Abstraction Modulo Stability for Reverse Engineering

Anna Becchi[1,2(✉)] and Alessandro Cimatti[1]

[1] Fondazione Bruno Kessler, Trento, Italy
{abecchi,cimatti}@fbk.eu
[2] University of Trento, Trento, Italy

Abstract. The analysis of legacy systems requires the automated extraction of high-level specifications. We propose a framework, called Abstraction Modulo Stability, for the analysis of transition systems operating in stable states, and responding with run-to-completion transactions to external stimuli. The abstraction captures the effects of external stimuli on the system state, and describes it in the form of a finite state machine. This approach is parametric on a set of predicates of interest and the definition of stability. We consider some possible stability definitions which yield different practically relevant abstractions, and propose a parametric algorithm for abstraction computation. The obtained FSM is extended with guards and effects on a given set of variables of interest. The framework is evaluated in terms of expressivity and adequacy within an industrial project with the Italian Railway Network, on reverse engineering tasks of relay-based interlocking circuits to extract specifications for a computer-based reimplementation.

Keywords: Timed Transition Systems · Property extraction · Simulations · Relay-based circuits

1 Introduction

The maintenance of legacy systems is known to be a very costly task, and the lack of knowledge hampers the possibility of a reimplementation with more modern technologies. Legacy systems may have been actively operating for decades, but their behavior is known only to a handful of people. It is therefore important to have automated means to reverse-engineer and understand their behavior, for example in the form of state machines or temporal properties.

We focus on understanding systems that exhibit self-stabilizing behaviors, i.e. that are typically in a stable state, and respond to external stimuli by reaching stability in a possibly different state. As an industrially relevant example, consider legacy Railway Interlocking Systems based on Relay technology (RRIS): these are electro-mechanical circuits for the control of railway stations, with thousands of components that respond to the requests of human operators to activate the shunting routes for the movement of the trains. They support a computational model based on "run-to-completion", where a change in a part

S. Shoham and Y. Vizel (Eds.): CAV 2022, LNCS 13371, pp. 469–489, 2022.
https://doi.org/10.1007/978-3-031-13185-1_23

of the circuit (e.g. a switch closing) may change the power in another part of the circuit, and in turn operate other switches, until a stable condition is (hopefully) reached. This is very different in spirit from typical "cycle-based" control implemented in computer-based systems such as SCADA.

In this paper, we tackle the problem of extracting abstract specifications of the possible behaviors of an infinite-state timed transition system. The idea is to understand how the system evolves from a stable state, in response to a given stimulus, to the next stable state. In addition, we are interested in knowing under which conditions the transitions are possible and which are the effects on selected state variables. All this information is presented in the form of an extended finite state machine, which can be seen as a collection of temporal specifications satisfied by the system.

We make the following contributions. First, we propose the general framework of *Abstraction Modulo Stability*, a white-box analysis of self-stabilizing systems with run-to-completion behavior. The set of abstract states is the grid induced by a set of given predicates of interest. The framework is generic and parameterized with respect to the notion of stability. Different notions of stability are possible, depending on several factors: remaining in a region is possible (for some paths) or necessary (for all paths); whether the horizon of persistence in the stable region is unbounded, or lower-bounded on the number of discrete transitions and/or on the actual time. The framework also takes into account the notion of reachability in the concrete space, in order to limit the amount of spurious behaviors in the abstract description. We illustrate the relations holding between the corresponding abstractions, depending on the strength of the selected notion of stability.

Second, we present a practical algorithm to compute stability abstractions. We face two key difficulties. In the general case, one abstract transition is associated to a sequence of concrete transitions, of possibly unbounded length, so that a fix point must be reached. Furthermore, we need to make sure that the sequence is starting from a reachable state. Contrast this with the standard SMT-based computation of predicate abstractions [15], where one transition in the abstract space corresponds to one concrete transition, and reachability is not considered.

Third, we show how to lift to the abstract space other relevant variables from the concrete space, so that each abstract transition is associated with guards and effects. This results in a richer abstraction where the abstract states (typically representing control modes) are complemented by information on the data flow of the additional variables (typically representing the actual control conditions in a given mode).

We experimentally evaluate the approach on several large RRIS implementing the control logic for shunting routes and switch controls. This research is strongly motivated by an ongoing activity on the migration of the Italian Railway Network from relay-based interlocking to computer-based interlocking [3]. Stability abstraction is the chosen formalism to reverse engineer the RRIS, and to automatically provide the actual specifications for computer-based interlock-

ing. We demonstrate the effectiveness of the proposed algorithms, and the crucial role of reachability in terms of precision of the abstractions.

Related Works. This work has substantial differences with most of the literature in abstraction. For example, Predicate Abstraction (PA) [11] can be directly embedded within the framework; furthermore, PA does not take into account concrete reachability; finally, an abstract transition is the direct result of a concrete transition, and not, as in our case, of a sequence of concrete transitions.

In [5] the authors propose to analyze abstract transitions between invariant regions with an approximated approach. In comparison, we propose a general framework, parameterized on the notion of stability. Additionally, we propose effective algorithms to construct automata from concrete behaviors only, and that represent symbolically the guards and the effects of the transitions.

The idea of weak bisimilarity [19], proposed for the comparison of observable behaviors of CCS, is based on collapsing sequences of silent, internal actions. The main difference with our approach is that weak bisimilarity is not used to obtain an abstraction for reverse engineering. Furthermore, in Abstraction Modulo Stability, observability is a property of states, and the silent actions are collapsed only when passing through unobservable (i.e., unstable) states.

Somewhat related are the techniques for specification mining, that have been extensively studied, for example in hardware and software. For example, DAIKON [9] extracts candidate invariant specifications from simulations. In our approach, the abstraction directly results in temporal properties that are guaranteed to hold on the system being abstracted. Yet, simulation-based techniques might be useful to bootstrap the computation of Abstraction Modulo Stability.

The work in [1] proposes techniques for the analysis of RRIS, assuming that a description of the stable states is already given. There are two key differences: first, the analysis of transient states is not considered; second, the extraction of a description in terms of stable states is a manual (and thus inefficient and error prone) task. For completeness, we mention the vast literature on the application of formal methods to railways interlocking systems (see e.g. [6,12,13,17,18]). Aside from the similarity in the application domain, these works are not directly related, given their focus on the verification of the control algorithms.

Structure of the Paper. In Sect. 2 we present the background notions. In Sect. 3 we present the framework of Abstraction Modulo Stability. In Sect. 4 we present the algorithms for computing abstraction. In Sect. 5 we present the experimental evaluation. In Sect. 6 we draw some conclusions and present the directions of future work.

2 Background

We work in the setting of Satisfiability Modulo Theories (SMT) [4], with quantifier-free first order formulae interpreted over the theory of Linear Real

Arithmetic (LRA). We use P, Q to denote sets of Boolean variables, p, q to denote truth assignments, and the standard Boolean connectives $\wedge, \vee, \neg, \rightarrow$ for conjunction, disjunction, negation and implication. $\top$ and $\bot$ define true and false respectively. For a set of variables V, let $\Psi_{\mathcal{T}}(V)$ denote the set of first-order formulae over a theory $\mathcal{T}$ with free variables in V. When clear from context we omit the subscript. Let $V' \doteq \{v' \mid v \in V\}$. For a formula $\phi \in \Psi(V)$, let ϕ' denote $\phi[V/V']$, i.e. the substitution of each variable $v \in V$ with v'.

