Software and Systems Modeling (2024) 23:403–426
https://doi.org/10.1007/s10270-023-01145-x

SPECIAL SECTION PAPER

P-stable abstractions of hybrid systems

Anna Becchi[1,2] · **Alessandro Cimatti**[1] · **Enea Zaffanella**[3]

Received: 1 July 2022 / Revised: 20 November 2023 / Accepted: 13 December 2023 / Published online: 29 January 2024

Abstract
Stability is a fundamental requirement of dynamical systems. Most of the works concentrate on verifying stability for a given stability region. In this paper, we tackle the problem of *synthesizing* $\mathbb{P}$*-stable abstractions*. Intuitively, the $\mathbb{P}$-stable abstraction of a dynamical system characterizes the transitions between stability regions in response to external inputs. The stability regions are not given—rather, they are synthesized as their most precise representation with respect to a given set of predicates $\mathbb{P}$. A $\mathbb{P}$-stable abstraction is enriched by timing information derived from the duration of stabilization. We implement a synthesis algorithm in the framework of Abstract Interpretation that allows different degrees of approximation. We show the representational power of $\mathbb{P}$-stable abstractions that provide a high-level account of the behavior of the system with respect to stability, and we experimentally evaluate the effectiveness of the algorithm in synthesizing $\mathbb{P}$-stable abstractions for significant systems.

Keywords $\mathbb{P}$-stable abstraction · Hybrid systems · Reverse engineering Abstract Interpretation · Predicate abstraction · Run-to-completion

1 Introduction

Context

Reactive systems are often designed to operate in some stable condition (in absence of external stimuli) and to reach a possibly different stable condition (in response to some external stimulus). A notable example are relay-based circuits, built out of electro-mechanical components, pervasively adopted in the railway domain for the control of stations. When analyzing these legacy systems, one is interested in characterizing the specification of a set of controlling actions in terms of their stable effects on the system state. These inputs may trigger a *run-to-completion* process, i.e., a sequence of internal changes, both *discrete* (like relay interactions) and *continuous* (like the charging process of a capacitor), toward the next stable state. The duration of these evolutions is also important: after an action, it may be necessary to wait some time before evaluating the state of the system or before accepting a new input.

System stability is hard to assess. First, it is not to be confused with a completely still situation (i.e., a zero-derivative point), since partly oscillating or limit behaviors can also be considered stable. Furthermore, a system may exhibit a large number of stable conditions, difficult to characterize by inspection. Finally, the discrete behavior depends crucially on the physical status: an engine may be powered or not depending on whether the magnetic field induced by a coil is sufficient to close a switch.

Communicated by Antonio Cerone and Frank de Boer.

A. Cimatti and E. Zaffanella contributed equally to this work.

✉ Anna Becchi
abecchi@fbk.eu

Alessandro Cimatti
cimatti@fbk.eu

Enea Zaffanella
enea.zaffanella@unipr.it

[1] Fondazione Bruno Kessler, Via Sommarive 18, 38123 Trento, TN, Italy

[2] Department of Information Engineering and Computer Science, University of Trento, Via Sommarive 9, 38123 Trento, TN, Italy

[3] Department of Mathematical, Physical and Computer Sciences, University of Parma, Parco Area delle Scienze 53/A, 43124 Parma, PR, Italy

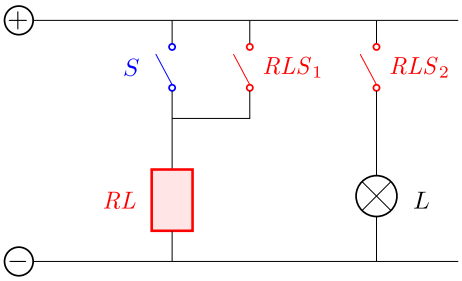


Fig. 1 A circuit controlling a lamp L, based on a switch S and a relay RL

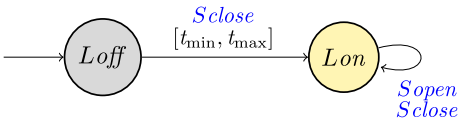


Fig. 2 An automaton describing the effects of the external actions on the lamp state

Running example

An inductor, also called coil, is an electrical component that intuitively stores energy in a magnetic field when electric current flows through it. The magnetic field is used to open or close a contact possibly belonging to another circuit. A *relay* is an electro-mechanical component formed by a coil and a set of associated switches.

In Fig. 1, we show a simple circuit whose behavior is based on a relay *RL*. For the sake of readability, we adopt a logical description of a relay, where the coil and the associated switches are drawn separately and their connection is identified by naming convention: relay *RL* controls switches RLS_1, RLS_2.

A human operator controls the circuit performing actions on the switch S, represented in blue. The *external action* of closing S makes relay *RL* (in red) be connected to a source of voltage. Current starts to flow through *RL* and when it receives enough current, i.e., if the switch S stays closed for a sufficient time for its charging process, it closes the corresponding switches RLS_1 and RLS_2 (in red). The closing of RLS_2 turns on the lamp L. The closing of RLS_1 is meant to keep the relay coil powered even when S gets opened. As a matter of fact, once the lamp is on, a following action on switch S has no impact on its status.

We consider the problem of extracting the relation between the state of lamp L and the sequence of performed actions by building the automaton represented in Fig. 2. Its states represent the truth value of the chosen predicates (the state of the lamp) and are connected by transitions labeled with the external events of the system. These transitions are not instantaneous: as said before, the first closing of the switch needs some time to charge the relay and turn on the lamp. Hence, the abstraction is enriched with timing information, defining how long the system takes to reach the new stable configuration.

Contribution

In this paper, we investigate the problem of characterizing the effects of events on a hybrid system by analyzing where the triggered behaviors stabilize. We propose the notion of $\mathbb{P}$-*stable abstraction* as the automaton that captures the essence of stabilization following each external input.

The granularity of the abstraction is induced by a given set of relevant predicates $\mathbb{P}$. Intuitively, an (abstract) state is associated with predicate valuations, and identifies the (concrete) states that are stable in the corresponding region. The transitions between abstract states describe the stabilization process of the concrete system when responding to an external stimulus. The abstraction is made accurate by requiring the stability regions to be *minimal*: the stability of a trajectory is defined in terms of the smallest region representable with $\mathbb{P}$ predicates in which the trajectory eventually converges. The synthesis of $\mathbb{P}$-stable abstractions directly results in a set of temporal properties that are satisfied by the concrete system, and can therefore be used in reverse-engineering and migrating to new technology. In order to capture the duration of stabilizations, a $\mathbb{P}$-stable abstraction is enriched with timing information characterizing the time spent in unstable states. This information can be used to synthesize the correct value to impose a slow-switching hypothesis on the external environment of the system [31].

Second, we prove that the problem can be recast in the framework of Abstract Interpretation (AI) [23] and propose a synthesis algorithm based on the exploration of the abstract state space. At the core of the algorithm is the computation of sufficient conditions for stability. The AI framework is fundamental to seamlessly approximate $\mathbb{P}$-stable abstractions and increase efficiency of the algorithm by reducing the precision of the abstract domain.

The proposed algorithm has been implemented in an analysis tool handling symbolic hybrid systems with piecewise-constants dynamics. An experimental evaluation, focusing on a set of parametric benchmarks representing circuits with run-to-completion behaviors, shows that the proposed techniques can obtain abstractions of rather complex hybrid systems.

Structure of the paper

In Sect. 2, we discuss related works. After introducing some background concepts in Sect. 3, in Sect. 4, we introduce the notion of $\mathbb{P}$-stability and we define $\mathbb{P}$-stable abstractions. In Sect. 5, we prove that the problem can be formulated in the general framework of Abstract Interpretation, and define how to approximate the construction. In Sect. 6, we present

an effective algorithm to synthesize approximated $\mathbb{P}$-stable abstractions, based on the exploration of the abstract state space. In Sect. 7, we comprehensively show the application of the algorithm to obtain the $\mathbb{P}$-stable abstraction of a water tanks system. In Sect. 8, we discuss the implementation and we experimentally evaluate the approach on relay-based circuits. Finally, in Sect. 9, we draw some conclusions and outline directions for future work.

2 Related works

State of the art

Stability is an important property of dynamical and hybrid systems which has been widely studied from different perspectives. Classic stability is defined by requiring that all the trajectories are asymptotically attracted by an equilibrium point x_e [18, 31]. Since classical asymptotic stability excludes oscillating behaviors, *region stability* [34, 35] requires that the trajectories eventually remain inside a given invariant region, even if no single equilibrium point exists. The alternative notion of *strong attractor* requires that all the trajectories of the system never leave the region once entered.

The problem is typically to verify the global stability of a given system, i.e., proving that every trajectory satisfies the required stability criterion (be it asymptotic or region stability). When global stability does not hold, an additional task is to compute the region of attraction, i.e., the set of states whose outgoing trajectories are stable.

Asymptotic stability can be proven by providing a Lyapunov function as a certificate that the energy of the system is decreasing until the equilibrium point is reached. Several methods have been proposed to this aim, for both dynamical and switched systems, with different levels of automation, soundness and scalability (see e.g., [19, 27]). Region stability verification cannot be directly tackled as a reachability problem, since its counterexamples are the paths that visit infinitely often the negation of the region. It is proved by reduction to liveness checking with combinations of reachability and SMT solving, or based on the use of (Cartesian) predicate abstraction [35].

Interestingly, in the case of switching systems, stability of the whole system is not implied by the stability of each modality. Some works (e.g. [14]) aim at finding conditions on switching sequences in order to ensure the stability of the composed system. Another approach to achieve global stability is to impose a *slow-switching* condition, i.e., sufficient time must elapse between subsequent inputs. [17, 33] prove the adequacy of such a time interval by analyzing the *average dwell time* of the system.

Finally, the work of [8] introduces a framework for abstracting symbolic timed transition systems which is parametric on a definition of stability. The latter is used as a criterion to prune the states that need to be visible for reverse engineering purposes at different levels of granularity.

Novelty

This work differs from the works mentioned above in several ways. First, in contrast to verifying stability with respect to a given region, we *synthesize* a $\mathbb{P}$-stable abstraction that characterizes all the system behaviors with respect to stabilization. Notice that we do not rely on a single convergence region being given. We explore the space of possible convergence regions induced by the set $\mathbb{P}$ of predicates and find the tightest representations. Second, the synthesized region is not a simple invariant of the system: rather, it is *possibly, eventually* invariant only for the trajectories triggered by the event under consideration. Hence, we simulate hybrid evolutions with a relation of possible attraction between two stable conditions: we want to express the existence of an eventually convergent trajectory rather than requiring stabilization for all paths. Another key difference is that the aforementioned approaches are mainly related to purely dynamical or closed hybrid systems. We adopt a more expressive framework, considering switched systems, open to external events. Specifically, our aim is to analyze the stabilization effects for external inputs, by considering the "closed" dynamic of the system. Finally, we take into account timing information.

This work is also quite distinct from predicate abstraction for hybrid systems [2, 3]: the main difference is that predicates are not evaluated in transient states, i.e., "abstract" transitions will connect predicates evaluated only in stable conditions. Consider, for example, that the length of the traces is not retained.

With respect to [8], $\mathbb{P}$-stable abstraction is defined on an exponentially larger lattice: a stable region can be a generic (possibly disjunctive) combination of the predicates, while in [8] abstract predicates are limited to complete assignments to $\mathbb{P}$ variables (a perspective more similar to predicate abstraction). Hence, differently from this work, the framework of [8] cannot detect stability in runs that oscillate in different predicates and does not deal with the minimality problem, i.e., finding the most precise characterization of a stable behavior.

This paper is an extended and revised version of [12] and [13]. Additional contributions include Propositions 2 and 3 and respective proofs in Sect. 5; an extensive presentation of the synthesis algorithm with pseudocodes and additional details on the implementation; a number of new illustrative examples and a fully worked out case study presented in Sect. 7; an extended experimental evaluation (Sect. 8) with new sets of benchmarks and improved results obtained with an optimized implementation.

3 Background

For a set S, we write $\wp(S)$ for the powerset of S and $|S|$ for its cardinality. The sets of natural, rational, real and nonnegative real numbers are denoted by $\mathbb{N}$, $\mathbb{Q}$, $\mathbb{R}$ and $\mathbb{R}_{\geq}$, respectively; we write $\mathbb{B} = \{\bot, \top\}$ for the set of Boolean values.

Given a sequence σ and an index $i \in \mathbb{N}$, let $\sigma[i]$ denote the i-th element of σ. We adopt the standard notion of first-order logic and Satisfiability Modulo Theory (SMT) [7], focusing on the theory of Linear Real Arithmetic (LRA). Given a set of Boolean variables L, let $\Psi(L)$ define the set of Boolean combinations over L. Given a set of real-valued variables V, let LPred_V define finite conjunctions of LRA predicates with free variables in V. We write $\Psi(L, V)$ to denote SMT(LRA) formulae obtained by Boolean combinations of Boolean variables in L and linear predicates over V.

Finite and timed automata

A *finite state automaton* is a tuple $\langle Q, Q_0, A, R\rangle$ where Q is a finite set of states, $Q_0 \subseteq Q$ is the set of initial states, A is a finite set of labels (including the empty label ε) and $R \subseteq (Q \times A \times Q)$ is a labeled transition relation between states. A clock predicate is a linear predicate over a clock variable c of the form $c \bowtie k$, where k is constant in $\mathbb{Q}$ and $\bowtie \in \{\leq, <, \geq, >\}$. A clock constraint is a finite conjunction of clock predicates. A *timed automaton* [1] $\langle Q, Q_0, C, A, \text{inv}, R\rangle$ is a finite-state automaton equipped with a finite set of clocks C, with state invariants $\text{inv}\colon Q \to \text{LPred}_C$ associating each state $q \in Q$ with its clock constraints $\text{inv}(q)$ and with $R \subseteq (Q \times A \times \text{LPred}_C \times \wp(C) \times Q)$, where edge $(q, a, g, r, q') \in R$ represents the transition from state q to q', labeled with a and guarded by clock constraints g; the set $r \subseteq C$ gives the set of clocks to be reset to zero with this transition. We adopt notation $q \xrightarrow{a,g,r} q'$. In a timed automaton with a single clock variable c, we write $q_i \xrightarrow{a,[m,M]} q_j$, meaning that q_i reaches q_j with a transition labeled with a in a time between m and M. Formally, we are omitting an intermediate state q with transitions $q_i \xrightarrow{a,c:=0} q \xrightarrow{\varepsilon,c\geq m} q_j$, where ε denotes the absence of an action label, i.e., a silent transition, and $\text{inv}(q) = (c \leq M)$. When clear from context we also omit the 'inv' component from the tuple.

Hybrid systems

Let $\dot{v}$ denote the time derivative dv/dt. A *linear hybrid system* with piecewise affine dynamics is a tuple $\mathcal{H} = \langle Loc, Var, Lab, \text{inv}, \text{init}, \text{flow}, \text{disc}\rangle$ where [36]

- Loc is a finite set of locations;
- $Var = \{v_1, \ldots, v_n\}$ is a finite set of continuous state variables;
- Lab is a finite set of synchronization labels;
- $\text{init}\colon Loc \to \text{LPred}_{Var}$ defines initial conditions for each location;
- $\text{inv}\colon Loc \to \text{LPred}_{Var}$ defines invariant conditions for each location;
- $\text{flow}\colon Loc \to \text{LPred}_{Var \cup \dot{Var}}$ defines the continuous transition relation;
- $\text{disc} \subseteq (Loc \times Lab \times Loc \times \text{LPred}_{Var \cup Var'})$ defines the labeled discrete transition relation.

A *state* of a hybrid system $\mathcal{H}$ is a tuple $\langle \ell, \boldsymbol{x}\rangle$ where $\ell \in Loc$ and $\boldsymbol{x} \in \mathbb{R}^n$. Let Σ denote the state space of $\mathcal{H}$ and $\text{init}(\mathcal{H})$ denote the set of states $s = \langle \ell, \boldsymbol{x}\rangle$ such that $\boldsymbol{x} \models (\text{inv}(\ell) \wedge \text{init}(\ell))$. A *run* (or a path) of hybrid system $\mathcal{H}$ is a possibly infinite sequence $\rho = (s_0 \xrightarrow{\delta_0} s_1 \xrightarrow{a_1} s_2 \xrightarrow{\delta_2} s_3 \xrightarrow{a_3} \ldots)$ where $\delta_i \in \mathbb{R}_{\geq}$, $a_i \in Lab$, $s_i = \langle \ell_i, \boldsymbol{x}_i\rangle \in \Sigma$, $s_0 \in \text{init}(\mathcal{H})$ and each step corresponds to either a continuous timed transition

$$\frac{\begin{array}{c}\delta \in \mathbb{R}_{\geq} \quad f\colon [0,\delta] \to \mathbb{R}^n \quad \dot{f}\colon (0,\delta) \to \mathbb{R}^n \\ f(0) = \boldsymbol{x} \qquad f(\delta) = \boldsymbol{x}' \\ \forall \epsilon \in [0,\delta]\colon f(\epsilon) \models \text{inv}(\ell) \\ \forall \epsilon \in (0,\delta)\colon (f(\epsilon), \dot{f}(\epsilon)) \models \text{flow}(\ell)\end{array}}{\langle \ell, \boldsymbol{x}\rangle \xrightarrow{\delta} \langle \ell, \boldsymbol{x}'\rangle}$$

or a discrete transition

$$\frac{(\ell, a, \ell', \mu) \in \text{disc} \quad (\boldsymbol{x}, \boldsymbol{x}') \models \mu \quad \boldsymbol{x}' \models \text{inv}(\ell')}{\langle \ell, \boldsymbol{x}\rangle \xrightarrow{a} \langle \ell', \boldsymbol{x}'\rangle.}$$

We write $\text{Run}(\mathcal{H})$ for the set of all runs of $\mathcal{H}$. In a run, we consider $\delta_i = 0$ whenever the i-th transition is a discrete one. A run ρ diverges if ρ is infinite and the sum $\sum_{i\geq 0} \delta_i$ diverges. We consider systems without Zeno behaviors, i.e., we exclude paths that execute infinitely many discrete steps in a finite time. If $\text{Run}(\mathcal{H}) \subseteq \text{Run}(\mathcal{H}')$, then $\mathcal{H}'$ is a *relaxation* of $\mathcal{H}$ and $\mathcal{H}$ is a *refinement* of $\mathcal{H}'$ [17].

