HAL
open science

Blaming in component-based real-time systems

Gregor Gössler, Lacramioara Astefanoaei

To cite this version:

Gregor Gössler, Lacramioara Astefanoaei. Blaming in component-based real-time systems. Proceedings of the 14th International Conference on Embedded Software - EMSOFT'14, Oct 2014, Delhi, India. 10.1145/2656045.2656048 . hal-01078214

HAL Id: hal-01078214

https://inria.hal.science/hal-01078214

Submitted on 28 Oct 2014

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Blaming in Component-Based Real-Time Systems*

Gregor Gössler and Lăcrămioara Aştefănoaei

October 28, 2014

Abstract

In component-based safety-critical real-time systems it is crucial to determine which component(s) caused the violation of a required system-level safety property, be it to issue a precise alert, or to determine liability of component providers. In this paper we present an approach for blaming in real-time systems whose component specifications are given as timed automata. The analysis is based on a single execution trace violating a safety property $\mathcal{P}$. We formalize blaming using counterfactual reasoning ("what would have been the outcome if component C had behaved correctly?") to distinguish component failures that actually contributed to the outcome from failures that had no impact on the violation of $\mathcal{P}$. We then show how to effectively implement blaming by reducing it to a model-checking problem for timed automata, and demonstrate the feasibility of our approach on the models of a pacemaker and of a chemical reactor.

1 Introduction

In component-based real-time systems such as autonomous cars or interconnected medical devices, determining the cause of an observed failure is a pivotal question. Establishing a causal link between the failure of one or more components and the violation of a system-level safety property is instrumental to blame components, be it to issue a precise alert or establish the contractual liability of component providers. However, we currently lack a general framework for blaming in component-based real-time systems.

In this paper, we address this problem by proposing a diagnostic mechanism that relies on counterfactual reasoning ("what would have been the outcome if component C had behaved correctly?") to distinguish component failures that actually contributed to the outcome from failures that had no impact on the system-level failure.

Analyzing causality is a hard task for a couple of reasons: an event e may not directly cause a failure f, but through a causal chain; f may be caused by any of two observed events e_1 and e_2, or by the conjunction of both. The quest to formalize causality goes back at least to Hume [12]:

> [...] we may define a cause to be an object, followed by another, and where all the objects similar to the first are followed by objects similar to the second. Or in other words where, if the first object had not been, the second never had existed.

Following Hume, analyzing causality of an event e is based on *counterfactuals*, that is, fictious scenarios that resemble the observed scenario, but where e does not happen. We illustrate this idea, and the subtleties in formalizing it, on a simple example. Consider a set of three processes sharing a resource, where mutual exclusion is ensured by time division, as shown in Figure 1. P_1 is required to access the resource within one time unit, P_2 after 3 to 4 time units, and P_3 after at least 7 time units; each process may keep the resource for at most 1 time unit.[1] Let us consider the failure scenario shown in Figure 2(a) where both P_2 and P_3 access the resource early. Intuitively,

*published at EMSOFT'14

[1] Although this is a made-up example, mutual exclusion actually relies on time slicing in some safety-critical systems [20].

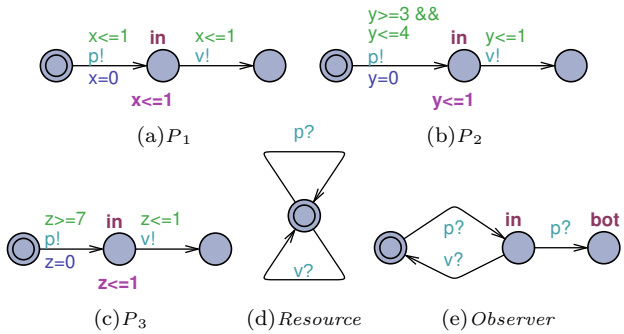


Figure 1: Mutual exclusion example.

the early access by P_2 entails a violation of mutual exclusion, whereas the early access by P_3 is not in conflict with any other process. Using the counterfactual approach, we may try to analyze whether the failure of P_2 is a cause for the violation of mutual exclusion, by replacing the trace of P_2 by a correct trace, as shown in Figure 2(b). However, the obtained set of traces is not a valid counterfactual scenario in the sense that the traces are inconsistent: they are not observations of any system execution. This is because the behavior of the resource remains the same, and consequently, there is a mismatch between the unchanged delay of 1 preceding the second access of the resource and the corrected delay of 2 in P_2.

P_1	1	p!	1	v!		4		1	
P_2	1		1	p!	v!	4		1	
P_3	1		1			4	p!	1	v!
resource	1	p?	1	p?	v? v?	4	p?	1	v?