A finite state automaton is a tuple $\mathcal{A} = \langle Q, L, Q_0, R\rangle$ where: Q is a finite set of states; L is the alphabet; $Q_0 \subseteq Q$ is the set of initial states; $R \subseteq (Q \times L \times Q)$ is the labeled transition relation. We also consider automata with transitions annotated by guards and effects expressed as SMT formulae over given sets of variables. For $(q_1, \ell, q_2) \in R$, we write $q_1 \xrightarrow{\ell}_{\mathcal{A}} q_2$. Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be two automata defined on the same set of states Q and on the same alphabet L including a label τ: we say that $\mathcal{A}_1$ weakly simulates $\mathcal{A}_2$, and we write $\mathcal{A}_1 \lesssim \mathcal{A}_2$, if whenever $q \xrightarrow{\ell}_{\mathcal{A}_1} q'$, then $q \xrightarrow{\ell}_{\mathcal{A}_2} \xrightarrow{\tau}^*_{\mathcal{A}_2} q'$, where $\xrightarrow{\tau}^*$ is a (possibly null) sequence of transitions labeled with τ.

A symbolic timed transition system is a tuple $\mathcal{M} = \langle V, C, \Sigma, \text{Init}, \text{Invar}, \text{Trans}\rangle$, where: V is a finite set of state variables; $C \subseteq V$ is a set of clock variables; Σ is a finite set of boolean variables encoding the alphabet; $\text{Init}(V)$, $\text{Invar}(V)$, $\text{Trans}(V, \Sigma, V')$ are SMT formulae describing the initial states, the invariant and the transition relation respectively. The clocks in C are real-valued variables. We restrict the formulae over clock variables to atoms of the form $c \bowtie k$, for $c \in C$, $k \in \mathbb{R}$ and $\bowtie \in \{\leq, <, \geq, >, =\}$. The clock invariants are convex. We allow the other variables in V to be either boolean or real-valued.

A state is an assignment for the V state variables, and let S denote the set of all the interpretations of V. We assume a distinguished clock variable $\mathit{time} \in C$ initialized with $\mathit{time} = 0$ in Init, representing the global time.

The system evolves following either a discrete or a timed step. The timed transition entails that there exists $\delta \in \mathbb{R}_+$ such that $c' = c + \delta$ for each clock variable $c \in C$, and $v' = v$ for all the other variables[1]. The discrete transition entails that $\mathit{time}' = \mathit{time}$ and can change the other variables instantaneously.

A valid trace π is a sequence of states $(s_0, s_1, \dots)$ that all fulfill the Invar condition, such that $s_0 \models \text{Init}$ and for all i, $(s_i, \ell_i, s_{i+1}) \models \text{Trans}(V, \Sigma, V')$ for some ℓ_i assignment to Σ. We denote with $\text{Reach}(\mathcal{M})$ the set of states that are reachable by a valid trace in $\mathcal{M}$. We adopt a hyper-dense semantics: in a trace π, time is weakly monotonic, i.e. $s_i.\mathit{time} \leq s_{i+1}.\mathit{time}$. We disregard Zeno behaviors, i.e. every finite run is a prefix of a run in which *time* diverges.

The states in which time cannot elapse, i.e. which are forced to take an instantaneous discrete transition, are called *urgent* states. We assume the existence of a boolean state variable $\mathit{urg} \in V$ which is true in all and only the urgent states. Namely, for every pair of states (s_i, s_{i+1}) in a path π where $s_i.\mathit{urg}$ is true, then $(s_i.\mathit{time} = s_{i+1}.\mathit{time})$.

[1] We abuse the notation and write $P = Q$ for $P \leftrightarrow Q$ when P and Q are Boolean variables.

We consider CTL+P [16], a branching-time temporal logic with the future and past temporal operators. A history $h = (s_0, ..., s_n)$ for $\mathcal{M}$ is a finite prefix of a trace of $\mathcal{M}$. For a CTL+P formula ψ, write $\mathcal{M}, h \models \psi$ meaning that after h, s_n satisfies ψ in $\mathcal{M}$. Operators AGψ, $E(\psi_1 \text{ U } \psi_2)$, $H\psi$ are used with their standard interpretations (*in every future ψ will always hold*, *there exists a future in which ψ_1 holds until ψ_2*, *in the current history ψ always held*, respectively).

3 The Framework of Abstraction Modulo Stability

3.1 Overview

We tackle the problem of abstracting a concrete system in order to mine relevant high-level properties about its behavior.

We are interested in how the system reacts to stimuli: when an action is performed, we want to skip the intermediate steps that are necessary to accomplish an induced effect, and evaluate how stable conditions are connected to each other. The definition of stability is the core filter that defines which states we want to observe when following a run-to-completion process, i.e., the run triggered by a stimulus under the assumption that the inputs remain stationary. In practice, several definitions of stability are necessary, each of them corresponding to a different level of abstraction.

An additional element of the desired abstraction is that relevant properties regard particular evaluations of the system. We consider a defined abstract space which intuitively holds the observable evaluations on the system, on which we will project the concrete states.

In this section we describe a general framework for *Abstraction Modulo Stability*, which is parametric with respect to the abstract domain and the definition of stability. The result will be a finite state system which simulates the original model, by preserving only the stable way-points on the abstract domain, and by skipping the transient (i.e., unstable and unobservable) states.

Finally, we define how the obtained abstract automata can be enriched with guards and effects for each transition.

Example 1. Consider as running example the timed transition system $\mathcal{S}$ shown in the right hand side of Fig. 1 which models a tank receiving a constant incoming flow of water, with an automatic safety valve.

$\mathcal{S}$ has a clock variable c which monitors the velocity of filling and emptying processes, and reads an input boolean variable in.flow. The status of this variable is controlled by the environment $\mathcal{E}$, shown in the left hand side of the figure. In the transition relation of $\mathcal{E}$, the variables in Σ encode the labels for the stimuli, which are variations of the input variable in.flow. In particular, if $\Sigma = \tau$, then in.flow is unchanged, and we say that the system $\mathcal{S}$ is not receiving any stimulus. $\mathcal{S}$ reacts accordingly to the updated in.flow$'$. The discrete transitions of $\mathcal{S}$ are labeled with guards and with resetting assignments on the clock variable (in the form *[guards]/resets*). The system starts in the Empty location. A discrete transition reacts to a true in.flow jumping in Filling and resetting $c' := 0$. The invariant $c \leq$

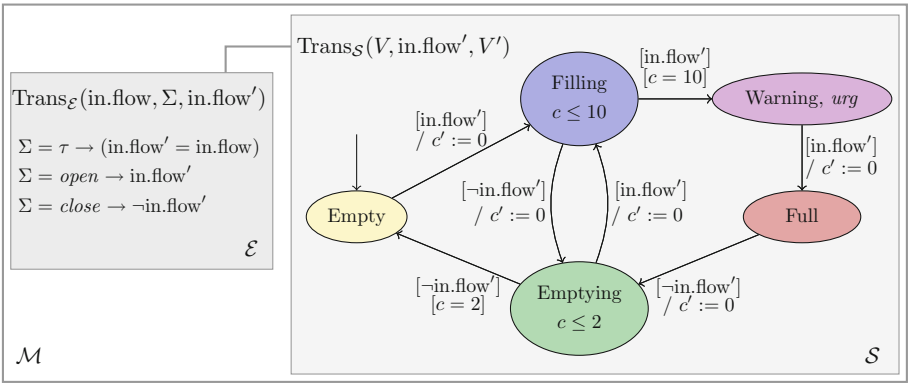

Fig. 1. A timed transition system representing a tank of water.