The length of a finite run $\rho = (s_0 \xrightarrow{\delta_0} s_1 \xrightarrow{a_1} s_2 \ldots s_{n-1})$ is the number of states n. For each run $\rho = (s_0 \xrightarrow{\delta_0} s_1 \xrightarrow{a_1} s_2 \ldots)$ and index m, the time spent to complete the prefix up to state s_m of ρ is $\tau_m(\rho) \doteq \sum_{i=0}^{m-1} \delta_i$. For a finite run of length n, $\tau(\rho)$ is a shortcut for $\tau_{n-1}(\rho)$.

Lattice theory

We briefly recall here introductory notions of lattice theory [15]. A partial order $\sqsubseteq$ over a set S is a binary relation that is reflexive, transitive and antisymmetric; the pair $(S, \sqsubseteq)$ is said to be a poset (PO). In the following let $(S, \sqsubseteq)$ be a poset. A chain is a subset of S where all the elements are comparable; the poset satisfies the ascending chain condition (ACC) if all its strictly increasing chains are finite. A subset $X \subseteq S$ is

upward closed if $X = \uparrow X \doteq \{y \in S \mid \exists x \in X \,.\, x \sqsubseteq y\}$. Notation for downward closure is dual. An upper bound of $X \subseteq S$ is an element $u \in S$ such that for all $x \in X, x \sqsubseteq u$; when it exists, the least upper bound (lub) of X is denoted $\sqcup X$; lower bounds and greatest lower bounds (glb) are defined dually and denoted $\sqcap X$. The poset $(S, \sqsubseteq)$ is a *complete partial order* (CPO) if it has a minimum element ($\bot$) and all its chains have a least upper bound. Given two CPOs $(S, \sqsubseteq_S)$ and $(T, \sqsubseteq_T)$, their *smash product* is the CPO $S \otimes T$:

$$\left\{(s,t) \in S \times T \;\middle|\; s \neq \bot_S \wedge t \neq \bot_T\right\} \cup \{\bot_{S\otimes T}\},$$

where $\bot_{S\otimes T}$ is a new element; the ordering is coordinate-wise and such that $\bot_{S\otimes T}$ precedes all elements. A function $f : S \to T$ is *monotonic* if, for each $x, y \in S, x \sqsubseteq_S y$ implies $f(x) \sqsubseteq_T f(t)$; for $X \subseteq S$, the image of f on X is the set $f(X) \doteq \{f(x) \mid x \in X\} \subseteq T$; f is *continuous* if, for each chain X of S, $\sqcup_T(f(X))$ exists and $f(\sqcup_S X) = \sqcup_T(f(X))$. [1] A pre-fixpoint (resp., post-fixpoint) for function $f : S \to S$ is an element $x \in S$ such that $x \sqsubseteq f(x)$ (resp., $f(x) \sqsubseteq x$); a fixpoint is both a pre-fixpoint and a post-fixpoint, i.e., $f(x) = x$. If $(S, \sqsubseteq)$ is a CPO with a least element $\bot$ and $f : S \to S$ is continuous, then f has a least fixpoint $lfp^{\preceq} f = \sqcup\{f^n(\bot) \mid n \in \mathbb{N}\}$, where $f^0(x) \doteq x$ and $f^{n+1}(x) \doteq f(f^n(x))$.

A poset $(S, \sqsubseteq)$ is a *complete lattice* if for all $X \subseteq S$ there exist both $\sqcup X$ and $\sqcap X$: we write $(S, \sqsubseteq, \sqcap, \sqcup, \bot, \top)$, where $\bot = \sqcap S = \sqcup\emptyset$ and $\top = \sqcup S = \sqcap\emptyset$ are the bottom and top elements, respectively. The lub and glb operators are also called the *join* and *meet* of the lattice, respectively. An *atom* of a lattice is a (non-bottom) element $a \in S$ such that there is no other (non-bottom) element $x \in S$ satisfying $\bot \sqsubset x \sqsubset a$.

Consider the set of formulae $\Psi(L, V)$ quotiented by logical equivalence (denoted with '$\Leftrightarrow$'). $\Psi(L, V)$ is a complete lattice ordered by logical entailment '$\Rightarrow$', where '$\vee$' and '$\wedge$' are the join and meet operators, respectively, while $\top$ and $\bot$ are the top and bottom elements, respectively. By identifying a formula with the set of its models, $\Psi(L, V)|_{\Leftrightarrow}$ induces the lattice $(\wp(U), \subseteq, \cap, \cup, \emptyset, U)$, where $U = \mathbb{B}^{|L|} \times \mathbb{R}^{|V|}$ is the set of all the possible models of a formula in $\Psi(L, V)$.

Abstract interpretation

We recall here some basic notions of Abstract Interpretation [23], which is a theory to formalize (and actually compute) sound over-approximations of the concrete semantics of a system. Intuitively, a Galois connection links a concrete semantic domain (C) with an abstract semantic domain (A) by means of an abstraction function (α) and a concretization function (γ). Given two posets $(C, \sqsubseteq_C)$ and $(A, \sqsubseteq_A)$ and two functions $\alpha : C \to A$ and $\gamma : A \to C$, the pair (α, γ) is a *Galois connection* [23], denoted $C \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} A$, if for all $c \in C, a \in A$ it holds that $\alpha(c) \sqsubseteq_A a \iff c \sqsubseteq_C \gamma(a)$. Equivalently, (α, γ) is a Galois connection if the abstraction function and the concretization function are mutually induced as the respective adjoints: for each $c \in C$ and $a \in A$

$$\alpha(c) = \sqcap_A\left\{a \in A \;\middle|\; c \sqsubseteq_C \gamma(a)\right\},$$
$$\gamma(a) = \sqcup_C\left\{c \in C \;\middle|\; \alpha(c) \sqsubseteq_A a\right\}.$$

Given two Galois connections $C \underset{\alpha_1}{\overset{\gamma_1}{\leftrightarrows}} A_1 \underset{\alpha_2}{\overset{\gamma_2}{\leftrightarrows}} A_2$, the pair (α, γ), with $\alpha \doteq \alpha_2 \circ \alpha_1$ and $\gamma \doteq \gamma_1 \circ \gamma_2$, is a Galois connection between C and A_2. Let $F_C : C \to C$ be a continuous concrete function; an abstract function $F_A : A \to A$ is a *sound* approximation of F_C if, for all $a \in A, \alpha(F_C(\gamma(a)) \sqsubseteq_A F_A(a)$; intuitively, a sound approximation does not lose in the abstract computation any of the concrete behaviors. The *most precise sound approximation* of F_C is $F_A^\sharp = \alpha \circ F_C \circ \gamma$.

The concrete least fixpoint computation for F_C can thus be over-approximated by a corresponding abstract fixpoint computation for F_A, building an increasing chain on the abstract domain A. Notice that this process may fail to finitely converge if the domain A has infinite ascending chains. Widening operators are used to guarantee (or accelerate) convergence of the abstract computation, possibly inducing further over-approximations, so that in general we obtain a post-fixpoint of F_A.

A *widening* operator for $(A, \sqsubseteq_A)$ is a function $\nabla : A \times A \to A$ such that [24, footnote 6]:

- $\forall x, y \in A : x \sqsubseteq_A y \Rightarrow y \sqsubseteq_A x \nabla y$;
- for each increasing chain $y_0 \sqsubseteq_A y_1 \sqsubseteq_A \ldots$, the increasing chain defined by $x_0 \doteq y_0$ and $x_{i+1} \doteq x_i \nabla (x_i \sqcup_A y_{i+1})$ for $i \in \mathbb{N}$ is not strictly increasing.

Hence, for a monotonic operator $F_A : A \to A$, the sequence with $x_0 \doteq \bot_A$ and

$$x_{i+1} \doteq \begin{cases} x_i & \text{if } F_A(x_i) \sqsubseteq_A x_i; \\ x_i \nabla \left(x_i \sqcup_A F_A(x_i)\right) & \text{otherwise;} \end{cases}$$

converges to a post-fixpoint of F_A after a finite number of iterations.

Temporal logic

In the rest of this paper, we adopt a notation inspired to model checking for specific patterns of computation-tree logic formulae. In particular, $\mathcal{H}, s \models \mathbf{AG}\phi$ means that for all $\rho \in \mathrm{Run}(\mathcal{H})$ outgoing from s (i.e., such that $\rho[0] = s$), for all $i \in \mathbb{N}, \mathcal{H}, \rho[i] \models \phi$. Similarly $\mathcal{H}, s \models \mathbf{EFAG}\phi$ means

[1] Note that, when clear from the context, in this paper we will also use the adjective *continuous* with the standard notion for functions on real-valued variables.

that there exists a run $\rho \in \mathrm{Run}(\mathcal{H})$ outgoing from s, and a $j \in \mathbb{N}$ such that $\mathcal{H}, \rho[j] \models \mathbf{AG}\phi$.

As examples, consider the concepts introduced in Sect. 2: the property of region stability (requiring that all trajectories will converge and never leave a region R) corresponds to the specification $\mathbf{AFAG}R$; the strong attractor variant (i.e., all trajectories never leave region R once entered) corresponds to the specification $\mathbf{A}\neg R\mathbf{UAG}R$. The existence of an eventually convergent trajectory in region R is expressed by the specification $\mathbf{EFAG}R$.

In systems having a global clock variable *clock*, we adopt a notation inspired to Metric Temporal Logic, in which temporal operators are enriched with interval constraints, e.g., $\mathbf{AF}[l, u]\phi$ means that all path will accomplish ϕ after l time and before u time units.

4 $\mathbb{P}$-stable abstraction

In this section, we characterize the stabilizing executions of a closed hybrid system with respect to the truth values of a given set of predicates. Given a set of events, we define an abstract timed automaton whose transitions simulate the stabilizing runs triggered by these events.

4.1 Evolutions in the closed system

We now define some additional concepts on hybrid systems extending the classical definitions presented in Sect. 3.

Closed system

In hybrid automata, as well as in switched systems [31], discrete behaviors can be distinguished between controlled (internal) events, for logic-based mechanism, and autonomous (external) events, modeling unpredictable environmental influences. In the following, we denote by $I \subseteq Lab$ the subset of labels of the hybrid system $\mathcal{H}$ corresponding to internal events; the set of external events is (implicitly) defined as $E \doteq Lab \setminus I$ (so that $E \cap I = \emptyset$).

Definition 1 (Closed system $\mathcal{H}^c$) Let $I \subseteq Lab$ be the set of internal events of a hybrid system

$$\mathcal{H} = \langle Loc, Var, Lab, \mathrm{inv}, \mathrm{init}, \mathrm{flow}, \mathrm{disc}\rangle;$$

the corresponding *closed system* is

$$\mathcal{H}^c \doteq \langle Loc, Var, I, \mathrm{inv}, \top, \mathrm{flow}, \mathrm{disc}^c\rangle,$$

where $\mathrm{disc}^c \doteq \{(\ell, i, \ell', \mu) \in \mathrm{disc} \mid i \in I\}$. The runs of $\mathcal{H}^c$ are called *closed evolutions* of $\mathcal{H}$.

Intuitively, the closed system $\mathcal{H}^c$ models the behaviors of $\mathcal{H}$ when all external events are blocked. It has no initial conditions because it is meant to show what are the runs originating from any state if no external event is received.

We denote with '$\overset{c}{\rightsquigarrow}$' the reflexive and transitive closure of the transition relation of $\mathcal{H}^c$. Namely, if $s \overset{c}{\rightsquigarrow} s'$ then there exists a sequence of transitions involving only continuous transitions and *internal* discrete transitions starting from s and reaching s'. When clear from context, we also use $s \overset{c}{\rightsquigarrow} s'$ to denote the corresponding set of runs from s to s' in the closed system.

Given a temporal formula φ and a state s, let $s \models^c \varphi$ be a shortcut for $\mathcal{H}^c, s \models \varphi$, meaning that s makes φ true limitedly to the closed system runs.

Slow switching

Let $\varepsilon(\rho)$ be the ordered sequence of indices for the external events of a run ρ: namely, letting $\rho = (s_0 \xrightarrow{\delta_0} s_1 \xrightarrow{a_1} s_2 \xrightarrow{\delta_2} \ldots)$, $\varepsilon(\rho)$ is the ordered sequence of positions k such that $a_k \in E$ for $s_k \xrightarrow{a_k} s_{k+1}$.

The sequence of *external switching time points* of a run ρ records the time at which the external events occur. More formally: it is defined as the sequence t of elements in $\mathbb{R}_{\geq}$ such that for all i, $t[i] = \tau_{\varepsilon[i]}(\rho)$.

For a hybrid automaton $\mathcal{H}$, a time $d \in \mathbb{R}_{\geq}$ induces a refinement $\mathcal{H}_d$ of $\mathcal{H}$ such that every run of $\mathcal{H}_d$ is associated with a sequence t of external switching time points satisfying $(t[i+1] - t[i]) > d$ for all i. Intuitively, the runs of $\mathcal{H}_d$ are the runs of $\mathcal{H}$ satisfying a *slow-switching* constraint on external events, i.e., at least d units of time must elapse between two subsequent events in E. $\mathcal{H}_d$ can be obtained from $\mathcal{H}$ as follows: first, by adding a clock variable (i.e., a state variable with flow $\dot{c} = 1$ in every location); second, by adding in all transitions labeled with external events the guard $c > d$ and the reset condition $c' = 0$.

4.2 From region stability to $\mathbb{P}$-stability

Given a region $R \subseteq \Sigma$ of the state space, we say that a state $s \in \Sigma$ is *internally stable in* R (for short, stable in R) if and only if R is invariant for all closed evolutions of s. State s is said to be *possibly attracted by* R (for short, attracted by R) if there exists a closed evolution of s reaching a state internally stable in R; dually, R is a possible eventually stable condition for s. More formally:

Definition 2 (Region stability) For each $s \in \Sigma$ and $R \subseteq \Sigma$,

$$\mathrm{STABLE}(s, R) \iff (s \models^c \mathbf{AG}R);$$
$$\mathrm{ATTR}(s, R) \iff (s \models^c \mathbf{EFAG}R).$$

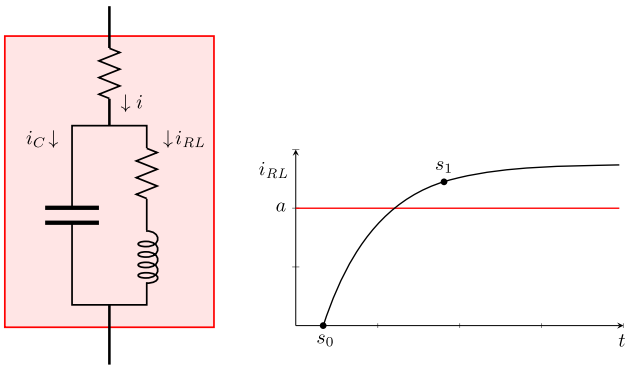


Fig. 3 A relay coil with delay and the corresponding hybrid system run

The "run-to-completion" closed evolutions of a state $s \in \Sigma$ are described by the regions that possibly attract it: due to the non-determinism of the closed system, taken into account by the existential path quantification, this can be a set of different regions, each of them representing a possible future stable condition. Given a state s, the set of regions R such that $\text{STABLE}(s, R)$ holds is upward closed (i.e., for all $R, R' \subseteq \Sigma$ such that $R \subseteq R'$, if $\text{STABLE}(s, R)$ then $\text{STABLE}(s, R')$). The *minimal* regions for which $\text{ATTR}(s, R)$ holds identify the set of most precise possible eventually stable conditions for s: minimal stability is assessed only if there is no other smaller region in which the system can converge in the future. In the infinite state setting, the quest for minimality is problematic. Take for example an asymptotic behavior to an equilibrium condition, like the discharging process of a capacitor. Its flow follows an exponential dynamic that, ideally, never reaches a null charge, so that a minimal region of convergence does not exist. Nonetheless, under a certain threshold the capacitor can be assumed to be discharged and the following decay of voltage has no impact on its behavior in the circuit. In other words, there are certain regions of interest, in which the exact trajectories are not relevant for the analysis of the system. This suggests to fix a priori a finite set of predicates $\mathbb{P}$ representing the properties we want to observe in order to discretize the state space: the run-to-completion behaviors will be described as the minimal attracting areas chosen from the (finite) set of regions induced by $\mathbb{P}$.

