Pattern Templates and Monitors for Verifying Safety Properties of Hybrid Automata

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

[1] VERIMAG, University of Grenoble-Alpes, Grenoble, France
{firstname.lastname}@univ-grenoble-alpes.fr
[2] GE Global Research, Munich, Germany
{firstname.lastname}@ge.com
[3] Robert Bosch GmbH, Corpor. Research, Stuttgart, Germany
{firstname.lastname}@de.bosch.com
[4] Austrian Institute of Technology, Vienna, Austria
{fistname.lastname}@ait.ac.at

Abstract. This paper aims to facilitate 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 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 available tools for reachability. In this paper, we consider specifications expressed in pattern templates, which are predefined phrases 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 give formal definitions for selected patterns in the formalism of hybrid automata and provide monitors which encode the properties as the reachability of an error state. By composing these monitor automata with the formal model under study, the property can be checked by off-the-shelf reachability tools. We illustrate the workflow on an electromagnetic brake use case.

1 Introduction

Model-based design (MBD) is a paradigm that enables the cost-effective and quick development of complex systems, such as control, energy, and communication 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 [46].

In this methodology, there is typically a sequence of steps that should be followed. The designer first models the physical plant, relying either on first principles or 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 by applying control rules. The composition of the controller and the plant yields the closed-loop (controlled) model. Once such a model is constructed or derived, the designer performs extensive simulation and testing in order to check its behavior against different configuration settings. The goal is to analyze and evaluate the controller design by inspecting the behavior of specific signals or variables over time. The analysis can be extensive and is typically performed with respect to some requirements (also called specifications or objectives). This step is a critical one and is known as validation. 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 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. In some cases, however, the evaluation may remain inconclusive. That is typically due to the fact that the designer manually inspects and assesses simulation behaviors, and they may have to rely on their domain expertise to reason about the design quality [33].

The necessity to provide guarantees of correctness and performance has motivated the development of formal verification techniques and the design of industrially oriented verification tools. Their goal is to guarantee that specifications are satisfied through a rigorous mathematical analysis of the system. 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 industrial 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, frequently have a safety-critical nature, and have applications in various domains such as health care, transportation, automation and robotics [48].

An appropriate mathematical model for design of such systems is hybrid systems [2]. As such, the field of formal verification of hybrid systems has gained wide attention from both academia and industry. Hybrid systems demonstrate joint discrete and continuous behaviors by combining the traditional models for discrete systems with classical differential- and algebraic- equations based models for dynamical systems [3]. Those systems are difficult to analyze, as any kind of nondeterminism in the system, like disturbances, measurement noise, parameter uncertainties, user input, or operating conditions, may have adverse effects on the performance. Different techniques for hybrid system verification have been proposed. They could be broadly divided into 3 categories: approaches based on symbolic representations (e.g. reachability analysis), abstraction, or logic. Survey papers can be found at [2, 20, 40].

Recently, there has been increased interest in fully automated verification tools, such as set-based reachability analysis [38]. That 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 [2]. On the

contrary, however, little effort has been made on formalizing requirements such that they can be verified automatically. The main reason concerns the semantic mismatch between industrial requirements, usually formulated in natural language, and the formal requirements.

In industry, system requirements are usually described in natural language or controlled natural language (CNL). A CNL is a subset of a natural language with a constrained grammar such as to reduce or eliminate ambiguity. The motivation is that requirements written in a CNL can be understood across different professional disciplines (e.g. technical, legal, marketing), but can also be translated into a formal representation and hence be used along a system engineering tool chain [37].

In the case of hybrid systems, the translation from CNL into a formal representation needs to be done manually which introduces the risk of translation errors. In particular, requirements which explicitly mention time or uncertainties in the environment models, may lead to complex expressions. We are therefore proposing a template–based, semi–automated translation of system requirements into a formal representation (so-called monitor automata).

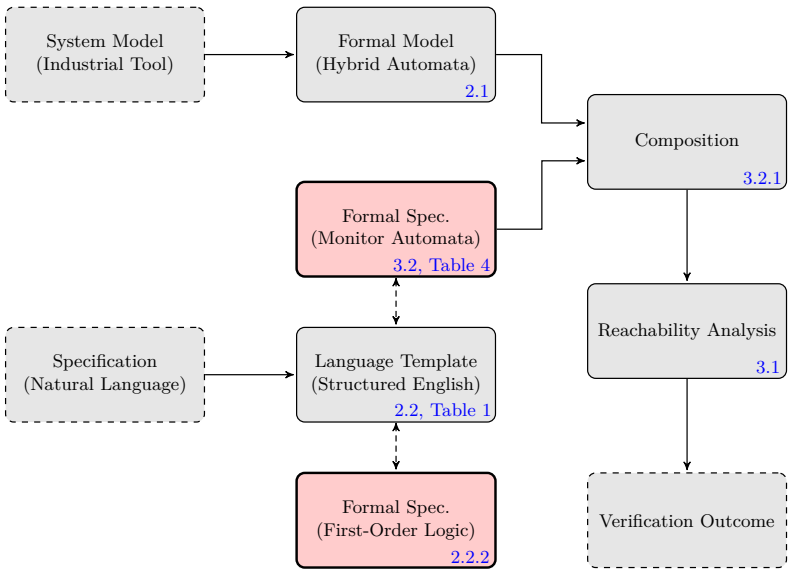


Fig. 1: Verification workflow against formal specifications.

A schematic of the workflow is depicted in Figure 1. A (safety) requirement in CNL is translated into a monitor automaton using pattern templates. The monitor automaton has the same syntax and respects the same semantics as the system model. As such, it can be combined (composed) combined with the system model and fed into a reachability tool. Given that the monitor automa-

ton encodes the forbidden states, the verification of a given set of requirements combined with a given system is successful if the intersection of the reachable states and the forbidden states is empty (see also [28] for the one of the earliest works on monitor automata).

In this paragraph, we cite related work on pattern templates – related work on temporal logic is discussed in Section 5. The use of a CNL notation employing templates for the specification of system requirements and their (semi–) automatic translation for verification has been proposed earlier. Pattern templates (also called specification templates) contain predicates which are to be substituted by logical expressions in order to formulate the actual requirements. Dwyer et al. [22] were among the first to introduce qualitative specification pattern templates and their translation into different logic expressions (e.g. linear temporal logic). Among others, Konrad and Cheng [36] extended Dwyer's original patterns to the real–time domain. Post et al. [47] applied and extended those patterns for the automotive industry. A generalisation to probabilistic specification patterns was introduced in [26]. For purely discrete systems, tools already exist that allow for the input of CNL expressions (e.g. in the form of the above mentioned pattern templates) and automatically translate them into formal expressions. Examples of such tools are Stimulus [5], Embedded Specifier [12], AutoFocus3 [11] and SpeAR [8]. On a system level and in the CNL representation, pattern templates for hybrid systems do not differ from pattern templates already available. However, the output of these tools is in the form of finite state machines, Büchi–automata or general ω–automata, and is not applicable to hybrid systems and the state-based properties we consider. Therefore, existing approaches for automatic translation into formal representation cannot be used.

The remainder of the paper is organized as follows: We present hybrid automata, pattern templates and the formalization of the requirements through pattern templates in Section 2. The monitor automata for the formalized pattern templates are described in Section 3. We show the applicability of the presented approach in Section 4 on an electromechanical brake use case. In Section 5, we give a brief overview of the state of the art. The paper concludes with Section 6.

2 Pattern Templates for Hybrid Automata

In this section, we introduce the framework of hybrid automata and the pattern templates. As a first major contribution, 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 3, we are going to see how to use these templates with reachability tools.

2.1 Hybrid Automata

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

Preliminaries 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 $U|^{\bar{X}}$. Given that a valuation u over X and a valuation v over Y *agree*, i.e., $u{\downarrow_{X \cap \bar{X}}} = v{\downarrow_{X \cap \bar{X}}}$, we use $u \sqcup v$ to denote the valuation w defined by $w{\downarrow_X} = u$ and $w{\downarrow_{\bar{X}}} = v$. Let $const_X(Y) = \{(v, v') | v, v' \in V(X), v{\downarrow_Y} = v'{\downarrow_Y}\}$.

