traces $\bar{\Pi}$ is an automaton $\mathcal{H}|_{\bar{\Pi}}$ with the following constituents.

$$M|_{\bar{\Pi}} = \bigcup_{\bar{\pi} \in \bar{\Pi}} \bigcup_{i \leq |\bar{\pi}|} \left\{ \mu_i^{\bar{\pi}} \mid \mu_i^{\bar{\pi}} \in M \right\}$$

$$\Lambda|_{\bar{\Pi}} = \bigcup_{\bar{\pi} \in \bar{\Pi}} \bigcup_{i < |\bar{\pi}|} \left\{ \lambda_i^{\bar{\pi}} \mid \lambda_i^{\bar{\pi}} \in \Lambda \right\}$$

$$E|_{\bar{\Pi}} = \{ (\mu, \lambda, \mu') \in E \mid \mu, \mu' \in M|_{\bar{\Pi}} \wedge \lambda \in \Lambda|_{\bar{\Pi}} \}$$

$$\gamma(e)|_{\bar{\Pi}} = [v_e^{\min}, v_e^{\max}] \quad \text{flow}(\mu)|_{\bar{\Pi}} = [v_\mu^{\min}, v_\mu^{\max}] \quad s_I|_{\bar{\Pi}} = s_I$$

Here, for $\varphi \in \{\min, \max\}$, the min and max values for guards and flows are:

$$v_e^\varphi = \varphi \{ x \mid \exists \bar{\pi}, \exists i < |\bar{\pi}| : x = x_i^{\bar{\pi}} \wedge e = e_i^{\bar{\pi}} \}$$

$$v_\mu^\varphi = \varphi \{ f \mid \exists \bar{\pi}, \exists i < |\bar{\pi}| : e_i^{\bar{\pi}} = (\mu', \lambda, \mu) \wedge x_i^{\bar{\pi}} + \delta_i^{\bar{\pi}} f = x_{i+1}^{\bar{\pi}} \}$$

¹This is the case for terminating systems. In the presence of infinite discrete paths, a threshold on the length of executions is usually imposed.

Intuitively, the projection strips the automaton of any information not reflected in the set of traces. This reduces the sets of modes, edges, and transition labels. By definition, the initial state occurs in all traces and thus remains the same. Guards and flows are reduced to the maximum and minimum value exhibited by some trace.

Note that the projection automaton $\mathcal{H}|_{\bar{\Pi}}$ is not meant to be constructed at any point; it merely serves as theoretical point of reference for the quality of the construction. It is easy to see that in general the projection reduces the expressiveness of an automaton, i.e., $\mathcal{L}(\mathcal{H}) \supseteq \mathcal{L}(\mathcal{H}|_{\bar{\Pi}})$. This, however, is not necessarily the case as the following theorem shows.

THEOREM 5.4 (PERFECT PROJECTION). *For any hybrid automaton \mathcal{H} there is a finite set of traces for which the projection onto these traces yields the identity, i.e., $\exists \bar{\Pi}^* \subseteq \mathcal{L}(\mathcal{H}) : \Pi^* \text{ finite} \wedge \mathcal{L}(\mathcal{H}|_{\bar{\Pi}^*}) = \mathcal{L}(\mathcal{H})$.*

The proof of Theorem 5.4 can be found in Appendix B. Note that the language equality cannot be extended to identical or isomorphic automata since \mathcal{H} can contain unreachable modes that are not reflected in its language and thus not in any trace. The theorem emphasizes the generality of the conservative construction: For an appropriate trace set, the projection of an automaton perfectly resembles the original system. Since the constructed automaton is conservative with respect to this very projection, it is also conservative with respect to the original system. This is independent of the exact structure of the underlying system.

5.3 Construction Guarantees

The first observations are that application of a merge and iterations of the construction do not reduce the language of an automaton.

LEMMA 5.5 (LOSSLESS MERGE). *Given a constructed hybrid automaton \mathcal{H}^+ and an equivalence relation \approx , merging \mathcal{H}^+ with respect to \approx yields a more permissive automaton, i.e., $\mathcal{L}(\mathcal{H}^+) \subseteq \mathcal{L}(\mathcal{H}^+ \downarrow_{\approx})$.*

The proof of the lossless merge can be found in Appendix B. After showing the property of a merged automaton, we state that our construction is lossless.

LEMMA 5.6 (LOSSLESS CONSTRUCTION). *Given a set of traces Π and specification Φ . For any iterations i and j , if $j \geq i$, then the set of edges, the flow, and the transition guards only grow over the iterations:*

$$E_i \subseteq E_j \wedge \text{flow}_i \subseteq \text{flow}_j \wedge \forall e \in E_i : \gamma_i(e) \subseteq \gamma_j(e)$$

PROOF. This lemma follows directly from the construction step (Definition 4.3). \square

This suffices to prove that the language of the constructed automaton at least includes all input traces.

THEOREM 5.7 (INPUT TRACE INCLUSION). *Given an adequate set of traces Π and specification Φ , the language of a constructed automaton \mathcal{H}^+ subsumes Π , i.e., $\Pi \subseteq \mathcal{L}(\mathcal{H}^+)$.*

The proof of Theorem 5.7 can be found in Appendix B. A stronger classification of the language of \mathcal{H}^+ requires some insight into its discrete structure in relation to the projection automaton of the original system. Specifically, $\mathcal{H}|_{\bar{\Pi}}$ has a finer discrete structure than \mathcal{H}^+ .

