A Toolchain for Verifying Safety Properties of Hybrid Automata via Pattern Templates

Goran Frehse[1], Nikolaos Kekatos[1], Dejan Nickovic[2], Jens Oehlerking[3], Simone Schuler[4], Alexander Walsch[4], and Matthias Woehrle[5]

Abstract— **In this paper, we provide a toolchain that facilitates the integration of formal verification techniques into model-based design. Applying verification tools to industrially relevant models requires three main ingredients: a formal model, a formal verification method, and a set of formal specifications. Our focus is on hybrid automata as the model and on reachability analysis as the method. Much progress has been made towards developing efficient and scalable reachability algorithms tailored to hybrid automata. However, it is not easy to encode rich formal specifications such that they can be interpreted by existing tools for reachability. Herein, we consider specifications expressed in pattern templates which are predefined properties with placeholders for state predicates. Pattern templates are close to the natural language and can be easily understood by both expert and non-expert users. We provide (i) formal definitions for selected patterns in the formalism of hybrid automata and (ii) monitors which encode the properties as the reachability of an error state. By composing these monitors with the formal model under study, the property can be checked by off-the-shelf fully automated verification tools. We illustrate the workflow on an electro-mechanical brake use case.**

I. Introduction

Model-based design (MBD) is a paradigm that enables the cost-effective and quick development of complex systems, such as control and energy systems. MBD has facilitated the detection and correction of errors in the early design stages and has established a common framework for communication throughout the whole design process [43].

In MBD, there is typically a sequence of steps that should be followed. Initially, the designer models the physical plant, relying either on first principles or on system identification; this model captures the dynamical characteristics of the physical parts of the system using mathematical equations. Then, the designer synthesizes a controller that regulates the behavior of the physical system. Subsequently, he performs extensive simulations to check the model behavior under different configuration settings. The aim is to analyze and evaluate the controller design by inspecting the behavior of specific variables over time. The analysis is typically performed with respect to some requirements. In practice, however, these requirements are high-level and often vague or informal. In case the system behavior is not satisfying with respect to these requirements, the designer has to manually modify the controller, e.g. by tuning the parameters or gains, and then repeat the validation step. Through these validation efforts, the design is deemed to be satisfactory or not. The evaluation may also remain inconclusive [30].

Over the past years, there have been a lot of efforts to bridge the gap between formal verification and industrial applications. The main focus has been on addressing the issues with the format mismatch and scale of industrially sized models. That is especially the case with the new generation of systems, embedded and cyber-physical, as they are complex with various interacting components and frequently have a safety-critical nature [46].

An appropriate modeling formalism for the design of such systems is hybrid systems [1]. Hybrid systems demonstrate joint discrete and continuous behaviors by combining the traditional models for discrete systems with classical differential and algebraic equation-based models for dynamical systems [2]. Those systems are difficult to analyze, as any kind of nondeterminism in the system, like disturbances, measurement noise, uncertainties, user inputs, or operating conditions, may have adverse effects on the performance [17].

Recently, there has been increased interest in fully automated verification tools for hybrid systems, such as set-based reachability analysis [37]. This is the case as they have been successful in finding bugs in real-world applications and there has been much progress towards efficient and scalable reachability algorithms [1]. On the contrary, relatively little progress has been made on formalizing requirements of hybrid systems such that they can be verified automatically. The main reason concerns the semantic mismatch between industrial requirements and formal requirements. Typically, formal requirements are expressed in temporal logic [44], whereas industrial requirements are described in natural language or controlled natural language (CNL) [36]. In this paper, we aim to mitigate this semantic mismatch and we are therefore proposing a semi–automated, template–based translation of industrial requirements into a formal representation (monitor automata) that enables the algorithmic verification of rich specifications.

A schematic of the proposed workflow is depicted in Figure 1. A (safety) requirement in CNL is translated into

[1]Goran Frehse and Nikolaos Kekatos are with VERIMAG, University of Grenoble Alpes, Grenoble, France {firstname.lastname}@univ-grenoble-alpes.fr

[2]Dejan Nickovic is with Austrian Institute of Technology, Vienna, Austria dejan.nickovic@ait.ac.at

[3]Jens Oehlerking is with Robert Bosch GmbH, Corpor. Research, Stuttgart, Germany jens.oehlerking@de.bosch.com

[4]Simone Schuler and Alexander Walsch are with GE Global Research, Munich, Germany {firstname.lastname}@ge.com

[5]Matthias Woehrle was with Robert Bosch GmbH, Corpor. Research, Stuttgart, Germany when this work was conducted. matthias.woehrle@alumni.ethz.ch

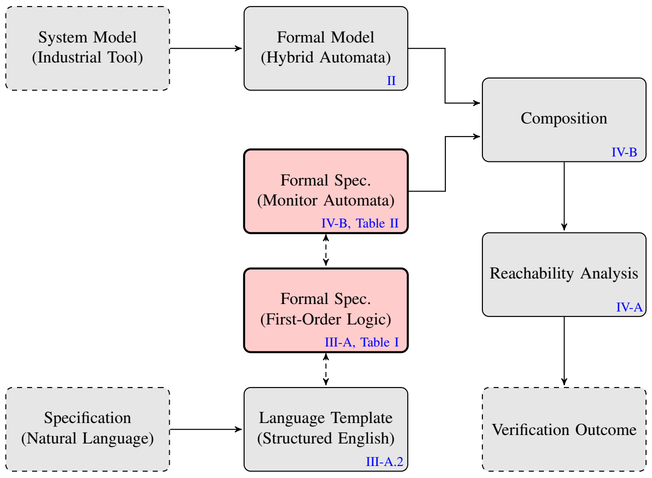

System Model (Industrial Tool)
Formal Model (Hybrid Automata)
II
Composition
IV-B
Formal Spec. (Monitor Automata)
IV-B, Table II
Reachability Analysis
IV-A
Formal Spec. (First-Order Logic)
III-A, Table I
Specification (Natural Language)
Language Template (Structured English)
III-A.2
Verification Outcome

Fig. 1: Proposed verification workflow: (i) start with an informal model and specs, (ii) formalize the model and the specs via pattern templates, (iii) feed the formal model and the specs into off-the-shelf reachability analysis tools. The contribution (pink) is the formalization using pattern templates tailored for model checking.

a monitor automaton using pattern templates [35]. The monitor automaton has the same syntax and respects the same semantics as the system model. As such, it can be composed with the system model and fed into a reachability tool. The monitor automaton encodes the requirements as the reachability of a designated error state (see [25] for one of the earliest works on monitor automata and [9] for analog and mixed-signal applications). Recently, a tool called formalSpec was implemented by some of the authors to automate the instantiation of monitor automata from CNL. The tool comes with a database of structured English phrases and the associated template monitors. The monitors of this paper can be used with formalSpec.