Example 1 In the left-hand side of Fig. 3, it is shown how a delay is added to the activation of a relay, whose use has already been introduced in Fig. 1. Focusing inside the relay component, we see that its inductor *RL* is connected in parallel with a capacitor C. When attached to a source of current, current flows and due to the Kirchhoff conservation laws we have that $i = i_C + i_{RL}$, where i_C and i_{RL} are the current flowing in the capacitor and in the inductor, while i is the value of the incoming current as shown in the figure. In the right-hand side of the figure, we show a run of the system projected on the value of i_{RL} and the (implicit) time variable t. Initially, the capacitor is fully discharged and acts as a short-circuit, attracting all the incoming current: therefore in state s_0, we have that $i_C = i$ and $i_{RL} = 0$. The capacitor voltage v_C follows the differential equation $c \cdot \dot{v_C} = i_C$ (where c is the constant value of its capacitance) that corresponds to an exponential primitive function w.r.t. to a time variable. This implies that the flow of current i_C charges the capacitor and its increasing voltage rejects current i_C; this results in a decrease of i_C and a corresponding increase of i_{RL}.

Since there is no external interaction, the run starting from s_0 and reaching s_1 is a *closed evolution* of the system. It *asymptotically* stabilizes to a condition in which $i_{RL} = i$ and $i_C = 0$ so that there is no minimal region that can express this run-to-completion process as a state invariant.

For the purposes of the component, the precise current value i_{RL} is relevant only when it induces a sufficient magnetic field to act on the associated contacts. From the internal characteristics of the components, we can compute the threshold 'a' above which the value of the magnetic field induced by *RL* is sufficient to open (or close) the switch. Hence, we can focus the stabilization analysis to the predicate $\varphi \doteq i_{RL} \geq a$. We have that $\text{ATTR}(s_0, \varphi)$ holds, because there exists an evolution of s_0 reaching s_1 which is stable in φ, i.e., $\text{STABLE}(s_1, \varphi)$.

$\mathbb{P}$-grid on Σ and minimally stable states

Let *BLoc* be a set of Boolean variables that symbolically encodes the set of locations *Loc*. Namely, a location $\ell \in Loc$ is a truth assignment of the *BLoc* variables. Given a finite set of predicates $\mathbb{P} \subseteq \Psi(BLoc, Var)$, we denote with $\Phi_{\mathbb{P}}$ the set of their (finite) Boolean combinations, omitting the subscript when clear from context. We say that $\mathbb{P}$ induces a grid in the state space, since every formula $\phi \in \Phi$ defines a $\mathbb{P}$-expressible region as the set of its models in Σ. Φ induces a *finite* sublattice of $\Psi(BLoc, Var)|_{\Leftrightarrow}$, ordered by logical entailment $\Rightarrow$; the top element is $\top = \vee\Phi$, corresponding to Σ, and the bottom is $\bot$, corresponding to the empty set. Both relations 'STABLE' and 'ATTR' formalized in Definition 2 can be evaluated on the regions of the $\mathbb{P}$-grid, where it is possible to define minimality.

Definition 3 (Minimal $\mathbb{P}$-stability) For each $s \in \Sigma$ and $\phi \in \Phi$, let

$$\text{NO\_STR\_ATTR}(s, \phi) \doteq \nexists \phi' \in \Phi \,.\, \begin{pmatrix} \phi' \neq \phi \wedge (\phi' \Rightarrow \phi) \\ \wedge\ \text{ATTR}(s, \phi') \end{pmatrix};$$

$$\text{STABLE}_{\min}(s, \phi) \doteq \text{STABLE}(s, \phi) \wedge \text{NO\_STR\_ATTR}(s, \phi);$$

$$\text{ATTR}_{\min}(s, \phi) \doteq \text{ATTR}(s, \phi) \wedge \text{NO\_STR\_ATTR}(s, \phi).$$

Relation '$\text{STABLE}_{\min}$' links a state $s \in \Sigma$ with a formula $\phi \in \Phi$ if ϕ is invariant for the evolutions of s in the

closed system and if there are no smaller (i.e., STRonger) $\mathbb{P}$-representable regions in which s can converge. If such a ϕ exists, it is unique (modulo logical equivalence) and state s is said to be *minimally* $\mathbb{P}$*-stable* (or simply stable). If a state has no formula in Φ such that '$\text{STABLE}_{\min}$' holds, then it is said to be *transient*, since there exists an evolution attracted by a smaller $\mathbb{P}$-region that does not contain it:

$$\text{TRANSIENT}(s) \doteq \left(\nexists \phi \in \Phi \,.\, \text{STABLE}_{\min}(s, \phi)\right).$$

Observe that '$\text{STABLE}_{\min}$' is a partial function on Σ to $\Phi|_{\Leftrightarrow}$. On the contrary, by definition, the relation '$\text{ATTR}_{\min}$' is a subset of relation 'ATTR' and, like the latter, it is non-deterministic: '$\text{ATTR}_{\min}$' links a state with the minimal regions in which one of its evolution stabilizes. By definition, if $\text{ATTR}_{\min}(s, \phi)$ then there exists a path $s \overset{c}{\rightsquigarrow} s'$ with $\text{STABLE}_{\min}(s', \phi)$.

Example 2 Reconsider the run shown in the right-hand side of Fig. 3. With $\varphi = (RL_i \geq a)$ and $\mathbb{P} = \{\varphi\}$, the lattice of formulae in Φ is given by $\{\bot, \varphi, \neg\varphi, \top\}$, with the obvious ordering: for all $\phi \in \Phi$, $\bot \Rightarrow \phi$ and $\phi \Rightarrow \top$, while φ and $\neg\varphi$ are uncomparable.

For s_0 we have that $\text{STABLE}(s_0, \top)$. However, s_0 is *not* minimally stable in any region, since it also holds that $\text{ATTR}(s_0, \varphi)$, and it is hence *transient*. On the other hand, since there is no smaller region that attracts it, $\text{ATTR}_{\min}(s_0, \varphi)$ holds. State s_1 instead is invariant and attracted by both $\top$ and φ: we conclude that $\text{STABLE}_{\min}(s_1, \varphi)$ and $\text{ATTR}_{\min}(s_1, \varphi)$ hold.

Well defined run-to-completion

Control logics in safety critical contexts is required to have a well defined semantics: the system must have a reliable response. Hence, it is natural to expect that a state cannot stay indefinitely in a transient condition. We formalize this requirement saying that a system $\mathcal{H}$ has a *well-defined run-to-completion semantics* for the set of predicates $\mathbb{P}$, written WD- RTC$(\mathcal{H}, \mathbb{P})$ for short, if a state of $\mathcal{H}$ cannot delay indefinitely its reaching a $\mathbb{P}$-stable condition, i.e., the time needed to reach a stable setting in all possible evolutions must be bounded.

If WD- RTC$(\mathcal{H}, \mathbb{P})$, then there exists a $k \in \mathbb{R}_{\geq}$ such that, for every state $s \in \Sigma$ it holds that $\mathbf{AF}[0, k]\neg\text{TRANSIENT}(s)$ in the closed system.

In general, a fine $\mathbb{P}$-grid is more likely to violate WD- RTC $(\mathcal{H}, \mathbb{P})$. Choosing coarser predicates means that we are interested in less restrictive properties. In fact, once a condition is permanently reached, it is not possible to observe what the system does inside it.

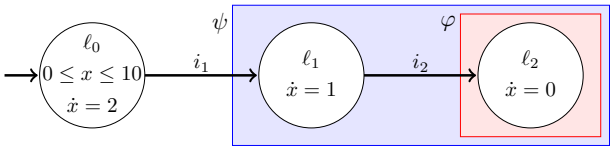


Fig. 4 Hybrid automaton whose run-to-completions are not well defined w.r.t. φ

Example 3 Consider the hybrid automaton $\mathcal{H}$ shown in Fig. 4, with $Var = \{x\}$ and $Loc = \{\ell_0, \ell_1, \ell_2\}$ with $\text{init}(\ell_0) = \top$ (and $\text{init}(\ell_1) = \text{init}(\ell_2) = \bot$).

Location ℓ_0 has invariant $(0 \leq x \leq 10)$ and flow $\dot{x} = 2$: these imply that the permanence in ℓ_0 is constrained between time 0 and 5, after which the system is forced to take the (internal) discrete transition labeled with i_1 and reach ℓ_1.

Since ℓ_1 has no invariant constraint, the system can stay in it indefinitely, letting x grow according to $\dot{x} = 1$. Nonetheless, at any moment, the system can choose to take the (internal) discrete transition labeled with i_2 and reach location ℓ_2 in which x has a constant evolution.

Let $\varphi = (\ell_2)$ be a predicate of interest: although all the states in Σ have an outgoing run stabilizing in φ, there exist runs that postpone indefinitely this event. [2]

Therefore, we say that $\mathcal{H}$ has not a well-defined run-to-completion semantics with respect to the grid induced by φ.

On the other hand, for a different predicate $\psi = (\ell_1 \vee \ell_2)$, WD- RTC$(\mathcal{H}, \{\psi\})$ holds, as the stabilization in ψ is guaranteed after 5 time units. Choosing this bigger predicate means that we are not interested in knowing what the system does inside ψ, i.e., if it is in ℓ_1 or ℓ_2.

4.3 $\mathbb{P}$-stable abstraction

$\mathbb{P}$-stability defines a simulation of stabilizing runs responding to external events.

Definition 4 ($\mathbb{P}$-stable abstraction) Let $E \doteq Lab \setminus I$ be the set of external events of a hybrid system $\mathcal{H} = \langle Loc, Var, Lab, \text{inv}, \text{init}, \text{flow}, \text{disc}\rangle$. Let $BLoc$ be a set of Boolean variables encoding the set Loc and $\mathbb{P} \subseteq \Psi(BLoc, Var)$ be a set of predicates, and Φ the set of their Boolean combinations. The (untimed) $\mathbb{P}$-stable abstraction of $\mathcal{H}$ is the finite state automaton $\mathcal{A} \doteq \langle \Phi, \Phi_0, E, \hookrightarrow\rangle$, where

$$\Phi_0 \doteq \left\{ \phi \in \Phi \;\middle|\; \begin{array}{l} \exists s_0 \in \text{init}(\mathcal{H}), s \in \Sigma \,. \\ s_0 \overset{c}{\rightsquigarrow} s \wedge \text{STABLE}_{\min}(s, \phi) \end{array} \right\}$$

[2] Notice that even by imposing a fairness condition that excludes the runs staying forever in ℓ_1, the delay would not be bounded.

and $\phi \overset{e}{\hookrightarrow} \phi'$ if and only if there exist $s, s_1, s' \in \Sigma$ such that

$$\text{STABLE}_{\min}(s, \phi), \quad s \xrightarrow{e} s_1 \overset{c}{\rightsquigarrow} s', \quad \text{STABLE}_{\min}(s', \phi').$$

Namely, a transition $\phi \overset{e}{\hookrightarrow} \phi'$ exists if it is *possible* that a state stable in ϕ, after receiving the external event e, stabilizes in the minimal invariant region ϕ'. Equivalently, there exist $s, s' \in \Sigma$ such that $\text{STABLE}_{\min}(s, \phi)$, $s \xrightarrow{e} s'$ and $\text{ATTR}_{\min}(s', \phi')$.

Intuitively, the runs of $\mathcal{A}$ represent the evolution of the truth values of the predicates in $\mathbb{P}$ in response to external events once stabilization is reached, upon "absorption" of the transient states. Initial states Φ_0 are defined similarly and represent the possible initial $\mathbb{P}$-stable regions, since $\mathcal{H}$ may start in a transient condition.

Stable-switching semantics

The definition of $\mathbb{P}$-stable abstraction captures the fact that we are studying the effects of the external inputs when $\mathcal{H}$ is *in a stable condition*: we disregard runs where external inputs are received in transient states. This "stable-switching" restriction can be intuitively thought of as the qualitative counterpart of the slow-switching hypothesis used as sufficient condition for reasoning about global stability in switched systems [31]. We informally define the restriction of $\mathcal{H}$ under stable switching as the hybrid automaton $\mathcal{H}^{ss}$ obtained by applying the guard $\neg$TRANSIENT to every external transition of $\mathcal{H}$. If WD- RTC$(\mathcal{H}, \mathbb{P})$ holds, then every state is guaranteed to eventually reach a non-transient state and possibly accept a new event.

We compare the definition of $\mathbb{P}$-stable abstraction with "classic" predicate abstraction [2, 3, 28, 30]. In our setting, the concretization of an abstract state ϕ is the set of states in Σ that are stable in ϕ, which is stricter than the set of models of ϕ. Most importantly, abstract transitions in $\mathbb{P}$-stable abstraction are witnessed by concrete paths of the form $s \xrightarrow{e} s_1 \overset{c}{\rightsquigarrow} s'$, i.e., a single external event possibly followed by multiple internal transitions. Thus, differently from predicate abstraction, a path in the abstraction may be significantly shorter than the corresponding concrete ones.

Let $\Gamma(\phi \overset{e}{\hookrightarrow} \phi')$ be the set of hybrid runs that are simulated by an abstract transition of $\mathcal{A}$; formally, for each $\phi_0 \in \Phi_0$ and for each $\phi \overset{e}{\hookrightarrow} \phi'$

$$\Gamma(\hookrightarrow \phi_0) \doteq \left\{ s \overset{c}{\rightsquigarrow} s' \;\middle|\; \begin{array}{l} s \in \text{init}(\mathcal{H}) \\ \text{STABLE}_{\min}(s', \phi_0) \end{array} \right\},$$

$$\Gamma(\phi \overset{e}{\hookrightarrow} \phi') \doteq \left\{ s \xrightarrow{e} s_1 \overset{c}{\rightsquigarrow} s' \;\middle|\; \begin{array}{l} \text{STABLE}_{\min}(s, \phi), \\ \text{STABLE}_{\min}(s', \phi') \end{array} \right\}.$$

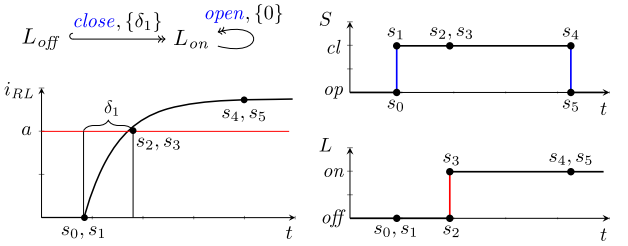


Fig. 5 Trace $s_0 \xrightarrow{close} s_1 \xrightarrow{\delta_1} s_2 \xrightarrow{rl} s_3 \xrightarrow{\delta_3} s_4 \xrightarrow{open} s_5$ projected on i_{RL}, the state of the switch S and of lamp L with respect to time t

Operator Γ is naturally extended to runs of $\mathcal{A}$ by concatenation of its applications to single transitions, and to set of runs. The $\mathbb{P}$-stable abstraction of $\mathcal{H}$ can be seen as a weak simulation [32] of $\mathcal{H}^{ss}$ which hides internal transient processes. Spurious behaviors may be introduced. In fact, two states that are stable in the same region ϕ are not necessarily connected by a concrete run and distinct areas of attraction can be represented with the same formula.

Proposition 1 *If* WD- RTC$(\mathcal{H}, \mathbb{P})$ *holds, then the stable-switching runs of* $\mathcal{H}$ *are simulated by the runs of* $\mathcal{A}$*, i.e.,* $\text{Run}(\mathcal{H}^{ss}) \subseteq \Gamma(\text{Run}(\mathcal{A}))$.

Proof Each run $\rho \in \text{Run}(\mathcal{H}^{ss})$ is of the form $s'_0 \overset{c}{\rightsquigarrow} s_0 \xrightarrow{e_0} s'_1 \overset{c}{\rightsquigarrow} s_1 \xrightarrow{e_1} s'_2 \overset{c}{\rightsquigarrow} s_2 \ldots$, where state $s'_0 \in \text{init}(\mathcal{H})$ and states $s_0, s_1, s_2, \ldots$ satisfy $\neg$TRANSIENT. By definition of TRANSIENT, for each i, there exists a ϕ_i such that $\text{STABLE}_{\min}(s_i, \phi_i)$; by Definition 4, ϕ_0 is an initial abstract state in Φ_0 and, for all i, $\phi_i \overset{e_i}{\hookrightarrow} \phi_{i+1}$ are abstract transitions in $\mathcal{A}$. It follows that the abstract run $\rho_{\mathcal{A}} = (\phi_0 \overset{e_1}{\hookrightarrow} \phi_1 \overset{e_2}{\hookrightarrow} \phi_2 \ldots)$ belongs to $\text{Run}(\mathcal{A})$ and, by definition of Γ, $\rho \in \Gamma(\rho_{\mathcal{A}})$. □

Example 4 Reconsider the circuit shown in Fig. 1 that can be modeled as a hybrid automaton with 4 locations and 16 state variables. External events include the opening/closing of switch S, while the internal discrete interactions are the automatic actions of the relay on the corresponding switches. The property of interest is the condition of lamp L. Letting i_L denote the intensity of the current passing through the lamp and 'l' be the required current to fire it, we choose $\mathbb{P} = \{i_L > l)\}$, so that the lattice induced by Φ includes $\{\bot, L_{on}, L_{off}, \top\}$ where $L_{on} \doteq (i_L > l)$ and $L_{off} \doteq \neg L_{on}$.

Figure 5 shows in three plots the same evolution of the system with respect to time, respectively, projected on the continuous variable i_{RL} (bottom-left plot), and the Boolean variable S (top-right plot) and the Boolean variable L (bottom-right plot).

Initially, all the switches are open and neither the relay coil RL nor the lamp L receive current. Hence, the system is (internally) stable in L_{off} (state s_0).