Definition 1 (Hybrid automaton). *[3, 29] A* hybrid automaton

$$H = (\mathsf{Loc}, \mathsf{Lab}, \mathsf{Edg}, X, \mathsf{Init}, \mathsf{Inv}, \mathsf{Flow}, \mathsf{Jump})$$

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*, also called its* alphabet*, which can be used to 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*, which 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 so determines how long the system can remain in the location;*
- *a set of* initial *states* $\mathsf{Init} \subseteq \mathsf{Inv}$*; every behaviour 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 such as*

  $$\dot{\mathbf{x}} = f(\mathbf{x});$$

  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.

Definition 2 (Run semantics). *An* execution *of a hybrid automaton H is a sequence*

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

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

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 duration δ_i is 0.*
2. Jumps: *If $\alpha_i \in \mathsf{Lab}$, 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.*

Note that the strict alternation of trajectories and jumps in Def. 2 is of no particular importance. Two consecutive jumps can be represented by inserting a trajectory with duration zero (which always exists), and two consecutive trajectories can be represented by inserting an uncontrolled assignment jump that does not modify the variables. Having an event at the end of the run will simplify the notation in the remainder of the paper.

2.2 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.

Our work builds upon the pattern templates introduced by Konrad and Cheng in [36]. While in [36], the patterns were formally defined using temporal logics (MTL), these definitions do not immediately carry over to monitoring with hybrid automata. In this respect, we select some common pattern templates, portrayed in Table 1, and define them in a formalism that is suitable for hybrid automata.

2.2.1 Preliminaries

We now introduce some notation to denote in a compact manner the states on runs and the times at which these states are taken by the run. This will allow us to express properties of runs in a clear and concise manner.

Let p be a predicate over the state variables, i.e., a function $\mathbb{R}^X \to \mathbb{B}$. 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 $\mathrm{Runs}(H)$. In the following 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

Table 1: Short overview of Pattern Templates [36]

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 120 seconds 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 model for no longer than 10 seconds at one time."
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 10 millseconds."
bounded response	Restricts the maximum amount of time that passes after a formula holds until another state formula becomes true. ABS System: "From direct client input of and response to rapid deceleration must occur within 0.015 seconds.
bounded invariance	Specifies the minimum amount of time a state formula must hold once another state formula is satisfied. *Engine starter system:* "If error 502 is sent to the Driver Information System, the braking system is inhibited for 10 seconds."

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

$$\mathrm{dom}(r) = \big\{(i, t) \;\big|\; 0 \leq i \leq N-1, t_{i-1} \leq t \leq t_i\} \cup \{(N, t_{N-1})\big\},$$

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

$$\mathrm{dom}_{-T}(r) = \big\{(i, t) \in \mathrm{dom}(r) \;\big|\; t < t_{N-1} - T\big\}.$$

The truncated domain will be used for properties that refer to future events that 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 let $r_{\mathsf{Loc}}(\tau) \in \mathsf{Loc}$ 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

$$\mathrm{Infi}\,(r, q) = \inf_{\tau \in \mathrm{dom}(r)} q(r(\tau)).$$

If r is clear from the context, we use the shorthand

$$\tau_{q.1} = \mathrm{Infi}(r, q).$$

Similarly, we look for the first time that a predicate p holds up to and before an event-time τ',

$$\mathrm{first}\,(r, \tau', q) = \inf_{\tau \in \mathrm{dom}(r), \tau \leq \tau'} q(r(\tau)).$$

To formally denote that a predicate holds at time τ for some nonzero amount of time, we define for a run r, a predicate p, and event-time τ,

$$\mathrm{persists}\,(r, p, \tau) = \quad \exists \delta > 0 : \forall \tau', \tau \leq \tau', d(\tau, \tau') \leq \delta : r(\tau').$$

2.2.2 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. In the following, we can therefore simply define what it means for a run r to satisfy the property ϕ, which we write as $r \models \phi$. Table 2 presents a list of clarifying remarks regarding the pattern templates.

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 $q(r(\tau_q))$ and $\tau \geq \tau_q$, holds $\neg p(r(\tau))$.

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 $q(r(\tau_q))$ and $d(\tau_q, \tau) \geq T$ holds $\neg p(r(\tau))$.

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 either:

Table 2: Remarks on pattern templates.

- The properties in this paper refer to state predicates $q, p, s : \mathbb{R}^X \to \{\text{true}, \text{false}\}$. These predicates describe states, not events. When p, q, s are always true or false, the monitor automata can be simplified.
- State predicates can express timing properties by adding an extra clock to the monitor, so that the time is now a state variable that can be used in q, p and s.
- We show so-called *triggered* versions of the properties, which only take effect after a predicate q holds. A run, for which $!q$ always holds, satisfies the property.
- There are more than one equivalent definitions for the properties (e.g. switch between universal and existential quantifiers). The selection of the most suitable one has been made to reflect the natural language of the pattern templates in Table 4.
- The universal quantifier of an empty set is always true.
- It is possible to check properties both for the bounded and unbounded time horizon. For some patterns, these two cases are distinguished explicitly.
- There is both a linguistic and practical difference between *becomes true* and *holds*. The former could be seen as an edge, i.e. the signal was false earlier and then became true. The latter could describe a property that was always true.
- The monitor automata are nondeterministic because this can lead to more compact automata.

(i) for all $\tau_p^*, \tau_q \in \text{dom}(r)$ with $q(r(\tau_q))$, $\tau_p^* \geq \tau_q$, holds $\neg p(r(\tau_p^*))$ (never q, or never p after q), or

(ii) if $\tau_q \in \text{dom}(r)$ with $q(r(\tau_q))$, then for $\tau_{q.1} = \text{Infi}\,(r, q)$ holds:
   (a) for all $\tau_p^*, \tau_{\bar{p}}^* \in \text{dom}(r)$ with $p(r(\tau_p^*))$, $\neg p(r(\tau_{\bar{p}}^*))$, $\tau_{q.1} \leq \tau_p^* < \tau_{\bar{p}}^*$, $d(\tau_{q.1}, \tau_{\bar{p}}^*) > T$ (p not becoming false within T after $\tau_{q.1}$), and
   (b) for all $\tau_p, \tau_{\bar{p}}, \tau'_{\bar{p}} \in \text{dom}(r)$ with $\tau_{q.1} \leq \tau_{\bar{p}} < \tau_p < \tau'_{\bar{p}}$, $p(r(\tau_p))$, $\neg p(r(\tau_{\bar{p}}))$ and $\neg p(r(\tau'_{\bar{p}}))$, it holds that $d(\tau_{\bar{p}}, \tau'_{\bar{p}}) > T$ (violations of p are more than T apart).

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 for all $\tau_q \in \text{dom}(r)$ with $q(r(\tau_q))$ either

(i) for all $\tau \in \text{dom}(r)$ with $\tau \geq \tau_q$, $\neg p(r(\tau))$ (never q, or p never holds after q), or

(ii) for all $\tau_p, \tau'_p \in \text{dom}(r)$ with $\tau_p \geq \tau_q$, $p(r(\tau_p))$, $p(r(\tau'_p))$ one of the following holds:
   (a) $d(\tau_p, \tau'_p) < T$ (τ'_p is early enough, including the $\tau_p = \tau'_p$ case), or
   (b) there is a $\tau_{\bar{p}}$ such that $\neg p(r(\tau_{\bar{p}}))$ and $\tau_p < \tau_{\bar{p}} < \tau'_p$ (p is false in between).

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 $\tau_q \in \mathrm{dom}(r)$ with $q(r(\tau_q))$ both following criteria hold:

(i) for all $\tau_p \in \mathrm{dom}(r)$ with $p(r(\tau_p))$ and $\tau_p \geq \tau_q$ there is a $\tau'_p \in \mathrm{dom}(r)$ such that $\tau_p < \tau'_p$, $d(\tau'_p, \tau_p) \leq T$ and $p(r(\tau'_p))$ (τ_p's with distance less than T).
(ii) there is a $\tau_p \in \mathrm{dom}(r)$ with $\tau_p \geq \tau_q$, $p(r(\tau_p))$ such that $d(\tau_q, \tau_p) \leq T$. (distance between τ_q and first τ_p is less than T).

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

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

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 for all $\tau_q \in \mathrm{dom}(r)$ with $q(r(\tau_q))$ one of the following holds:

(i) for all $\tau \in \mathrm{dom}(r)$ with $\tau \geq \tau_q$, $\neg p(r(\tau))$ (never q, or p never holds after q), or
(ii) for all $\tau_p \in \mathrm{dom}(r)$ with $\tau_p \geq \tau_q$ and $p(r(\tau_p))$, there is a $\tau_s \in \mathrm{dom}(r)$ such that $\tau_p \leq \tau_s$, $d(\tau_s, \tau_p) \leq T$ and $\mathrm{persists}\,(r, \tau_s, s)$.