(a) Scenario violating mutual exclusion.

P_1	1	p!	1	v!		4		1	
P_2	1		**2**	**p!**	**v!**	**3**		**1**	
P_3	1		1			4	p!	1	v!
resource	1	p?	1	p?	v? v?	4	p?	1	v?

(b) The trace of P_2 is corrected.

Figure 2: A failure scenario and a simple counterfactual scenario. Each line is a component trace; vertical alignments of delays and interactions are added to make the scenario more readable. Substituted suffixes are written in bold.

In this work we introduce the notion of unaffected prefixes, which allows us to take the impact of component failures on other components into account. Based on unaffected prefixes we leverage the definition of necessary causality introduced in [9] in order to formalize blaming in real-time systems. We consider a set of black-box components, each equipped with a specification in the form of a timed automaton. On a given execution trace, causality of the components having violated their specifications is analyzed with respect to the violation of a system-level safety property. As in the setting of timed automata we usually have to cope with an infinite number of counterfactuals, we also propose a symbolic encoding.

The paper is organized as follows. Section 2 reviews related work. Section 3 introduces definitions for handling timed automata and traces. Section 4 formalizes our definition and analysis of logical causality, and proposes a symbolic encoding. In Section 5 we apply the approach to two case studies. Sections 6 concludes.

2 Related Work

Generally speaking, causality analysis frameworks may take several pieces of information into account: the actual, observed behavior (e.g. execution traces, or inputs and outputs), a causal model of the system, and the expected behavior (in the form of specifications). Most existing approaches consider subsets of these entities. There is no single generally agreed-on definition of causality; rather, several definitions have been proposed to meet different needs.

In Halpern and Pearl's influential structural equations model [11], counterfactual reasoning is used to define *actual causality* on a model of causal dependencies between propositional variables. However, as their approach requires the structural equations to be given, it cannot be applied to single execution traces of black-box components.

In [9] a language-based framework for causality analysis on execution traces has been introduced. Causality of the failures of components in an index set $\mathcal{I}$ for the violation of a system-level property $\mathcal{P}$ is analyzed in two steps. First, *temporal causality* in the sense of Lamport [17] is analyzed by computing the *cone of influence* spanned by the failures of components in $\mathcal{I}$. Then, *logical causality* is determined by substituting the cone with the possible *correct* suffixes of the impacted components. If all obtained counterfactual scenarios satisfy $\mathcal{P}$, the violation of $\mathcal{P}$ is blamed on the failures of components in $\mathcal{I}$. Real-time systems can be cast in this framework by modeling time progress as explicit synchronizations between all components. However, the proposed analysis of temporal causality would then compute a cone of influence that grossly over-estimates the actual impact of the component failures, thus leading to false positives.

A causality analysis for real-time systems similar to our framework is presented in [24]. The main difference with respect to the approach we present here is that [24] does not systematically take dependencies between component traces into account. Therefore fault propagation may be under-estimated, resulting in the possibility of inconsistent counterfactual scenarios and hence, false positives by vacuity. To our knowledge, [24] is the only approach to cope with blaming in real-time systems.

In [6] causality of a vector of boolean signals with respect to the violation of an LTL property is analyzed using a variant of [11]. The approach determines, for each signal, the instants where the signal value is, in some context, a cause for the violation of the property. This technique is limited to synchronous Boolean signals and LTL specifications. [15] extends the definition of actual causality of [11] to totally ordered sequences of events, and uses this definition to construct from a set of traces a fault tree. Using a probabilistic model, the fault tree is annotated with probabilities. The accuracy of the diagnostic depends on the number of traces used to construct the model.

With the goal of localizing faults in source code, [5] uses model-checking to determine program faults as pieces of code that belong to an error trace but not to any correct trace. Given a counter-example in model-checking, [10] uses a distance metric to determine an explanation of the property violation as the difference between the error trace and a closest correct trace. [23] explains a failed assertion c in a program by computing the weakest precondition of c along a counter-example trace and checking at which point the weakest precondition becomes false. More recently, a game-theoretic approach to locate and fix faults has been proposed in [14]. Delta debugging [26] is a technique for automatically isolating a cause of some error in transformational systems where the transformation is a black box on which different inputs can be replayed.

Fault diagnosis (see e.g. [22]) and fault localization aim at determining what unobservable error events occurred, and where in a system. In contrast, our goal is to establish the existence of a causal chain between observed component failures — that is, errors in the sense of [4] — and a system failure.