Starting from a stable state in L_{off}, when the external event '*close*' is received, the switch S is closed (state s_1), and i_{RL} starts to increase with a continuous dynamics implementing the delay of its activation. After δ_1 time, i_{RL} reaches the activation threshold 'a' (state s_2) and it closes the corresponding switches with an internal discrete transition labeled '*rl*'. The closing of RLS_2 turns on the lamp L (state s_3). The system is now stable in L_{on}.

Along the hybrid run $s_0 \xrightarrow{close} s_1 \xrightarrow{\delta_1} s_2 \xrightarrow{rl} s_3$ it holds that:

$$
\begin{aligned}
&s_0 \models (L_{off}, S_{open}, i_{RL} = 0, di_{RL}/dt = 0, i_L = 0),\\
&\quad \text{STABLE}_{\min}(s_0, L_{off}),\\
&s_1 \models (L_{off}, S_{closed}, i_{RL} = 0, di_{RL}/dt = f(i_{RL}), i_L = 0),\\
&\quad \text{ATTR}_{\min}(s_1, L_{on}),\\
&s_2 \models (L_{off}, S_{closed}, i_{RL} = a, di_{RL}/dt = f(i_{RL}), i_L = 0),\\
&\quad \text{ATTR}_{\min}(s_2, L_{on}),\\
&s_3 \models (L_{on}, S_{closed}, i_{RL} = a, di_{RL}/dt = f(i_{RL}), i_L > l),\\
&\quad \text{STABLE}_{\min}(s_3, L_{on})
\end{aligned}
$$

This run corresponds to a single transition $L_{off} \xrightarrow{close} L_{on}$ in the $\mathbb{P}$-stable abstraction, shown in the top-left of Fig. 5.

Since RLS_1 has been closed, RL receives current even if the switch S gets opened. It follows that when later receiving the external event '*open*' in state s_4 (with $\text{STABLE}_{\min}(s_4, L_{on})$), the system switches to state s_5 and the lamp stays on (namely, $\text{STABLE}_{\min}(s_5, L_{on})$). In the $\mathbb{P}$-stable abstraction, we will have the transition $L_{on} \xrightarrow{open} L_{on}$.

4.4 Timed $\mathbb{P}$-stable abstraction

We now characterize the time needed to reach a stable condition after receiving an external input.

Definition 5 (Convergence time of $\phi \overset{e}{\hookrightarrow} \phi'$) For each abstract transition $\phi \overset{e}{\hookrightarrow} \phi'$ its *convergence time* is the interval in $\mathbb{R}_{\geq}$ $ct(\phi \overset{e}{\hookrightarrow} \phi') \doteq [lb, ub]$, where

$$
lb = \inf \left\{ \tau(\rho) \;\middle|\; \rho \in \Gamma(\phi \overset{e}{\hookrightarrow} \phi') \right\},
$$

$$
ub = \sup \left\{ \tau_m(\rho) \;\middle|\; \begin{array}{l} \rho \in \Gamma(\phi \overset{e}{\hookrightarrow} \phi'), \\ \neg \text{STABLE}_{\min}(\rho[m], \phi') \end{array} \right\}.
$$

The convergence time represents the time spent in the transient states. If the system is stable in ϕ and an external event e is received, before $\min ct(\phi \overset{e}{\hookrightarrow} \phi')$ time, the system is certainly in a transient state; after $\max ct(\phi \overset{e}{\hookrightarrow} \phi')$ time the system will certainly be stable in ϕ'. Similar considerations apply to initial conditions: each $\phi \in \Phi_0$ is associated with a convergence time $ct(\hookrightarrow \phi)$ that represents the time needed to stabilize at start up.

Since we assume WD- RTC$(\mathcal{H}, \mathbb{P})$, the convergence time of Definition 5 is always bounded from above (i.e., $ub < +\infty$). In fact, an unbounded convergence time witnesses that the system can indefinitely postpone its stabilization in ϕ', which is likely an undesirable behavior. Hence, the convergence time is an effective way to detect the violation of the WD- RTC$(\mathcal{H}, \mathbb{P})$ hypothesis and yields diagnostic information to either debug the model or change the set of predicates $\mathbb{P}$.

Definition 6 (Timed $\mathbb{P}$-stable abstraction)

Given a hybrid system $\mathcal{H}$ and its (untimed) $\mathbb{P}$-stable abstraction $\mathcal{A} = \langle \Phi, \Phi_0, E, \hookrightarrow \rangle$ the corresponding *timed* $\mathbb{P}$-stable abstraction is

a timed automaton having as initial state a new state $\star$ and having

- transition $\star \overset{\varepsilon,[m,M]}{\hookrightarrow\!\!\!\!\!\rightarrow} \phi$, for each $\phi \in \Phi_0$, with $[m, M] = ct(\hookrightarrow \phi)$;
- transition $\phi \overset{e,[m,M]}{\hookrightarrow\!\!\!\!\!\rightarrow} \phi'$, for each $\phi \overset{e}{\hookrightarrow} \phi'$, with $[m, M] = ct(\phi \overset{e}{\hookrightarrow} \phi')$.

Starting from the initial state $\star$, each path reaches the first stable condition ϕ in the corresponding initialization time $ct(\hookrightarrow \phi)$. Then, after an external event e, it nondeterministically jumps to the next stable condition within the interval imposed by the associated convergence time.

We remark that the convergence time constants may not be rational due to the dynamics of the hybrid automaton being abstracted. We believe that this is not a problem because the timed abstraction is used for representational purposes only.[3]

The convergence time information defines the runs we are abstracting with a *slow-switching* characterization, rather than a stable-switching one. Considering the greatest of the delays

$$
ct = \max\{ M \mid \phi \overset{e,[m,M]}{\hookrightarrow\!\!\!\!\!\rightarrow} \phi' \text{ in } \mathcal{A} \},
$$

the refinement $\mathcal{H}_{ct}$ (see Sect. 4.1), which allows external inputs with a delay of (at least) ct, ensures that the system has always sufficient time to reach stability. Note that ct exists in $\mathbb{R}$ because neither of the convergence times is unbounded: ct characterizes how much to wait in order to safely assess stability and be ready for the next input. We obtain that $\text{Run}(\mathcal{H}_{ct}) \subseteq \text{Run}(\mathcal{H}^{ss})$, because the slow-switching hypothesis excludes the runs with a high external events frequency; since $\mathcal{A}$ abstracts the runs of $\mathcal{H}^{ss}$, as stated in Proposition 1, then the restricted concrete semantics of $\mathcal{H}_{ct}$ is compliant with $\mathcal{A}$. In other words, by imposing the slow-switching constraint on $\mathcal{H}$, we obtain $\text{Run}(\mathcal{H}_{ct}) \subseteq \Gamma(\text{Run}(\mathcal{A}))$.

[3] In practice, the computation of the timed abstraction is approximated and guarantees that the constants are indeed rational.

Example 5 Reconsider the circuit of Fig. 1, whose $\mathbb{P}$-stable abstraction has been analyzed in Example 4. We can compute the timing information of the stabilization processes, obtaining $ct(L_{off} \xrightarrow{close} L_{on}) = [\delta_1, \delta_1]$; the other abstract transitions are instantaneous (i.e., their convergence time is 0, as they have no transient states). By waiting δ_1 after each external event, the system $\mathcal{H}_{\delta_1}$ follows the behaviors described by the $\mathbb{P}$-stable abstraction: namely, we know how long the switch S must stay closed in order to turn on (and keep on) the lamp.

4.5 Computation of $\mathbb{P}$-stable abstraction

In general, the synthesis of $\mathbb{P}$-stable abstraction of Definition 4 and 6 faces two main problems: attractor analysis in $\mathcal{H}$, and minimality. As for the first problem, reachability is undecidable for infinite state systems: checking the validity of the formula **EFAG**ϕ, needed for the run-to-completion processes, is not computable in general.

Even when adding hypotheses ensuring finite computability, e.g., in the finite state setting, the approach could be highly inefficient. As a matter of fact, since we require the minimality of the predicates that express the next stable state, a naive algorithm would explore the abstract domain lattice, which has a doubly exponential size in the number of predicates, and search for the minimal ones that verify **EFAG**ϕ.

For these reasons, approximations are necessary.

5 $\mathbb{P}$-stable abstraction via abstract interpretation

In this section, we rephrase the concept of $\mathbb{P}$-stable abstraction in the framework of Abstract Interpretation. This framework provides us with a formal setting to reason about (finitely computable) correct approximations and also search for a good balance between precision and efficiency.

5.1 $\mathbb{P}$-stable abstraction as Galois connection

Concrete semantics

As shown in Sect. 4, $\mathbb{P}$-stable abstraction studies how some discrete events connect regions of convergence, taken from the grid induced by $\mathbb{P}$. The concrete domain is the powerset of states $(\wp(\Sigma), \subseteq)$ and the concrete function we want to approximate models the effects of external events in E. Consider a state-transformer computing the post-image (i.e., the set of successors) of a set of source states $S \subseteq \Sigma$, following transition '$\xrightarrow{e}$':

$$\mathrm{post}(S, e) = \left\{ s' \in \Sigma \;\middle|\; \exists s \in S, s \xrightarrow{e} s' \right\}.$$

While being computable, this function is not adequate in our context, since it is not enforcing the *stable-switching* constraint. Hence, we replace $\mathcal{H}$ with its refinement $\mathcal{H}^{ss}$ by adding a temporal guard to each event in E:

$$\mathrm{post}_{ss}(S, e) \doteq \left\{ s' \in \Sigma \;\middle|\; \begin{array}{l} \exists s \in S \,.\, s \xrightarrow{e} s', \\ \neg\, \mathrm{TRANSIENT}(s) \end{array} \right\}.$$

In general, post_{ss} operator is not computable in finite time, as it may require a fixpoint computation to check the guard $\mathrm{TRANSIENT}(s)$.

Abstraction

In order to abstract a state into the tightest formulae in Φ that attract it, we choose as abstract domain the powerset of Φ. This domain can take into account multiple target regions given by the non-determinism of the attraction relation. We define a Galois connection between $(\wp(\Sigma), \subseteq)$ and $(\wp(\Phi), \subseteq)$ using the monotone functions defined, for each $S \subseteq \Sigma$ and $F \subseteq \Phi$, as follows:

$$\alpha_1(S) \doteq \left\{ \phi \in \Phi \;\middle|\; \exists s \in S \,.\, \mathrm{ATTR}_{\min}(s, \phi) \right\},$$

$$\gamma_1(F) \doteq \left\{ s \in \Sigma \;\middle|\; \begin{array}{l} \forall \phi \in \Phi \,. \\ \mathrm{ATTR}_{\min}(s, \phi) \implies \phi \in F \end{array} \right\}.$$

This construction is inspired to a similar one proposed in [25] for transition systems. Note that the partial order of the abstract domain is totally unrelated with the implication order defined on Φ (i.e., each single formula $\phi \in \Phi$ becomes an atom $\{\phi\}$ in the powerset and atoms are pairwise uncomparable). Hence, the non monotonicity on $(\Phi, \Rightarrow)$ of the function $\lambda\phi.\{s \mid \mathrm{ATTR}_{\min}(s, \phi)\}$ does not compromise the monotonicity on $(\wp(\Phi), \subseteq)$ of the function γ_1.

Proposition 2 *(α_1, γ_1) is a Galois connection:* $(\wp(\Sigma), \subseteq) \underset{\alpha_1}{\overset{\gamma_1}{\leftrightarrows}} (\wp(\Phi), \subseteq)$.

Proof Functions α_1 and γ_1 are monotone. In order to prove that (α_1, γ_1) is a Galois Connection between $(\wp(\Sigma), \subseteq)$ and $(\wp(\Phi), \subseteq)$, we show that γ_1 is the adjoint of α_1. Let $\mathrm{AT}(s) \doteq \{\phi \in \Phi \mid \mathrm{ATTR}_{\min}(s, \phi)\}$. It is easy to see that

$$\alpha_1(S) = \bigcup\nolimits_{s \in S} \mathrm{AT}(s), \quad \gamma_1(F) = \{s \in \Sigma \mid \mathrm{AT}(s) \subseteq F\}.$$

For each $F \subseteq \Phi$, the adjoint of α_1 is $\bigcup \left\{ S \subseteq \Sigma \;\middle|\; \alpha_1(S) \subseteq F \right\} = \bigcup \left\{ s \in \Sigma \;\middle|\; \mathrm{AT}(s) \subseteq F \right\} = \gamma_1(F)$. □

Using (α_1, γ_1), the best correct approximation for function post_{ss} is

$$\mathrm{post}^{\sharp}_1(\{\phi\}, e) \doteq \alpha_1(\mathrm{post}_{ss}(\gamma_1(\{\phi\}), e)),$$

which is indeed equivalent to the relation introduced in Definition 4.

Proposition 3 *Let $\mathcal{A}$ be the $\mathbb{P}$-stable abstraction of $\mathcal{H}$. For $\phi, \phi' \in \Phi$ and $e \in E$, $\phi \overset{e}{\hookrightarrow} \phi'$ in $\mathcal{A}$ if and only if $\phi' \in \mathrm{post}_1^\sharp(\{\phi\}, e)$; also, $\Phi_0 = \alpha_1(\mathrm{init}(\mathcal{H}))$.*

Proof We first prove that, for each $s \in \Sigma$ and $\phi \in \Phi$,

$$\neg(\textsc{transient}(s)) \wedge s \in \gamma_1(\{\phi\}) \iff \textsc{stable}_{\min}(s, \phi). \quad (1)$$

Namely, if s is minimally stable in some formula $\psi \in \Phi$ and it is minimally attracted only by $\phi \in \Phi$, then $\psi = \phi$:

$$\begin{aligned}
&\neg\textsc{transient}(s) \wedge s \in \gamma_1(\{\phi\}) \\
&\text{[by def of TRANSIENT]} \\
&\iff \exists \psi \in \Phi \,.\, \textsc{stable}_{\min}(s, \psi) \wedge s \in \gamma_1(\{\phi\}) \\
&\text{[by def of } \gamma_1\text{]} \\
&\iff \exists \psi \in \Phi \,.\, \textsc{stable}_{\min}(s, \psi) \wedge \textsc{at}(s) = \{\phi\} \\
&\text{[by def of STABLE}_{\min}\text{ and ATTR}_{\min}\text{]} \\
&\iff \textsc{stable}_{\min}(s, \phi)
\end{aligned}$$

We now prove the first part of the statement:

$$\begin{aligned}
&\phi' \in \mathrm{post}_1^\sharp(\{\phi\}, e) \\
&\text{[by def of } \mathrm{post}_1^\sharp\text{]} \\
&\iff \phi' \in \alpha_1(\mathrm{post}_{ss}(\gamma_1(\{\phi\}), e)) \\
&\text{[by def of } \alpha_1\text{]} \\
&\iff \exists s_1 \,.\, \left(s_1 \in \mathrm{post}_{ss}(\gamma_1(\{\phi\}), e) \wedge \textsc{attr}(s_1, \phi')\right) \\
&\text{[by def of } \mathrm{post}_{ss}\text{]} \\
&\iff \exists s, s_1 \,.\, \begin{pmatrix} \neg\textsc{transient}(s) \wedge s \in \gamma_1(\{\phi\}) \wedge \\ s \xrightarrow{e} s_1 \wedge \textsc{attr}(s_1, \phi') \end{pmatrix} \\
&\text{[by (1)]} \\
&\iff \exists s, s_1 \,.\, \begin{pmatrix} \textsc{stable}_{\min}(s, \phi) \wedge \\ s \xrightarrow{e} s_1 \wedge \textsc{attr}(s_1, \phi') \end{pmatrix} \\
&\text{[by def of ATTR]} \\
&\iff \exists s, s_1, s' \,.\, \begin{pmatrix} \textsc{stable}_{\min}(s, \phi) \wedge \\ s \xrightarrow{e} s_1 \overset{c}{\rightsquigarrow} s' \wedge \textsc{stable}_{\min}(s', \phi') \end{pmatrix} \\
&\text{[by def of } \overset{e}{\hookrightarrow}\text{]} \\
&\iff \phi \overset{e}{\hookrightarrow} \phi'.
\end{aligned}$$

Finally,

$$\alpha_1(\mathrm{init}(\mathcal{H})) = \left\{ \phi \in \Phi \;\middle|\; \begin{array}{l} \exists s_0 \in \mathrm{init}(\mathcal{H}) \,. \\ \textsc{attr}_{\min}(s_0, \phi) \end{array} \right\} = \Phi_0$$

directly follows from the definitions of α_1 and Φ_0. □

5.2 Approximating the $\mathbb{P}$-stable abstraction

We now consider lighter (but coarser) abstract domains, trading precision for efficiency. A first simplification is obtained by overapproximating disjunctive stable regions with their join, yielding a deterministic abstract system based on a conservative abstraction of all the possible behaviors of the concrete system. To this end, we consider $(\Phi, \Rightarrow)$ as abstract domain, rather than its powerset, and we compose the previous Galois connection with $(\wp(\Phi), \subseteq) \underset{\alpha_2}{\overset{\gamma_2}{\leftrightarrows}} (\Phi, \Rightarrow)$, where, for each $F \subseteq \wp(\Phi)$ and $\phi \in \Phi$:

$$\begin{aligned}
&\alpha_2(F) \doteq \bigvee F, \\
&\gamma_2(\phi) \doteq {\downarrow}\{\phi\} = \{\phi' \in \Phi \mid \phi' \Rightarrow \phi\}.
\end{aligned}$$

Since the number of formulae on $\mathbb{P}$ is doubly exponential in the number $n = |\mathbb{P}|$ of predicates, we further approximate this domain by using its Cartesian relaxation.