For a bounded time horizon, $r \models \phi$ iff one of the following holds:

(i) for all $\tau_q, \tau \in \mathrm{dom}(r)$ with $q(r(\tau_q))$, $\tau \geq \tau_q$, holds $\neg p(r(\tau))$ (never q, or p never holds after q), or
(ii) for all $\tau_q, \tau_p \in \mathrm{dom}_{-T}(r)$ with $\tau_p \geq \tau_q$, $q(r(\tau_q))$ and $p(r(\tau_p))$, there is a $\tau_s \in \mathrm{dom}(r)$ such that $\tau_p \leq \tau_s$, $d(\tau_p, \tau_s) \leq T$ and $\mathrm{persists}\,(r, \tau_s, s)$.

Remark 1. The reason why we require $\tau \in \mathrm{dom}_{-T}(r)$ in the bounded time horizon (with the restricted domain being right-open) is the following: We assume an optimistic interpretation of bounded runs, in the sense that 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 one of the following holds:

(i) for all $\tau_q \in \mathrm{dom}(r)$ with $q(r(\tau_q))$, there is no τ_p with $\tau_p \geq \tau_q$ such that $p(r(\tau_p))$ (never q, or p never holds after q), or
(ii) for all $\tau_p \in \mathrm{dom}(r)$ with $\tau_p \geq \tau_q.1$, $p(r(\tau_p))$, and for all $\tau \in \mathrm{dom}(r)$ such that $\tau_p \leq \tau$, $d(\tau_p, \tau) < T$, the predicate $s(r(\tau))$ is true.

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

2.3 Application of Formalized Pattern Templates

Formalizing practical requirements is considered to be an important but difficult task even for experts [19,31]. This section aims to illustrate the use of templates and is divided in two parts. In the first, we express common control objectives with our patterns.

2.3.1 Typical Control Objectives

Some common control objectives can be expressed with the formalized pattern templates. In the following, we consider the untriggered version of the requirements, which is equal to setting $q :=$ true in our patterns. For simplicity, we assume a reference signal x_{ref} that is positive and constant as well as that $x(0) < x_{ref}$ holds.

- Safety: *The state x of the system should always remain inside the acceptable operating range expressed as the safe region S.*
  This requirement is matched by the **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.*
  This property could be encoded as the **bounded response** pattern, where $p :=$ true and $s := \{d(x, x_{target}) \leq \varepsilon\}$, with $d(x, y)$ being a function that computes the distance between states x and y.

- Overshoot: *The state x of the system should not exceed an overshoot of $ov\%$ with respect to the reference signal (x_{ref}).*
  This property can be formulated as an **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.*
  This property can be described by the **absence (timed)** pattern, where $T := T_{set}$, p:= $\{x \leq (100 - per)\% \cdot x_{ref} \vee x \geq (100 + per)\% \cdot x_{ref}\}$.

– Rise-Time: *The state x of the system should reach 90% of the reference x_{ref} at time T_{rise}.*
This property can be mapped to the **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.*
This property can be expressed with the **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 more than one pattern templates. Note that the scenario of a varying reference signal can be captured with the introduction of predicate q.

2.3.2 Industrial Use Cases

This section shows the applicability of the pattern templates given within this paper to different application domains. We have exemplary selected three different application domains, wind turbines [35, 49], automated driving [30, 50] and braking systems [24, 51]. Table 3 summarizes some of the application requirements and their mapping to pattern templates. Some of the requirements can be difficult to translate into a formal representation without a given pattern template. Note that the universality can be described via the absence property, i.e. replacing p in the definitions by $!p$.

3 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.

3.1 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 also satisfies one of the jump conditions (some intermediate points along the trajectory are usually kept as well). Then one picks one of the successor states of the jump and repeats the process. Like numerical simulation, reachability analysis directly follows the transition semantics of hybrid automata, but considers sets of states instead of single states [20].

Table 3: Representative examples of requirements in natural language and their mapping to pattern templates for different application domains.

Appl. Domain	Requirement	Category
wind turbines	The pitch rate of the turbine blades shall be smaller than the maximal pitch rate.	universality
	The absolute difference between the commanded pitch angle and the measured pitch angle can only be larger than the maximum difference for less than c time units.	maximum duration
	The absolute difference between two individual pitch angles can only be larger than the maximum difference for less than c time units.	maximum duration
automated driving	The largest communication sequence flow duration shall be less than *TBD* seconds.	maximum duration
	When an acknowledgeable message arrives, an *ACK* message shall be sent to the sender by the receiver within the maximal waiting time.	maximum duration (triggered)
	If an *ACK* message exceeds the maximal waiting time, the message being acknowledged shall be considered as lost.	universality
	If a message is considered as lost, it shall be resent.	
	A vehicle shall send `MVR_FINISHED` messages to its session partner after finishing successfully its planned manoeuvre.	universality (triggered)
	After a vehicle i received a propose message with the information {ID-x, constraints-x, Tfinish-x} and after vehicle i sent an accept message with ID-x, every planned contingency manoeuvre of vehicle i must satisfy constraints-x, until the point of time Tfinish-x is reached.	minimum duration (triggered)
	After starting execution of the manoeuvre primitive k it is always the case that if the actual disturbances and measurement errors are below error-bounds-k then the deviation of the state from the reference trajectory is below state-bounds-k for at least duration-k.	bounded response (triggered)
brake system	The caliper must reach $x_0 = 0.05$ dm after the braking request is issued within 20 ms with a precision of 4%.	bounded response
	The caliper speed at contact must be below 2 mm/s.	absence

Just like numerical simulation, reachability computation has to use approximations if the dynamics of the system are complex. Working with sets instead of points, approximate reachability can be conservative in the sense that the computed sets are sure to cover all solutions. Computation costs generally increase sharply in terms of the number of continuous variables. Scalable approximations are available for certain types of dynamics, as discussed later in this section, 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, e.g., in [6, 20].

The reachable set consists of all the states that can be visited by a trajectory of the hybrid system starting in a specified set of initial states. Reachability analysis has often been motivated by safety verification, which consists in checking whether the intersection of the reachable set with a set of bad 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 bad (forbidden) states, the hybrid system is guaranteed to be safe [6].

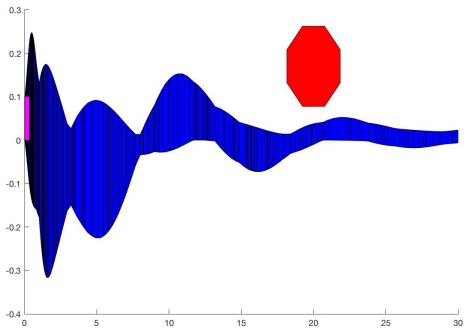


Fig. 2: Safety verification of a 20-dimensional helicopter model [25]. Conducted reachability analysis with SpaceEx tool over a time horizon of 30 seconds. Blue: reachable sets, red: unsafe area, magenta: initial set.

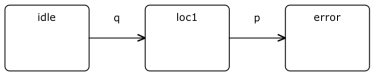
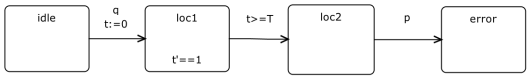
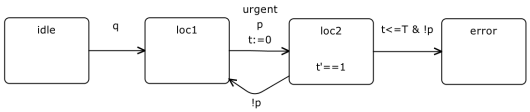
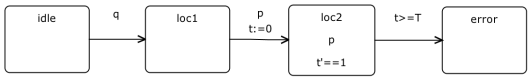
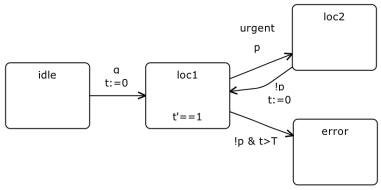
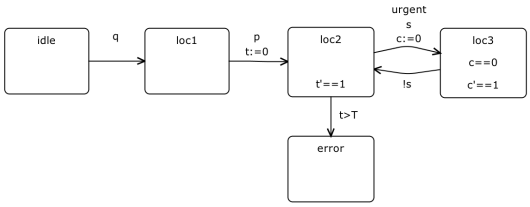
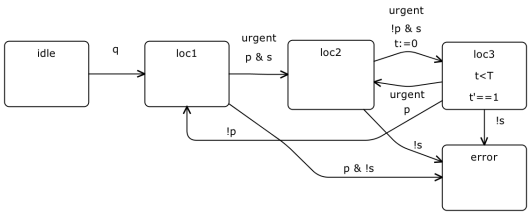
3.2 Monitor Automata for Reachability

In this section, we define monitor automata that, composed with the system under test, encode the requirements as reachability properties as follows. 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 monitor automata are shown in Table 4.