In this paper, we provide a semi-automated toolchain to conduct formal verification of hybrid systems against rich formal specifications. To accomplish this task, we introduce two new elements. First, we give formal definitions for pattern templates [35] that are suitable for hybrid automata (run semantics). Second, we provide monitor automata whose correctness we have formally proved and which can be handled by off-the-shelf fully automated verification tools. The proofs have been omitted for lack of space. They can be found in the technical report [21].

The remainder of the paper is organized as follows. In Section II, we introduce the preliminaries. The pattern templates and their formalization are presented in Section III. The monitor automata are described in Section IV. We apply the introduced monitors to an electromechanical brake use case in Section V. In Section VI, we give an overview of the state of the art. The paper concludes with Section VII.

II. Background

In this part, we present the preliminaries and give a formal definition of a hybrid automaton and its run semantics.

Given a set $X = \{x_1, \ldots, x_n\}$ of variables, a *valuation* is a function $v : X \to \mathbb{R}$. Let $V(X)$ denote the set of valuations over X. Let $\dot{X} = \{\dot{x}_1, \ldots, \dot{x}_n\}$ and $X' = \{x'_1, \ldots, x'_n\}$. The *projection* of v to variables $Y \subseteq X$ is $v{\downarrow_Y} = \{x \to v(x) | x \in Y\}$. The *embedding* of a set $U \subseteq V(X)$ into variables $\bar{X} \supseteq X$ is the largest subset of $V(Y)$ whose projection is in U, written as $U|^{\bar{X}}$.

A *hybrid automaton* ([2], [26]) $H = (\mathsf{Loc}, \mathsf{Lab}, \mathsf{Edg}, X, \mathsf{Init}, \mathsf{Inv}, \mathsf{Flow}, \mathsf{Jump})$ consists of

- a finite set of *locations* $\mathsf{Loc} = \{\ell_1, \ldots, \ell_m\}$ which represents the discrete states,
- a finite set of *synchronization labels* Lab, which coordinate state changes between several automata,
- a finite set of edges $\mathsf{Edg} \subseteq \mathsf{Loc} \times \mathsf{Lab} \times \mathsf{Loc}$, also called *transitions*, that determines which discrete state changes are possible using which label,
- a finite set of *variables* $X = \{x_1, \ldots, x_n\}$, partitioned into uncontrolled variables U and controlled variables Y; a *state* of H consists of a location ℓ and a value for each of the variables, and is denoted by $s = (\ell, \mathbf{x})$;
- a set of states Inv called *invariant* or *staying condition*; it restricts for each location the values that x can possibly take and determines how long the system can remain in the location;
- a set of *initial* states $\mathsf{Init} \subseteq \mathsf{Inv}$; every behavior of H must start in one of the initial states;
- a *flow relation* Flow, where $\mathsf{Flow}(\ell) \subseteq \mathbb{R}^{\dot{X}} \times \mathbb{R}^X$ determines for each state $(\ell, \mathbf{x})$ the set of possible derivatives $\dot{\mathbf{x}}$, e.g., using a differential equation $\dot{\mathbf{x}} = f(\mathbf{x})$. Given a location ℓ, a *trajectory* of duration $\delta \geq 0$ is a continuously differentiable function $\xi : [0, \delta] \to \mathbb{R}^X$ such that for all $t \in [0, \delta]$, $(\dot{\xi}(t), \xi(t)) \in \mathsf{Flow}(\ell)$. The trajectory *satisfies the invariant* if for all $t \in [0, \delta]$, $\xi(t) \in \mathsf{Inv}(\ell)$.
- a *jump relation* Jump, where $\mathsf{Jump}(e) \subseteq \mathbb{R}^X \times \mathbb{R}^{X'}$ defines for each transition $e \in \mathsf{Edg}$ the set of possible successors $\mathbf{x}'$ of $\mathbf{x}$; jump relations are typically described by a *guard* set $\mathcal{G} \subseteq \mathbb{R}^X$ and an assignment (or reset) $\mathbf{x}' = r(\mathbf{x})$ as $\mathsf{Jump}(e) = \{(\mathbf{x}, \mathbf{x}') \mid \mathbf{x} \in \mathcal{G} \wedge \mathbf{x}' = r(\mathbf{x})\}$. A jump can be cast as urgent, which means that time cannot elapse when the state is in the guard set.

We define the behavior of a hybrid automaton with a *run*: starting from one of the initial states, the state evolves according to the differential equations whilst time passes, and according to the jump relations when taking an (instantaneous) transition. Special events, which we call *uncontrolled assignments*, model an environment that can make arbitrary changes to the uncontrolled variables.

An *execution* of a hybrid automaton H is a sequence

$$(\ell_0, \mathbf{x}_0) \xrightarrow{\delta_0, \xi_0} (\ell_0, \xi_0(\delta_0)) \xrightarrow{\alpha_0} (\ell_1, \mathbf{x}_1) \xrightarrow{\delta_1, \xi_1} (\ell_1, \xi_1(\delta_1)) \ldots \xrightarrow{\alpha_{N-1}} (\ell_N, \mathbf{x}_N),$$

with $\alpha_i \in \mathsf{Lab} \cup \{\tau\}$, satisfying for $i = 0, \ldots, N-1$:

1) *Trajectories:* In location ℓ_i, ξ_i is a trajectory of duration δ_i with $\xi_i(0) = \mathbf{x}_i$ and it satisfies the invariant. It does not go through urgent guard sets unless δ_i is 0.