Cartesian abstraction

The Cartesian abstraction [6] of $(\Phi, \Rightarrow)$ is the set of formulae that can be obtained as conjunctions of (possibly negated) predicates in $\mathbb{P}$. It can be formally defined by considering, for each predicate $P_i \in \mathbb{P}$, the corresponding lattice of knowledge values $\mathcal{P}_{P_i} \doteq (\{\bot_i, p_i, \overline{p}_i, \top_i\}, \sqsubseteq_i)$, where $\bot_i \sqsubset_i p_i \sqsubset_i \top_i$ and $\bot_i \sqsubset_i \overline{p}_i \sqsubset_i \top_i$. Then, the lattice $(\mathbb{K}, \sqsubseteq) = \left(\otimes_{P_i \in \mathbb{P}} \mathcal{P}_{P_i}\right)$ is obtained by combining all these lattices using the smash product operator '$\otimes$', which avoids redundant representations of the bottom element.

Hence, every $k \in \mathbb{K}$ is either $\bot_{\mathbb{K}}$ or a vector $(k_1, \ldots, k_n)$, where each $k_i \in \{p_i, \overline{p}_i, \top_i\}$.

We will write $k \in \mathbb{K}$ to denote the corresponding formula in Φ: while $\bot_{\mathbb{K}}$ is the bottom element of Φ (i.e., $\bot$ itself), the vector $(k_1, \ldots, k_n)$ represents the formula

$$\left(\bigwedge_{k_i = p_i} P_i \wedge \bigwedge_{k_i = \overline{p}_i} \neg P_i \right).$$

Note that $(\mathbb{K}, \sqsubseteq)$ is a meet sublattice of $(\Phi, \Rightarrow)$; in particular, we will abuse notation by writing $\Rightarrow$ (resp., $\wedge$) to denote the partial order (resp., the greatest lower bound operator) on $\mathbb{K}$. Joins are not preserved because the Cartesian domain overapproximates disjunctions between predicates. We define a Galois connection $(\Phi, \Rightarrow) \underset{\alpha_3}{\overset{\gamma_3}{\leftrightarrows}} (\mathbb{K}, \Rightarrow)$ with

$$\begin{aligned}
&\alpha_3(\phi) \doteq \bigwedge \{k \in \mathbb{K} \mid \phi \Rightarrow k\}, \\
&\gamma_3(k) \doteq k.
\end{aligned}$$

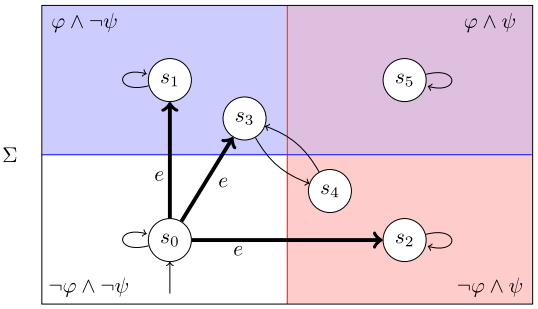


Fig. 6 Non-deterministic transitions on a $\mathbb{P}$ grid

Composition of Galois connections

We approximate the system built in Definition 4, which is yield by (α_1, γ_1), with the compositions:

$$(\wp(\Sigma), \subseteq) \underset{\alpha_1}{\overset{\gamma_1}{\leftrightarrows}} (\wp(\Phi), \subseteq) \underset{\alpha_2}{\overset{\gamma_2}{\leftrightarrows}} (\Phi, \Rightarrow) \underset{\alpha_3}{\overset{\gamma_3}{\leftrightarrows}} (\mathbb{K}, \Rightarrow).$$

Definition 7 (Approximated system $\mathcal{A}^\sharp$) Let $\alpha \doteq \alpha_3 \circ \alpha_2 \circ \alpha_1$ and $\gamma \doteq \gamma_1 \circ \gamma_2 \circ \gamma_3$. The finite state automaton $\mathcal{A}^\sharp \doteq \langle \mathbb{K}, \mathbb{K}_0, E, \hookrightarrow \rangle$ has initial states $\mathbb{K}_0 \doteq \alpha(\text{init}(\mathcal{H}))$ and $k \overset{e}{\hookrightarrow} k'$ if and only if $k' = \text{post}^\sharp(k, e)$, where $\text{post}^\sharp$ is the best correct approximation of function post_{ss}:

$$\text{post}^\sharp(k, e) = \alpha(\text{post}_{ss}(\gamma(k), e)).$$

By definition, $\gamma \circ \alpha$ is an approximation of $\gamma_1 \circ \alpha_1$: since all the concrete functions we are dealing with are monotone, the analysis done with the pair (α, γ) is a sound abstraction of the one done with (α_1, γ_1). It follows that the automaton $\mathcal{A}^\sharp$ of Definition 7 is a sound overapproximation of the automaton $\mathcal{A}$ of Definition 4. Spurious behaviors can be introduced mainly because non-deterministic trajectories are conservatively abstracted with a single transition. With α_2 we lose the ability to distinguish between states that are stable in a region with the ones that are stable in a greater one: we lose the minimality of the stable predicates, in exchange for the monotonicity of the concretization function γ_2 on Φ. Also, α_3 overapproximates the disjunctions on $\mathbb{K}$.

The following example highlights the effects of α_2 and α_3 on the $\mathbb{P}$-stable abstract automaton.

Example 6 Consider the automaton in the state space Σ shown in Fig. 6, and the grid induced by $\mathbb{P} = \{\varphi, \psi\}$. Φ is formed by all the possible Boolean combinations of predicates φ and ψ. Let $\phi_0 \doteq \neg\varphi \wedge \neg\psi$ and let state s_0 be internally stable in region ϕ_0; note that $\phi_0 \in \mathbb{K} \subseteq \Phi$ and $\{\phi_0\} \in \wp(\Phi)$. We have that $\text{STABLE}_{\min}(s_0, \phi_0)$ and

$$\{s_0\} = \gamma_1(\{\phi_0\}) = (\gamma_1 \circ \gamma_2)(\phi_0) = (\gamma_1 \circ \gamma_2 \circ \gamma_3)(\phi_0)$$

When s_0 receives an external event e, it non-deterministically jumps to states s_1, s_2 or s_3 which, by evolving according to *internal* transitions, all stabilize in different regions: states s_1 and s_2 have an (internal) self-loop in region $\phi_1 \doteq (\varphi \wedge \neg\psi)$ (in blue) and region $\phi_2 \doteq (\neg\varphi \wedge \psi)$ (in red), respectively; the runs reaching s_3 instead oscillates between states s_3 and s_4, which are inside ϕ_1 and ϕ_2, respectively. The minimality imposed by predicate $\text{STABLE}_{\min}$ distinguishes these three combinations with different stabilizing results (i.e., abstract states):

$$\text{post}_{ss}(\{s_0\}, e) = \{s_1, s_2, s_3\}, \begin{array}{l} \text{STABLE}_{\min}(s_1, \phi_1) \\ \text{STABLE}_{\min}(s_2, \phi_2) \\ \text{STABLE}_{\min}(s_3, \phi_1 \vee \phi_2). \end{array}$$

It follows that

$$\begin{aligned} \alpha_1(\text{post}_{ss}(\gamma_1(\{\phi_0\}))) &= \alpha_1(\{s_1, s_2, s_3\}) \\ &= \{\phi_1, \phi_2, \phi_1 \vee \phi_2\}, \end{aligned}$$

and the automaton $\mathcal{A}$ of Definition 4 has three transitions $\phi_0 \overset{e}{\hookrightarrow} \phi_1, \phi_0 \overset{e}{\hookrightarrow} \phi_2$ and $\phi_0 \overset{e}{\hookrightarrow} (\phi_1 \vee \phi_2)$. When considering a next external event, we consider the abstract states individually, distinguishing the source states between $\gamma_1(\{\phi_1\}) = \{s_1\}$, $\gamma_1(\{\phi_2\}) = \{s_2\}$ and $\gamma_1(\{\phi_1 \vee \phi_2\}) = \{s_3, s_4\}$. When using a precise abstraction (α_1, γ_1), the states that are minimally stable in $(\phi_1 \vee \phi_2)$ are different from the states that are minimally stable in either only ϕ_1 or only ϕ_2.

When approximating the system using (α_2, γ_2) we weaken the minimality constraint: by merging the three different regions in their join in Φ, we have that the states that are stable in $\phi_1 \vee \phi_2$ include the ones that are stable in ϕ_1 or ϕ_2 alone. Namely:

$$\begin{aligned} &(\alpha_2 \circ \alpha_1)(\text{post}_{ss}((\gamma_1 \circ \gamma_2)(\phi_0))) \\ &\quad = \alpha_2(\{\phi_1, \phi_2, \phi_1 \vee \phi_2\}) = \phi_1 \vee \phi_2. \end{aligned}$$

When also applying the Cartesian relaxation (α_3, γ_3) the disjunction is overapproximated: the smallest region in $\mathbb{K}$ containing $\phi_1 \vee \phi_2 = (\varphi \wedge \neg\psi) \vee (\neg\varphi \wedge \psi)$ is the top element:

$$\begin{aligned} &\alpha(\text{post}_{ss}(\gamma(\phi_0))) \\ &\quad = (\alpha_3 \circ \alpha_2 \circ \alpha_1)(\text{post}_{ss}((\gamma_1 \circ \gamma_2 \circ \gamma_3)(\phi_0))) \\ &\quad = (\alpha_3 \circ \alpha_2 \circ \alpha_1)(\{s_1, s_2, s_3\}) \\ &\quad = (\alpha_3 \circ \alpha_2)(\{\phi_1, \phi_2, \phi_1 \vee \phi_2\}) \\ &\quad = \alpha_3(\phi_1 \vee \phi_2) \\ &\quad = (\top_\varphi, \top_\psi) = \top. \end{aligned}$$

It follows that in the approximated abstract automaton $\mathcal{A}^\sharp$ of Definition 7, there is a unique abstract transition $\phi_0 \overset{e}{\hookrightarrow} \top$.

Observe that $\gamma(\top) = \{s_0, s_1, s_2, s_3, s_4, s_5\}$ includes all the states, even s_0 and s_5. When considering the next external event, we are allowed to start from any state, therefore introducing spurious runs.

Extending the reasoning done in Sect. 4.2, for each $k, k' \in \mathbb{K}$ and transition $k \xhookrightarrow{e} k'$ in $\mathcal{A}^\sharp$, let $\Gamma^\sharp(k \xrightarrow{e} k')$ be the set of runs abstracted by it. Since the stabilization criterion is more slack, we obtain that $\Gamma(\text{Run}(A)) \subseteq \Gamma^\sharp(\text{Run}(\mathcal{A}^\sharp))$.

The computation of convergence time information can be extended as well: the time needed by the approximated system to stabilize after an external input will be lower than the one computed for $\mathcal{A}$. Letting

$$ct^\sharp = \max\{M \mid k \xhookrightarrow{e,[m,M]} k' \text{ in } \mathcal{A}^\sharp\},$$

we have that $\mathcal{H}_{ct^\sharp}$ is a *relaxation* of $\mathcal{H}_{ct}$ and $\text{Run}(\mathcal{H}_{ct^\sharp}) \subseteq \Gamma^\sharp(\text{Run}(\mathcal{A}^\sharp))$. Namely, $\mathcal{H}_{ct^\sharp}$ defines a new (refined) concrete semantics that is compliant with $\mathcal{A}^\sharp$. Moreover, for each $t \geq ct^\sharp$, we know that the abstraction soundly analyzes the evolution of predicates along the runs of $\mathcal{H}_t$.

6 Algorithm for $\mathbb{P}$-stable abstraction

In this section, we outline the algorithm for the computation of the approximated $\mathbb{P}$-stable abstract automaton $\mathcal{A}^\sharp$ of Definition 7, as an extension of the reachability analysis for hybrid systems.

6.1 Main procedure

Pseudocode 1 shows the main procedure for the computation of $\mathcal{A}^\sharp$. It receives as input the hybrid automaton (whose events are already divided into I and E) and the set of predicates $\mathbb{P}$. In a nutshell, the abstract automaton is the result of a reachability driven fixpoint computation performed in the abstract space $\mathbb{K}$: abstract states are incrementally found and added to automaton $\mathcal{A}^\sharp$, together with the abstract transitions connecting them.

In the procedure, an abstract state is represented by a pair $\langle k, S\rangle \in \mathbb{K} \times \wp(\Sigma)$, where S tracks the (currently reachable) set of states that are stable in k. The initial abstract state $\langle k_0, S_0\rangle$ is built at line 4 by the helper function ABSTRACT$_\mathcal{H}$ (explained in detail in Sect. 6.2): intuitively, this function takes as inputs a set of states (in this case init($\mathcal{H}$)) and the predicates $\mathbb{P}$, and returns the region ($S_0 \in \wp(\Sigma)$) where the input states will stabilize, the abstract element ($k_0 \in \mathbb{K}$) representing such a region, and the convergence time ($ct_0 \in \mathbb{R}$). The abstract state $\langle k_0, S_0\rangle$ is added as initial state to the automaton $\mathcal{A}^\sharp$ and it is used to initialize a *waiting list* (line 5), holding the abstract states to be visited (as newly discovered or updated).

The main loop (lines 6–14) of the procedure analyzes the elements in the waiting list. At each iteration of the loop, an abstract state is extracted from the waiting list and the effects of all the enabled external events in E are considered. The post image S_e of the (single) discrete transition labeled with e starting from S is computed (line 9) and passed as input to the function ABSTRACT$_\mathcal{H}$ (line 10), which computes the target stable state $\langle k', S'\rangle$ reachable from S_e and convergence time ct. The new state and transition $k \xhookrightarrow{e,ct} k'$ are added to $\mathcal{A}^\sharp$ (in lines 11–12); if $\langle k', S'\rangle$ involves newly discovered information, i.e., if it is the first time that k' is found, or if it is already in $\mathcal{A}^\sharp$ but with a different convergence region S', the waiting list is updated accordingly by adding k' in it. In Sect. 6.3 we explain in detail how the procedure 'add_state' updates the elements in the abstract automaton.

6.2 Computation of $\alpha(S)$

Procedure ABSTRACT$_\mathcal{H}$, shown in Pseudocode 2, is meant to implement the abstraction function α: it takes a set of source states S and computes the element k' (in the Cartesian abstraction built on $\mathbb{P}$) that characterizes their final stabilization. As hinted before, while doing this it also computes the set S' of reachable states that are stable in k' and the convergence time ct.

First, the call to CLOSED_EVOLVE$_\mathcal{H}$ computes the evolution of the input set of states in the closed system. Such reachability analysis can be performed in various ways. We use the finite powerset of convex polyhedra with a tailored version of the fixpoint computation from [10, 29], where discrete steps (for the internal transitions only) are interleaved with calls to the time elapse operator, modeling the continuous evolution steps. All the contributions are conservatively stored in evol$_S$ until a fixpoint is reached. As in other reachability tools using polyhedra [10, 26], a delayed widening technique [16] can be adopted to guarantee termination: the user can choose to apply a convergence accelerator after a parametric number of iterations.

Once the fixpoint of the internal evolution of S has been computed in evol$_S$, function ABSTRACT$_\mathcal{H}$ proceeds by calling BUILD_ABSTRACT_STATE, which detects where these states are stabilizing, i.e., in which predicates of $\mathbb{P}$ they can stay indefinitely. In order to detect stability, we modify the input hybrid automaton by adding a new variable *clock*, keeping track of the time spent during the evolution. This preprocessing step is performed, once for all, at the beginning of the main procedure (line 2 of Pseudocode 1): for every location $\ell \in Loc$, we update $\text{inv}(\ell) := \text{inv}(\ell) \wedge (clock \geq 0)$ and $\text{flow}(\ell) := \text{flow}(\ell) \wedge (\dot{clock} = 1)$, while for all the discrete transitions $\ell \xrightarrow{a,\mu} \ell'$ in disc we update

Pseudocode 1 Build the approximated $\mathbb{P}$-stable abstraction of $\mathcal{H}$.

```
1: function BUILD_ABSTRACTION(H, P)
2:     H := add_clock_variable(H);
3:     ⟨A♯, waiting⟩ := ⟨∅, ∅⟩;
4:     ⟨k0, S0, ct0⟩ := ABSTRACT_H(init(H), P);
5:     ⟨A♯, waiting⟩ := add_init_state(A♯, ⟨k0, S0⟩, ct0, waiting);
6:     while waiting ≠ ∅ do
7:         ⟨k, S⟩ := pop(waiting);
8:         for all e ∈ E such that e enabled in S do
9:             Se := post_H(S, e);
10:            ⟨k′, S′, ct⟩ := ABSTRACT_H(Se, P);
11:            ⟨A♯, waiting⟩ := add_state(A♯, ⟨k′, S′⟩, waiting);
12:            ⟨A♯, waiting⟩ := add_trans(A♯, ⟨k, e, ct, k′⟩, waiting);
13:        end for
14:    end while
15:    return A♯;
16: end function
```

Pseudocode 2 Build the cartesian formula describing the stabilization of S w.r.t. predicates $\mathbb{P}$.

```
1: function ABSTRACT_H(S, P)
2:     evolS := CLOSED_EVOLVE_H(S);
3:     ⟨k′, S′, ct⟩ := BUILD_ABSTRACT_STATE(evolS, P);
4:     return ⟨k′, S′, ct⟩;
5: end function
```

```
1: function CLOSED_EVOLVE_H(S, P)
2:     evolS := S;                                          ▷ currently reached states
3:     X := S;                                              ▷ single iteration
4:     while true do
5:         X := dpost(X, disc^c_H, inv_H);                  ▷ internal discrete step
6:         X := X ↗ flow_H;                                 ▷ add time elapse
7:         if X ⊆ evolS then                                ▷ fixpoint check
8:             return evolS;
9:         end if
10:        if X ⊆|clock evolS then                          ▷ untimed fixpoint test
11:                                                         ▷ here X has been visited twice, with different clock values
12:            X := X.remove_upper_bounds(clock);           ▷ let clock diverge
13:        end if
14:        evolS := evolS ∪ X;                              ▷ add the contribution of the new iteration
15:    end while
16:    return evolS;
17: end function
```

```
1: function BUILD_ABSTRACT_STATE(evolS, P)
2:     let n = |P|; let k′ = (k′1, ..., k′n); let ct = [m, M];
3:     (k′1, ..., k′n) := (⊤1, ..., ⊤n); S′ := evolS; [m, M] = [0, 0];
4:     for all i ∈ {1 ... n} do
5:         t+ := max_clock(evolS ∩ Pi);
6:         t− := max_clock(evolS ∩ ¬Pi);
7:         if t+ = +∞ and t− ≠ +∞ then
8:             k′i := pi;
9:             S′ := S′ ∩ Pi;
10:            M := max{M, t−};
11:        else if t+ ≠ +∞ and t− = +∞ then
12:            k′i := p̄i;
13:            S′ := S′ ∩ ¬Pi;
14:            M := max{M, t+};
15:        end if
16:    end for
17:    m := min_clock(S′);
18:    return ⟨k′, S′, ct⟩;
19: end function
```

$$\mu := \begin{cases} \mu \wedge (clock' = 0), & \text{if } a \in E; \\ \mu \wedge (clock' = clock), & \text{if } a \in I. \end{cases}$$

Having computed the closed evolutions of S in this extended system, we identify stability in all the predicates in which evol_S has an unbounded *clock*.