3.2.1 Parallel Composition

We now give a formal definition of the standard way to couple two hybrid automata. We will use this operation to connect the system under test with its monitor. Intuitively, both automata must agree on every change of a variable. The operator is similar to the composition operator in [4].

Table 4: Pattern templates and translation to monitor automata.

Pattern name	**Language Template**	**Monitor Automaton**
absence	After q, it is never the case that p holds.	
absence (timed)	When T time units are measured, after q was first satisfied, it is never the case that p holds.	
minimum duration	After q, it is always the case that once p becomes satisfied, it holds for at least T time units.	
maximum duration	After q, it is always the case that once p becomes satisfied, it holds for less than T time units.	
bounded recurrence	After q, it is always the case that p holds (for nonzero time) at least every T time units.	
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.	
bounded invariance	After q, it is always the case that if p holds, then s holds for at least T time units.	

The jump relations of synchronized transitions result from the conjunction of the participating transitions. Independent transitions, i.e., those that do not synchronize, are allowed to change variables arbitrarily and the variables over which their jump relation is not defined are set to remain constant.

Definition 3 (Composition of HA). *The* parallel composition *of hybrid automata H_1 and H_2 is the hybrid automaton $H = H_1||H_2$*

- $Loc = Loc_1 \times Loc_2$,
- $Lab = Lab_1 \cup Lab_2$,
- $Edg = \{((\ell_1, \ell_2), \alpha, (\ell_1', \ell_2')) \mid (\alpha \in Lab_1 \Rightarrow (\ell_1, \alpha, \ell_1')) \wedge (\alpha \in Lab_2 \Rightarrow (\ell_2, \alpha, \ell_2'))\}$,
- $X = X_1 = X_2$ *(by assumption),* $Y = Y_1 \cup Y_2$, $U = (U_1 \cup U_2) \setminus Y$,
- $Jump((\ell_1, \ell_2), a, (\ell_1', \ell_2'))$ *with* $\mu = \{(v, v') \in \mu_i\}$ *iff for* $i = 1, 2$,
  - $a \in Lab_i$ *and* $(\ell_i, a_i, \mu_i, \ell_i') \in Edg_i$, *or*
  - $a \notin Lab_i$, $\ell_i' = \ell_i$, *and* $\mu_i = const_{X_i}(Z_i)$, *where* $Z_1 = Y_1 \setminus Y_2$ *and* $Z_2 = Y_2 \setminus Y_1$;
- $Flow(\ell_1, \ell_2) = Flow_1(\ell_1) \cap Flow_2(\ell_2)$;
- $Inv(\ell_1, \ell_2) = Inv_1(\ell_1) \cap Inv_2(\ell_2)$;
- $Init(\ell_1, \ell_2) = Init_1(\ell_1) \cap Init_2(\ell_2)$.

Without loss of generality we can assume that H and M have the same variables. If M has a variable not in H, e.g., a clock variable for measuring the time between events, we can add it to H without restricting it in the invariants, guards, or flows. Note that all transitions in M have the label τ, so they do not synchronize with any transitions in H.

A run $r_{H||M}$ in $H||M$ is given by locations $\ell_i = (\ell_i^H, \ell_i^M)$, continuous states $\mathbf{x}_i$, trajectories ξ_i, durations δ_i, and labels α_i. Let r_H be the projection of the run onto H, obtained by replacing ℓ_i with ℓ_i^H, and let r_M be the projection of the run onto M, obtained by replacing ℓ_i with ℓ_i^M and α_i with τ. Then by definition, we have that for any run $r_{H||M}$ in $\mathrm{Runs}(H||M)$, $r_H \in \mathrm{Runs}(H)$ and $r_M \in \mathrm{Runs}(M)$.

3.2.2 Operations on Runs

We use the following shorthand notation to improve the readability of the proofs. As shorthand, we will define a run by the sequence $(\ell_i, \mathbf{x}_i, \delta_i, \xi_i, \alpha_i)_{i=0,\ldots,N-1}$. Given a run r and an event-time $\tau^* = (k^*, t^*) \in \mathrm{dom}(r)$, the run can be split into the *prefix* up to τ^*, and the *postfix* after τ^*. The prefix is extended with a silent transition, which by definition can be injected anywhere:

$$\mathrm{prefix}\,(r, (k^*, t^*)) = (\ell_i, \mathbf{x}_i, \delta_i, \xi_i, \alpha_i)_{i=0,\ldots,k^*-1}; (\ell_{k^*}, \mathbf{x}_{k^*}, t^* - t_{k^*-1}, \xi_{k^*}, \tau). \quad (1)$$

$$\mathrm{postfix}\,(r, (k^*, t^*)) = \big(\ell_{k^*}, r(k^*, t^*), \delta_{k^*} - t_{k^*-1}, \xi^*, \alpha_{k^*}\big); \\ (\ell_i, \mathbf{x}_i, \delta_i, \xi_i, \alpha_i)_{i=k^*+1,\ldots,N-1}, \quad (2)$$

where $r(k^*, t^*) = \xi_{k^*}(t^* - t_{k^*-1})$, and $\xi^*(t) = \xi_{k^*}(t - t_{k^*-1})$ is the trajectory $\xi_{k^*}(t)$ shifted backwards in time by t_{k^*-1}. Similarly, the *infix* between event-times $\tau_a = (k_a, t_a) \in \mathrm{dom}(r)$, $\tau_b = (k_b, t_b) \in \mathrm{dom}(r)$, with $\tau_a \leq \tau_b$, is

$$\mathrm{infix}\,(r, (k_a, t_a), (k_b, t_b)) \;=\; \mathrm{prefix}\,(\mathrm{postfix}\,(r, (k_a, t_a)), (k_b - k_a, t_b - t_a)). \quad (3)$$

It is straightforward that the concatenation

$$\mathrm{prefix}\,(r, \tau) \;;\; \mathrm{postfix}\,(r, \tau)$$

is a run of H. Similarly, the concatenation

$$\mathrm{prefix}\,(r, \tau_a) \;;\; \mathrm{infix}\,(r, \tau_a, \tau_b) \;;\; \mathrm{postfix}\,(r, \tau_b)$$

is a run of H. With a slight abuse of notation, we write $r \times \ell^*$ to denote the run $(\ell_i \times \ell^*, \mathbf{x}_i, \delta_i, \xi_i, \alpha_i)_{i=0,\ldots,N-1}$. This is not necessarily a run of $H||M$, but it can be one, such as under the following condition.

Lemma 1. *Let $r = (\ell_i, \mathbf{x}_i, \delta_i, \xi_i, \alpha_i)$ be a run of H. If a location ℓ_M in M has (i) no invariant constraints and (ii) no urgent outgoing transitions, then $r \times \ell_M$ is a run of $H||M$.*

Lemma 2. *Let $r = (\ell_i, \mathbf{x}_i, \delta_i, \xi_i, \alpha_i)$ be a run of H. If a location ℓ_M in M has (i) no invariant constraints and (ii) one urgent outgoing transition with guard condition p, leading to location ℓ'_M that has (iii) no invariant constraints and (iv) no urgent outgoing transitions then*

$$\mathrm{prefix}\,(r, \tau_{p.1}) \times \ell_M \;;\; \mathrm{postfix}\,(r, \tau_{p.1}) \times \ell'_M$$

is a run of $H||M$, where $\tau_{p.1} = \mathrm{Infi}\,(r, p)$ is the smallest event time where p holds.

We call a monitor M non-blocking if for any run r_H of H, there is a corresponding run $r_{H||M}$ of $H||M$ such that r_H is the projection of $r_{H||M}$ onto H. Simply put, there is no deadlock that caused a run to be terminated.

3.2.3 Correctness Proofs

A monitor automaton is correct if its error location is reachable exactly when the system H violates the property. Formally, let h be a run of H that violates a given property ϕ. Then we first show (a) that there exists a run r of $H||M$ that reaches the error location. Second, we show (b) that for any run r of $H||M$ that reaches the error location, the run projected onto H violates the property. We intuitively explain the proofs for the **absence** pattern. All the remaining proofs can be found at the Appendix.