2) *Jumps:* If $\alpha_i \in$ Lab, there is a transition $(\ell_i, \alpha_i, \ell_{i+1}) \in \mathsf{Edg}$ with jump relation $\mathsf{Jump}(e)$ such that $(\xi_i(\delta_i), \mathbf{x}_{i+1}) \in \mathsf{Jump}(e)$ and $\mathbf{x}_{i+1} \in \mathsf{Inv}(\ell_{i+1})$.
3) *Uncontrolled assignments:* If $\alpha_i = \tau$, then $\ell_i = \ell_{i+1}$ and $\xi_i(\delta_i) \downarrow_Y = \mathbf{x}_{i+1} \downarrow_Y$. This represents arbitrary assignments that the environment might perform on the uncontrolled variables $U = X \setminus Y$.

A *run* of H is an execution that starts in one of the initial states, i.e. $(\ell_0, \mathbf{x}_0) \in \mathsf{Init}$. A state $(\ell, \mathbf{x})$ is *reachable* if there exists a run with $(\ell_i, \mathbf{x}_i) = (\ell, \mathbf{x})$ for some i.

III. Pattern Templates for Hybrid Automata

Pattern templates are predefined properties with placeholders for state predicates and were introduced in [23]. As a first major contribution of this paper, we define a set of pattern templates in a formalism that is suitable for hybrid automata and show their usability on practical applications. Later, in Section 4, we are going to see how to use these templates with reachability tools.

A. Formalizing Pattern Templates for Hybrid Automata

In this section, we list the requirements considered in this paper, give a compact (intuitive) definition in structured English, and a formal definition based on the runs of the hybrid automaton.

The pattern templates in [35] were formally defined using temporal logics (MTL). These definitions, however, do not immediately carry over to monitoring with hybrid automata (see Section VI). In this respect, we select some common pattern templates, portrayed in Table I, and define them in a formalism that is suitable for hybrid automata. Note that the properties in this paper refer to predicates that describe states, not events. In addition, the predicates can express timing properties by adding an extra clock to the monitor, so that the time becomes a state variable. We consider *triggered* versions of the properties that only take effect after a predicate q holds. A run, for which $!q$ always holds, satisfies the property.

1) Preliminaries:

Let p be a predicate over the state variables, i.e. a function $\mathbb{R}^X \rightarrow \mathbb{B}$. We write the shorthand $p(\mathbf{x})$ to denote that p is true for $\mathbf{x}$. Let the set of runs of a hybrid automaton H be $\mathrm{Runs}(H)$. We consider a run $r \in R$ given by locations ℓ_i, continuous states $\mathbf{x}_i$, trajectories ξ_i, and durations δ_i. To simplify the formalization of the properties, we introduce some further notation for the timing of states on runs. For a run r ,the *event-times* are $t_i = \sum_{j=0}^{i} \delta_i$, so the jump number i takes place at time t_i for $i = 0, \ldots, N-1$. For notational convenience, let $t_{-1} = 0$.

We introduce a total order on the time points of the run by looking at pairs (i, t), where i is an index and t is the global time. Formally, let the *event-time* be $\mathbb{T} = \mathbb{N}^0 \times \mathbb{R}^{\geq 0}$. To clarify the difference, we denote real time with t and event-time with $\tau \in \mathbb{T}$. We use the lexicographical order on event-times, formally $(i, t) < (i', t') \Leftrightarrow (i < i') \vee (i = i' \wedge t < t')$. The event-time allows us to uniquely identify discrete and continuous states on the run. The *event-time domain* of a run r is the set of pairs $\mathrm{dom}(r) = \{(i,t) \mid 0 \leq i \leq N-1, t_{i-1} \leq t \leq t_i\} \cup \{(N, t_{N-1})\}$, where the latter term captures that the last state in the run, $(\ell_N, \mathbf{x}_N)$ is taken at time t_{N-1} (total duration of the run).

The *open truncated* event-time domain of a run r excluding the last T time units is the set of pairs $\mathrm{dom}_{-T}(r) = \{(i,t) \in \mathrm{dom}(r) \mid t < t_{N-1} - T\}$. The truncated domain is used for properties that refer to future events and are not covered by the domain of the run. We take an optimistic view of such cases: if the property holds on the truncated domain, then it is considered to hold on the run.

For a given $\tau = (i, t) \in \mathrm{dom}(r)$, let $r(\tau) \in \mathbb{R}^X$ be the continuous state $\xi_i(t - t_i)$ and $r_{\mathsf{Loc}}(\tau) \in \mathsf{Loc}$ be the discrete state (location) ℓ_i. This denotes the time elapsed between two event-times $\tau = (i, t), \tau' = (i', t')$ as $d(\tau, \tau') = t' - t$.

Sometimes, we are interested in the first time that a predicate holds. If the predicate, say q, is true over a left-open interval, the infimum shall be used. Let $\tau_{q.1} = \inf_{\tau \in \mathrm{dom}(r)} q(r(\tau))$. To formally denote that a predicate holds at τ for some nonzero amount of time, we define for a run r, a predicate p, and event-time τ, $\mathrm{persists}(r, p, \tau) = \exists \delta > 0 : \forall \tau', \tau \leq \tau', d(\tau, \tau') \leq \delta : r(\tau')$.

2) Formal Definitions:

We define the properties of a hybrid automaton via its runs. A hybrid automaton H satisfies a property ϕ if and only if all runs $r \in \mathrm{Runs}(H)$ satisfy ϕ. We use the convention: (i) $r \models \phi$ when a run r satisfies the property ϕ, (ii) p following q means that there are τ_q and τ_p with $\tau_p \geq \tau_q$ such that $p(r(\tau_p))$ and $q(r(\tau_q))$ hold, (iii) τ_q, τ_p, $\tau_{\overline{p}}$, τ'_p respectively imply that $q(r(\tau_q))$, $p(r(\tau_p))$, $\neg p(r(\tau_{\overline{p}})$, $p(r(\tau'_p))$ hold.

Absence. *After q, it is never the case that p holds.*

$r \models \phi$ iff for all $\tau_q, \tau \in \mathrm{dom}(r)$ with $\tau \geq \tau_q$, holds $\neg p(r(\tau))$.

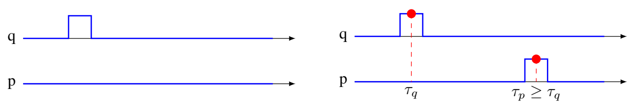


Fig. 2: Absence pattern: satisfied (left), violated (right).