This is where the benefits of the Cartesian approximation are evident. As a matter of fact, this step can be implemented with a linear number of checks, testing one predicate $P_i \in \mathbb{P}$ at a time and building the abstract element $(k'_1, \ldots, k'_n) \in \mathbb{K}$, where $n = |\mathbb{P}|$: if the clock variable is unbounded in $P_i \cap \text{evol}_S$ and bounded in $\neg P_i \cap \text{evol}_S$ then we assign $k'_i = p_i$ and update S' and M accordingly (lines 8–10 of function BUILD_ABSTRACT_STATE); similarly, if the *clock* is bounded in $P_i \cap \text{evol}_S$ and unbounded in $\neg P_i \cap \text{evol}_S$, then we assign $k'_i = \overline{p}_i$ (lines 12–14). If the clock variable is unbounded in both $P_i \cap \text{evol}_S$ and $\neg P_i \cap \text{evol}_S$, then there are stable states in both P_i and $\neg P_i$, so that $k'_i = \top_i$. In this way, even if WD- RTC$(\mathcal{H}, \mathbb{P})$ does not hold, the algorithm automatically enlarges the formula and ensures that the abstract state k is reached in a bounded time.

The introduction of variable *clock* requires some additional cares in the fixpoint computation of evol_S (function CLOSED- EVOLVE$_{\mathcal{H}}$). When checking if the states reached at the new iteration are included in the already computed ones, we need to discard the diverging *clock* variable.[4] Given two polyhedra, the *untimed* inclusion test $\mathcal{P}_1 \subseteq |_{clock} \mathcal{P}_2$ can be checked by comparing polyhedra $\mathcal{P}_1$ and $\mathcal{P}_2$ projected on variables in *Var*, i.e., removing the variable *clock*. If $X \subseteq |_{clock} \text{evol}_S$ (knowing that $X \not\subseteq \text{evol}_S$) then X holds states that have been revisited after some time, i.e., along a lasso-shaped trace with a positive duration. We need to make the time diverge along it, intuitively storing the fact that its states can be visited infinitely often. Therefore, we drop the superior constraints for *clock* in X (this can be simply done by adding a ray generator [11]) and continue with the iteration. When X will be visited again, the (timed) fixpoint of line 7 will succeed.

6.3 Computation of $\gamma(k)$

When it comes to compute $\text{post}_{ss}(\gamma(k), e)$, the stable-switching constraint forces to identify the states in $\gamma(k)$ that are not transient, i.e., that are actually stable in k. We exploit the fact that $\mathcal{A}^\sharp$ is built on the fly, following the contributions of each abstract transition until fixpoint: in this process, every new location is introduced (or visited again) by finding states that are stable in it as the result of a closed evolution. This is why, in procedure BUILD_ABSTRACTION, we keep track of the states S associated with every currently discovered state k in the automaton $\mathcal{A}^\sharp$. Namely, each abstract state k is associated with an increasing chain of set of states $\{S^i_k\}_{i\geq 0}$, where S^i_k is the set of states associated with k up to iteration i of the main loop. The fixpoint of this chain represents all the states that are stable in k: we obtain that $\text{post}^\sharp(k, e) = \alpha(\text{post}_{ss}(R, e))$, where $R = lfp^{\subseteq}\{S^i_k\}_{i\geq 0}$.

[4] This reasoning is similar to the one applied in [22] for the detection of lasso-shaped trace with diverging clock variables.

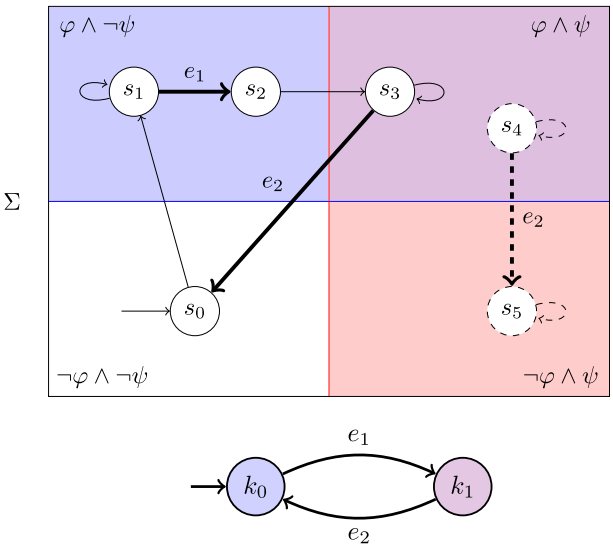


Fig. 7 Abstraction of the stable-switching runs on a $\mathbb{P}$ grid, with $k_0 = (\varphi \wedge \neg\psi)$ and $k_1 = (\phi \wedge \psi)$

Since the sequence $\{S^i_k\}_{i\geq 0}$ is not guaranteed to converge, in order to effectively compute its least fix point we use widenings when introducing a new element. Namely, let $\langle k, S' \rangle$ be the new pair obtained at line 10 of Pseudocode 1 at iteration $i + 1$. Procedure 'add_state' will compute

$$S^{i+1}_k \doteq \begin{cases} S^i_k & \text{if } S' \subseteq S^i_k \\ S^i_k \nabla \left(S^i_k \cup S' \right) & \text{otherwise.} \end{cases}$$

Widenings can be applied at each iteration or delayed after a parametric number of iterations in which $S^{i+1}_k \doteq S^i_k \cup S'$.

This technique has a double benefit: for the sake of efficiency, it allows us to exploit previous computations to obtain the stable states in k; for the sake of precision, it restricts the search to states that are reachable in $\mathcal{H}^{ss}$.

The following example shows how this procedure can improve the result by removing unreachable behaviors.

Example 7 Consider the automaton $\mathcal{H}$ in the state space Σ shown in Fig. 7, and the grid induced by $\mathbb{P} = \{\varphi, \psi\}$.

The system is initialized in state s_0 and its evolution in the closed system (i.e., only considering internal events) reaches a self-loop in state s_1: since $s_1 \models \varphi$ and $s_1 \not\models \psi$, the abstraction algorithm assigns $k_0 := (\varphi \wedge \neg\psi)$ and $S_{k_0} := \{s_1\}$, which are used to initialize the waiting list.

After extracting $\langle k_0, S_{k_0} \rangle$ from the waiting list, the external event e_1 is considered, leading to the computation of $S_{e_1} =$

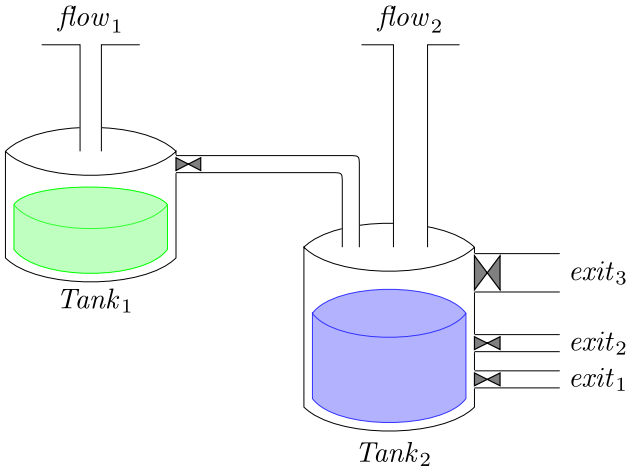


Fig. 8 Two tanks hybrid system

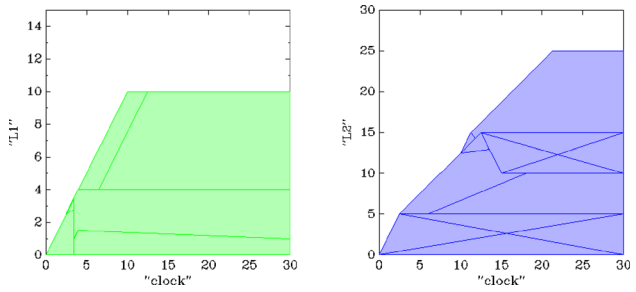


Fig. 9 Reachable states space projected on variables L_1 (left-hand side) or L_2 (right-hand side) and *clock* with time horizon constraint $0 \leq clock \leq 30$

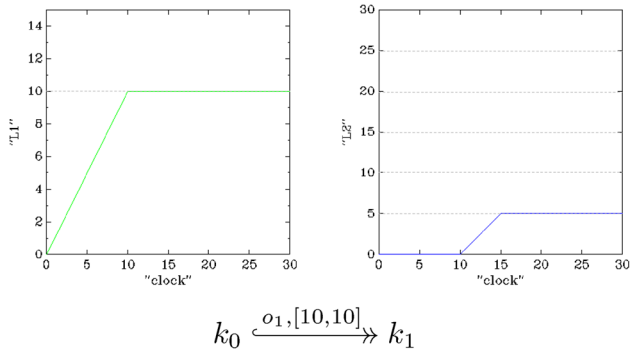


Fig. 10 Closed evolution of the system of Fig. 8 to a new $\mathbb{P}$-stable state after processing an external event, projected on *clock* and L_1 (green line) or L_2 (blue line)

$\{s_2\}$; the following evolution in the closed system reaches stability in s_3 and the new abstract state is built as $k_1 = (\varphi \wedge \psi)$ with $S_{k_1} = \{s_3\}$ (which are in turn added to the waiting list).

When processing $\langle k_1, S_{k_1} \rangle$ from the waiting list, state s_3 is considered as a source state for e_2, leading to $S_{e_2} = \{s_0\}$, whose internal evolution converges once again in s_1; since s_1 already belongs to S_{k_0}, there is no need to update it (namely, $S_{k_0} \nabla \{s_1\} = S_{k_0}$); hence, the waiting list is kept empty and the algorithm terminates after detecting a fixpoint. The resulting automaton $\mathcal{A}^\sharp$ is shown in the bottom of the figure.

This example shows the precision improvement of computing the stable states in $\gamma(k)$ by keeping track of the currently reached states in the closed evolutions. When determining $\text{post}^\sharp(k_1, e_2)$ one can start from $S_{k_1} = \{s_3\}$ only. In this way, s_4 is not considered, even if it is stable in k_1, because it is not reachable in $\mathcal{H}^{ss}$.

This would result in an overapproximation of the automaton: the evolutions starting from $\{s_3, s_4\}$ would stabilize in $\{s_1, s_5\}$, and $\top$ would be the target state.

7 Two tanks example

In this section, we show an application of our abstraction on a system composed by two tanks, shown in Fig. 8.

The external events of the system act on the incoming flows $flow_1$ and $flow_2$: when open, they inject in the corresponding tank a constant flow of water. Both tanks start empty. If $flow_1$ is opened, the level of Tank1 increases according to dynamics $\dot{L_1} = 1$. When L_1 reaches level 10, it triggers an internal discrete interaction between the two tanks, opening a connection: the flow is redirected to Tank2, which starts receiving the same amount of water incoming in Tank1. If $flow_2$ is opened, the derivative $\dot{L_2}$ for the level of Tank2 increases of 2 units. Tank2 internally regulates its level by enabling/disabling its three possible exits $exit_1$, $exit_2$ and $exit_3$: these are automatically opened when L_2 reaches levels 5, 15 and 25 and automatically closed when L_2 decreases under levels 4, 10 and 20, respectively. When open, $exit_1$ and $exit_2$ cause a decrease of 1 unit in the derivative $\dot{L_2}$, whereas $exit_3$ causes a decrease of 2 units.

In Fig. 9, we show the result of a classical reachability analysis, relating the level L_2 of Tank2 with the time variable *clock*, bounded within $0 \leq clock \leq 30$. Even though it correctly represents the reachable set (represented in the figure as the union of several, possibly overlapping polyhedra), the plots do not convey any information on the dynamic of the system. We can infer that Tank2 cannot be filled before $clock = 20$, but it is impossible to obtain information on the effects of single events, the corresponding stabilization and the associated convergence time. Moreover, in order to take into account timing information, which requires the addition of the *clock* variable, the analysis must be limited to a fixed time horizon, since otherwise the reachability analysis will never find a fixpoint due to the *clock* variable diverging.

To better understand the system, we analyze the evolutions triggered by each event. From the initial condition in which both tanks are empty, if only $flow_1$ is opened, Tank1 stabilizes

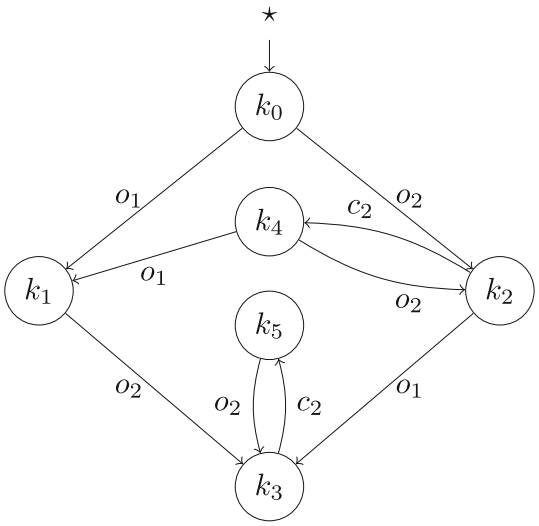


Fig. 11 Timed $\mathbb{P}$-stable abstraction for the two tanks system

$\mathbb{P}$-stable regions
$S_{k_0} = (L_1 = 0 \wedge L_2 = 0)$
$S_{k_1} = (L_1 = 10 \wedge 0 \leq L_2 \leq 5)$
$S_{k_2} = (L_1 = 0 \wedge 5 < L_2 \leq 15)$
$S_{k_3} = (L_1 = 10 \wedge 20 \leq L_2 \leq 25)$
$S_{k_4} = (L_1 = 0 \wedge 4 \leq L_2 \leq 5)$
$S_{k_5} = (L_1 = 10 \wedge 10 \leq L_2 \leq 15)$

Transitions convergence time	
$ct_0 = 0$	$ct(k_0 \hookrightarrow k_1) = 10$
$ct(k_0 \hookrightarrow k_2) = \frac{5}{2}$	$ct(k_1 \hookrightarrow k_3) \in [10, \frac{35}{3}]$
$ct(k_2 \hookrightarrow k_3) = 15$	$ct(k_2 \hookrightarrow k_4) \in [0, 10]$
$ct(k_3 \hookrightarrow k_5) \in [5, 10]$	$ct(k_4 \hookrightarrow k_1) = 10$
$ct(k_4 \hookrightarrow k_2) \in [0, 1]$	$ct(k_5 \hookrightarrow k_3) \in [5, 10]$

Fig. 12 Stable states and convergence times of the two tanks system

with $L_1 = 10$ and Tank2 stabilizes with $L_2 = 5$, due to the opening of $exit_1$ (see Fig. 10).

If instead, only $flow_2$ is open, Tank2 increases with $\dot{L_2} = 2$ until reaching $L_2 = 5$; then, due to the opening of $exit_1$, the level still increases in Tank2 with $\dot{L_2} = 1$, so that it stabilizes its level at $L_2 = 15$, due to the opening of $exit_2$.

Finally, if $flow_2$ is open after $flow_1$, i.e., after Tank1 is already filled, the level of water in Tank2 oscillates between level 25 and 20 due to the opening and closing of $exit_3$.

In Fig. 11, we show the abstract automaton obtained by the abstraction, with $\mathbb{P} = \{(L_2 \geq 10), (L_2 = 0), (L_2 \leq 5), (L_2 \leq 15), (20 \leq L_2 \leq 25)\}$ as the set of predicates and $E = \{o_1, o_2, c_2\}$ of external events, corresponding to the opening of $flow_1$ and the opening/closing of $flow_2$. Observe that these are the input of the analysis that the user must provide, depending on what features they want to analyze or what events must be considered: as an example, one could also add the simultaneous opening of the two flows in E.