Sufficient Conditions.
Since $r \not\models \phi$, there exist $\tau_q, \tau_p \in \mathrm{dom}(r)$ with $q(r(\tau_q))$, $\tau_p \geq \tau_q$, and $p(r(\tau_p))$. With Lemma 1 and the definition of a jump,

$$\mathrm{prefix}\,(h, \tau_q) \times \mathrm{idle} \;;\; \mathrm{infix}\,(h, \tau_q, \tau_p) \times \mathrm{loc1} \;;\mathrm{postfix}\,(h, \tau_p) \times \mathrm{error}$$

is a run of $H||M$.

Necessary Conditions.
To get from idle to error, M had to take first a transition with guard q and then a transition with guard p. Consequently, there exist τ_q and τ_p with $\tau_q \leq \tau_p$, $q(r_H(\tau_q))$ and $p(r_H(\tau_p))$. τ_q and τ_p are witnesses that violate ϕ.

4 Application Example

In this section, we illustrate the workflow on an industrial use case on electromagnetic brakes and highlight how the introduced pattern templates and associated monitor automata can facilitate the verification process.

4.1 Toolchain

There are several tools for hybrid systems reachability analysis, such as C2E2 [21], CORA [1], Flow* [14], HyCreate [7], HyReach [39], SoapBox [27], SpaceEx [25]. In the context of this work, we use SpaceEx; a tool for set-based reachability analysis. SpaceEx is suitable for safety verification problems, which are reachability problems with forbidden states.

SpaceEx supports hybrid automata models and provides hierarchy, templates and instantiations of components. The composition of multiple hybrid automata is supported by SpaceEx and it is automatically conducted by the tool, once the model is specified. The design of the formal model is facilitated with the use of SpaceEx Model Editor (MO.E.), a graphical editor for creating models of complex hybrid systems out of nested components [23].

The construction of verification models out of simulation models (Simulink), which are supported by SpaceEx and other available tools, is considered in [34, 43]. A tool, formalSpec, that enables the translation of requirements in CNL to the corresponding monitor automata in SpaceEx is presented in [13].

4.2 Electromagnetic Brake Use Case

The electromagnetic brake (EMB) use case is described in [51]. The requirements that shall be enforced are presented in [24]. The proposed workflow is shown in Figure 3.

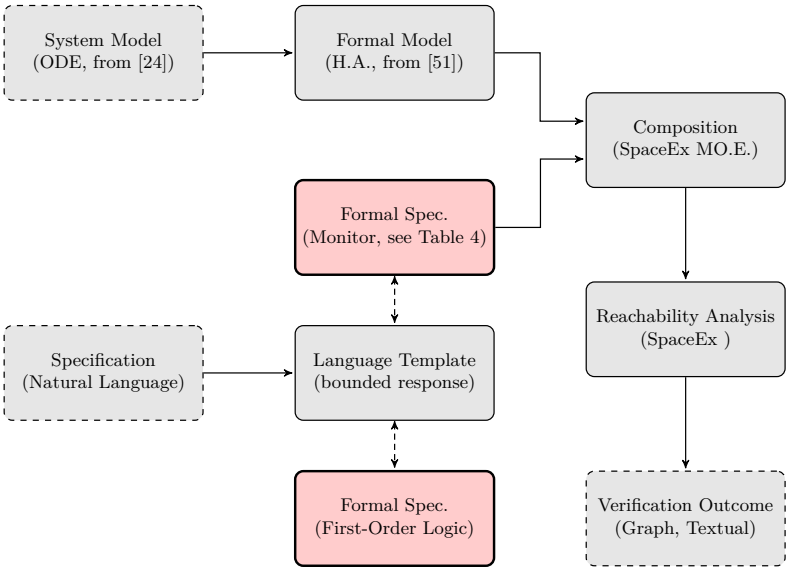


Fig. 3: Verification workflow against formal specifications (ODE: ordinary differential equations, H.A.: hybrid automata, MO.E.: Model Editor).

Industrial Model: The system under analysis consists of an experimental electro-mechanical braking system, together with its controller, implemented in software. The controller comprises both a feedback and feedforward control part. Figure 4 shows an illustration of the the braking system.

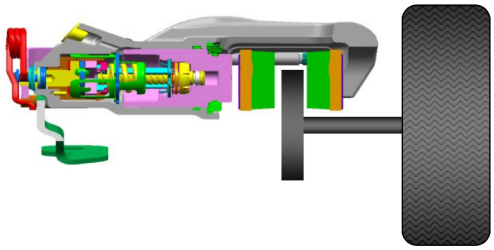


Fig. 4: Schematic of the electro-mechanical brake with electrical engine on left hand side, brake disc connected to wheel on right hand side and brake caliper around the brake disc [51].

Formal Model: The braking system can be described with PWA hybrid automata. It is expressed in SpaceEx format in [51] and it consists of 8 base components (single HA) and 4 network components (networks of HA).

Specifications: Let us recall the two braking specifications provided in 3.

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 caliber 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.

Monitor: For the first specification, described by the bounded response pattern, we use the corresponding monitor automaton of Table 4. We can either replace manually the p, q, s, and T values, or use the formalSpecs tool [13]. Then, the monitor automaton in the SpaceEx Model Editor is shown in Figure 5.

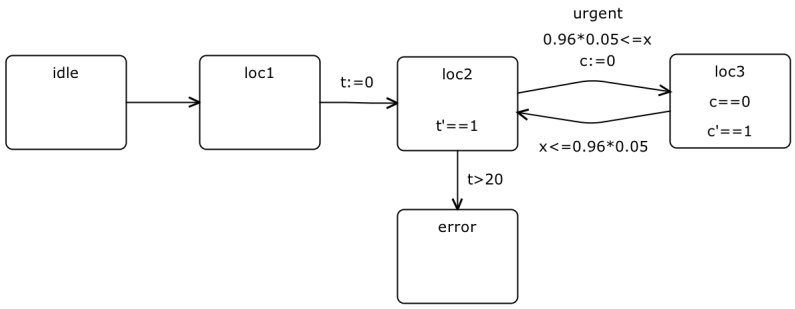


Fig. 5: Monitor automaton of the bounded response pattern (in Model Editor).

Composition This step corresponds to the parallel composition of the formal model with the corresponding monitor. In essence, the variables/signals that appear in the monitor should be connected with the corresponding variables/signals of the formal model. In our case, only the caliber position x should be considered. The variables t and c are local and only used inside the monitor automaton. FIgure 6 shows the composed system in the Model Editor.

In principle, this mapping/binding step could be automated and performed with the formalSpecs tool.

Reachability Analysis SpaceEx is used for computing the reachable sets. The safety verification problem is tackled by introducing as a set of forbidden states and checking whether the error state is reachable or not. In particular, the forbidden state is "loc(monitor_1)==error". The, we run SpaceEx for approximately 40 seconds and it finds a fixed point after 434 iterations.

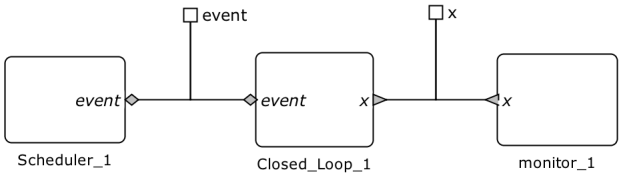


Fig. 6: Composition of formal model (scheduler & closed loop) with the monitor automaton (bounded response).

Verification Outcome The verification outcome is that the error state is not reachable and the property is satisfied. This result is displayed in two different ways. First, there is a textual response streamlining whether the forbidden states have been reached or not, and there is a plot showing the evolution of the reachable set for the caliber position x. The graph is portrayed in Figure 7.

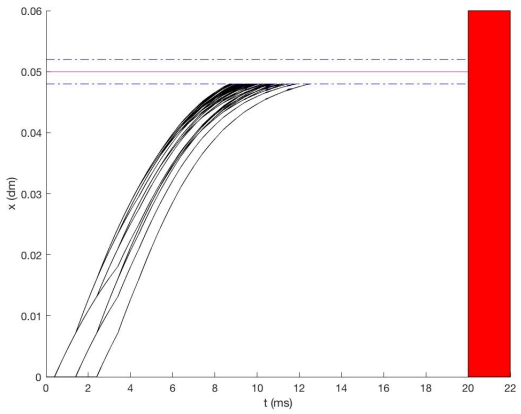


Fig. 7: Reachable sets of the caliber position computed with SpaceEx

5 Related Work

Formal verification of hybrid systems is a scientifically and technically challenging problem. Historically, the main emphasis has been on developing efficient reachability algorithms, thus improving the scalability of the underlying methods and tools. In contrast, there has been relatively little effort invested in enabling verification of hybrid systems against rich formal specifications.