Absence (timed). *When T time units are measured, after q was first satisfied, it is never the case that p holds.*

$r \models \phi$ iff for all $\tau_q, \tau \in \mathrm{dom}(r)$ with $d(\tau_q, \tau) \geq T$, holds $\neg p(r(\tau))$.

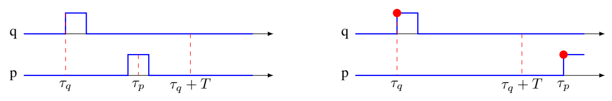


Fig. 3: Timed absence pattern: satisfied (left), violated (right).

Minimum duration. *After q, it is always the case that once p becomes satisfied, it holds for at least T time units.*

$r \models \phi$ iff p following q implies that for $\tau_{q.1}$ holds:

1) for all $\tau_p, \tau_{\overline{p}} \in \mathrm{dom}(r)$ with $\tau_{q.1} \leq \tau_p < \tau_{\overline{p}}$, $d(\tau_{q.1}, \tau_{\overline{p}}) > T$ (p not becoming false within T after $\tau_{q.1}$), and

TABLE I: Short overview of Pattern Templates [35]

Pattern name	Description & Example
absence	Specifies a state formula that must not hold. *ABS system:* "The ABS controller should never allow a wheel skidding."
minimum duration	Describes the minimum amount of time a state formula has to hold once it becomes true. *Engine starter system:* "The system has a minimum 'off' period of 120s before it reenters the cranking mode."
maximum duration	Captures that a state formula always holds for less than a specified amount of time. *Engine starter system:* "The system can only operate in engine cranking mode for no longer than 10s."
bounded recurrence	Denotes the amount of time in which a state formula has to hold at least once. *ABS system:* "The ABS controller checks for skidding every 10ms."
bounded response	Restricts the maximum amount of time that passes after a formula holds until another formula becomes true. *ABS System:* "From direct client input, detection and response to rapid deceleration must occur within 0.015s."
bounded invariance	Specifies the minimum amount of time a state formula must hold once another state formula is satisfied. *Engine starter system:* "If the error 502 is sent to the Drive Information System, the braking system is inhibited for 10s."

2) for all $\tau_p, \tau_{\bar{p}}, \tau'_{\bar{p}} \in \mathrm{dom}(r)$ with $\tau_{q.1} \le \tau_{\bar{p}} < \tau_p < \tau'_{\bar{p}}$, it holds that $d(\tau_{\bar{p}}, \tau'_{\bar{p}}) > T$ (violations of p are more than T apart).

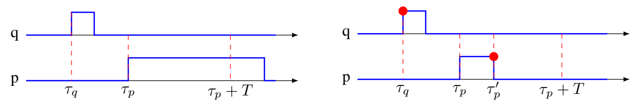


Fig. 4: Minimum duration pattern: satisfied (left), violated (right).

Maximum duration. *After q, it is always the case that once p becomes satisfied, it holds for less than T time units.*

$r \models \phi$ iff p following q implies that for all $\tau_p, \tau'_p \in \mathrm{dom}(r)$ with $\tau_p \ge \tau_q$ one of the following holds:

1) $d(\tau_p, \tau'_p) < T$ (τ'_p is early enough, including the $\tau_p = \tau'_p$ case), or
2) there is a $\tau_{\bar{p}}$ such that $\tau_p < \tau_{\bar{p}} < \tau'_p$ (p is false in between).

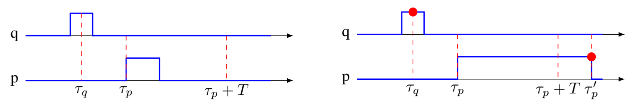


Fig. 5: Maximum duration pattern: satisfied (left), violated (right).

Bounded recurrence. *After q, it is always the case that p holds at least every T time units.*

For the unbounded case, $r \models \phi$ iff for all $\tau_q \in \mathrm{dom}(r)$ both following criteria hold:

(i) for all $\tau_p \in \mathrm{dom}(r)$ with $\tau_p \ge \tau_q$ there is a $\tau'_p \in \mathrm{dom}(r)$ such that $\tau_p < \tau'_p$, $d(\tau_p, \tau'_p) \le T$ (τ_p's with distance less than T),

(ii) there is a $\tau_p \in \mathrm{dom}(r)$ with $\tau_p \ge \tau_q$ such that $d(\tau_q, \tau_p) \le T$ (distance between τ_q and first τ_p is less than T).

For a bounded time horizon, $r \models \phi$ iff for all $\tau_q \in \mathrm{dom}_{-T}(r)$ both following criteria hold:

(i) for all $\tau_p \in \mathrm{dom}_{-T}(r)$ with $\tau_p \ge \tau_q$ there is a $\tau'_p \in \mathrm{dom}(r)$ such that $\tau_p < \tau'_p$ and $d(\tau_p, \tau'_p) < T$,

(ii) there is a $\tau_p \in \mathrm{dom}(r)$ with $\tau_p \ge \tau_q$ and $d(\tau_q, \tau_p) \le T$.

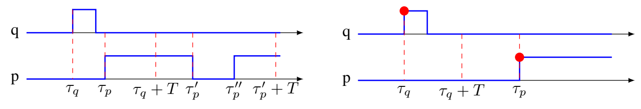


Fig. 6: Bounded recurrence pattern: satisfied (left), violated (right).

Bounded response (persisting). *After q, it is always the case that if p holds, then s persists (holds for nonzero time) after at most T time units.*

For an unbounded time horizon, $r \models \phi$ iff p following q implies that for all $\tau_q \in \mathrm{dom}(r)$, it holds that for all $\tau_p \in \mathrm{dom}(r)$ with $\tau_p \ge \tau_q$, there is a $\tau_s \in \mathrm{dom}(r)$ such that $\tau_p \le \tau_s$, $d(\tau_s, \tau_p) \le T$, and $\mathrm{persists}\,(r, \tau_s, s)$.

For a bounded time horizon, $r \models \phi$ iff p following q implies that for all $\tau_q, \tau_p \in \mathrm{dom}_{-T}(r)$ with $\tau_p \ge \tau_q$, there is a $\tau_s \in \mathrm{dom}(r)$ such that $\tau_p \le \tau_s$, $d(\tau_p, \tau_s) \le T$, and $\mathrm{persists}\,(r, \tau_s, s)$.