LEMMA 5.8 (DISCRETE REFINEMENT). *For an adequate specification Φ and set of traces Π for a hybrid automaton \mathcal{H} , the reconstruction \mathcal{H}^+ is coarser than the projection of \mathcal{H} onto Π , i.e., $\exists \sim_+ : \mathcal{H}^+ \downarrow_{\sim_+} \approx \mathcal{H}|_{\Pi}$.*

PROOF. The proof proceeds in two steps. First, for an arbitrary trace π through $\mathcal{H}|_{\Pi}$ it generates a trace π' through \mathcal{H}^+ . Second, it constructs the equivalence relation \sim_+ based on these trace pairs.

STEP 1: For a given $\tilde{\pi} \in \mathcal{L}(\mathcal{H}|_{\Pi})$, the proof inductively constructs $\tilde{\pi}'$ with $\tilde{\pi}' \in \mathcal{L}(\mathcal{H}^+)$ such that the observable traces for $\tilde{\pi}$ and $\tilde{\pi}'$ are equal. Moreover, for any step i : $\psi(\tilde{\pi}[0..i]) = \psi(\tilde{\pi}'[0..i])$. The induction base is trivial since both traces originate in the fixed initial state, which corresponds to the initial state of the specification automaton. INDUCTION STEP: Suppose the observable traces for $\tilde{\pi}[0..i]$ and $\tilde{\pi}'[0..i]$ are equal and $\psi(\tilde{\pi}[0..i]) = \psi(\tilde{\pi}'[0..i])$. Suppose further the last action label was λ and the next is λ' .

Since the label combination λ, λ' appears in $\tilde{\pi}$, it is also present in a mode μ in \mathcal{H}^+ by Lemma 5.2 with $\alpha(\tilde{\pi}[0..i]) = \alpha(\tilde{\pi}'[0..i]) = \alpha(\mu')$. Further, $\psi(\tilde{\pi}[0..i])$ has an incoming λ label, so by Definition 4.4, $\psi(\tilde{\pi}[0..i]) \sim_{\exists \lambda} \mu'$ due to their shared action label. Hence, by Lemma 5.5, $\alpha(\tilde{\pi}[0..1])$ has an outgoing edge with label λ' , which proves that the discrete edge is present.

Now, it suffices to show that $\psi(\tilde{\pi}[0..i]) = \psi(\tilde{\pi}'[0..i])$. There are possibilities: *both* traces take a transition in terms of ψ or neither one does. This follows from Criterion 5.1.2 and Lemma 5.6 stating that both automata are refinements of \mathcal{A}^Φ . If both take such a transition, the claim follows because \mathcal{A}^Φ is deterministic and both automata refine \mathcal{A}^Φ . If neither one does, both remain in the same state in \mathcal{A}^Φ ; the claim follows from the induction hypothesis. STEP 2: The trace pairs $\tilde{\pi}, \tilde{\pi}'$ induce an equivalence relation:

$$\sim_+ = \{(\psi(\tilde{\pi}[0..i]), \psi(\tilde{\pi}'[0..i])) \mid i \in \mathbb{N}\}$$

This relation is a witness for the claim that $\mathcal{H}|_{\Pi}$ is a refinement of \mathcal{H}^+ due to the trace inclusion proven in Step 1. \square

Note that the refinement can be a true refinement since the merge criterion might falsely relate modes that are distinct in $\mathcal{H}|_{\Pi}$ but share some discrete behavior, as discussed before.

COROLLARY 5.9. \sim_+ is finer than \sim_M .

THEOREM 5.10 (CONSERVATIVE CONSTRUCTION). *Let \mathcal{H} be a hybrid automaton with an adequate set of traces Π and specification Φ . The constructed automaton \mathcal{H}^+ over-approximates the language of the projection automaton $\mathcal{H}|_{\Pi} : \mathcal{L}(\mathcal{H}|_{\Pi}) \subseteq \mathcal{L}(\mathcal{H}^+)$.*

PROOF. Let $\tilde{\pi} \in \mathcal{L}(\mathcal{H}|_{\Pi})$. The proof constructs a trace $\pi \in \mathcal{L}(\mathcal{H}^+)$ such that π is an observable counterpart for $\tilde{\pi}$. The initial real-valued state of π is $x_0^{\tilde{\pi}}$ with $\psi(\pi[1]) = \mu_1^{\tilde{\pi}}$. By construction, $\mu_1^{\tilde{\pi}} \sim_+ \mu_1|_{\Pi} = \mu_0^{\tilde{\pi}}$.