Temporal causality in the sense of Lamport [17] determines the set of events in the execution of a distributed system that are known to have been posterior to a given event e, but they are not necessarily *caused* by e in a logical sense. Finally, there is a large body of work on defining causally consistent operational semantics. Basically, the goal is to provide a constructive way of extending an execution prefix to a possibly infinite run satisfying a set of constraints, see [19] for an example. Being constructive, these approaches do not rely on counterfactual reasoning.

3 Preliminaries

We recall the definitions of the timed automata and networks of timed automata (NTA) we use. These are a subclass of the timed automata implemented in Uppaal [7].

Definition 1 (Timed automaton) *A timed automaton is a tuple* $(L, l^0, X, inv, ch, \Sigma, E)$ *where:*

- L *is a finite set of locations, and* $l^0 \in L$ *is the initial location;*
- X *is a finite set of clocks;*
- $inv : L \to \mathcal{C}$ *assigns an invariant to each location;*
- ch *is a finite set of channels, partitioned into the set* ch^s *of synchronous channels and the set* ch^b *of broadcast channels;*
- $\Sigma \subseteq \{c! \mid c \in ch\} \cup \{c? \mid c \in ch\} \cup \{\tau\}$ *is a finite action alphabet;*
- $E \subseteq L \times (\Sigma \times \mathcal{C} \times 2^X) \times L$ *is a set of edges labeled with an action, a guard, and a set of clocks to be reset. Edges labeled by* $c?$ *with* c *being a broadcast channel have the guard true.*

$\mathcal{C}$ *is the set of clock constraints. A clock constraint is defined by the grammar* $C ::= true \mid false \mid x\#c \mid x - y\#c \mid C \wedge C$ *with* $x, y \in X$, $\# \in \{<, \leq, =, \geq, >\}$ *and* $c \in \mathbb{Z}$. *Invariants are restricted to constraints of the type* $x \leq c$.

Definition 2 (Semantics of a timed automaton) *The semantics of a timed automaton* $T = (L, l^0, X, inv, ch, \Sigma, E)$ *is given by the labeled transition system (LTS)* $sem(T) = (Q, \Sigma, \to, q^0)$ *where:*

- $Q = L \times V$ *denotes the states of* T;
- $q^0 = (l^0, \mathbf{0})$ *denotes the initial state of* T *(we require that* $inv(l^0)(\mathbf{0})$*);*
- $\to \subseteq Q \times (\Sigma \cup \mathbb{R}_{\geq 0}) \times Q$ *denotes the transitions according to the rules:*
  - $(l, v) \xrightarrow{d} (l, v + d)$ *if* $\forall d' \in [0, d]$, $inv(l)(v + d')$ *(time progress);*
  - $(l, v) \xrightarrow{a} (l', v')$ *if there exists a transition* $\big(l, (a, g, r), l'\big)$ *in* E *such that* $g(v) \wedge inv(l')(v')$ *with* $v' = v[r]$ *(action transition).*

V *is the set of all clock valuation functions* $v : X \to \mathbb{R}_{\geq 0}$, *and* $\mathbf{0}$ *is the valuation where all clocks are* 0. *For a constraint* C, $C(v)$ *denotes the evaluation of* C *in* v. *The notation* $v + d$ *represents a new valuation function* v' *defined as* $v'(x) = v(x) + d$ *while* $v[r]$ *represents* v' *defined as* $v'(x) = 0$ *if* $x \in r$, *and* $v'(x) = v(x)$ *otherwise.*

Definition 3 (Semantics of a NTA) *The semantics of a network* $T = \|_{i=1}^{n} T_i$ *of* n *timed automata* T_i *with disjoint sets* X_i *of clocks,* $sem(T_i) = \big(Q_i, \Sigma_i, \to_i, (l_i^0, \mathbf{0})\big)$, *and channels* $ch^s \cup ch^b$, *is given by the LTS* $(Q, \Sigma, \to, q^0)$ *where:*

- $Q = \big\{(\vec{l}, v) \mid \forall i = 1, ..., n : (l_i, v_i) \in Q_i\big\}$;
- $\Sigma = \big\{(a_1, ..., a_n) \in \Sigma_1^\epsilon \times ... \times \Sigma_n^\epsilon \mid \big(\exists c \in ch^s\ \exists i, j : a_i = c! \wedge a_j = c? \wedge \forall k \notin \{i, j\} : a_i = \epsilon\big) \vee \big(\exists c \in ch^b\ \exists i : a_i = c! \wedge \forall j \neq i : a_j \in \{c?, \epsilon\}\big)\big\} \cup \{\tau\}$ *with* $\Sigma_i^\epsilon = \Sigma_i \cup \{\epsilon\}$;
- $q^0 = \big((l_1^0, ..., l_n^0), \mathbf{0}\big)$;
- $\to \subseteq Q \times (\Sigma \cup \mathbb{R}_{\geq 0}) \times Q$ *given by:*
  - $(\vec{l}, v) \xrightarrow{d} (\vec{l}, v + d)$ *if* $\forall i : (l_i, v_i) \xrightarrow{d}_i (l_i, v_i + d)$ *(time progress);*