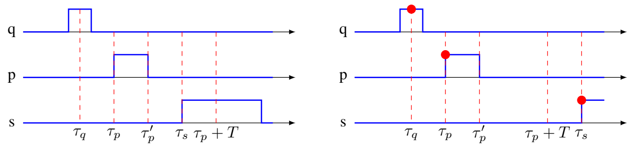


Fig. 7: Bounded response pattern: satisfied (left), violated (right).

Remark 1: We require $\tau \in \mathrm{dom}_{-T}(r)$ in the bounded time horizon (with the restricted domain being right-open) as we assume an optimistic interpretation of bounded runs. If there is a continuation of the run for which the system satisfies the property, then the bounded run satisfies the property. If the restricted domain was right-closed, then a run ending with $\neg s$ could violate the property, but have a continuation that (in zero time) sets s to true, which then should satisfy the property.

Remark 2: We require s to hold for nonzero time, formally with the use of $\mathrm{persists}\,(\cdot)$, because the monitor automaton may give a false alarm otherwise.

Bounded invariance. *After q, it is always the case that if p holds, then s holds for at least T time units.*

$r \models \phi$ iff p following q implies that for all $\tau_p \in \mathrm{dom}(r)$ with $\tau_p \ge \tau_{q.1}$ and for all $\tau \in \mathrm{dom}(r)$ such that $\tau_p \le \tau$, $d(\tau_p, \tau) < T$, the predicate $s(r(\tau))$ is true.

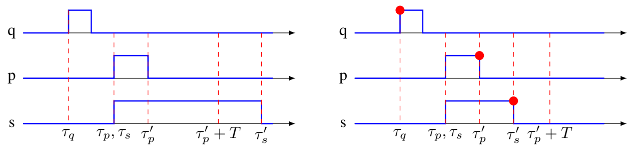


Fig. 8: Bounded invariance pattern: satisfied (left), violated (right).

Remark 3: Note that in the case that predicates $s = p$, then p has to hold forever (by recursion).

B. Application of Formalized Pattern Templates

Formalizing practical requirements is a challenging task, even for experts [16], [27]. Control specifications can be expressed with Temporal Logic [32] or with Simulink monitors [5]. In this section, we show how some common control specifications can be expressed with our formalized pattern templates. We consider the untriggered version of the requirements ($q :=$ true). We assume a constant, positive reference signal x_{ref} as well as that $x(0) < x_{ref}$ holds.

Safety. *The state x of the system should always be inside the acceptable operating range expressed as safe region S.*
absence pattern with $p := \{x \notin S\}$.

Target Reachability. *The state x of the system should be within distance ε of the target (x_{target}) within T time units.*
bounded response pattern, where $p :=$ true and $s := \{||x, x_{target}|| \leq \varepsilon\}$.

Overshoot. *The state x of the system should not exceed an overshoot of $ov\%$ with respect to the reference x_{ref}.*
absence pattern, where $p := \{x > (100 + ov)\% \cdot x_{ref}\}$.

Settling Time. *The state x of the system should reach and stay within a $per\%$ of the reference x_{ref} within T_{set} time units.*
absence (timed) pattern, where $T := T_{set}$, p:= $\{x \leq (100 - per)\% \cdot x_{ref} \lor x \geq (100 + per)\% \cdot x_{ref}\}$.

Rise-Time. *The state x of the system should reach 90% of the reference x_{ref} at time T_{rise}.*
bounded response pattern, where p :=true, $T := T_{rise}$, and $s := \{x >= 0.9 * x_{ref}\}$.

Undershoot. *After reaching the reference x_{ref}, the state x of the system should not fall below a threshold of $u\%$ with respect to the reference.*
bounded invariance pattern, where $T := \infty$, $p := x \geq x_{ref}$ and $s := \{x \geq (100 - u)\% \cdot x_{ref}\}$.

Remark 4: In several cases above, the monitor can be simplified (when p :=true, $T := 0$, etc.) or be expressed with multiple pattern templates. A varying reference signal can be captured with the introduction of predicate q.

IV. Verifying Pattern Templates using Monitor Automata

In this section, we briefly present reachability analysis and provide monitor automata which encode the requirements as reachability problems. These monitor automata constitute the second main contribution of this paper, as they can be composed with the system under study and thus can be straightforwardly used by reachability tools.

A. Reachability Analysis

Set-based reachability analysis can be seen as a generalization of numerical simulation. In numerical simulation, one picks an initial state and tries to compute a successor state that lies on one of the solutions of the corresponding flow constraint and satisfies one of the jump conditions. Then, one of the successor states of the jump is picked and the process is repeated. Reachability analysis directly follows the transition semantics of hybrid automata, but considers sets of states instead of single states [17].

The reachable set consists of all the states that can be visited by a trajectory of the hybrid system starting in specified initial states. Reachability analysis is often motivated by safety verification, which consists in checking whether the intersection of the reachable set with a set of error (forbidden) states is empty. When the reachable set of a hybrid system is not exactly computable, we try to compute an overapproximation so that if it does not intersect the set of error states, the hybrid system is guaranteed to be safe [4].

Computation costs generally increase sharply with respect to the number of continuous variables. Scalable approximations are available for certain types of dynamics, but this performance comes at a price in accuracy. The trade-off between runtime and accuracy remains a central problem in reachability analysis. Surveys of reachability techniques for hybrid automata can be found in [4], [17].

B. Monitor Automata for Reachability

In this section, we define monitor automata that, composed with the system under test, encode the requirements as reachability properties. Consider a system under test H and a monitor automaton M. The goal is that H satisfies a property ϕ if and only if the location *error* is unreachable in the parallel composition $H||M$. We prove correctness of M by showing that every violating run of H has a corresponding run in $H||M$ that reaches the error location, and vice versa. The proofs have been omitted for lack of space and can be found in the technical report [21]. The monitor automata are shown in Table II.