For the induction, consider a delay transition $x_i^{\tilde{\pi}}, \mu_i^{\tilde{\pi}}, \delta_i^{\tilde{\pi}}, x_{i+1}^{\tilde{\pi}}$ where $\mu_i^{\tilde{\pi}} \sim_+ \mu^+$ by Lemma 5.8. Let $x_i^{\tilde{\pi}} + f \delta_i^{\tilde{\pi}} = x_{i+1}^{\tilde{\pi}}$ for $f \in \mathbb{R}^n$. By definition of the projection automaton (Definition 5.3), Π contains traces π^\uparrow and π^\downarrow exhibiting the flow f^\uparrow and f^\downarrow at point $a^\uparrow \leq |\pi^\uparrow|$ and $a^\downarrow \leq |\pi^\downarrow|$, respectively, while traversing $\mu_i^{\tilde{\pi}}$ with $f^\downarrow \leq f \leq f^\uparrow$. By construction, there are modes $\mu^{a^\uparrow} \in M_{a^\uparrow}^+$ and $\mu^{a^\downarrow} \in M_{a^\downarrow}^+$ with $flow_{a^\uparrow}(\mu^{a^\uparrow}) = f^\uparrow$ and $flow_{a^\downarrow}(\mu^{a^\downarrow}) = f^\downarrow$. Lemmas 5.5 and 5.6

guarantee that further construction steps and merges retain this information. Moreover, Lemma 5.8 implies that $\mu^{a^\uparrow}, \mu^{a^\downarrow}$, and μ^+ are equal with respect to \sim_+ . By Corollary 5.9, they are also equal with respect to \sim_M . Thus, $flow(\mu^+) = Conv(\{flow(\mu) \mid \mu \in \zeta^+\})$ with $f^\downarrow, f^\uparrow, f \in flow(\mu^+)$. As a result, π may contain the subsequence $x_i^{\tilde{\pi}}, \delta_i^{\tilde{\pi}}, x_{i+1}^{\tilde{\pi}}$ representing the delay transition.

For discrete transitions, consider $e = (\mu_i^{\tilde{\pi}}, \lambda_i^{\tilde{\pi}}, \mu_{i+1}^{\tilde{\pi}})$. We show that $e^+ = (\mu_s^+, \lambda_i^{\tilde{\pi}}, \mu_t^+)$ is a valid transition assuming that $\mu_s^+ = \psi(\pi[0..i])$, i.e., the trace constructed so far ended in μ_s^+ . There are three cases:

CASE A) Both $\mu_i^{\tilde{\pi}} \sim_\Phi \mu_{i+1}^{\tilde{\pi}}$ and $\mu_s^+ \sim_\alpha \mu_t^+$. Intuitively, this means that both automata remain in the same state of the specification automaton. In this case, by construction: $\gamma_i(e^+) = \Gamma^\Phi(\alpha_i(e^+)) = \mathbb{1}$ and by Lemmas 5.5 and 5.6: $\gamma_i(e^+) \subseteq \gamma^+(e^+)$. Thus, the guard is satisfied trivially. The existence of the edge in the constructed automaton follows from Lemma 5.8.

CASE B) Neither $\mu_i^{\tilde{\pi}} \sim_\Phi \mu_{i+1}^{\tilde{\pi}}$ nor $\mu_s^+ \sim_\alpha \mu_t^+$. Intuitively, this means neither automaton remains in the same state of the specification automaton. In this case, $\gamma_i(e^+) = \Gamma^\Phi(\alpha_i(e^+))$. By Criterion 5.1.2 and Definition 5.3, we know that $\gamma|_{\Pi}(e) \implies \gamma(e)$ and $\gamma(e) \implies \Gamma^\Phi(\alpha_i(e^+))$. Again, by Lemma 5.6 and Lemma 5.5 we know $\gamma_i(e^+) \subseteq \gamma^+(e^+)$ and the existence of the edge in the constructed automaton follows from Lemma 5.8.

CASE C) Either $\mu_i^{\tilde{\pi}} \not\sim_\Phi \mu_{i+1}^{\tilde{\pi}}$ or $\mu_s^+ \not\sim_\alpha \mu_t^+$ but not both. This case is impossible for adequate specifications (Definition 5.1.1) and by the definition of \sim_Φ / \sim_α .

Thus, the discrete transition exists and is applicable in the reconstructed automaton. This concludes the proof. \square

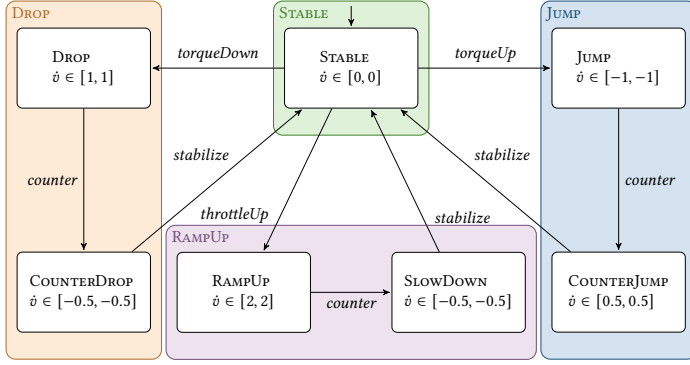
6 EXPERIMENTS

The empirical evaluation shows the scalability and precision of the approach presented in this paper. It is based on a prototype implementation in Rust² and the code is open source. All experiments were conducted on an Intel i5-7200u with 8GB RAM.