The authors in [18] 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 [44]. Jeannin and Platzer presented in [32] 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 [15] 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 [41, 42] 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 [45] for the relevant references). More recently, property-based model-checking of hybrid systems was proposed in [16,17], 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 [9, 10].

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.

We first observe that there is a cultural gap between formal verification methods and the engineers. While the researcher typically appreciates the beauty, 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 her design practices and hence requires a steep learning curve to master it. 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.

In addition, we recurrently encounter similar classes of requirements in many application areas. The main aspects that change between various requirements are specific parameters. As a consequence, 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

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

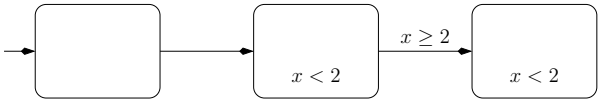


Fig. 8: Observer automaton for φ. The error location is omitted.

The formula φ, illustrated in Figure 8 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 neighbourhood 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.

6 Conclusions

In this paper, we tackle the problem of hybrid systems verification against formal requirements. We rely upon the notion of pattern templates, as they have shown promising results in discrete systems and are close to the natural language. We define these 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. Then, we illustrate how the introduced pattern templates and associated monitors can facilitate the verification process. By composing the monitors with the system model under study, the safety verification problem is transformed into the reachability problem of an error state. We show The results obtained from an industrial braking case study indicate that template monitors can facilitate the applicability of hybrid system verification tools to industrial settings.

From an industrial viewpoint, these monitor automata can be reused across different domains and are expected to add a good level of risk reduction when translating requirements to a formal specification. Doing so does not require any knowledge of formal methods or hybrid systems by the user. In addition, they are applicable to any development process in every industry that starts with text based safety requirements (which is a de-facto standard).

Acknowledgments

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

References

1. M. Althoff. An introduction to CORA 2015. In *ARCH@ CPSWeek*, pages 120–151, 2015.
2. R. Alur. Formal verification of hybrid systems. In *Embedded Software (EMSOFT), 2011 Proceedings of the International Conference on*, pages 273–278. IEEE, 2011.
3. 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*, 138(1):3–34, 1995.
4. R. Alur, T. A. Henzinger, and P.-H. Ho. Automatic symbolic verification of embedded systems. *IEEE Transactions on Software Engineering*, 22(3):181–201, 1996.
5. Argosim. Stimulus. `http://argosim.com`, 2015.
6. 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 IEEE International Conference on Control Applications, 2006 IEEE International Symposium on Intelligent Control, 2006 IEEE*, pages 1582–1587. IEEE, 2006.
7. S. Bak. HyCreate: A tool for overapproximating reachability of hybrid automata. *Retrieved January*, 17:2016, 2013.
8. A. Bita. SpeAR — specification and analysis for requirements tool. `https://github.com/AFifarek/SpeAR`, accessed on March 11, 2016.
9. D. Bresolin. HyLTL: a temporal logic for model checking hybrid systems. In *Proceedings Third International Workshop on Hybrid Autonomous Systems, HAS 2013, Rome, Italy, 17th March 2013.*, pages 73–84, 2013.
10. D. Bresolin. Improving HyLTL model checking of hybrid systems. In *Proceedings Fourth International Symposium on Games, Automata, Logics and Formal Verification, GandALF 2013, Borca di Cadore, Dolomites, Italy, 29-31th August 2013.*, pages 79–92, 2013.
11. M. Broy, F. Huber, and B. Schätz. AutoFocus — ein Werkzeugprototyp zur Entwicklung eingebetteter Systeme. In *Informatik Forschung und Entwicklung*, pages 121–134, 1999.
12. BTC. Embedded specifier. `http://www.btc-es.de`, 2015.
13. A. Busboom, S. Schuler, and A. Walsch. formalSpec: Semi-automatic formalization of system requirements for formal verification. In *ARCH@ CPSWeek*, pages 106–114, 2016.
14. X. Chen, E. Ábrahám, and S. Sankaranarayanan. Flow*: An analyzer for non-linear hybrid systems. In *International Conference on Computer Aided Verification*, pages 258–263. Springer, 2013.
15. A. Cimatti, A. Griggio, S. Mover, and S. Tonetta. Verifying LTL properties of hybrid systems with k-liveness. In *International Conference on Computer Aided Verification*, pages 424–440. Springer, 2014.
16. A. Cimatti, M. Roveri, and S. Tonetta. Requirements validation for hybrid systems. In *Computer Aided Verification, 21st International Conference, CAV 2009, Grenoble, France, June 26 - July 2, 2009. Proceedings*, pages 188–203, 2009.

17. A. Cimatti, M. Roveri, and S. Tonetta. HRELTL: A temporal logic for hybrid systems. *Inf. Comput.*, 245:54–71, 2015.
18. J. M. Davoren and A. Nerode. Logics for hybrid systems. *Proceedings of the IEEE*, 88(7):985–1010, 2000.
19. A. Dokhanchi, B. Hoxha, and G. Fainekos. Metric interval temporal logic specification elicitation and debugging. In *Formal Methods and Models for Codesign (MEMOCODE), 2015 ACM/IEEE International Conference on*, pages 70–79. IEEE, 2015.
20. L. Doyen, G. Frehse, G. Pappas, and A. Platzer. Verification of hybrid systems. In *Edmund M. Clarke, Thomas A. Henzinger and Helmut Veith, editors, Handbook of Model Checking*. Springer, 2017.
21. P. S. Duggirala, S. Mitra, M. Viswanathan, and M. Potok. C2E2: A verification tool for stateflow models. In *TACAS*, pages 68–82, 2015.
22. M. B. Dwyer, G. S. Avrunin, and J. C. Corbett. Patterns in property specifications for finite-state verification. In *Proc. 21th Int. Conf. Software Engineering*, pages 411–420, 1999.
23. G. Frehse. An introduction to SpaceEx v0.8, 2010.
24. G. Frehse, A. Hamann, S. Quinton, and M. Woehrle. Formal analysis of timing effects on closed-loop properties of control software. In *Real-Time Systems Symposium (RTSS), 2014 IEEE*, pages 53–62. IEEE, 2014.
25. 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 *Computer Aided Verification*, pages 379–395. Springer Berlin/Heidelberg, 2011.
26. L. Grunske. Specification patterns for probabilistic quality properties. In *Proc. 30th Int. Conf. Software Engineering*, pages 31–40, 2008.
27. W. Hagemann, E. Möhlmann, and A. Rakow. Verifying a PI controller using SoapBox and Stabhyli: Experiences on establishing properties for a steering controller. In *1St int. workshops on applied verification for continuous and hybrid systems (ARCH'14), EPic series in computer science*, volume 34, 2014.
28. N. Halbwachs, F. Lagnier, and P. Raymond. Synchronous observers and the verification of reactive systems. In *3rd Int. Conf. on Algebraic Methodology and Software Technology, AMAST'93*. Springer Verlag, 1993.
29. T. A. Henzinger, P. W. Kopke, A. Puri, and P. Varaiya. What's decidable about hybrid automata? *Journal of Computer and System Sciences*, 57:94–124, 1998.
30. D. Hess, C. Löper, and T. Hesse. Safe cooperation of automated vehicles, 2017.
31. B. Hoxha, N. Mavridis, and G. Fainekos. VISPEC: a graphical tool for easy elicitation of mtl requirements. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, Hamburg, Germany*, 2015.
32. J.-B. Jeannin and A. Platzer. dtl2: Differential temporal dynamic logic with nested temporalities for hybrid systems. In *International Joint Conference on Automated Reasoning*, pages 292–306. Springer, 2014.
33. X. Jin, A. Donzé, J. V. Deshmukh, and S. A. Seshia. Mining requirements from closed-loop control models. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 34(11):1704–1717, 2015.
34. N. Kekatos, M. Forets, and G. Frehse. Constructing verification models of nonlinear Simulink systems via syntactic hybridization. In *Conference on Decision and Control (CDC)*. IEEE, 2017.
35. N. Kekatos, M. Forets, and G. Frehse. Modeling the wind turbine benchmark with pwa hybrid automata. In *ARCH17. 4th International Workshop on Applied Verification of Continuous and Hybrid Systems*, volume 48 of *EPiC Series in Computing*, pages 100–113. EasyChair, 2017.