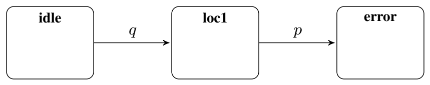
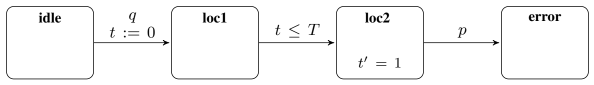
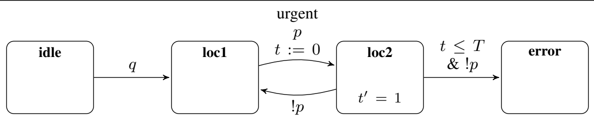
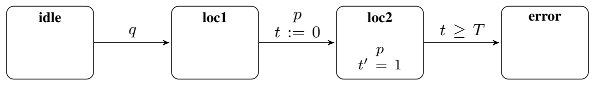
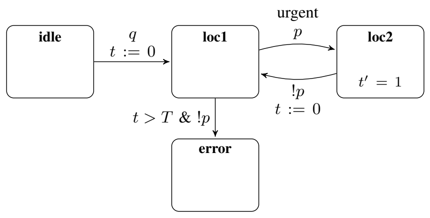
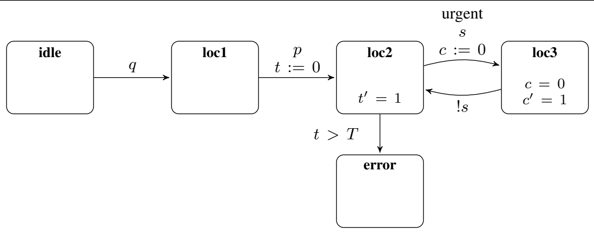
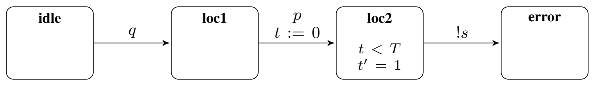
V. Application Example

In this section, we illustrate the workflow on an industrial use case on electro-mechanical brakes (EMB) and highlight how the introduced pattern templates and associated monitor automata can facilitate the verification process. The EMB use case is described in [47]. The requirements that shall be enforced are presented in [20]. The steps of the proposed workflow are as follows.

a) Industrial Model: The model is designed with Simulink. It consists of an experimental electro-mechanical braking system, a feedforward and a feedback controller.

b) Formal Model: The Simulink to SpaceEx (SL2SX) translator [40] is used to construct the formal model. The model is expressed in the SpaceEx format [19] and it consists of 8 base components (single HA) and 4 network components (networks of HA). General, nonlinear Simulink systems can

TABLE II: Pattern templates and translation to monitor automata.

Pattern name	Language Template	Monitor Automaton
absence	After q, it is never the case that p holds[a].	
absence (timed)	When T time units are measured, after q was first satisfied, it is never the case that p holds.	 idle q $t := 0$ loc1 $t \leq T$ loc2 $t' = 1$ p error
minimum duration	After q, it is always the case that once p becomes satisfied, it holds for at least T time units.	 idle q loc1 urgent p $t := 0$!p loc2 $t' = 1$ $t \leq T$ & !p error
maximum duration	After q, it is always the case that once p becomes satisfied, it holds for less than T time units.	 idle q loc1 p $t := 0$ loc2 p $t' = 1$ $t \geq T$ error
bounded recurrence	After q, it is always the case that p holds at least every T time units.	 idle q $t := 0$ loc1 urgent p !p $t := 0$ loc2 $t' = 1$ $t > T$ & !p error
bounded response (persisting)	After q, it is always the case that if p holds, then s persists (holds for nonzero time) after at most T time units.	 idle q loc1 p $t := 0$ loc2 $t' = 1$ urgent s $c := 0$!s loc3 $c = 0$ $c' = 1$ $t > T$ error
bounded invariance	After q, it is always the case that if p holds, then s holds for at least T time units.	 idle q loc1 p $t := 0$ loc2 $t < T$ $t' = 1$!s error

[a] We use the verb *hold* to describe a property that was always true. On the contrary, *becomes satisfied* corresponds to an edge, i.e. the signal was false earlier and then became true.

be transformed to SpaceEx models in the form of piecewise affine hybrid automata, as shown in [33] and [34].

c) Specifications: Two braking specifications are provided in [20].

1) "The caliper must reach $x_0 = 0.05$ dm after the braking request is issued within 20 ms with a precision of 4%". This property can be mapped to the **bounded response** pattern, where $T := 20$, $q := true$, $p :=$ true (braking request), $s := \{0.96 \cdot x_0 \leq x\}$ and x represents the caliper position.
2) "The caliper speed at contact must be below 2 mm/s". This property can be mapped to the **absence** pattern, where $q := true$, $p := \{v \geq 2\}$, and v represents the caliper speed.

d) Monitor: For the first specification, we use the corresponding monitor automaton of Table II for the bounded response pattern. The monitor is generated with formalSpec tool [11]. The formal model and the monitor are expressed as hybrid automata and comply with SpaceEx format.

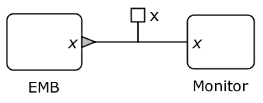


Fig. 9: Composition of the formal model (EMB) with the monitor automaton (bounded response), shown in SpaceEx Model Editor.

e) Composition: This step corresponds to the parallel composition of the formal model with the corresponding monitor. In essence, the variables that appear in the monitor should be connected with the corresponding variables of the formal model. In our case, only the caliper position x should be considered. The variables t and c are local and only used inside the monitor automaton. Figure 9 shows the composed system in the Model Editor [19].

f) Reachability Analysis: SpaceEx [22] is used for computing the reachable sets. The safety verification problem is tackled by introducing a set of error states and checking whether they are reachable or not. In practice, we check if

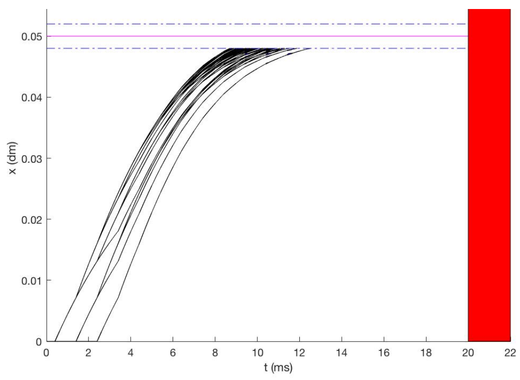


Fig. 10: Reachable sets of the caliper position computed with SpaceEx.