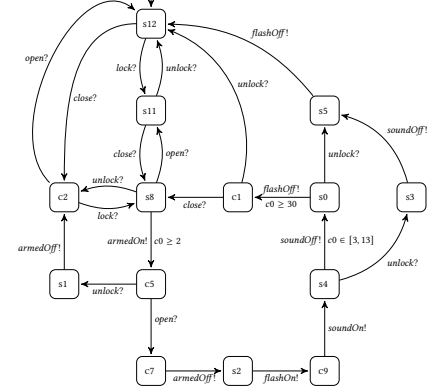
6.1 Aircraft System

As a first proof of concept, consider the running example from Figure 1a. For adequate input traces, the output will always be structurally equal with varying dynamics. This can be seen in Figure 1b, which shows the results of two construction processes. The dynamics in black are constructed from three hand-picked traces of length eight. Two of these traces travers all three TRAVEL modes, whereas the last one skips the course adjustment modes and loops in one of the LANDING modes instead. As can be seen, by picking the state values for the traces in such a way that they represent the extreme behavior, the reconstruction of the dynamics is perfect. Conversely, the constructed dynamics based on an adequate trace set of ten traces obtained by conducting random walks on the original system is shown in gray. The traces can be found in Appendix A. Evidently, the reconstruction closely resembles the original system both structurally and in terms of dynamics despite being based on a small set of random traces.

²<https://www.rust-lang.org>



(a) Hybrid automaton approximating the engine timing control system. A single trace of length ten enables perfect reconstruction. Colored states indicate the specification automaton.



(b) Timed automaton for the car alarm system. Perfect reconstruction requires seven traces of length twelve.

Figure 2: Example automata by Medhat et al. [14] (a) and Tappler et al. [20] (b).

6.2 Scalability

Recall the complexity of each step of the construction algorithm, i.e., extraction, construction, and merging, from Section 4.4. The extraction only requires a single pass over the specification and is thus negligible. The construction and merges depend on the dimensionality of the system and the number and length of traces. The merge also depends on the number of equivalence classes with respect to \sim_M in the best case, which is the size of the output automaton.

For this reason, the scalability evaluation considers exactly these three factors: dimensionality, number and length of traces, and output-size. To this end, it automatically generates an automaton with matching specification and adequate trace set. The automaton is shaped like a binary tree of variable depth d (scales the length of traces) where each of the $2^{d+1} - 1$ nodes is a control mode with dynamics of variable dimension (scales the dimensionality). The specification summarizes a variable number of modes with equal depth (scales the output size) and generates a variable number of adequate traces (scales the number of traces) enabling the respective merges.

Figure 3a and Figure 3b show the running time and memory consumption for varying sizes of the original automaton. The green line represents runs where \sim_M only equates identities, prohibiting any merges. For the blue line, two modes are equal if they have the same depth in the underlying automaton. The number and size of the traces required for an adequate trace set scales linearly with the size and the depth, respectively. Independent of the existence of merges, the running time lies below a second for automata with less than 2^{10} modes and increases steadily afterwards — as expected considering the asymptotic complexity. Even for automata with 2^{15} modes, the construction terminated after less than an hour (green line) or half an hour (blue line). The memory consumption behaves similarly, starting to rise significantly around 2^7 owing both to the increased number of traces stored in memory, and the resulting size

of \mathcal{H}_{Π}^+ . Note that the memory consumption almost exclusively stems from the construction process; merging only deallocates memory. In terms of running time, the lion’s share comes from merging due to the difference in time-complexity. At a size of 2^9 modes the running time of the construction process amounts to 3.061% and decreases to 0.015% for automata with 2^{15} modes.

The dimensionality impacts the running time to a lesser extent. Raising the dimension from 1,000 to 7,000 for an automaton size of 10^{10} increases the running time from around 4 s to 14 s. The limiting factor here is the memory consumption: each additional dimension increases the memory consumption of every guard condition, mode, and trace step. As a result, 10,000 dimensions requires around 5 GB of memory, which is also reflected in a relatively steep increase in running time to 96 s. Lastly the number of traces has an almost identical impact on both the running time and memory consumption. The scale of the impact lies in-between the one of the dimensionality and the size of the automaton. Raising the number of traces from 3,000 to 16,000 increases the running time roughly 40-fold. Detailed results for the dimensionality and number of traces can be found in Appendix C.

6.3 Comparison Against Other Approaches

The construction of a hybrid automaton requires the determination of both the discrete structure and continuous behavior. Regarding the former, the conservative construction relies on the information provided by the specification and refines abstract states according to trace information. This restricts revisions to a local level. In absence of such a specification, other approaches resort to learning algorithms. Medhat et al. [14] use a modification of Angluin’s L^* algorithm [5] to learn the discrete structure separately from the dynamics. Tappler et al. [20] on the other hand learn a timed automaton with genetic programming. Note that these automata alleviate the need to infer dynamics. We will evaluate the conservative construction against both of these approaches.

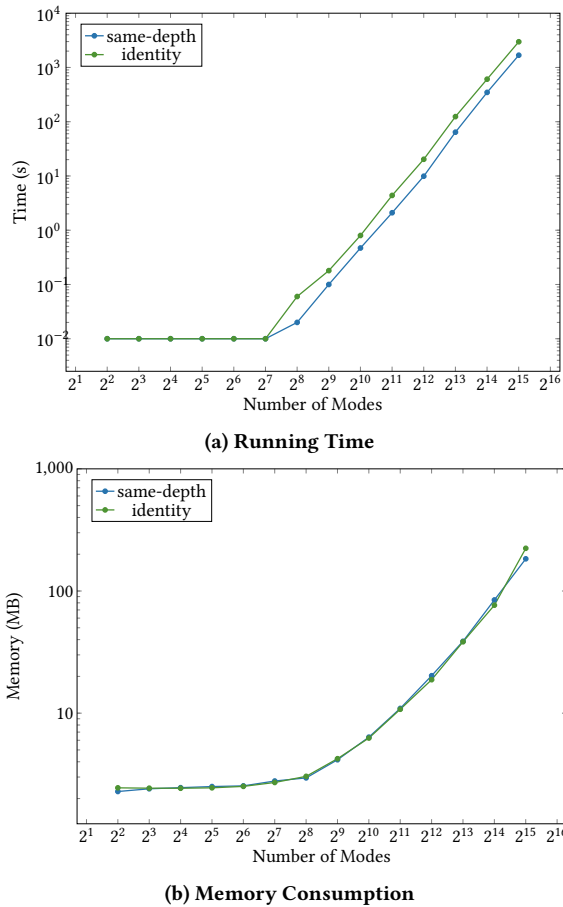


Figure 3: The results of the scalability analysis for different sizes of the original automaton and specifications enabling many (blue) or no (green) merges.