10 of Filling forces the system to transit to a Warning location after 10 time units, corresponding to the time needed to reach a critical level. Warning is urgent: as soon as $\mathcal{S}$ reaches this state, it is forced to take the next discrete transition. The urgency of location Warning models the causality relation between the evaluation on the level of water and the instantaneous opening of a safety valve. Due to the latter, in location Full the system dumps all the incoming water and keeps the level of water stable. If the input is closed, $\mathcal{S}$ transits in Emptying. In this condition, water is discharged faster: after 2 time units the system is again in Empty. Transitions between Filling and Emptying describe the system's reaction to a change of the input while in charging/discharging process.

We consider as predicates of interest exactly the five locations of the system. The stability abstraction of the composed system is meant to represent the stable conditions reached after the triggering events defined by Σ.

3.2 Abstraction Modulo Stability

Consider a symbolic timed transition system $\mathcal{M} = \langle X, C, \Sigma, \text{Init}, \text{Invar}, \text{Trans}\rangle$ whose discrete transitions are labeled by assignments to Σ representing stimuli. A stimulus corresponds to a variation of some variables $I \subseteq V$ which we call *input* variables. Namely, we can picture $\mathcal{M}$ as a closed system partitioned into an *environment* $\mathcal{E}$ which changes the variables I, and a open system $\mathcal{S}$ which reads the conditions of the updated variables I and reacts accordingly: $\text{Trans}(X, \Sigma, X') = \text{Trans}_\mathcal{E}(I, \Sigma, I') \wedge \text{Trans}_\mathcal{S}(V, I', V')$, with $V = X \setminus I$.

In particular, we assume a distinguished assignment τ to the labels Σ, corresponding to the absence of stimuli: $\text{Trans}_\mathcal{E}[\Sigma/\tau] = (I \leftrightarrow I')$. The transition labeled with τ is the *silent* or *internal* transition. It corresponds to the discrete changes which keep the inputs stationary (i.e., unchanged) and the timed transitions. We write $\mathcal{M}^\tau$ for the restriction of $\mathcal{M}$ which evolves only with the silent transition τ, i.e., under the assumption that no external interrupting action is performed on $\mathcal{S}$, so that $I \leftrightarrow I'$ is entailed by the transition relation. We assume

that $\mathcal{M}$ is never blocked waiting for an external action: this makes $\mathcal{M}^\tau$ always responsive to τ transition. Moreover, we assume that Zeno behaviors are not introduced by this restriction.

We define a framework for abstracting $\mathcal{M}$ parametric on an abstract domain Φ and a stability definition σ.

Abstract Domain. Between the variables of the system $\mathcal{M}$, consider a set of boolean variables $P \subseteq X$ representing important predicates. The abstract domain Φ is the domain of the boolean combinations of P variables.

Stability Definition. Let $\sigma(X)$ be a CTL+P formula providing a stability criterion.

Definition 1 (σ-Stability). *A concrete state s with history $h = (s_0, \ldots, s)$ is σ-stable if and only if*

$$\mathcal{M}^\tau, h \models \sigma.$$

Note that the stability is evaluated in $\mathcal{M}^\tau$, i.e. under the assumption that the inputs are stationary: at the reception of an external stimulus, a σ-stable might move to a new concrete state which does not satisfy σ. We say that a state s is σ-stable in a region $p \in \Phi$ if it is σ-stable and $s \models p$.

The states for which $\mathcal{M}^\tau, (s_0, \ldots, s) \not\models \sigma$, are said σ-*unstable*. These states might be transient during a convergence process which leads to the next stable state. In the following we will omit the prefix σ when clear from context.

Definition 2 (Abstraction Modulo σ-Stability). *Given a concrete system $\mathcal{M} = \langle X, C, \Sigma, \mathrm{Init}, \mathrm{Invar}, \mathrm{Trans}\rangle$, with $P \subseteq X$ boolean variables, the* abstraction modulo σ-stability *of $\mathcal{M}$ is a finite state automaton $\mathcal{A}_\sigma = \langle \Phi, 2^\Sigma, \mathrm{Init}_\sigma, \mathrm{Trans}_\sigma\rangle$. For each $p_0 \in \Phi$, $p_0 \models \mathrm{Init}_\sigma$ if and only if there exists a state $s_0 \in S$ such that $s_0 \models \mathrm{Init}$, and with $h_0 = (s_0)$*

$$\mathcal{M}^\tau, h_0 \models \mathrm{E}(\neg\sigma \ \mathrm{U} \ (\sigma \wedge p_0)).$$

For each $p_1, p_2 \in \Phi$, $\ell \in 2^\Sigma$, the triple $(p_1, \ell, p_2) \models \mathrm{Trans}_\sigma$ if and only if there exist states $s_0, s_1, s_2 \in S$ and histories $h_1 = (s_0, \ldots, s_1)$, $h_2 = (s_2)$ such that $(s_1, \ell, s_2) \models \mathrm{Trans}$, and such that

$$\mathcal{M}^\tau, h_1 \models \sigma \wedge p_1, \qquad \mathcal{M}^\tau, h_2 \models \mathrm{E}(\neg\sigma \ \mathrm{U} \ (\sigma \wedge p_2)).$$

Abstract automaton $\mathcal{A}_\sigma$ simulates with a single abstract transition a run of the concrete system $\mathcal{M}$ that connects two σ-stable states with a single event and possibly multiple steps of internal τ transitions. We call such convergence process a *run-to-completion* triggered by the initial event.

Observe that the abstraction is led by the definition of σ-stability. It preserves only the abstract regions in which there is a σ-stable state. The transient states are not exposed, hence disregarding also the behaviors of $\mathcal{M}$ in which a new external stimuli interrupts a convergence still in progress. In other words, it represents the effects of stimuli accepted only in stable conditions.

In this way, $\mathcal{A}_\sigma$ satisfies invariant properties that would have been violated in σ-unstable states, transient along an internal run-to-completion.

Reachability-Aware Abstraction. Abstractions modulo stability can be tightened by considering only concrete reachable states in $\mathcal{M}$. In fact, in the setting of reverse engineering, considering unreachable states may result in an abstraction that includes impossible behaviors that have no counterpart in the concrete space. This is done by enforcing that the first state of h_1 in Definition 2 to be reachable in $\mathcal{M}$. This is an orthogonal option to the choice of the stability definition σ.

3.3 Instantiating the Framework

The level of abstraction of $\mathcal{A}_\sigma$, i.e., the disregarded behaviors, is directly induced by the chosen definition of σ. Its adequacy depends on both the application domain and the objective of the analysis. We now explore some possibilities that we consider relevant in practice.

Predicate Abstraction. Firstly, we show that the Abstraction Modulo Stability framework is able to cover the known *predicate abstraction* [11,14]. With a trivial stability condition

$$\sigma_1 \doteq \top,$$

every concrete state s is stable and is projected in the abstract region it belongs to ($p = \exists(X \setminus P) \,.\, s$). In this way, all concrete transitions (including the timed ones) are reflected in the corresponding $\mathcal{A}_{\sigma_1}$.