36. S. Konrad and B. H. C. Cheng. Real-time specification patterns. In *Proc. 27th Int. Conf. Software Engineering*, pages 372–381. IEEE, 2005.
37. T. Kuhn. A survey and classification of controlled natural languages. *Computational Linguistics*, 40(1):121–170, 2014.
38. E. A. Lee and S. A. Seshia. *Introduction to embedded systems: A cyber-physical systems approach*. MIT Press, 2016.
39. I. B. Makhlouf, N. Hansen, and S. Kowalewski. HyReach: A reachability tool for linear hybrid systems based on support functions. In *ARCH@ CPSWeek*, pages 68–79, 2016.
40. O. Maler. Algorithmic verification of continuous and hybrid systems. In *Infinity*, 2013.
41. O. Maler and D. Nickovic. Monitoring temporal properties of continuous signals. *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*, pages 71–76, 2004.
42. O. Maler and D. Nickovic. Monitoring properties of analog and mixed-signal circuits. *International Journal on Software Tools for Technology Transfer*, 15(3):247–268, 2013.
43. S. Minopoli and G. Frehse. SL2SX translator: from simulink to spaceex models. In *Proceedings of the 19th International Conference on Hybrid Systems: Computation and Control*, pages 93–98. ACM, 2016.
44. 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*, volume 3707, pages 217–233, 2005.
45. D. Nickovic. Monitoring and measuring hybrid behaviors. A tutorial. In *Runtime Verification - 6th International Conference, RV 2015 Vienna, Austria, September 22-25, 2015. Proceedings*, pages 378–402, 2015.
46. G. Nicolescu and P. J. Mosterman. *Model-based design for embedded systems*. CRC Press, 2009.
47. A. Post, I. Menzel, and A. Podelski. *Requirements Engineering: Foundation for Software Quality*, chapter Applying Restricted English Grammar on Automotive Requirements—Does it Work? A Case Study, pages 166–180. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.
48. R. R. Rajkumar, I. Lee, L. Sha, and J. Stankovic. Cyber-physical systems: the next computing revolution. In *Proceedings of the 47th Design Automation Conference*, pages 731–736. ACM, 2010.
49. S. Schuler, F. D. Adegas, and A. Anta. Benchmark problem: hyrid modelling of a wind turbine. In *Proc. of the Workshop on Applied Verification for Continuous and Hybrid Systems (ARCH)*, 2016.
50. S. Schuler, A. Walsch, and M. Woehrle. Unifying control and verification of cyber–physical systems (UnCoVerCPS), Deliverable D1.1 — Assessment of languages and tools for the automatic formalisation of system requirements. `http://cps-vo.org/node/24197`, 2016.
51. T. Strathmann and J. Oehlerking. Verifying properties of an electro-mechanical braking system. In *ARCH@ CPSWeek*, pages 49–56, 2015.

Appendix A Correctness Proofs

Recall that a monitor automaton is correct if its error location is reachable exactly when the system H violates the property. Formally, let h be a run of H that violates a given property ϕ. Then we first show (a) that there exists a run r of $H||M$ that reaches the error location. Second, we show (b) that for any run r of $H||M$ that reaches the error location, the run projected onto H violates the property.

A.1 Sufficient Conditions.

absence. Since $r \not\models \phi$, there exist $\tau_q, \tau_p \in \mathrm{dom}(r)$ with $q(r(\tau_q))$, $\tau_p \geq \tau_q$, and $p(r(\tau_p))$.

With Lemma 1 and the definition of a jump,

$$\mathrm{prefix}\,(h, \tau_q) \times \mathrm{idle} \;\; ; \;\; \mathrm{infix}\,(h, \tau_q, \tau_p) \times \mathrm{loc1} \;\; ; \mathrm{postfix}\,(h, \tau_p) \times \mathrm{error}$$

is a run of $H||M$.

absence (timed). Since $r \not\models \phi$, there exist $\tau_q, \tau_p \in \mathrm{dom}(r)$ with $q(r(\tau_q))$, $d(\tau_q, \tau_p) \geq T$, and $p(r(\tau_p))$.

M can remain in idle location during $\mathrm{prefix}\,(h, \tau_q)$, then transition to loc1 and remain there during $\mathrm{infix}\,(h, \tau_q, \tau_p)$. M can then transition to loc2 with Lemma 1. In loc2, M can take the transition to error, as p holds.

minimum duration. $r \not\models \phi$, so there is $\tau_p \geq \tau_q$ with $p(r(\tau_p^*))$ and $q(r(\tau_q))$, and one of the following is true:

(a) there are $\tau_p, \tau'_{\bar{p}}$ with $\tau_{q.1} \leq \tau_p < \tau'_{\bar{p}}$, $p(r(\tau_p))$, $\neg p(r(\tau'_{\bar{p}}))$, and $d(\tau_{q.1}, \tau_{\bar{p}}) \leq T$, or
(b) there are $\tau_p, \tau_{\bar{p}}, \tau'_{\bar{p}} \in \mathrm{dom}(r)$ with $\tau_{q,1} \leq \tau_{\bar{p}} < \tau_p < \tau'_{\bar{p}}$, $p(r(\tau_p))$, $\neg p(r(\tau_{\bar{p}}))$, $\neg p(r(\tau'_{\bar{p}}))$, and $d(\tau_{\bar{p}}, \tau'_{\bar{p}}) \leq T$.

In case (a), let $\tau_{p.1} = \mathrm{first}\,(r, \tau_{q.1}, p)$, so $\tau_{p.1} \leq \tau_p$. M can remain in idle location during $\mathrm{prefix}\,(h, \tau_{q.1})$, then transition to loc1 and with Lemma 2 remain there during $\mathrm{infix}\,(h, \tau_q, \tau_{p.1})$. M can then transition to loc2, setting t to zero with Lemma 1. M can remain in loc2 during infix $\left(h, \tau_p, \tau'_{\bar{p}}\right)$. Since $d(\tau_{q,1}, \tau'_{\bar{p}}) \leq T$, we have $t \leq T$. M can then transition to error.

In case (b), we first show that M can be at loc1 at $\tau_{\bar{p}}$. After τ_q, M can go to loc2 as soon as p is satisfied, and move back to loc1 as soon as p is violated. We can therefore assume that M can be loc1 at $\tau_{\bar{p}}$. We match the remainder of the run in analogy to case (a), replacing $\tau_{q.1}$ by $\tau_{\bar{p}}$.

In the following, we only highlight the differences with the aforementioned proofs (in terms of what happens before τ_q and τ_p).

maximum duration. $r \not\models \phi$ implies that

(i) there is $\tau_p \geq \tau_q$ with $r(\tau_p)$ satisfying p and $r(\tau_q)$ satisfying q, and
(ii)(a) there is τ_p' with $p(r(\tau_p'))$ and $d(\tau_p, \tau_p') \geq T$, and
(ii)(b) there is no $\tau_{\bar{p}}$ such that $\neg(p(r(\tau_{\bar{p}}))$ and $\tau_p < \tau_{\bar{p}} < \tau_p'$.

At τ_p, M can be either in loc1 or loc2. In loc1, M can take the transition to loc2, as p holds. Once in loc2, M can wait there for T time units, since with (ii)(a) and (ii)(b), p still holds. M can then transition to error.

bounded recurrence. For the unbounded case (i), there is $\tau_p \in \mathrm{dom}(r)$ with $p(r(\tau_p))$ and $\tau_p \geq \tau_q$, such that there is no $\tau_p' > \tau_p$, with $d(\tau_p', \tau_p) \leq T$ and $p(r(\tau_p'))$ (τ_p's with distance less than T).

If M is in loc2 at τ_p, it takes the transition from loc1 to loc2. If M is in loc2, there are two subcases:

(a) p does not hold within T time units after τ_p, in which case M can go to loc1, wait for more than T time and then go to error.
(b) p holds after τ_p, which means that it holds at a time τ_p' with $d(\tau_p, \tau_p') > T$. Let $\delta = d(\tau_p, \tau_p') - T$. Then M can wait for $\delta/3$ time in loc2, after which p is false. Then M can go to loc1, wait for $T + \delta/3$ time, and since only $T + 2\delta/3$ time has passed since τ_p, p is still false. Since $t > T$, M can go to error.

bounded response (persisting). For the unbounded time horizon, $r \not\models \phi$ implies that

(i) there is $\tau_p \geq \tau_q$ with $r(\tau_p)$ satisfying p and $r(\tau_q)$ satisfying q, and
(ii) there is no $\tau_s \geq \tau_p$ such that $r(\tau_s)$ satisfies s, $d(\tau_p, \tau_s) \leq T$ and persists (r, τ_s, s).