Anguin’s L^ and Clustering.* Medhat et al. [14] use an adaptation of L^* and clustering to identify the discrete and continuous behavior of a system, respectively. In their case study, they use a Simulink model of a closed-loop engine timing control system. Figure 2a shows an approximate representation of the system as hybrid model. Their construction uses eight traces to generate an automaton that resembles the underlying system for new traces up to an error of 2.6%. Note that a detailed comparison is exacerbated by a lack of information regarding the running time, memory consumption, and length of the input traces. The conservative construction can perfectly reconstruct the automaton with one hand-picked trace of length ten within less than 1 ms. When using random walks, an average of 35 traces of length fifteen suffices for the perfect reconstruction. In this example, the specification always summarizes modes belonging to a certain operation of the system, i.e., a drop, jump, ramp-up and the stable configuration.

Genetic Programming. Tappler et al. [20] use genetic programming to successively adapt a candidate automaton to encompass all input traces. As an example, they consider a timed automaton modeling a car alarm system (see Figure 2b). A sufficiently precise

reconstruction requires 2,000 randomly generated traces and took a mean of around 100 min. When using seven hand-selected traces, the conservative construction can perfectly reconstruct the system within less than 1 ms, disregarding resets. With random walks, an average of 35 traces of length fifteen is necessary for the perfect reconstruction. Note that the nature of the car alarm system does not allow for a meaningful summary of several modes in the specification. However, in terms of performance, the specification is irrelevant owing to the small size of the system.

7 RELATED WORK

The theory of hybrid automata was first studied by Henzinger [11] as the real-time extension of timed automata [3]. Learning the complex structure of timed and hybrid automata is a line of research that resulted in deterministic and stochastic reconstruction algorithms.

Niggemann et al. [15] present the tool HyBUTLA that builds prefix trees of the traces and applies merges when appropriate. Since Angluin’s L^* algorithm [5] is a prominent solution for learning discrete automata, several extensions for timed automata were proposed [4, 10]. Based on that, Medhat et al. [14] split the learning process of a hybrid automaton into two steps. They first learn the discrete model of the automaton with L^* and then capture the dynamics using clustering. Both of these techniques can potentially be replaced or integrated into different frameworks. Hence, their approach is complementary to the conservative construction: substituting the clustering for the simpler solve function can yield better precision at the price of conservativeness. Focusing on the medical application domain, HyMN [12] learns patient specific parameters for hybrid automata deterministically. Soto et al. [18] synthesize a hybrid automaton with an online algorithm without relying on a specification as discrete template. While precision is very high and completeness is shown, learning a trace prompts a global analysis of the previously learned hybrid automaton, which incurs performance penalties. The conservative construction avoid this complexity by using the specification automaton and the adequacy criterion. This way, revisions are local and still retain correctness guarantees. Other approaches for learning hybrid automata using mathematical models for node identification were proposed by Summerville et al. [19] and Breschi et al. [7].

If large datasets of traces are available, stochastic learning of hybrid automata is feasible. Tappler et al. [20] use genetic programming to reconstruct timed automata both in an offline and online setting [2]. Santana et al. [16] build hybrid automata with the Expectation-Maximization algorithm to iteratively define the model parameters. An unsupervised learning approach was presented by Lee et al. [13], whereas Birgelen and Niggemann [21] use self-organizing feature maps. Despite the success of machine learning, the results do not provide provable guarantees.

8 CONCLUSION

This paper presented a construction algorithm for conservative hybrid automata from development artifacts in the shape of a runtime monitoring specification and pre-recorded execution traces. The construction is validated mathematically by proving that the result is an over-approximation under certain assumptions on the inputs. An additional empirical evaluation revealed both the extraordinary

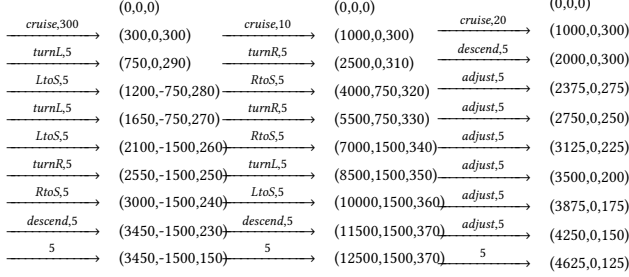
scalability of the construction and that even randomly generated inputs regularly satisfy the input requirements. Considering that – in a realistic setting – these inputs are high-quality artifacts acquired during development of the system, they should no longer be left under-utilized. Treating them as the valuable assets they are allows for constructing precise, conservative hybrid automata in a scalable fashion.