The abstract automaton makes evident what are the consequences of each chosen event. Every abstract transition is associated with a precise evolution process and with timing information: the tables of Fig. 12 show the $\mathbb{P}$-stable region computed for each of the reachable abstract states and the convergence time intervals derived for each transition (i.e., the values $\langle k, S_k, ct \rangle$ resulting from the fixpoint computation of Pseudocode 1).

The evolutions presented in Figs. 10 and 13 are the states computed by Algorithm CLOSED_EVOLVE$_{\mathcal{H}}$ of Pseudocode 2 when computing the respective transitions. The procedure performs a reachability analysis, starting from the stable states in the source abstract location. The computation is done in a space with the *clock* variable, *without* time horizon. The *clock* variable is used to compute the next stable state: the algorithm detects stable loops by searching for states that are revisited with different clock values.

For example, Fig. 13a shows the evolution of the stable states in k_3 (i.e., $20 \leq L_2 \leq 25$) after the closing of $flow_2$. In the states $L_2 = 10$, the *clock* variable is unbounded and a fixpoint is obtained. Hence, the next $\mathbb{P}$-stable states are computed by checking at the grid of predicates, shown as dashed gray lines in the figure. The most precise region in the grid including the stable states is $(10 \leq L_2 \leq 15) = k_5$. Finally, the convergence time [5, 10] is computed by taking the lower and upper clock values for the entering in the detected stable region, as shown with red lines in the figure.

The stable states stored for the newly discovered abstract location k_5 are the state of the evolution after the convergence time: $10 \leq L_2 \leq 15$. This set is used to repeat the procedure to compute the effect of event o_2 on k_5. The evolution shown in Fig. 13b deserves some additional considerations. Stability is given by an oscillating behavior due to the opening and closing (i.e., discrete transitions) of $exit_3$. The classic reachable states computation would fail in detecting the fixpoint, since the *clock* variable changes every time a state is revisited. For this reason we perform an *untimed* check in line 10 of Algorithm CLOSED_EVOLVE$_{\mathcal{H}}$. In this case, it detects that the states discovered when $clock \in [15, 20]$ are exactly the same states reached previously, when $clock \in [5, 10]$. Hence, the algorithm removes the upper bounds for *clock*, and "lets it diverge" in an infinite strip.

8 Experimental evaluation

8.1 Implementation

A prototype implementation for the algorithm of Sect. 6 has been developed on top of a new stand-alone analyzer. The analyzer accepts an automaton described in a syntax similar to Timed nuXmv [22], extended with features from the HyDI language [21]. The hybrid automaton is defined with a set

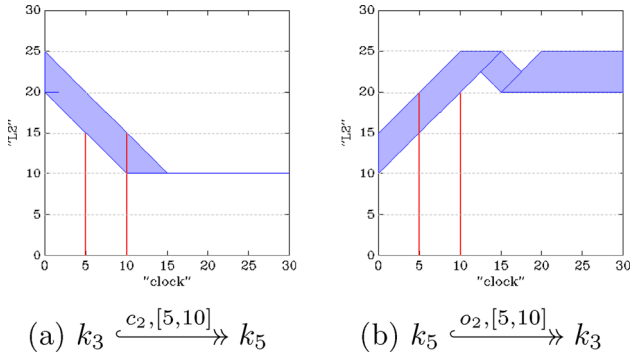


Fig. 13 Closed evolution of the system of Fig. 8 to a new $\mathbb{P}$-stable state after processing an external event, projected on *clock* and L_2. Red lines show the min and max convergence time

of Boolean variables B (including the Boolean encoding of enumerative variables) and continuous variables X. Initial condition, invariant, flow and transition relation are provided as LRA formulae on these variables: $\text{init}_{\mathcal{H}}, \text{inv}_{\mathcal{H}} \in \Psi(B, X)$ and $\text{disc}_{\mathcal{H}} \in \Psi(B \cup B', X \cup X')$. The accepted dynamic is piecewise-constant, i.e., $\text{flow}_{\mathcal{H}} \in \Psi(B, \dot{X})$. A continuous variable x can be also declared of type *real*, resp. *clock*, to express that $\text{flow}_{\mathcal{H}}$ entails $\dot{x} = 0$, resp. $\dot{x} = 1$.

LRA formulae are represented as Multi-Terminal Binary Decision Diagrams (MTBDDs) having as leaves (finite sets of) convex polyhedra. These are implemented by combining the CUDD library [37] and the PPLite library [9, 11] for the handling of the Boolean and continuous part, respectively.

The analyzer is able to perform a sound fixpoint reachability analysis of the hybrid automaton given in input. Additionally, given a set of predicates, it is able to build the $\mathbb{P}$-stable abstraction.

Since the input language is quite low level, the analyzer accepts models compiled by other tools. For example, the two tanks system described in Sect. 7 was initially modeled by exploiting the full expressivity of Timed NUXMV, which then provided the flat model to analyze.

We also experimented the interaction with NORMA [5], a tool for the analysis of relay-based circuits. NORMA provides a graphical interface to model electro-mechanical circuits and translate them in SMV models. It is currently used by the Italian Railway Network to digitalize and analyze Railway Interlocking Systems. With the NORMA front-end we can model various circuits and validate the $\mathbb{P}$-stable abstraction results. For example, we used NORMA to model the circuit of Fig. 1 and experiment with it by changing electro-mechanical features (like the delay of the relay component) or by adding new components.

8.2 Scalability evaluation

To experimentally evaluate the scalability of our analyzer, we programmatically generated several models representing

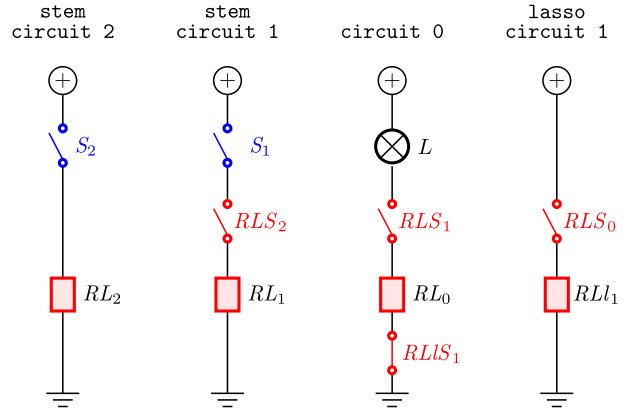


Fig. 14 Example of the R configuration, with $n = 2$, $\ell = 1$ and $c = 1$. Internal discrete interactions between a relay RL_i and its switch RLS_i are denoted in red, externally controlled switches S_i in blue. The S configuration corresponds to consider all S_i with $i < n$ always closed

relay-based circuits with run-to-completion behaviors. Let a circuit be a set of components connected by wires, and let a network be a set of circuits, possibly connected by logical bindings between relays and contacts (recall the naming convention for which a relay named *RL* controls a switch named *RLS*). The generated models are obtained from the general pattern *mode_n_ℓ_c*, where *mode* is a label identifying a specific subclass of models and $n \geq 0$, $\ell \geq 0$ and $c \geq 1$ are natural parameters controlling the complexity of the model. Intuitively, *mode_n_ℓ_c* describes a collection of c identical and independent replicas of the same network, composed by a linear sequence of $n + 1$ circuits, called the "stem portion", possibly connected to a sequence of other ℓ circuits, called the "lasso portion".

The i-th circuit in the stem portion (where $n > i > 0$) is composed by a voltage generator, two switches (S_i and RLS_{i+1}), a delayed relay (RL_i) and a ground, connected in series. The relay RL_i of the i-th circuit controls the switch RLS_i of the $(i-1)$-th circuit (on its right): RL_i is active if and only if RLS_i is closed.

The i-th circuit in the lasso portion is composed by a voltage generator, a switch $RLlS_{i-1}$, a relay RLl_i and a ground, connected in series; when charged, relay RLl_i controls the closing of the corresponding switch $RLlS_i$ in the $(i+1)$-th component, if $i < \ell$.

Circuit 0 has a lamp L, which fires only if it is traversed by enough current, and, when $\ell > 0$, is provided with another switch $RLlS_\ell$. The latter is initially closed and is controlled by the last circuit of the lasso portion: RLl_ℓ is active if and only if $RLlS_\ell$ is *open*. Hence, the relay charging process in the lasso portion produces an oscillating behavior, making the lamp flash with a frequency that depends on the relays' delays.

In Fig. 14, we show the instance `R_2_1_1`: since $c = 1$, we have a single network; since $n = 2$, the first $n + 1 = 3$ circuits (from left to right) form the stem portion; and, since

Table 1 $\mathbb{P}$-stable abstraction of models with run-to-completion behaviors

$\mathcal{H}$					$\mathcal{A}^\sharp$			$\mathcal{H}$					$\mathcal{A}^\sharp$		
Test	B	X	E	$\mathbb{P}$	Loc	Trans	Time(s)	Test	B	X	E	$\mathbb{P}$	Loc	Trans	Time(s)
S_1_0_1	5	7	2	2	2	4	<1	S_1_1_3	18	54	6	3	22	132	4476
S_1_1_1	7	18	2	1	2	4	<1	S_1_2_3	24	75	6	3	–	–	TO
S_1_2_1	9	25	2	1	2	4	<1	S_2_0_3	18	42	6	12	8	48	8
S_1_3_1	11	32	2	1	2	4	1	S_2_1_3	24	75	6	9	–	–	MO
S_1_4_1	13	39	2	1	2	4	2	S_3_0_3	24	63	6	18	8	48	218
S_1_5_1	15	46	2	1	2	4	10	S_3_1_3	30	96	6	15	–	–	TO
S_2_0_1	7	14	2	4	2	4	<1	S_1_0_4	16	28	8	8	16	128	2
S_2_1_1	9	25	2	3	2	4	<1	S_1_1_4	24	72	8	4	–	–	TO
S_2_2_1	11	32	2	3	2	4	1	S_2_0_4	24	56	8	16	16	128	185
S_2_3_1	13	39	2	3	2	4	2	S_2_1_4	32	100	8	12	–	–	TO
S_2_4_1	15	46	2	3	2	4	11	S_1_0_5	19	35	10	10	32	320	19
S_2_5_1	17	53	2	3	2	4	36	S_1_1_5	29	90	10	5	–	–	MO
S_3_0_1	9	21	2	6	2	4	<1	R_2_0_1	9	18	4	4	4	16	<1
S_3_1_1	11	32	2	5	2	4	<1	R_2_1_1	11	29	4	3	4	16	<1
S_3_2_1	13	39	2	5	2	4	2	R_2_2_1	13	36	4	3	4	16	2
S_3_3_1	15	46	2	5	2	4	7	R_2_3_1	15	43	4	3	4	16	10
S_3_4_1	17	53	2	5	2	4	32	R_2_4_1	17	50	4	3	4	16	44
S_3_5_1	19	60	2	5	2	4	122	R_2_5_1	19	57	4	3	4	16	181
S_4_0_1	11	28	2	8	2	4	<1	R_3_0_1	12	29	6	6	7	42	<1
S_4_1_1	13	39	2	7	2	4	1	R_3_1_1	14	40	6	5	7	42	5
S_4_2_1	15	46	2	7	2	4	8	R_3_2_1	16	47	6	5	7	42	28
S_4_3_1	17	53	2	7	2	4	33	R_3_3_1	18	54	6	5	7	42	121
S_4_4_1	19	60	2	7	2	4	113	R_3_4_1	20	61	6	5	7	42	405
S_4_5_1	21	67	2	7	2	4	375	R_3_5_1	22	68	6	5	7	42	1303
S_5_0_1	13	35	2	10	2	4	<1	R_4_0_1	16	40	8	8	11	88	10
S_5_1_1	15	46	2	9	2	4	6	R_4_1_1	18	51	8	7	19	152	1692
S_5_2_1	17	53	2	9	2	4	28	R_4_2_1	20	58	8	7	14	112	6590
S_5_3_1	19	60	2	9	2	4	102	R_4_3_1	22	65	8	7	–	–	TO
S_5_4_1	21	67	2	9	2	4	301	R_5_0_1	19	51	10	10	16	160	102
S_5_5_1	23	74	2	9	2	4	1237	R_5_1_1	21	62	10	9	–	–	TO
S_1_0_2	9	14	4	4	4	16	<1	R_2_0_2	16	36	8	8	19	152	17
S_1_1_2	13	36	4	2	8	32	22	R_2_1_2	20	58	8	6	22	176	4162
S_1_2_2	17	50	4	2	7	28	983	R_2_2_2	24	72	8	6	–	–	TO
S_1_3_2	21	64	4	2	4	16	2321	R_3_0_2	22	58	12	12	65	780	1649
S_1_4_2	25	78	4	2	–	–	MO	R_3_1_2	26	80	12	10	–	–	MO
S_2_0_2	13	28	4	8	4	16	<1	R_2_0_3	22	54	12	12	98	1176	1328
S_2_1_2	17	50	4	6	6	24	160	R_2_1_3	28	87	12	9	–	–	MO
S_2_2_2	21	64	4	6	4	16	918	P_1_0_1	3	8	2	1	2	4	<1
S_2_3_2	25	78	4	6	–	–	TO	P_2_0_1	8	16	2	1	2	4	<1
S_3_0_2	17	42	4	12	4	16	2	P_1_0_2	5	16	4	2	4	16	<1
S_3_1_2	21	64	4	10	4	16	527	P_2_0_2	15	32	4	2	–	–	TO
S_3_2_2	25	78	4	10	–	–	MO	P_1_0_3	6	24	6	3	8	48	<1
S_4_0_2	21	56	4	16	4	16	38	P_2_0_3	21	48	6	3	–	–	TO
S_4_1_2	25	78	4	14	–	–	TO	N_1_0_1	13	42	4	2	4	10	<1
S_5_0_2	25	70	4	20	4	16	437	N_1_1_1	14	61	2	2	–	–	TO

Table 1 continued

$\mathcal{H}$					$\mathcal{A}^\sharp$			$\mathcal{H}$					$\mathcal{A}^\sharp$		
Test	B	X	E	$\mathbb{P}$	Loc	Trans	Time(s)	Test	B	X	E	$\mathbb{P}$	Loc	Trans	Time(s)
S_5_1_2	29	92	4	18	–	–	TO	N_2_0_1	18	65	6	3	7	29	3
S_1_0_3	12	21	6	6	8	48	<1	N_3_0_1	24	88	8	4	6	30	1558

$\ell = 1$, we also have the lasso portion, which is formed by the two rightmost circuits.

The described setting is easily adjustable to increase the size of the model, change the stable behaviors or the convergence times. The *mode* label can take four possible values, with the following meaning:

- S (sequence): the only external event is the opening/closing of the leftmost switch S_n, which is initially open; the other switches S_{n-1}, …, S_1 in the stem portion are initially closed and cannot change their status. The lamp is supposed to be fired (for the first time) after $n \cdot [m, M]$ units of time from the closing of the switch S_n, where $[m, M]$ is the delay of the relays.
- R (receptive): the external environment controls all the switches S_n, S_{n-1}, …, S_1 in the stem portion. Here, the lamp is supposed to be fired only if all the switches are closed, with a delay that depends on the distance of the switch that was closed last.
- N (Norma): these models (which were not considered in [12]) are equivalent to those having the R label, but they have been obtained via manual modeling using the NORMA front-end.
- P (pwc dynamic): these models are similar to those labeled by S, but they differ in the implementation of the delayed relays. In all the previous cases, a delayed relay is implemented with a local clock variable and additional locations, corresponding to transient states "charging" and "discharging", as in [20]. In the P models, instead, the capacitor inside each *RL* is explicitly modeled and its flow is approximated with a piecewise-constant dynamic with a (static) location splitting.[5]

The $\mathbb{P}$-stable abstraction for each model considers as predicates of interest the status of the lamp and all relays.

Table 1 shows the results of the performed tests. Column "test" represents the configuration of the model *mode_n_ℓ_c*. Column "B" and "X" hold, respectively, the number of Boolean and continuous variables (including a variable for the global time) in the concrete model $\mathcal{H}$. Column "E" reports the number of external events, while "$\mathbb{P}$" is the number of considered predicates. Columns "$\mathcal{A}^\sharp$loc" and "$\mathcal{A}^\sharp$trans" describe the result of the $\mathbb{P}$-stable abstraction procedure, showing the number of abstract locations and transitions, respectively. Finally, column "time" reports the time spent in the analyzer: this value includes the compiling phase, in which the MTBDDs for the automaton's INIT, INVAR, TRANS, FLOW and predicates $\mathbb{P}$ are built, before being analyzed by the $\mathbb{P}$-stable abstraction algorithm. All benchmarks ran on a 2.40GHz processor, with a time limit set to 5 h and a memory limit set to 64GB (labels TO and MO are shown when these limits are reached).

The reported results were obtained performing reachability analysis in the domain of convex polyhedra, with widening applied at each add_state call (see Sect. 6.3). This domain may cause overapproximations in the computation of closed evolutions. As an example, the abstraction of R with $n = 2, \ell = 0, c = 2$ has more states than the expected composition of abstractions of the same test with $c = 1$ (recall that the replicas are fully independent from each other). New abstract locations are obtained when enlarging the stable states and introducing spurious behaviors. By running the same test in the domain of finite powerset of polyhedra and with a delayed widening technique, we obtain exactly the composition of the abstractions of the single replica of the network in 980 seconds.