– $(\vec{l}, v) \xrightarrow{\tau} (\vec{l'}, v')$ *if there exists* i *such that* $(l_i, v_i) \xrightarrow{\tau}_i (l'_i, v'_i)$ *and* $\forall j \neq i : (l'_j, v'_j) = (l_j, v_j)$ *(internal transition);*

– $(\vec{l}, v) \xrightarrow{\alpha} (\vec{l'}, v')$ *with* $\alpha = (a_1, ..., a_n)$ *if there exist:*

1. *either a channel* $c \in ch^s$ *and indices* i, j *such that* $a_i = c!$, $a_j = c?$, $(l_i, v_i) \xrightarrow{c!}_i (l'_i, v'_i)$, $(l_j, v_j) \xrightarrow{c?}_j (l'_j, v'_j)$, *and* $\forall k \notin \{i, j\} : (l'_k, v'_k) = (l_k, v_k)$ *(synchronization);*
2. *or a channel* $c \in ch^b$ *and an index* i *such that* $a_i = c!$, $(l_i, v_i) \xrightarrow{c!}_i (l'_i, v'_i)$, $\forall j \in J : a_j = c?$ *and* $(l_j, v_j) \xrightarrow{c?}_j (l'_j, v'_j)$, *and* $\forall k \notin \{i\} \cup J : a_k = \epsilon$ *and* $(l'_k, v'_k) = (l_k, v_k)$, *where* $J = \{j \mid \exists l''_j, v''_j : (l_j, v_j) \xrightarrow{c?}_j (l''_j, v''_j)\}$ *(broadcast)*

where we write $\vec{l}$ *for the vector* $(l_1, ..., l_n)$, *and* v_i *stands for the restriction of* v *to the domain* X_i.

The alphabet consists of a set of tuples over the component alphabets, indicating for communications the action of each component, and an internal action τ. The intuition behind Definition 3 is as follows. Time can progress uniformly if all timed automata agree on time progress. Internal τ transitions interleave, resulting in an internal transition of T. Two components may interact with complementary actions over a synchronous channel whenever both components are ready to do so. Whenever a send action on a broadcast channel c is enabled in some component, the broadcast may take place and all components having an enabled $c?$ transition take it simultaneously. The symbol ϵ stands for the absence of a component in a communication over a (synchronous or broadcast) channel.

A *timed trace* over alphabet Σ is a word in $\big((\Sigma \setminus \{\tau\}) \cup \mathbb{N}\big)^*$ [2]. Given a timed trace $tr = \alpha_1\alpha_2 \cdots \alpha_k$ and an index $i \in \mathbb{N}$, let $tr[1..i] = \alpha_1\alpha_2 \cdots \alpha_i$ and let $tr[i] = \alpha_i$. A *run* of a timed automaton or network of timed automata with semantics $(Q, \Sigma, \rightarrow, q^0)$ is a finite sequence of transitions $r = (l_0, v_0) \xrightarrow{\alpha_1} (l_1, v_1) \xrightarrow{\alpha_2} ... \xrightarrow{\alpha_m} (l_m, v_m)$ such that $(l_0, v_0) = q^0$. The timed trace accepted by r is obtained by removing from $\alpha_1\alpha_2 \cdots \alpha_m$ all τ-symbols. Let $\mathcal{L}(T)$ be the set of all timed traces accepted by some run of T.

Definition 4 (Projection π) *Consider a network of timed automata* T_i *with alphabets* Σ_i, *and let* $T = \|_{i=1}^n T_i$ *with alphabet* Σ. *The projection of an action* $\alpha = (a_1, ..., a_n) \in \Sigma$ *on* Σ_k, *written* $\pi_k(\alpha)$, *is the component action* a_k. *The projection of a time delay* d *is* d. *The projection of an internal action* τ *is* ϵ. *The projection of a timed trace* $tr = \alpha_1\alpha_2 \cdots$ *of* T *on* Σ_k, *written* $\pi_k(tr)$, *is the timed trace of* T_k *obtained by removing all* ϵ*-symbols in* $\pi_k(\alpha_1)\,\pi_k(\alpha_2)\,\cdots$.