Non-urgent Abstraction. Urgent states are the ones in which time cannot elapse, and are forced to transit with a discrete transition. They are usually exploited to decompose a complex action made of multiple steps and to faithfully model the causality along a cyclical chain of events. Unfortunately, by construction, urgent states introduce *transient* conditions which may be physically irrelevant. In practice, in the analysis of the system's behaviors, one may want to disregard the intermediate steps of a complex instantaneous action.

To this aim, we apply the Abstraction Modulo Stability framework and keep only the states in which time can elapse for an (arbitrarily small) time bound T.

$$\sigma_2(X) \doteq \neg urg.$$

The obtained abstract automaton $\mathcal{A}_{\sigma_2}$ has transitions that correspond to *instantaneous* run-to-completion processes, skipping urgent states until time is allowed to elapse.

Example 2. On the left hand side of Fig. 2 we show the abstraction of the tank system obtained using σ_1. An abstract transition connects two predicates (recall that in this example predicates correspond to concrete locations) if they are connected in $\mathcal{S}$, by either a discrete or a timed transition.

On the right hand side of Fig. 2 we show the abstraction obtained using σ_2. With respect to $\mathcal{A}_{\sigma_1}$, here location Warning is missing, since time cannot elapse in it, and an abstract transition connects directly Filling to Full.

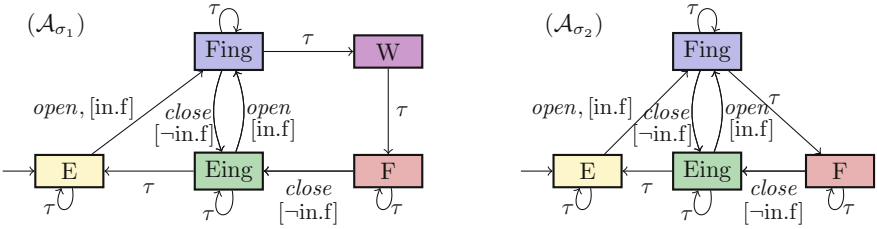

Fig. 2. Abstractions modulo σ_1 and σ_2 on the tank running example.

Eq-predicate Abstractions. Let $Eq(P)$ be a formula expressing implicitly that the interpretations of the abstract predicates are not changing during a transition (either a discrete or a timed step).

We now address the intuitive definition: *"a stable state is associated with behaviors that preserve the abstract predicates for enough time, i.e., if the system is untouched, then the predicates do not change value for a sufficient time interval"*. One can choose to measure the permanence of s in $p \in \Phi$ in terms of number of steps (e.g., at least K concrete steps, with $K \in \mathbb{N}_+$), or in terms of continuous-time (e.g., for at least T time, with $T \in \mathbb{R}_+$), or both.

This intuitive definition can be interpreted both backward and forward. In this paragraph we illustrate the backward perspective.

Consider the doubly bounded definition

$$\sigma_3^{T,K}(X) \doteq \mathrm{H}^{>T,>K} Eq(P),$$

where: $\mathcal{M}^\tau, h \models \sigma_3^{T,K}$, if and only if $h = (s_0 \dots s_i)$, with $i \geq K$ and for some $p \in 2^P$

$$\begin{pmatrix} \forall j \in [(i-K), i] : s_j \models p \ \wedge \\ s_i.time - s_{i-K}.time > T \end{pmatrix}.$$

Such characterization of stability captures the states that have been in the same predicate assignment for at least K steps *and* at least T time has elapsed in such frame. Several variants of this definition are possible, e.g. by using only one bound.

This definition is referred to as *backward* since we consider the history of the system: a stable state has a past trajectory that remained in the same abstract region for enough time/steps. It is practically relevant in contexts where it is useful to highlight the dwell time of the system in a given condition. The only visible behaviors are the ones that were exposed for sufficient time/steps.

It can be easily seen that if a history h satisfies $\sigma_3^{T_2,K}$, then it also satisfies $\sigma_3^{T_1,K}$, with $T_1 \leq T_2$.

Notably, for the instantiations of σ_3 with $K = 1$, a state is stable if it has just finished a timed transition elapsing at least T time. In the following, we omit the superscript K from $\sigma_3^{T,K}$ when $K = 1$. We have that if a history h satisfies σ_3^T, then it also satisfies σ_2. Namely, while every urgent state (i.e., a

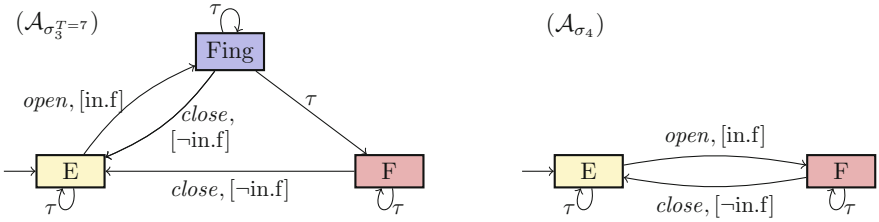

Fig. 3. Abstractions modulo $\sigma_3^{T=7}$ and σ_4 on the tank running example.

transient state for σ_2) is transient also for σ_3^T, for σ_3^T also become transient the non-urgent states that are accidentally traversed in 0 time, for example because an exiting discrete transition is immediately enabled.

Future Eq-predicate Abstractions. In contrast to the backward evaluation of σ_3, one can think of assessing stability forward, by looking at the future(s)[2] of the state. A possible definition in this perspective would be

$$\sigma_4(X) \doteq \mathrm{AG}Eq(P),$$

asking that, as long as only τ transitions are taken, the system will never change the evaluation of predicates. Namely, once a state is σ_4-stable, it can change the predicates only with an external event, and the abstract states in $\mathcal{A}_{\sigma_4}$ are closed under τ transitions. This is similar in spirit to the notion of P-stable abstraction of [5], with the difference that in the latter arbitrary regions are considered.

Within this perspective, alternative definitions can be obtained by interchanging the existential/universal path quantifiers (e.g., $\mathrm{EG}Eq(P)$ characterizes a state for which there exists a future that never changes the predicate evaluations), or by bounding the "globally" operator (e.g., $\mathrm{AG}^{>K}Eq(P)$ captures a state which is guaranteed to expose the same evaluations of predicates in the next K steps). Observe that all these variants would assess σ-stability of a state *before* it has actually proven to expose the same predicates for enough time/steps.

Example 3. On the left hand side of Fig. 2 we show the abstraction obtained with $\sigma_3^{T,K}$ definition, using $T = 7$ and $K = 1$. State Emptying is unstable, since time cannot elapse in it more than T time: namely, from Full, at the reception of the stimulus which opens in.flow, all the τ-paths lead to Empty in less than T time. On the other hand, Fing is kept, since the system may stay in this location for enough time to be considered relevant.

On the right hand side of Fig. 2 we show the abstraction obtained with σ_4. Here, the stable states are only Empty and Full: the others are abstracted since they are not invariant for the τ internal transition. Each external event directly leads to the end of a timed process which converges in the next stable state. Note that in this setting, an abstract transition labeled with τ can only be self loops.

[2] Note that, in contrast to the backward case where the past is unique, in the forward case we adopt a branching time view with multiple futures.

Here, $\mathcal{A}_{\sigma_4}$ corresponds to the P-stable abstraction because the chosen abstract domain Φ is able to express the "minimally stable" regions [5] of $\mathcal{M}$.

Observe that $\mathcal{A}_{\sigma_4}$ would be also obtained by increasing the time bound of σ_3^T, e.g., with $T = 15$.