At τ_p, M can be either in loc1, loc2, or loc3. In loc1, M can transition immediately to loc2 (because p is true). In loc2, there are two options. If s is false ($\nexists \tau_s : d(\tau_p, \tau_s) \leq T$), M can stay there for more than T time units. M can then transition to error. If s is not always false, there is a τ_s such that $\neg$persists (r, τ_s, s). At τ_s, M instantaneously moves to loc3 and then back to loc2 when s does not hold. From loc3, if $\neg s$, M can transition to loc2. If s and $\neg$persists (r, τ_s, s), M can transition to loc2, since $\neg$persists $(r, \tau_s, s) : \exists \tau_s' > \tau_s$ with $d(\tau_s, \tau_s') = 0$ and $\neg s(\tau_s')$.

For the bounded time horizon, $r \not\models \phi$ implies that

(i) there is $\tau_p \geq \tau_q$ with $r(\tau_p)$ satisfying p and $r(\tau_q)$ satisfying q, and
(ii) $\tau_q, \tau_p \in \mathrm{dom}_{-T}(r)$ and there is no $\tau_s \in \mathrm{dom}(r)$ such that $\tau_p \leq \tau_s$, $d(\tau_p, \tau_s) \leq T$ and persists (r, τ_s, s).

The proof is analogous to the unbounded case.

bounded invariance. $r \not\models \phi$ implies that

(i) there is $\tau_p \geq \tau_q$ with $r(\tau_p)$ satisfying p and $r(\tau_q)$ satisfying q, and
(ii) there is a τ with $\tau \geq \tau_p$ such that $d(\tau_p, \tau) < T$ and $s(r(\tau))$ is false.

At τ_p, M can be either in loc1, loc2, or loc3. In loc1, there are two options. If s holds, then M transitions immediately to loc2 (because p and s are true). If s does not hold, M can transition to the error state. If M stays in loc2 from τ_p to τ, then it can go to the error state because s is false at τ. Whenever p gets false, time can no longer elapse in loc2, M goes to loc3, and the clock t is initialized. In loc3, time can elapse for T time units, while $\neg p$ holds. Since the distance $d(\tau_p, \tau) < T$, M can wait in loc3 until τ, at which point M can go to error. If p becomes true in loc3, M goes immediately to loc2. By induction, M always lets time elapse until τ is reached: in loc2 while p holds and in loc3 while loc3 p does not hold.

A.2 Necessary Conditions.

For the necessary condition, we need to show that a run r in $H||M$ that ends in location error implies a run in H that violates the property. Let r_H be the projection of the run onto H (removing the locations and clocks of M). It is straightforward that r_H is a run of H. In the following, we show that $r_H \not\models \phi$. Note that r starts in location idle. Note also that any event-times of r are also event-times of r_H.

absence. To get from idle to error, M had to take first a transition with guard q and then a transition with guard p. Consequently, there exist τ_q and τ_p with $\tau_q \leq \tau_p$, $q(r_H(\tau_q))$ and $p(r_H(\tau_p))$. τ_q and τ_p are witnesses that violate ϕ.

absence(timed). To get from idle to error, M had to take first a transition with guard q, wait for T time units, and then take a transition with guard p. Consequently, there exist τ_q and τ_p with $d(\tau_q, \tau_p) \geq T$, $q(r_H(\tau_q))$ and $p(r_H(\tau_p))$. τ_q and τ_p are witnesses that violate ϕ.

minimum duration. Similarly to the above proof of the absence pattern, we can stipulate the existence of τ_q, τ_p and $\tau'_{\bar{p}}$ with $\tau_q \leq \tau_p \leq \tau'_{\bar{p}}$, $q(r_H(\tau_q))$, $p(r_H(\tau_p))$ and $\neg p(r_H(\tau'_{\bar{p}}))$. τ_q and τ_p are witnesses that violate case (i).

For case (ii), let $\tau_{q.1} = \text{first}\,(r, 0, q)$, so that $\tau_{q.1} \leq \tau_q$. Without loss of generality, we can assume that τ_p is the last event-time on r where M entered loc2, so $t = d(\tau_p, \tau'_{\bar{p}})$. Because of the transition guard from loc2 to error, $d(\tau_p, \tau'_{\bar{p}}) \leq t \leq T$. There are two subcases:

(a) If there is no $\tau_{\bar{p}}$ with $\tau_{q.1} \leq \tau_{\bar{p}} \leq \tau_p$ and $\neg p(r_H(\tau_{\bar{p}}))$, we can conclude that $\tau_{q.1} = \tau_{p.1}$, where $\tau_{p.1} = \text{first}\,(r, \tau_{q.1}, p)$. In this case, the run in M goes from idle to loc1 to loc2, so $\tau_{q.1} = \tau_{p.1} = \tau_p$. Consequently, $d(\tau_{q.1}, \tau'_{\bar{p}}) = d(\tau_p, \tau'_{\bar{p}}) \leq T$, which violates case (a).

(b) Otherwise, we have $\tau_{q.1} \leq \tau_{\bar{p}} \leq \tau_{p.1} \leq \tau'_{\bar{p}}$. We will show that there is a $\tau^* \leq \tau_p$, with $d(\tau^*, \tau_p) = 0$ and where $r(\tau^*)$ violates p. Then $d(\tau^*, \tau'_{\bar{p}}) \leq T$, which violates case (b). We now show the existence of τ^*, by first identifying some $\tau' \leq \tau_p$ such that M is in loc1 for all $\tau' \leq \tau \leq \tau_p$, and for which $r(\tau')$ violates p. Consider that we can assume that loc1 was entered either from idle with p being violated (otherwise case (a) applies), or from loc2, which also means p is violated. Since the transition from loc1 to loc2 is urgent, p can not hold for any τ with $\tau' \leq \tau < \tau_p$ where $d(\tau, \tau_p) > 0$ (no time can elapse while p is true). So there exists a τ^* with $\tau' \leq \tau^* \leq \tau_p$ with $d(\tau^*, \tau_p) = 0$.

maximum duration. Let τ_p be the last event-time on r where M entered loc2. As the loc2 has invariant p and the transition guard from loc2 to error has the constraint $t \geq T$, we know that at least T time units have elapsed in loc2. That means that there exist τ_p and τ'_p so that $d(\tau_p, \tau'_p) \geq T$ without any $\tau_{\bar{p}}$ in between them. Therefore, τ_q, τ_p and the absence of $\neg p$ witnesses the violation of ϕ.

bounded recurrence. Since time can only elapse in loc2 while $\neg p$ and t is reset on all incoming transitions, we know that $\neg p$ holds for more than T time units, which violates the property.

bounded response (persisting). Similarly to the above proof of the absence pattern, we can stipulate the existence of τ_q and τ_p. Let τ_p be the last event-time on r where M entered loc2. Cycles between loc2 and loc3 take zero time: because of the urgent transition from loc2 to loc3, s was false during this time, with the possible exception of switching to true and back to false in zero time (which doesn't satisfy the definition of "persists"). Because the transition guard from loc2 to error has the constraint $t > T$, we know that more than T time units have elapsed in loc2. Therefore, τ_q, τ_p and the absence of s witness the violation of ϕ.

bounded invariance. Assuming that M is in the error location, due to the guard conditions of the incoming transitions, we know that at some point τ on the run, s did not hold. If the transition from loc1 was taken, p also held at τ, which immediately violates the property.

In loc2, we know from the incoming guard conditions, that p held at some point τ_p with $\tau_p \leq \tau$. Without loss of generality, let τ^* be the latest (supremum) point on the run where p holds in loc2: $\forall \varepsilon > 0, \exists \tau_p : d(\tau_p, \tau^*) \leq \varepsilon$. From τ^* onwards, $\neg p$ holds. Therefore, time could only elapse loc3, and for no more than T time units. In consequence, $d(\tau^*, \tau) < T$. With the above, we get $\forall \varepsilon > 0, \exists \tau_p : d(\tau_p, \tau) < T + \varepsilon$. This is equivalent to $d(\tau_p, \tau) < T$, which violates the property.