"loc(Monitor)==error". SpaceEx finds a fixed point after 434 iterations and 31.702s. The computation time for the same model without the monitor is 29.427s. As such, the induced overhead is around 7.73%. Note that SpaceEx composes (flattens) the model on-the-fly during reachability analysis.

g) Verification Outcome: The error state is not reachable and the property is satisfied. The reachable set for the caliper position x is portrayed in Figure 10.

VI. Related Work

Conducting hybrid system verification against rich formal specifications is a scientifically and technically challenging problem. The authors in [15] studied the topological aspects of hybrid systems in the context of propositional modal μ-calculus. Mysore, et al., studied the verification problem of semi-algebraic hybrid systems for TCTL (Timed Computation Tree Logic) properties and proved undecidability [41]. Jeannin and Platzer presented in [29] a differential temporal dynamic logic to specify temporal properties of hybrid systems. This logic complemented with a theorem prover could enable verification of nested temporalities for hybrid systems. The authors in [12] studied the verification of hybrid systems with K-liveness but restricted the system model to a small subclass of hybrid automata.

Signal Temporal Logic (STL) was proposed in [38], [39] as a high-level declarative language for expressing properties of hybrid systems. However, it has been mainly applied to the lighter problem of runtime verification (monitoring) of individual hybrid traces (see [42] for the relevant references). More recently, property-based model-checking of hybrid systems was proposed in [13], [14], where the specification language used is HRELTL, a hybrid extension of the discrete-time linear temporal logic enhanced with the regular expression operators. A similar approach of model checking HyLTL, another hybrid extension of LTL, was developed and presented in [7], [8].

In this paper, we opt to use a template language to express informal requirements rather than a full-blown declarative language based on temporal logic. Certainly, a declarative specification language such as STL offers a degree of freedom and flexibility that cannot be easily matched by pattern templates. Nevertheless, we see multiple advantages in choosing this approach.

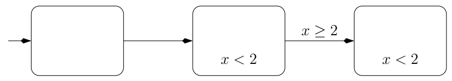


Fig. 11: Observer automaton for φ. The error location is omitted.

We first observe that there is a cultural gap between formal verification methods and the engineers. While the researcher typically appreciates the conciseness and the elegance of a temporal logic formula, an engineer without formal methods background often sees an unintuitive language that is too distant from his design practices and hence requires a steep learning curve to master it. The pattern templates proposed in this paper have the important advantage of being much closer to the natural language used in the informal requirements, while still retaining formal and rigorous semantics. Such templates can play an important role in bridging the gap between formal verification research and its users.

We recurrently encounter similar classes of requirements in many application areas. The main aspects that change between various requirements are specific parameters. As such, we believe that parameterized specification templates are amply sufficient to properly cover most requirements used in practice. This black-box approach enables engineers to use existing libraries of templates with little effort.

Finally, we see a number of technical challenges in applying property-based verification of temporal logic-based specifications to the class of hybrid systems used in this paper. Let us first examine STL specifications, that are defined over continuous-time and real-valued variable domains. An observer for a simple STL specification such as $\Box\Diamond_{[99,100]}(x < 2)$ requires considerable memory resources – the underlying automaton needs 200 real-valued clock variables, resulting in a considerable dimensionality overhead for any model checking approach. In addition, the continuous-time semantics of STL has an extreme precision, both in the time and in the value domain. Consider the STL formula

$$\varphi = \Diamond(y \geq 2 \wedge ((y < 2)\; \mathcal{S}\; \mathsf{true}) \wedge ((y < 2)\; \mathcal{U}\; \mathsf{true})).$$

The formula φ, illustrated in Figure 11 requires the existence of some time t, where y is greater or equal to 2 during a zero duration period (y is strictly smaller than 2 in both the left and the right neighborhood of t). The hybrid automata considered in this paper cannot distinguish events with such precision because the intersection between the location invariants and location guards must be non-empty. It follows that hybrid automata cannot be used as the property φ observers. The specification languages HyLTL and HRELTL are both defined over traces that alternate between continuous trajectories and discrete events. In contrast to our template language, these languages use untimed temporal operators, therefore not being appropriate to express relative real-time constraints between states and events in the system.

The use of pattern templates for system requirements and their (semi–) automatic translation to formal specifications have been proposed earlier. Dwyer et al. [18] were among the first to introduce qualitative specification pattern templates and their translation into different logic expressions. Among others, Konrad and Cheng [35] extended Dwyer's original patterns to the real–time domain. Application of the patterns in the automotive industry can be found in [31], [45]. A generalization to probabilistic pattern templates was proposed in [24]. For discrete systems, there are tools that accept as an input CNL expressions (e.g. in the form of pattern templates) and automatically translate them into formal specifications. Examples of such tools are Stimulus [3], Embedded Specifier [10], AutoFocus3 [28], and SpeAR [6]. Pattern templates for hybrid systems do not differ from pattern templates already available. Yet, no existing tool can translate them into a formal representation that is applicable to hybrid systems and enables the verification of rich properties.

VII. CONCLUSIONS

In this paper, we facilitate the verification of hybrid systems against rich formal requirements by employing pattern templates and we provide a comprehensive toolchain. We define selected patterns in a formalism which is suitable for hybrid automata and applicable over both bounded and unbounded time. For these patterns, we give monitor automata with correctness proofs. The proofs are omitted for lack of space and can be found in the technical report [21]. By composing the monitors with the system model under study, the safety verification problem is transformed into the reachability problem of an error state. Results obtained from an industrial braking use case indicate that monitor automata can facilitate the applicability of hybrid system verification tools to industrial settings. These monitors are applicable to the development process of industries that utilize text-based safety requirements and they yield risk reduction when translating requirements into formal specifications.

ACKNOWLEDGMENT

The authors gratefully acknowledge financial support by the European Commission project UnCoVerCPS under grant number 643921.

REFERENCES

[1] R. Alur. Formal verification of hybrid systems. In *EMSOFT*, 2011.

[2] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical computer science*, 1995.

[3] Argosim. Stimulus. `http://argosim.com`, 2015.

[4] E. Asarin, T. Dang, G. Frehse, A. Girard, C. Le Guernic, and O. Maler. Recent progress in continuous and hybrid reachability analysis. In *Computer Aided Control System Design*, 2006.

[5] A. Balsini, M. Di Natale, M. Celia, and V. Tsachouridis. Generation of simulink monitors for control applications from formal requirements. In *SIES Symposium*, 2017.