As the examples show, different stability definitions induce abstract automata with different numbers of states and transitions. The following proposition states what is the effect on the abstract automata of making stricter the stability definition. Let us write $p_1 \xrightarrow{\ell}_{\sigma} p_2$ meaning that $(p_1, \ell, p_2) \models \text{Trans}_\sigma$ in $\mathcal{A}_\sigma$.

Proposition 1. *Let σ and σ' be two stability definitions such that every history that is σ-stable, is also σ'-stable, and let $\mathcal{A}_\sigma$ and $\mathcal{A}_{\sigma'}$ be the corresponding abstractions modulo stability of the same concrete model $\mathcal{M}$. Then, $\mathcal{A}_\sigma$ weakly simulates $\mathcal{A}_{\sigma'}$.*

Proof. By definition, if $p_1 \xrightarrow{\ell}_{\sigma} p_2$, then there exists $(s_1, \ell, s_2) \models \text{Trans}$ with (1) $\mathcal{M}^\tau, h_1 \models \sigma \wedge p_1$, and (2) $\mathcal{M}^\tau, h_2 \models \text{E}(\neg\sigma \text{ U } (\sigma \wedge p_2)$, with $h_1 = (s_0 \ldots, s_1)$ and $h_2 = (s_2)$. Since every σ-stable history is also σ'-stable, from (1) we obtain that $\mathcal{M}^\tau, h_1 \models \sigma' \wedge p_1$, and from (2) we derive

$$\begin{aligned} \mathcal{M}^\tau, h_2 \models \text{EF}(\sigma \wedge p_2) &\implies \mathcal{M}^\tau, h_2 \models \text{EF}(\sigma' \wedge p_2) \\ &\implies \mathcal{M}^\tau, h_2 \models \text{E}(\neg\sigma' \text{ U } (\sigma'\text{EX}(\neg\sigma' ... \text{ U } (\sigma' \wedge p_2)...))) \end{aligned}$$

Hence, $p_1 \xrightarrow{\ell}_{\sigma'} \xrightarrow{\tau}{}^{*}_{\sigma'} p_2$ and $\mathcal{A}_\sigma \lesssim \mathcal{A}_{\sigma'}$.

Corollary 1. *For every bounds $T_1 \leq T_2 \in \mathbb{R}_+$*

$$\mathcal{A}_{\sigma_3^{T_2}} \lesssim \mathcal{A}_{\sigma_3^{T_1}} \lesssim \mathcal{A}_{\sigma_2} \lesssim \mathcal{A}_{\sigma_1}$$

3.4 Extending with Guards and Effects

Abstract transitions in $\mathcal{A}_\sigma$ are labeled with the stimulus that has triggered the abstracted run-to-completion process. Recall that a stimulus $\ell \in 2^\Sigma$ is connected to a (possibly null) variation of the inputs I by $\text{Trans}_\mathcal{E}(I, \Sigma, I')$. A *guard* for an abstract transition (p_1, ℓ, p_2) is a formula on I' variables entailed by $\text{Trans}_\mathcal{E}[\Sigma/\ell]$ which describes the configurations of inputs that, starting from p_1 with event ℓ, lead to p_2. In order to enrich the description of the *effects* of an abstract transition, we also consider a subset of state variables $O \subseteq V$, called *output* variables. Observe that an abstract transition may be witnessed by multiple concrete paths, each with its own configuration of inputs and outputs. Hence, we can keep track of a precise correlation between guards and effects with a unique relational formula on I and O variables. This formula is obtained as a disjunction of all the configurations of inputs and outputs in the concrete states accomplishing stability in p_2 (since the configuration of I set by the stimulus is preserved by τ along the run-to-completion process).

Example 4. The stability abstractions shown in Figs. 2 and 3 are equipped with *guard* constraints, as evaluations on the original input variable in.flow, (shown in square brackets near the label of the stimuli).

4 Algorithms for Stability Abstractions

In order to build the abstract automaton structure we have to check whether there exists a (reachable) σ-stable state in p_1, with $(s_1, \ell, s_2) \models \text{Trans}$ and $\mathcal{M}^\tau, s_2 \models \text{E}(\neg\sigma \text{ U } (\sigma \wedge p_2))$, for every pair $(p_1, p_2) \in \Phi \times \Phi$. Reachability analysis and (C/)LTL model checking for infinite state systems are undecidable problems. The work in [5] computes overapproximations of the regions that are invariant for silent transitions (i.e., addresses an unbounded stability criterion AGϕ), exploiting the abstract interpretation framework. This approach also over-approximates multiple stable targets that may be given by the non-determinism of the concrete system.

Here, instead, we deal precisely with the non-determinism of the underlying concrete system by collecting information about actual, visible consequences of an action, by focusing on *bounded* stability definitions. In fact, we consider stability criteria that do not require fixpoint computations in the concrete system, *and* we under-approximate the reachability analysis fixing a bound for unstable paths. Namely, our algorithm follows an iterative deepening approach, which considers progressively longer unstable run-to-completion paths, seeking for the next stable condition.

Intuitively we search for concrete witnesses for an abstract transition (p_1, ℓ, p_2) by searching for a concrete path connecting a concrete σ-stable state s_1 in p_1 and a σ-stable state in p_2, with a bounded reachability analysis from s_1.

Notice that the algorithm builds a *symbolic* characterization for the stability automaton. In fact, instead of enumerating all $(p_1, p_2) \in \Phi \times \Phi$ and check if they are connected by some concrete path, we incrementally build a formula characterizing all the paths of $\mathcal{M}^\tau$ connecting two σ-stable states. Then, we project such formula on the P variables, hence obtaining symbolically all the abstract transitions having a witness of that length. This intuition is similar to [15] to efficiently compute predicate abstractions.

Moreover, having a formula representing finite paths of $\mathcal{M}^\tau$ connecting two σ-stable states, we can extract guards and effects with a projection on I and O variables. Namely, while checking the existence of an abstract transition, we also synthesize the formula on I and O annotating it.

A significant characteristic of our approach, also with respect to the classical instantiation of predicate abstraction, is that we refine the abstract transitions by forcing the concrete states to be reachable from the initial condition.

In the following we describe the general algorithm for computing abstractions parametric on the stability definition σ, and then show how the criteria proposed in Sect. 3.3 can be actually passed as parameter.

4.1 Symbolic Algorithm for Bounded Stability

Consider the symbolic encoding of automaton $\mathcal{M} = \langle X, C, \Sigma, \text{Init}, \text{Invar}, \text{Trans}\rangle$,[3] and a classification of the variables in X distinguishing P boolean predicates variables, I input variables, O output variables.

[3] For exposition purposes, let Trans entails both Invar and Invar$'$.

We address the computation of the formulae $\text{Init}_\sigma(P)$ and $\text{Trans}_\sigma(P, I, O, P')$, for a stability definition provided as a formula $\sigma(X_0, \ldots, X_n)$ with $n \in \mathbb{N}$.

The algorithm performs a reachability analysis based on two bounds:

- $U \in \mathbb{N}$, as the bound for the length for unstable paths.
- $L \in \mathbb{N}$, with $L \geq n + 1$, as the bound for the length of the run witnessing an abstract transition, starting from the initial state, used for the reachability-aware refinement.