REFERENCES

- [1] Florian-Michael Adolf, Peter Faymonville, Bernd Finkbeiner, Sebastian Schirmer, and Christoph Torens. Stream runtime monitoring on UAS. *CoRR*, abs/1804.04487, 2018.
- [2] Bernhard K. Aichernig, Andrea Pferscher, and Martin Tappler. From passive to active: Learning timed automata efficiently. In Ritchie Lee, Susmit Jha, and Anastasia Mavridou, editors, *NASA Formal Methods - 12th International Symposium, NFM 2020, Moffett Field, CA, USA, May 11–15, 2020, Proceedings*, volume 12229 of *Lecture Notes in Computer Science*, pages 1–19. Springer, 2020.
- [3] Rajeev Alur and David L. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 126(2):183–235, 1994.
- [4] Jie An, Mingshuai Chen, Bohua Zhan, Naijun Zhan, and Miaomiao Zhang. Learning one-clock timed automata. *CoRR*, abs/1910.10680, 2019.
- [5] Dana Angluin. Learning regular sets from queries and counterexamples. *Inf. Comput.*, 75(2):87–106, 1987.
- [6] Jan Baumeister, Bernd Finkbeiner, Sebastian Schirmer, Maximilian Schwenger, and Christoph Torens. Rtlola cleared for take-off: Monitoring autonomous aircraft. *CoRR*, abs/2004.06488, 2020.
- [7] Valentina Breschi, Dario Piga, and Alberto Bemporad. Learning hybrid models with logical and continuous dynamics via multiclass linear separation. In *55th IEEE Conference on Decision and Control, CDC 2016, Las Vegas, NV, USA, December 12–14, 2016*, pages 353–358. IEEE, 2016.
- [8] Peter Faymonville, Bernd Finkbeiner, Malte Schledjewski, Maximilian Schwenger, Marvin Stenger, Leander Tenstrup, and Hazem Torfah. Streamlab: Stream-based monitoring of cyber-physical systems. In Isil Dillig and Serdar Tasiran, editors, *Computer Aided Verification - 31st International Conference, CAV 2019, New York City, NY, USA, July 15–18, 2019, Proceedings, Part I*, volume 11561 of *Lecture Notes in Computer Science*, pages 421–431. Springer, 2019.
- [9] Peter Faymonville, Bernd Finkbeiner, Maximilian Schwenger, and Hazem Torfah. Real-time stream-based monitoring. *CoRR*, abs/1711.03829, 2017.
- [10] Olga Grinchtein, Bengt Jonsson, and Martin Leucker. Learning of event-recording automata. In Yassine Lakhnech and Sergio Yovine, editors, *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems, Joint International Conferences on Formal Modelling and Analysis of Timed Systems, FORMATS 2004 and Formal Techniques in Real-Time and Fault-Tolerant Systems, FTRTFT 2004, Grenoble, France, September 22–24, 2004, Proceedings*, volume 3253 of *Lecture Notes in Computer Science*, pages 379–396. Springer, 2004.
- [11] Thomas A. Henzinger. The theory of hybrid automata. In *Proceedings, 11th Annual IEEE Symposium on Logic in Computer Science, New Brunswick, New Jersey, USA, July 27–30, 1996*, pages 278–292. IEEE Computer Society, 1996.
- [12] Imane Lamrani, Ayan Banerjee, and Sandeep K. S. Gupta. Hymn: Mining linear hybrid automata from input output traces of cyber-physical systems. In *IEEE Industrial Cyber-Physical Systems, ICPS 2018, Saint Petersburg, Russia, May 15–18, 2018*, pages 264–269. IEEE, 2018.
- [13] Gilwoo Lee, Zita Marinho, Aaron M. Johnson, Geoffrey J. Gordon, Siddhartha S. Srinivasa, and Matthew T. Mason. Unsupervised learning for nonlinear piecewise smooth hybrid systems. *CoRR*, abs/1710.00440, 2017.
- [14] Ramy Medhat, S. Ramesh, Borzoo Bonakdarpour, and Sebastian Fischmeister. A framework for mining hybrid automata from input/output traces. In *2015 International Conference on Embedded Software, EMSOFT 2015, Amsterdam, Netherlands, October 4–9, 2015*, pages 177–186, 2015.
- [15] Oliver Niggemann, Benno Stein, Asmir Vodencarevic, Alexander Maier, and Hans Kleine Büning. Learning behavior models for hybrid timed systems. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, July 22–26, 2012, Toronto, Ontario, Canada, 2012*.
- [16] Pedro Henrique Santana, Spencer Lane, Eric Timmons, Brian Charles Williams, and Carlos Forster. Learning hybrid models with guarded transitions. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25–30, 2015, Austin, Texas, USA*, pages 1847–1853, 2015.
- [17] Maximilian Schwenger. Monitoring cyber-physical systems: From design to integration. In Jyotirmoy Deshmukh and Dejan Nickovic, editors, *Runtime Verification - 20th International Conference, RV 2020, Los Angeles, CA, USA, October 6–9, 2020, Proceedings*, volume 12399 of *Lecture Notes in Computer Science*, pages 87–106. Springer, 2020.
- [18] Miriam Garcia Soto, Thomas A. Henzinger, Christian Schilling, and Luka Zeleznik. Membership-based synthesis of linear hybrid automata. In Isil Dillig and Serdar Tasiran, editors, *Computer Aided Verification - 31st International Conference, CAV 2019, New York City, NY, USA, July 15–18, 2019, Proceedings, Part I*, volume 11561 of *Lecture Notes in Computer Science*, pages 297–314. Springer, 2019.
- [19] Adam Summerville, Joseph C. Osborn, and Michael Mateas. CHARDA: causal hybrid automata recovery via dynamic analysis. In Carles Sierra, editor, *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19–25, 2017*, pages 2800–2806. ijcai.org, 2017.
- [20] Martin Tappler, Bernhard K. Aichernig, Kim Guldstrand Larsen, and Florian Lorber. Time to learn - learning timed automata from tests. In Étienne André and Mariëlle Stoelinga, editors, *Formal Modeling and Analysis of Timed Systems - 17th International Conference, FORMATS 2019, Amsterdam, The Netherlands, August 27–29, 2019, Proceedings*, volume 11750 of *Lecture Notes in Computer Science*, pages 216–235. Springer, 2019.
- [21] Alexander von Birgelen and Oliver Niggemann. Using self-organizing maps to learn hybrid timed automata in absence of discrete events. In *22nd IEEE International Conference on Emerging Technologies and Factory Automation, ETFA 2017, Limassol, Cyprus, September 12–15, 2017*, pages 1–8. IEEE, 2017.