[6] A. Bita. SpeAR — specification and analysis for requirements tool. `https://github.com/AFifarek/SpeAR`, 2016.

[7] D. Bresolin. HyLTL: a temporal logic for model checking hybrid systems. In *Hybrid Autonomous Systems Workshop*, 2013.

[8] D. Bresolin. Improving HyLTL model checking of hybrid systems. In *GandALF Symposium*, 2013.

[9] A. A. Bruto da Costa and P. Dasgupta. Formal interpretation of assertion-based features on ams designs. *IEEE Design & Test*, 2015.

[10] BTC. Embedded specifier. `http://www.btc-es.de`, 2015.

[11] A. Busboom, S. Schuler, and A. Walsch. FORMALSPEC: Semi-automatic formalization of system requirements for formal verification. In *ARCH Workshop*, 2016.

[12] A. Cimatti, A. Griggio, S. Mover, and S. Tonetta. Verifying LTL properties of hybrid systems with K-Liveness. In *CAV Conf.*, 2014.

[13] A. Cimatti, M. Roveri, and S. Tonetta. Requirements validation for hybrid systems. In *CAV Conference*, 2009.

[14] A. Cimatti, M. Roveri, and S. Tonetta. HRELTL: A temporal logic for hybrid systems. *Inf. Comput.*, 2015.

[15] J. M. Davoren and A. Nerode. Logics for hybrid systems. *IEEE Proceedings*, 2000.

[16] A. Dokhanchi, B. Hoxha, and G. Fainekos. Metric interval temporal logic specification elicitation and debugging. In *MEMOCODE*, 2015.

[17] L. Doyen, G. Frehse, G. Pappas, and A. Platzer. *Verification of hybrid systems*. Handbook of Model Checking, 2017.

[18] M. B. Dwyer, G. S. Avrunin, and J. C. Corbett. Patterns in property specifications for finite-state verification. In *ICSE Conference*, 1999.

[19] G. Frehse. An introduction to SpaceEx v0.8, 2010.

[20] G. Frehse, A. Hamann, S. Quinton, and M. Woehrle. Formal analysis of timing effects on closed-loop properties of control software. In *RTSS Symposium*, 2014.

[21] G. Frehse, N. Kekatos, and D. Nickovic. Formally correct monitors for hybrid automata. *Verimag Research Report*, 2017.

[22] G. Frehse, C. Le Guernic, A. Donzé, S. Cotton, R. Ray, O. Lebeltel, R. Ripado, A. Girard, T. Dang, and O. Maler. SpaceEx: Scalable verification of hybrid systems. In *CAV Conference*, 2011.

[23] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. Design patterns–elements of reusable object-oriented software. 1995.

[24] L. Grunske. Specification patterns for probabilistic quality properties. In *ICSE Conference*, 2008.

[25] N. Halbwachs, F. Lagnier, and P. Raymond. Synchronous observers and the verification of reactive systems. In *AMAST Conference*, 1993.

[26] T. A. Henzinger, P. W. Kopke, A. Puri, and P. Varaiya. What's decidable about hybrid automata? *JCSS*, 1998.

[27] B. Hoxha, N. Mavridis, and G. Fainekos. VISPEC: a graphical tool for easy elicitation of MTL requirements. In *IROS*, 2015.

[28] F. Huber, B. Schätz, A. Schmidt, and K. Spies. AutoFocus– a tool for distributed systems specification. In *FTRTFT Symposium*, 1996.

[29] J.-B. Jeannin and A. Platzer. dTL2: Differential temporal dynamic logic with nested temporalities for hybrid systems. In *IJCAR*, 2014.

[30] X. Jin, A. Donzé, J. V. Deshmukh, and S. A. Seshia. Mining requirements from closed-loop control models. *TCAD Journal*, 2015.

[31] A. Kane. Runtime monitoring for safety-critical embedded systems. *Dissertation*, 2015.

[32] J. Kapinski, X. Jin, J. Deshmukh, A. Donze, T. Yamaguchi, H. Ito, T. Kaga, S. Kobuna, and S. Seshia. ST-Lib: A library for specifying and classifying model behaviors. *SAE Technical Report*, 2016.

[33] N. Kekatos, M. Forets, and G. Frehse. Constructing verification models of nonlinear Simulink systems via syntactic hybridization. In *CDC Conference*, 2017.

[34] N. Kekatos, M. Forets, and G. Frehse. Modeling the wind turbine benchmark with PWA hybrid automata. In *ARCH Workshop*, 2017.

[35] S. Konrad and B. Cheng. Real-time specification patterns. *ICSE*, 2005.

[36] T. Kuhn. A survey and classification of controlled natural languages. *Computational Linguistics*, 2014.

[37] E. A. Lee and S. A. Seshia. *Introduction to embedded systems: A cyber-physical systems approach*. MIT Press, 2016.

[38] O. Maler and D. Nickovic. Monitoring temporal properties of continuous signals. In *FORMATS Conference*, 2004.

[39] O. Maler and D. Nickovic. Monitoring properties of analog and mixed-signal circuits. *STTT Journal*, 2013.

[40] S. Minopoli and G. Frehse. SL2SX translator: from Simulink to SpaceEx models. In *HSCC Conference*, 2016.

[41] V. Mysore, C. Piazza, and B. Mishra. Algorithmic algebraic model checking II: Decidability of semi-algebraic model checking and its applications to systems biology. In *ATVA*, 2005.

[42] D. Nickovic. Monitoring and measuring hybrid behaviors. A tutorial. In *RV Conference*, 2015.

[43] G. Nicolescu and P. J. Mosterman. *Model-based design for embedded systems*. CRC Press, 2009.

[44] A. Pnueli. The temporal logic of programs. In *FOCS*, 1977.

[45] A. Post, I. Menzel, and A. Podelski. Applying restricted English grammar on automotive requirements–does it work? a case study. In *RREFSQ Workshop*, 2011.

[46] R. R. Rajkumar, I. Lee, L. Sha, and J. Stankovic. Cyber-physical systems: the next computing revolution. In *DAC Conference*, 2010.

[47] T. Strathmann and J. Oehlerking. Verifying properties of an electro-mechanical braking system. In *ARCH Workshop*, 2015.