Pseudocode 1. Reachability-aware symbolic computation of the abstract transition relation Trans_σ

```
1:  function EXTRACT-ABSTRACT-TRANS(Init, Trans)
2:      Trans_σ := ⊥;
3:      S := new_solver();
4:      S.ASSERT(Init(X_0));
5:      for all j ∈ [0, L) do
6:          S.ASSERT(Trans(X_j, Σ_j, X_{j+1}));
7:          if j < n + 1 then continue;
8:          S.PUSH();
9:          S.ASSERT(σ(X_{j−n}, ..., X_j));                    ▷ stable slot at j
10:         for all i ∈ reversed[j − 1 − U, j) do
11:             if i + 1 < j then
12:                 S.ASSERT(I_{i+1} = I_{i+2} ∧ ¬σ(X_{i+1−n}, ..., X_{i+1}));   ▷ unstable path
13:             S.PUSH();
14:             S.ASSERT(σ(X_{i−n}, ..., X_i) ∧ ⋀_{i−n≤h<i} I_h = I_{h+1});   ▷ stable slot at i
15:             S.ASSERT(¬Trans_σ[P/P_i, I/I_j, O/O_j, P'/P_j]);
16:             Trans_σ^{(i,j)} ← S.PROJECT-ON(P_i, I_j, O_j, P_j);
17:             Trans_σ ← Trans_σ ∨ Trans_σ^{(i,j)}[P_i/P, I_j/I, O_j/O, P_j/P'];
18:             S.POP();
19:         S.POP();
        return Trans_σ
```

Computation of Trans_σ. Pseudocode 1 shows the algorithm for extraction of the transition relation Trans_σ. It builds a formula

$$\text{Init}(X_0) \wedge \bigwedge_{0 \leq h \leq j} \text{Trans}(X_h, X_{h+1}) \wedge \left(\begin{array}{c} \sigma(X_{i-n}, ..., X_i) \wedge \bigwedge\limits_{i-n \leq h < i} I_h = I_{h+1} \wedge \\ \bigwedge\limits_{i < h < j} (I_h = I_{h+1} \wedge \neg\sigma(X_{h-n}, ..., X_h)) \wedge \\ \sigma(X_{j-n}, ..., X_j) \end{array} \right)$$

for each i, j with $0 \leq j - i \leq U$ and $j < L$. The procedure exploits the incrementality of the SMT solvers which organize assertions in a stack: the push/pop interface allows the addition of layers, in which to insert new formulae with the

ASSERT primitive. In this way, we can progressively build the path and avoid its recomputation for every pair i, j. Namely, for each $j < L$, firstly we build the path until j (line 6) and assert σ-stability in j (line 9). Then we progressively try i going backward (in order to better exploit incrementality), constrain the I variables to be unchanged, and σ-unstability (lines 11–12).

Function S.PROJECT-ON() (line 16) performs an existential quantification of the formula currently present in the solver stack. We preserve variables P_i and P_j, which characterize the two stable states connected by the transition. Variables I_j and O_j are also preserved: in this way, we extract the guards and the effects formulae directly within the building of the abstract transition. Notice that, due to the input stability hypothesis preserved during the unstable path, the input configuration read in j is the same read immediately following the external event in $i + 1$.

Every found contribute $\text{Trans}_\sigma^{(i,j)}$ is then merged in a single Trans_σ, after substitution of the variables in P, I, O, P'. Observe that an important optimization is to block the negation of the already computed formula Trans_σ (shifted in the current i, j indices) before each projection (line 15), in order to avoid recomputing the same transitions.

Reachability-Awareness. A reachability-unaware version would drop the first part of the formula characterizing the path from 0 to $i - n$.

The described algorithm is reachability-aware, meaning that every considered stable state is, by construction, reachable from the initial condition Init. This is important to extract actually concretizable behaviors, and is a main difference with respect to the classical predicate abstraction technique: it is well known that mere the projection on the boolean predicates of the single transition relation may introduce several spurious behaviors.

Note that the reachability-aware improvement is based on *concrete* reachability. In contrast, the algorithm of [5], exploits abstract reachability until fixpoint in the abstract automaton, possibly incurring in further overapproximations induced by the use of convergence accelerators.

Computation of Init_σ. The algorithm for the extraction of the initial state Init_σ is similar: it builds a formula

$$\text{Init}(X_0) \wedge \bigwedge_{0 \leq h \leq i} (\text{Trans}(X_h, X_{h+1}) \wedge I_h = I_{h+1}) \wedge \sigma(X_{i-n}, ..., X_i)$$

for every $i \leq U$. Init_σ is the collection of the contributes $\text{Init}_\sigma^{(i)}$, obtained by fixing a stable slot in the last position i and projecting on P_i variables.

4.2 Instantiating the Algorithm

The bounded stability definitions presented in Sect. 3 can be unrolled and expressed in the form $\sigma(X_0, \ldots, X_n)$

Predicate Abstraction. $\sigma_1(X_0) = \top$ trivially needs only the current variables. Observe that in this case we can use a $U = 1$ bound, since the unstability constraint is always unsatisfiable.

Non-urgent Abstraction. Having a classification of urgent conditions, also $\sigma_2(X_0) = \neg urg_0$ can be established looking only at the current variables (it only needs $n = 0$).

Eq-predicate Abstraction. More generally, given K and T bounds, we encode that the abstract region has not changed for the last K steps and that at least T time has elapsed using $n = K$ and

$$\sigma_3^{T,K}(X_0 \dots X_K) = \bigwedge_{h<K} (P_h = P_{h+1}) \wedge (time_0 + T < time_K).$$

5 Experimental Evaluation

We evaluate the applicability and the adequacy of stability abstractions for the reverse engineering of real-world Relay-based Railway Interlocking Systems.

Relay-Based Railway Interlocking Systems (RRIS). RRIS are complex electro-mechanical circuits used for the control stations and train traffic. Such systems receive stimuli from an external environment, including both human operators (e.g., performing actions on buttons) and physical entities (e.g., a train passing on some sensors). In response, they control railway elements, like signaling lights or railway switches. Internally, they use relays to propagate signals: relays are electro-mechanical components which, when activated, change the position of an associated contact after a (possibly null) delay.

The controlling logic implemented by RRIS is hidden by complex legacy internal optimizations performed over the years by numerous electro-mechanical engineers. For this reason, it is hard to understand their high-level behavior and highlight the connections between stimuli and observable railway properties.

The experimental evaluation is based on real-world RRIS schematics that are intended to control level crossing and shunting routes. Using the tool NORMA [2], the considered RRIS have been modeled and automatically converted in timed transition systems in the syntax of Timed NUXMV [7]. The obtained models involve several real-valued variables (modeling voltages and currents in the circuits), changing accordingly to the configuration of the boolean variables (modeling the switches of the circuit). The discrete state changes when an external event updates the position of a switch, or as a consequence of the activation of an internal relay. Hence, these systems react to an external variation with a chain of internal transitions. The duration of the triggered run-to-completion process is important: urgent states are widely used to model the causality relation between the activation of an instantaneous relay and the action performed on the associated switch; timed relays may impose a low delay, so that the internal response is actually very fast and almost non observable.