A ADEQUATE TRACES OF AIRCRAFT SYSTEM

This section of the appendix presents the adequate traces used to learn the version of the aircraft system depicted in Figure 1b with the dynamics written in black.



B PROOFS OF SECTION 5

This section contains the missing proofs of Section 5.

THEOREM 5.4 (PERFECT PROJECTION). *For any hybrid automaton \mathcal{H} there is a finite set of traces for which the projection onto these traces yields the identity, i.e., $\exists \tilde{\Pi}^* \subseteq \mathcal{L}(\mathcal{H}) : \Pi^* \text{ finite} \wedge \mathcal{L}(\mathcal{H}|_{\tilde{\Pi}^*}) = \mathcal{L}(\mathcal{H})$.*

PROOF. The proof selects traces from $\mathcal{L}(\mathcal{H})$ enabling the perfect projection. For each $e \in E$, $\tilde{\Pi}^*$ contains a trace $\tilde{\pi}$ with $e \in \tilde{\pi}$. This immediately entails that the projected edge set, set of actions, set of modes, and initial mode are accurate when disregarding unreachable entities. For each mode, $\tilde{\Pi}^*$ encompasses four traces per dimension: one minimizing and one maximizing the flow and continuous state variable of the mode and dimension. The minimization and maximization is over the set of traces rather than over the mode itself. As a result, the projection of the flow is perfect. Lastly, for each mode, outgoing edge, and dimension, there are two traces in $\tilde{\Pi}^*$ which maximize and minimize the continuous state value before taking the transition. Hence, the projection of the guard condition is lossless in terms of the language of the automaton.

In conclusion, $\mathcal{L}(\mathcal{H}|_{\tilde{\Pi}^*}) = \mathcal{L}(\mathcal{H})$ with

$$|\tilde{\Pi}^*| = |E| + n(5|M| + \sum_{\mu \in M} \text{outdeg}(\mu))$$

where n is the dimension of \mathcal{H} . \square

LEMMA 5.5 (LOSSLESS MERGE). *Given a constructed hybrid automaton \mathcal{H}^+ and an equivalence relation \approx , merging \mathcal{H}^+ with respect to \approx yields a more permissive automaton, i.e., $\mathcal{L}(\mathcal{H}^+) \subseteq \mathcal{L}(\mathcal{H}^+ \downarrow_{\approx})$.*

PROOF. By contradiction: Assume there is a trace $\pi \in \mathcal{L}(\mathcal{H}^+) \setminus \mathcal{L}(\mathcal{H}^+ \downarrow_{\approx})$. As the merge operation is defined by unifying modes, π either (1) takes a discrete transition or (2) traverses a continuous state not permitted in the merged automaton. Definition 4.6 “bends” edges such that they originate and end in the respective representatives. Hence, the construction retains all edges up to elimination of duplicates due to set semantics, ruling out (1). Regarding (2), Definition 4.6 builds the convex hull for all flow and guard definitions of the merged states. The convex hull is at least as permissive as its constituents, rendering a less permissive behavior impossible, which concludes the proof. \square

THEOREM 5.7 (INPUT TRACE INCLUSION). *Given an adequate set of traces Π and specification Φ , the language of a constructed automaton \mathcal{H}^+ subsumes Π , i.e., $\Pi \subseteq \mathcal{L}(\mathcal{H}^+)$.*

PROOF. Let $\pi \in \Pi$ be an arbitrary input trace. We first prove by induction that any subsequence of length i of π is included in the language of \mathcal{H}_i^+ . Recall that $\psi_i(\pi)$ denotes the mode in which $\pi[0, i]$ ends.

INDUCTION BASE: $i = 0$. By construction $(x_0^\pi, \mu_I) \in \mathcal{L}(\mathcal{H}_0^+)$. Moreover, $\psi_0(\pi) = \mu_I$ marks the (start and) end point of the trace.