The benchmark suite contains almost one hundred benchmarks including the 34 tests that were presented in [12] and new ones, generated in the same way but with higher parameters. Also the N category is new. While the algorithm for $\mathbb{P}$-stable abstraction is in practice unchanged with respect to the implementation of [12], the analyzer was considerably improved, especially in the interaction between CUDD and PPLite libraries. On the common 34 benchmarks, the new implementation achieves an average speedup factor of 40.

9 Conclusions

In this paper, we tackled the problem of synthesizing an abstract representation of the stabilizing behavior of hybrid automata. We defined $\mathbb{P}$-stable abstractions that have two key distinguishing features: first, they provide the most precise account—with respect to the given set of predicates—of the

[5] A similar technique to handle piecewise-affine flows is adopted also in tools PHAVer [26] and PHAVerLite [10]: in these analyzers, an on-the-fly location splitting approximates a piecewise-linear dynamic (like $c \cdot \dot{v_C} = i_C$) with a piecewise-constant one, up to a desired level of precision. Here instead we have explicitly split the location in the model, and manually computed the piecewise-constant overapproximation on the lower and upper bounds.

evolution between stable conditions in response to external events; second, they include timing information derived from the duration of the stabilization process, which provides suitable values for slow-switching control. We proved that the problem of synthesizing $\mathbb{P}$-stable abstractions can be cast in the framework of Abstract Interpretation, and presented a general synthesis algorithm which allows approximating $\mathbb{P}$-stable abstractions with precision depending on the abstract domain being adopted. We showed that $\mathbb{P}$-stable abstractions are very informative from a representational standpoint. The experimental evaluation demonstrates the applicability of the procedure to models representing relay-base circuits, with substantial improvements with a previous implementation.

In the future, we will investigate the use of symbolic techniques such as SMT to complement Abstract Interpretation and further improve the scalability and the precision of the engine. On the application side, the synthesis of $\mathbb{P}$-stable abstraction is currently being integrated within an industrial tool chain of the Italian Railway Network [4, 20]. Specifically, the aim is to reverse-engineer legacy relay-based railways interlocking systems, using the $\mathbb{P}$-stable abstraction as reference specification for a computer-based equivalent solution.

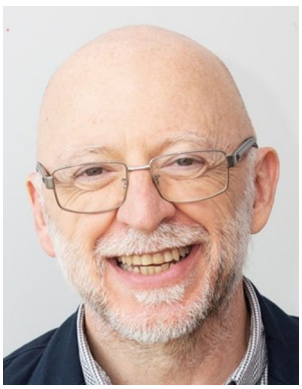
Funding Open access funding provided by Università degli Studi di Trento within the CRUI-CARE Agreement. This work has been partly supported by the project "AI@TN" funded by the Autonomous Province of Trento and by the PNRR project FAIR - Future AI Research (PE00000013), under the NRRP MUR program funded by the NextGenerationEU.

References

1. Alur, R., Dill, D.L.: A theory of timed automata. Theor. Comput. Sci. **126**(2), 183–235 (1994). https://doi.org/10.1016/0304-3975(94)90010-8
2. Alur, R., Dang, T., Ivancic, F.: Reachability analysis of hybrid systems via predicate abstraction. In: Tomlin, C.J., Greenstreet, M.R. (eds) Hybrid Systems: Computation and Control, 5th International Workshop, HSCC 2002, Stanford, CA, USA, March 25–27, 2002, Proceedings, Lecture Notes in Computer Science, vol 2289. Springer, pp 35–48, (2002) https://doi.org/10.1007/3-540-45873-5_6
3. Alur, R., Dang, T., Ivancic, F.: Counter-example guided predicate abstraction of hybrid systems. In: Garavel, H., Hatcliff, J. (eds) Tools and Algorithms for the Construction and Analysis of Systems, 9th International Conference, TACAS 2003, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2003, Warsaw, Poland, April 7–11, 2003, Proceedings, Lecture Notes in Computer Science, vol 2619. Springer, pp 208–223, (2003). https://doi.org/10.1007/3-540-36577-X_15
4. Amendola, A., Becchi, A., Cavada, R., et al.: A model-based approach to the design, verification and deployment of railway interlocking system. In: Margaria, T., Steffen, B. (eds) Leveraging Applications of Formal Methods, Verification and Validation: Applications - 9th International Symposium on Leveraging Applications of Formal Methods, ISoLA 2020, Rhodes, Greece, October 20–30, 2020, Proceedings, Part III, Lecture Notes in Computer Science, vol 12478. Springer, pp 240–254, (2020). https://doi.org/10.1007/978-3-030-61467-6_16
5. Amendola, A., Becchi, A., Cavada, R., et al.: NORMA: a tool for the analysis of relay-based railway interlocking systems. In: Fisman, D., Rosu, G. (eds) Tools and Algorithms for the Construction and Analysis of Systems–28th International Conference, TACAS 2022, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Munich, Germany, April 2–7, 2022, Proceedings, Part I, Lecture Notes in Computer Science, vol 13243. Springer, pp. 125–142 (2022). https://doi.org/10.1007/978-3-030-99524-9_7
6. Ball, T., Podelski, A., Rajamani, S.K.: Boolean and cartesian abstraction for model checking C programs. Int. J. Softw. Tools Technol. Transf. **5**(1), 49–58 (2003). https://doi.org/10.1007/s10009-002-0095-0
7. Barrett, C.W., Sebastiani, R., Seshia, S.A., et al.: Satisfiability modulo theories. In: Biere, A., Heule, M., van Maaren, H., et al (eds) Handbook of Satisfiability–Second Edition, Frontiers in Artificial Intelligence and Applications, vol 336. IOS Press, pp. 1267–1329 (2021). https://doi.org/10.3233/FAIA201017
8. Becchi, A., Cimatti, A.: Abstraction modulo stability for reverse engineering. In: Shoham, S., Vizel, Y. (eds) Computer Aided Verification–34th International Conference, CAV 2022, Haifa, Israel, August 7–10, 2022, Proceedings, Part I, Lecture Notes in Computer Science, vol. 13371. Springer, pp 469–489 (2022). https://doi.org/10.1007/978-3-031-13185-1_23
9. Becchi, A., Zaffanella, E.: An efficient abstract domain for not necessarily closed polyhedra. In: Podelski, A. (ed) Static Analysis–25th International Symposium, SAS 2018, Freiburg, Germany, August 29-31, 2018, Proceedings, Lecture Notes in Computer Science, vol. 11002. Springer, pp. 146–165 (2018). https://doi.org/10.1007/978-3-319-99725-4_11
10. Becchi, A., Zaffanella, E.: Revisiting polyhedral analysis for hybrid systems. In: Chang, B.E. (ed) Static Analysis–26th International Symposium, SAS 2019, Porto, Portugal, October 8–11, 2019, Proceedings, Lecture Notes in Computer Science, vol 11822. Springer, pp. 183–202 (2019). https://doi.org/10.1007/978-3-030-32304-2_10
11. Becchi, A., Zaffanella, E.: PPLite: Zero-overhead encoding of NNC polyhedra. Inf. Comput. **275**(104), 620 (2020). https://doi.org/10.1016/j.ic.2020.104620
12. Becchi, A., Cimatti, A., Zaffanella, E.: Synthesis of P-stable abstractions. In: de Boer, F.S., Cerone, A. (eds.) Software Engineering and Formal Methods–18th International Conference, SEFM 2020, Amsterdam, The Netherlands, September 14-18, 2020, Proceedings, Lecture Notes in Computer Science, vol. 12310. Springer, pp. 214–230 (2020). https://doi.org/10.1007/978-3-030-58768-0_12
13. Becchi, A., Cimatti, A., Zaffanella, E.: Reverse engineering with P-stable abstractions. In: Monica, D.D., Pozzato, G.L., Scala, E. (eds) Proceedings of the 3rd Workshop on Artificial Intelligence and Formal Verification, Logic, Automata, and Synthesis hosted by the Twelfth International Symposium on Games, Automata, Logics,

and Formal Verification (GandALF 2021), Padua, Italy, September 22, 2021, CEUR Workshop Proceedings, vol 2987. CEUR-WS.org, pp 91–95 (2021). https://ceur-ws.org/Vol-2987/paper16.pdf
14. Benerecetti, M., Faella, M., Minopoli, S.: Automatic synthesis of switching controllers for linear hybrid systems: Safety control. Theor. Comput. Sci. **493**, 116–138 (2013). https://doi.org/10.1016/j.tcs.2012.10.042
15. Birkhoff, G.: Lattice Theory, Colloquium Publications, vol. XXV, 3rd edn. American Mathematical Society, Providence, Rhode Island, USA (1967)
16. Blanchet, B., Cousot, P., Cousot, R., et al.: A static analyzer for large safety-critical software. In: Cytron, R., Gupta, R. (eds) Proceedings of the ACM SIGPLAN 2003 Conference on Programming Language Design and Implementation 2003, San Diego, California, USA, June 9–11, 2003. ACM, pp 196–207 (2003). https://doi.org/10.1145/781131.781153
17. Bogomolov, S., Mitrohin, C., Podelski, A.: Composing reachability analyses of hybrid systems for safety and stability. In: Bouajjani, A., Chin, W. (eds) Automated Technology for Verification and Analysis - 8th International Symposium, ATVA 2010, Singapore, September 21–24, 2010. Proceedings, Lecture Notes in Computer Science, vol 6252. Springer, pp 67–81 (2010). https://doi.org/10.1007/978-3-642-15643-4_7
18. Branicky, M.: Stability of hybrid systems: State of the art. pp 120 – 125 vol.1 (1998). https://doi.org/10.1109/CDC.1997.650600
19. Brayton, R., Tong, C.: Stability of dynamical systems: A constructive approach. Circ. Syst. IEEE Trans. CAS **26**, 224–234 (1979). https://doi.org/10.1109/TCS.1979.1084637
20. Cavada, R., Cimatti, A., Mover, S., et al.: Analysis of relay interlocking systems via SMT-based model checking of switched multi-domain Kirchhoff networks. In: Bjørner, N.S., Gurfinkel, A. (eds) 2018 Formal Methods in Computer Aided Design, FMCAD 2018, Austin, TX, USA, October 30–November 2, 2018. IEEE, pp. 1–9 (2018). https://doi.org/10.23919/FMCAD.2018.8603007
21. Cimatti, A., Mover, S., Tonetta, S.: Hydi: A language for symbolic hybrid systems with discrete interaction. In: 37th EUROMICRO Conference on Software Engineering and Advanced Applications, SEAA 2011, Oulu, Finland, August 30–September 2, 2011. IEEE Computer Society, pp. 275–278 (2011). https://doi.org/10.1109/SEAA.2011.49
22. Cimatti, A., Griggio, A., Magnago, E., et al.: Extending nuXmv with timed transition systems and timed temporal properties. In: Dillig, I., Tasiran, S. (eds) Computer Aided Verification - 31st International Conference, CAV 2019, New York City, NY, USA, July 15–18, 2019, Proceedings, Part I, Lecture Notes in Computer Science, vol 11561. Springer, pp 376–386 (2019). https://doi.org/10.1007/978-3-030-25540-4_21
23. Cousot, P., Cousot, R.: Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: Graham, R.M., Harrison, M.A., Sethi, R. (eds) Conference Record of the Fourth ACM Symposium on Principles of Programming Languages, Los Angeles, California, USA, January 1977. ACM, pp. 238–252 (1977). https://doi.org/10.1145/512950.512973
24. Cousot, P., Cousot, R.: Comparing the galois connection and widening/narrowing approaches to abstract interpretation. In: Bruynooghe M, Wirsing M (eds) Programming Language Implementation and Logic Programming, 4th International Symposium, PLILP'92, Leuven, Belgium, August 26–28, 1992, Proceedings, Lecture Notes in Computer Science, vol 631. Springer, pp. 269–295, (1992). https://doi.org/10.1007/3-540-55844-6_142
25. Cousot, P., Cousot, R.: Refining model checking by abstract interpretation. Autom. Softw. Eng. **6**(1), 69–95 (1999). https://doi.org/10.1023/A:1008649901864
26. Frehse, G.: Phaver: Algorithmic verification of hybrid systems past hytech. In: Morari M, Thiele L (eds) Hybrid Systems: Computation and Control, 8th International Workshop, HSCC 2005, Zurich, Switzerland, March 9-11, 2005, Proceedings, Lecture Notes in Computer Science, vol 3414. Springer, pp. 258–273 (2005). https://doi.org/10.1007/978-3-540-31954-2_17
27. Giesl, P., Hafstein, S.F.: Computation and verification of Lyapunov functions. SIAM J. Appl. Dyn. Syst. **14**(4), 1663–1698 (2015). https://doi.org/10.1137/140988802
28. Graf, S., Saïdi, H.: Construction of abstract state graphs with PVS. In: Grumberg O (ed) Computer Aided Verification, 9th International Conference, CAV '97, Haifa, Israel, June 22-25, 1997, Proceedings, Lecture Notes in Computer Science, vol 1254. Springer, pp. 72–83 (1997). https://doi.org/10.1007/3-540-63166-6_10
29. Halbwachs, N., Proy, Y., Raymond, P.: Verification of linear hybrid systems by means of convex approximations. In: Charlier BL (ed) Static Analysis, First International Static Analysis Symposium, SAS'94, Namur, Belgium, September 28-30, 1994, Proceedings, Lecture Notes in Computer Science, vol 864. Springer, pp. 223–237 (1994). https://doi.org/10.1007/3-540-58485-4_43
30. Lahiri, S.K., Bryant, R.E., Cook, B.: A symbolic approach to predicate abstraction. In: Jr. WAH, Somenzi F (eds) Computer Aided Verification, 15th International Conference, CAV 2003, Boulder, CO, USA, July 8-12, 2003, Proceedings, Lecture Notes in Computer Science, vol 2725. Springer, pp. 141–153 (2003). https://doi.org/10.1007/978-3-540-45069-6_15
31. Liberzon, D.: Switching in Systems and Control. Systems & Control: Foundations & Applications, Birkhäuser (2003). https://doi.org/10.1007/978-1-4612-0017-8
32. Milner, R.: Communication and concurrency. PHI Series in computer science, Prentice Hall (1989)
33. Mitra, S., Liberzon, D.: Stability of hybrid automata with average dwell time: an invariant approach. In: 43rd IEEE Conference on Decision and Control, CDC 2004, Nassau, Bahamas, December 14–17, 2004. IEEE, pp. 1394–1399 (2004). https://doi.org/10.1109/CDC.2004.1430238
34. Podelski, A., Wagner, S.: Model checking of hybrid systems: From reachability towards stability. In: Hespanha, J.P., Tiwari, A. (eds) Hybrid Systems: Computation and Control, 9th International Workshop, HSCC 2006, Santa Barbara, CA, USA, March 29-31, 2006, Proceedings, Lecture Notes in Computer Science, vol. 3927. Springer, pp. 507–521, (2006). https://doi.org/10.1007/11730637_38
35. Podelski, A., Wagner, S.: Region stability proofs for hybrid systems. In: Raskin, J., Thiagarajan, P.S. (eds) Formal Modeling and Analysis of Timed Systems, 5th International Conference, FORMATS 2007, Salzburg, Austria, October 3–5, 2007, Proceedings, Lecture Notes in Computer Science, vol. 4763. Springer, pp 320–335 (2007). https://doi.org/10.1007/978-3-540-75454-1_23
36. Schupp, S., Ábrahám, E., Chen, X., et al.: Current challenges in the verification of hybrid systems. In: Berger, C., Mousavi, M.R. (eds) Cyber Physical Systems. Design, Modeling, and Evaluation - 5th International Workshop, CyPhy 2015, Amsterdam, The Netherlands, October 8, 2015, Proceedings, Lecture Notes in Computer Science, vol. 9361. Springer, pp. 8–24 (2015). https://doi.org/10.1007/978-3-319-25141-7_2
37. Somenzi, F.: CUDD: CU Decision Diagram Package Release (1998)

Anna Becchi is a Ph.D. student since 2021 in the University of Trento, Italy, in collaboration with the research institute Fondazione Bruno Kessler in Trento, Italy. She joined the Embedded Systems unit in Fondazione Bruno Kessler in 2019. Her research interests include automated formal verification and reverse-engineering of cyber-physical systems leveraging SMT-based model checking of infinite state transition systems. She also worked on techniques based on abstract interpretation, with focus on the domain of convex polyhedra, applied to both static analysis and reachability in hybrid systems.



Alessandro Cimatti is the director of the Center for Digital Industry at Fondazione Bruno Kessler, Trento, Italy. His research interests concern formal verification of industrial critical systems, decision procedures and their applications, safety analysis, diagnosis and diagnosability, planning and runtime verification. Cimatti is the author of more than 250 papers in the fields of formal methods and artificial intelligence. For his fundamental works on Bounded Model Checking and on Satisfiability Modulo Theories, Cimatti has received the TACAS 2014 Most Influential Paper award, the ETAPS 2017 Test of Time award and the CAV Award in 2018 and 2021.

Enea Zaffanella is an Associate Professor in Computer Science at the University of Parma (Italy). His research interest is in the development of analysis and verification tools based on the theory of Abstract Interpretation. He received his PhD from the University of Leeds (UK), where he was working on the analysis of variable sharing for logic languages. Later he has been working on the design and implementation of open-source abstract domain libraries for numeric domains, in particular the PPL and PPLite libraries for convex polyhedra, as well as their integration in analysis tools for cyber-physical systems and mainstream programming languages, with a focus on the design of formally correct approximation strategies that can improve the precision/efficiency tradeoff of the analyses.