Table 1. Result of the abstraction of `routesN` RRIS benchmarks with different stability definitions.

							reach. unaware			reach. aware		
test	X	I	O	P	Φ	σ	$\mathcal{A}_\sigma$states	$\mathcal{A}_\sigma$trans	time	$\mathcal{A}_\sigma$states	$\mathcal{A}_\sigma$trans	time
routes01	54	2	1	3	8	σ_1	8	40	01s	7	13	26s
						σ_2	4	6	20s	3	4	3m 09s
						$\sigma_{3,T=1}$	3	4	15s	2	2	1m 57s
						$\sigma_{3,T=7}$	3	4	15s	2	2	1m 57s
routes02	90	4	2	6	48	σ_1	48	768	22s	11	22	1m 02s
						σ_2	7	13	46s	6	11	6m 16s
						$\sigma_{3,T=1}$	5	9	38s	4	7	4m 32s
						$\sigma_{3,T=7}$	5	9	38s	4	7	4m 20s
routes04	166	8	3	12	4096	σ_1	–	–	TO	49	97	3h 7m 03s
						σ_2	29	83	1h 42m 29s	25	48	2h 56m 46s
						$\sigma_{3,T=1}$	17	52	1h 41m 17s	13	24	2h 42m 04s
						$\sigma_{3,T=7}$	17	52	1h 41m 10s	13	24	2h 41m 55s

Abstraction Modulo Stability of RRIS. The Timed nuXmv model checker was used to convert the models produced by NORMA in untimed transition systems in SMV. The algorithm presented in Sect. 4 has been implemented using the pySMT library [10] and the MathSAT5 SMT solver [8]. It requires in input a classification of the variables X, selecting the predicates P, the inputs I and the outputs O, which can be directly provided by railway domain experts. We choose as P the status of some relays or (boolean variables associated with) linear predicates on the electrical variables, representing, as an example, the status of a lamp.

Table 1 and 2 report the number of variables X, P, I, O for each benchmark. Column Φ reports the size of the resulting abstract domain, obtained by considering all the *consistent* combinations of P predicates (with respect to the invariant of the model).

We show the results of the Abstraction Modulo Stability considering the stability definitions described in Sect. 3.3, using the algorithm of Sect. 4 with bounds $L = 40$ and $U = 15$. All the experiment ran on a 2.4 GHz CPU, with time out (TO) set to 15 h, and memory limit set to 20 GB.

Columns "$\mathcal{A}_\sigma$states" and "$\mathcal{A}_\sigma$trans" hold the number of abstract states and transitions respectively, computed counting the configurations of the predicate variables in the abstract automaton $\mathcal{A}_\sigma$. As stated in Corollary 1, the corresponding abstract automata have progressively less states.

Stability abstractions were used by railway experts from the Italian Railway Network company (RFI) to understand two main families of legacy RRIS.

Routes. `routesN` is a RRIS regulating the activation/deactivation of `N` shunting routes concurring for the same resources. The implemented logic takes care of avoiding the simultaneous activation of conflicting routes. In such RRIS the inputs are the switches controlled by a human operator, attempting to enable/disable a route; the outputs are the status of some internal entities that

we want to monitor; the predicates are the status of lamps representing whether the routes have been registered.

In the `routes` benchmarks the delays used in the run-to-completion processes are very small, so that in the abstract automata obtained (Table 1) there is no difference between $\sigma_3^{T=1}$ and $\sigma_3^{T=7}$ (i.e., if a state has stayed in the same predicate for 1 time unit, then it can also stay there for 7). These abstract automata clearly highlight what are the consequences of the requests of a human operator with respect to the active/inactive status of the routes involved. As an example, the abstraction `routes02` (a circuit handling two routes) has only 4 stable states which show that the routes are incompatible and one of them has priority on the other, and disregards all the intermediate steps that the concrete system needs to progressively check the availability of the resources. These steps are visible with a less strict stability definition, like σ_1 or σ_2.

Table 1 also evaluates the effectiveness of the reachability refinement. When dropping the prefix starting from the initial states of the concrete system, the algorithm would consider several spurious behaviors. Especially in these benchmarks, the resulting abstract automaton would also show the unreachable states (e.g., the ones in which two routes are in conflict), therefore reducing the relevance for the reverse engineering purpose. Moreover, the *reach.unaware* computation may be harder to compute as it has to explore more transitions and more models in the guards and effects formulae.

Railway Switch. `r-switch` is a RRIS modeling a railway switch. It has several externally controlled switches and only 4 relevant observations, defining its abstract state. The schema can be instantiated as nominal (N) or faulty (F), by injecting faulty behaviors in some physical components. We consider three versions: `r-switch1` interacts with a free environment, showing a wide number of circuit configurations; `r-switch2` and `r-switch3`, instead, exploit some assumptions on the environment and expose less inputs, and, although using different internal implementations, are supposed to guarantee the same controlling logic.

Table 2 reports the features of the abstract automata obtained for these benchmarks. Here, during a run-to-completion process, some states dwell in the same predicate for a time $1 \leq t \leq 7$, so that are visible in $\sigma_3^{T=1}$ but skipped by $\sigma_3^{T=7}$ when reporting the corresponding abstract transition.

Again, the *reach.unaware* option reports more transitions. The difference is especially evident in the nominal versions, as the faulty concrete system already covers more behaviors. Even when the number of abstract transitions is the same, the *reach.aware* option reports more precise guards and effects, i.e., each annotating formula on I and O has less models.

By looking at the abstract automata, the user could recover what are the triggering reasons that make the system reach certain states (e.g., the ones that are shown in `r-switch1` and not in `r-switch2`). Namely, $\mathcal{A}_\sigma$ could highlight the enabling conditions for certain behaviors, which may apply far from the final observable consequence and were hard to inspect by hand. In this way, the user could also collect what assumptions are needed to avoid certain behaviors (e.g.,

Table 2. Result of the abstraction of `r-switch` RRIS benchmarks with different stability definitions.

							reach. unaware			reach. aware		
test	X	I	O	P	Φ	σ	$\mathcal{A}_\sigma$ states	$\mathcal{A}_\sigma$ trans	time	$\mathcal{A}_\sigma$ states	$\mathcal{A}_\sigma$ trans	time
r-switch1-N	128	18	3	4	12	σ_1	–	–	TO	12	78	8h 12m
						σ_2	12	112	4h 12m	12	94	2h 42m
						$\sigma_{3,T=1}$	12	112	7h 47m	12	86	2h 30m
						$\sigma_{3,T=7}$	12	112	7h 24m	12	66	2h 07m
r-switch1-F	128	18	3	4	12	σ_1	–	–	TO	–	–	TO
						σ_2	–	–	TO	12	112	7h 60m
						$\sigma_{3,T=1}$	12	112	13h 12m	12	112	5h 24m
						$\sigma_{3,T=7}$	12	112	14h 05m	12	112	4h 45m
r-switch2-N	127	17	3	4	12	σ_1	12	102	8h 18m	12	74	3h 29m
						σ_2	10	86	1h 56m	10	74	1h 18m
						$\sigma_{3,T=1}$	10	86	2h 12m	10	66	1h 10m
						$\sigma_{3,T=7}$	10	86	2h 31m	10	54	58m
r-switch2-F	127	17	3	4	12	σ_1	–	–	TO	12	90	10h 34m
						σ_2	10	86	4h 21m	10	86	2h 42m
						$\sigma_{3,T=1}$	10	86	4h 30m	10	86	2h 12m
						$\sigma_{3,T=7}$	10	86	4h 33m	10	86	1h 39m
r-switch3-N	121	16	3	4	12	σ_1	12	102	3h 28m	12	74	2h 08m
						σ_2	10	86	52m	10	74	52m
						$\sigma_{3,T=1}$	10	86	1h 34m	10	66	51m
						$\sigma_{3,T=7}$	10	86	1h 32m	10	54	44m
r-switch3-F	121	16	3	4	12	σ_1	–	–	TO	12	90	4h 21m
						σ_2	10	86	2h 46m	10	86	1h 38m
						$\sigma_{3,T=1}$	10	86	2h 01m	10	86	1h 22m
						$\sigma_{3,T=7}$	10	86	2h 16m	10	86	1h 24m

in understanding what changes were made from `r-switch1` to `r-switch2` or `r-switch3` schemas).