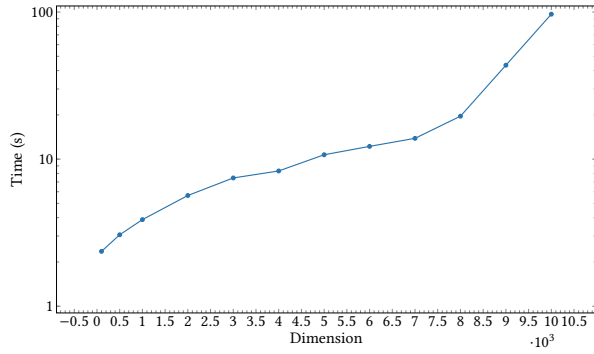
INDUCTION STEP: $i - 1 \rightarrow i$. Consider $\pi[0, i] = \pi[0, i - 1], \lambda_i, \delta_i, x_i$. By induction hypothesis, $\pi[0, i - 1] \in \mathcal{L}(\mathcal{H}_{i-1}^+)$. By Lemma 5.6, the language membership carries over to \mathcal{H}_i^+ . By construction, $e = (\mu_{i-1}, \lambda_i, \psi_i(\pi)) \in E_i$ with guard $\gamma_i(\pi) = \Gamma^\Phi(e^\alpha)$. Here, $\Gamma^\Phi(e^\alpha)$ is either \top if the transition is not present Φ , or the condition from the respective trigger in Φ . In the latter case, by construction of α_i , the respective trigger (and thus guard) condition is satisfied in x_{i-1} . This enables the discrete transition and ensuring that the trace ends in $\psi_i(\pi)$. The delay transition is valid because of the definition of solve. Hence, $\pi \subseteq \mathcal{L}(\mathcal{H}_{|\pi|}^+)$. By Lemma 5.5, this result carries over to \mathcal{H}^+ , i.e., $\pi \subseteq \mathcal{L}(\mathcal{H}^+)$. \square

C ADDITIONAL EVALUATIONS RESULTS

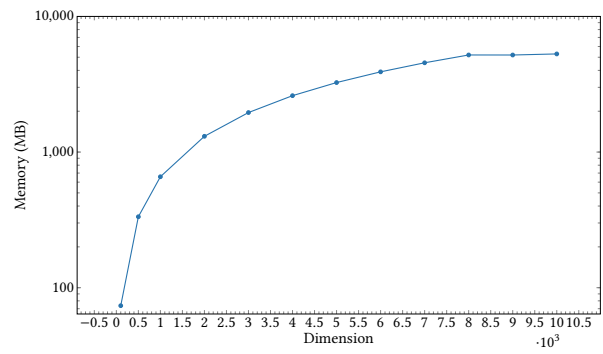
This section presents supplementary material for the evaluation in Section 6.

Figure 4 shows that the construction scales extremely well for an increasing dimension, both in terms of running time and memory consumption assuming the number of modes is a constant $2^{11} - 1$. The increase in running time mainly stems from the merge operation, which computes the convex hull of the dynamics for each dimension separately. However, the impact is rather low since the running time is less than 100 s even for 10^4 dimensions. Similarly, the memory consumption increases solely owing to the necessity to store the multi-dimensional dynamics of each mode.

Figure 5 shows the result when constructing five-dimensional automata with the same number of modes, but varying number of traces. Each trace has a length of eleven. Increasing the number of traces results in an increase in running time since the construction has to traverse the current automaton for each traces. It also increases the memory consumption because traces lead to the creation of new modes. However, the construction scales well in both performance metrics with a running time of around 17 min and memory consumption of 200 MB for 16,000 traces.

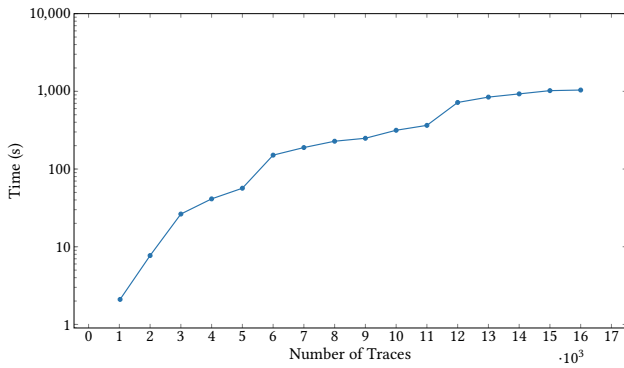


(a) Running Time

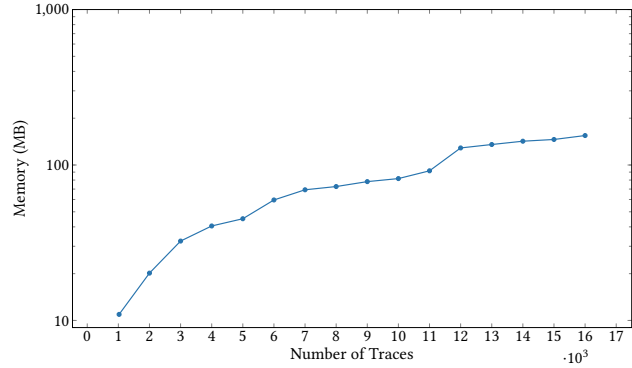


(b) Memory Consumption

Figure 4: Performance of the reconstruction for varying dimensions. The original automaton has $2^{11} - 1$ modes and each of the around 1,000 trace has length eleven.



(a) Running Time



(b) Memory Consumption

Figure 5: Performance of the reconstruction for a varying number of traces. The original automaton has $2^{11} - 1$ modes and five dimensions. Each trace has length eleven.