Finally, as expected, `r-switch2` and `r-switch3` have exactly the same abstract automata for every stability definition and nominal/faulty configuration, since they are two different implementations for the same observable properties.

P-Stable Abstractions. We also tried the implementation of [5], for approximated P-stable abstractions (σ_4), which uses BDDs and convex polyhedra. On small handcrafted models like the tank system used as running example we could run all the approaches and confirm the output automata described in Sect. 3. Nonetheless, in the analysis of RRIS the approach of [5] turned out to be impractical, and was unable to deal with any of the considered RRIS models, due to the high number of variables.

More importantly, in our case studies, σ_4 would likely result in abstractions that are too aggressive, hiding states that are practically interesting, such as the ones that emerge from the analysis of run-to-completion processes with non negligible duration.

6 Conclusions

In this paper we presented a framework for the reverse engineering of legacy systems. Starting from a symbolic timed transition system, the framework supports the construction of abstractions in the form of state machines with guards and effects over transitions. The abstractions are parameterized on the notion of stability. We propose an SMT-based algorithm for abstraction computation, and we instantiate it over several notions of stability.

The results have been evaluated within an industrial project with the Italian Railway Network, on reverse-engineering tasks of complex relay-based interlocking circuits. The experimental analysis demonstrated that the approach is practical, and able to construct abstractions for complex real-world circuits. Taking reachability into account allowed us to produce tighter, more informative representations of the system under inspection. Railway signaling engineers involved in the project considered the proposed approach adequate in terms of expressiveness and able to provide substantial support in understanding the legacy RRIS.

In the future, we will define an "anytime" version of algorithms, so that the abstraction can be incrementally visualized as the computation proceeds, and leverage parallelization to increase the efficiency. Given the positive feedback from the RFI experts, we plan to integrate the proposed abstraction techniques abstraction within a RRIS modeling front-end, and to apply them on a larger set of interlockings.

References

1. de Almeida Pereira, D.I.: Analysis and formal specification of relay-based railway interlocking systems. (Analyse et spécification formelle des systèmes d'enclenchement ferroviaire basés sur les relais). Ph.D. thesis, École centrale de Lille, Villeneuve-d'Ascq, France (2020)
2. Amendola, A., et al.: NORMA: a tool for the analysis of relay-based railway interlocking systems. In: Fisman, D., Rosu, G. (eds.) TACAS 2022. LNCS, vol. 13243, pp. 125–142. Springer, Cham (2022). https://doi.org/10.1007/978-3-030-99524-9_7
3. Amendola, A., et al.: A model-based approach to the design, verification and deployment of railway interlocking system. In: Margaria, T., Steffen, B. (eds.) ISoLA 2020. LNCS, vol. 12478, pp. 240–254. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-61467-6_16
4. Barrett, C.W., Sebastiani, R., Seshia, S.A., Tinelli, C.: Satisfiability modulo theories. In: Handbook of Satisfiability, Frontiers in Artificial Intelligence and Applications, vol. 185, pp. 825–885. IOS Press (2009)
5. Becchi, A., Cimatti, A., Zaffanella, E.: Synthesis of P-stable abstractions. In: de Boer, F., Cerone, A. (eds.) SEFM 2020. LNCS, vol. 12310, pp. 214–230. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-58768-0_12
6. ter Beek, M.H., Borälv, A., Fantechi, A., Ferrari, A., Gnesi, S., Löfving, C., Mazzanti, F.: Adopting formal methods in an industrial setting: the railways case. In: ter Beek, M.H., McIver, A., Oliveira, J.N. (eds.) FM 2019. LNCS, vol. 11800, pp. 762–772. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-30942-8_46

7. Cimatti, A., Griggio, A., Magnago, E., Roveri, M., Tonetta, S.: Extending nuXmv with timed transition systems and timed temporal properties. In: Dillig, I., Tasiran, S. (eds.) CAV 2019. LNCS, vol. 11561, pp. 376–386. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-25540-4_21
8. Cimatti, A., Griggio, A., Schaafsma, B.J., Sebastiani, R.: The MathSAT5 SMT solver. In: Piterman, N., Smolka, S.A. (eds.) TACAS 2013. LNCS, vol. 7795, pp. 93–107. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36742-7_7
9. Ernst, M.D., Perkins, J.H., Guo, P.J., McCamant, S., Pacheco, C., Tschantz, M.S., Xiao, C.: The daikon system for dynamic detection of likely invariants. Sci. Comput. Program. **69**(1–3), 35–45 (2007)
10. Gario, M.E.G., Micheli, A.: PySMT: a solver-agnostic library for fast prototyping of SMT-based algorithms. In: International Workshop on Satisfiability Modulo Theories (SMT) (2015)
11. Graf, S., Saidi, H.: Construction of abstract state graphs with PVS. In: Grumberg, O. (ed.) CAV 1997. LNCS, vol. 1254, pp. 72–83. Springer, Heidelberg (1997). https://doi.org/10.1007/3-540-63166-6_10
12. Haxthausen, A.E., Kjær, A.A., Le Bliguet, M.: Formal development of a tool for automated modelling and verification of relay interlocking systems. In: Butler, M., Schulte, W. (eds.) FM 2011. LNCS, vol. 6664, pp. 118–132. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-21437-0_11
13. Hong, L.V., Haxthausen, A.E., Peleska, J.: Formal modelling and verification of interlocking systems featuring sequential release. Sci. Comput. Program. **133**, 91–115 (2017)
14. Lahiri, S.K., Bryant, R.E., Cook, B.: A symbolic approach to predicate abstraction. In: Hunt, W.A., Somenzi, F. (eds.) CAV 2003. LNCS, vol. 2725, pp. 141–153. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45069-6_15
15. Lahiri, S.K., Nieuwenhuis, R., Oliveras, A.: SMT techniques for fast predicate abstraction. In: Ball, T., Jones, R.B. (eds.) CAV 2006. LNCS, vol. 4144, pp. 424–437. Springer, Heidelberg (2006). https://doi.org/10.1007/11817963_39
16. Laroussinie, F., Schnoebelen, P.: Specification in CTL+past for verification in CTL. Inf. Comput. **156**(1–2), 236–263 (2000). https://doi.org/10.1006/inco.1999.2817
17. Limbrée, C.: Formal verification of railway interlocking systems. Ph.D. thesis, Catholic University of Louvain, Louvain-la-Neuve, Belgium (2019)
18. Limbrée, C., Cappart, Q., Pecheur, C., Tonetta, S.: Verification of railway interlocking - compositional approach with OCRA. In: Lecomte, T., Pinger, R., Romanovsky, A. (eds.) RSSRail 2016. LNCS, vol. 9707, pp. 134–149. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-33951-1_10
19. Milner, R.: Calculi for synchrony and asynchrony. Theor. Comput. Sci. **25**, 267–310 (1983). https://doi.org/10.1016/0304-3975(83)90114-7