Hence, internal actions are unobservable in the projection.

The *canonical form* of a timed trace tr over Σ, written $\mathsf{CF}(tr)$, is obtained by replacing in tr any maximal sub-sequence $d_1d_2 \cdots d_m$ of time delays by a single time delay $d_1 + d_2 + \cdots + d_m$, and removing all zero delays.

Definition 5 (Prefix $\preceq$, $\sqcap$, $\sqcup$) *A timed trace* tr' *is a prefix of* tr, *written* $tr' \preceq tr$, *if there exists an index* ℓ *such that either* $\mathsf{CF}(tr') = \mathsf{CF}(tr[1..\ell])$, *or there exist* $d_1, d_2 \in \mathbb{N}$ *such that* $tr[\ell] = d_1 + d_2$ *and* $\mathsf{CF}(tr') = \mathsf{CF}(tr[1..\ell - 1].d_1)$.

For a set P *of prefixes of a given trace let* $\bigsqcap P$ *and* $\bigsqcup P$ *denote the minimal and the maximal element of* $\{\mathsf{CF}(w) \mid w \in P\}$ *with respect to* $\preceq$, *respectively. We write* $tr_1 \sqcap tr_2$ *for* $\bigsqcap\{tr_1, tr_2\}$.

For instance, $1.1.p!.1 \preceq 2.p!.2.v!$; $\bigsqcap\{1.1,\ 2.p!.2\} = 2$; $\bigsqcup\{1.1,\ 2.p!.2\} = 2.p!.2$.

Next we define a timed automaton that accepts exactly a given timed trace by reaching its final state.

Definition 6 ($\mathcal{T}$) *Let* tr *be a timed trace over* Σ. *We suppose w.l.o.g. that* $tr = d_0a_1...d_{n-1}a_nd_n$ *for* $n \geq 0$ *with* $d_i \in \mathbb{N}$ *and* $a_i \in \Sigma$, *that is, discrete and time steps alternate. Let* $nbcr(i)$ *denote the number of receptions on a broadcast channel in* $a_1a_2...a_i$ *(we put* $nbcr(0) = 0$*), and let*

[2] Integer delays are a realistic assumption since logged events will be time-stamped by some discrete clock.

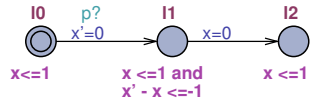


Figure 3: Timed automaton recognizing the trace $1.p?.1$ by reaching the final state $(\mathsf{l2}, \mathsf{x} = 1)$.

$m = n + nbcr(n)$. We construct a timed automaton $\mathcal{T}(tr) = (L, l_0, \{x, x'\}, inv, ch, \Sigma, E)$ accepting exactly tr as follows. Let $L = \{l_0, ..., l_m\}$ with $inv(l_0) = (x \leq d_0)$, and $E = \{(l_{j-1}, (b_j, g_j, r_j), l_j) \mid 1 \leq j \leq m\}$. The invariants and transition labels are defined as follows. For $1 \leq i \leq n$,

- *if a_i is a reception on a broadcast channel, we use two subsequent transitions to model its timing:*
  1. *$b_{k_i} = a_i$, $g_{k_i} = true$, $r_{k_i} = \{x'\}$, $inv(l_{k_i}) = (x \leq d_{i-1} \wedge x' - x \leq -d_{i-1})$;*
  2. *$b_{k_i+1} = \tau$, $g_{k_i+1} = true$, $r_{k_i+1} = \{x\}$, $inv(l_{k_i+1}) = (x \leq d_i)$.*
- *Otherwise, $b_{k_i} = a_i$, $g_{k_i} = (x = d_{i-1})$, $r_{k_i} = \{x\}$, and $inv(l_{k_i}) = (x \leq d_i)$*

where $k_i = i + nbcr(i-1)$.

Let $complete^{\mathcal{T}}(tr) = (l_m, x \geq d_n)$ identify the states where tr has been accepted.

Since Uppaal does not support guarded receive transitions on broadcast channels (see Definition 1), we encode such a reception by a sequence of two transitions. In the first transition, an auxiliary clock x' is reset to measure the time elapsed since the reception. The invariant of the location between both transitions ensures that only timely broadcasts are accepted. For instance, Figure 3 shows $\mathcal{T}(tr)$ for $tr = 1.p?.1$, where p is a broadcast channel. The encoding of binary synchronization in a single transition is straight-forward.

Whether $tr \in \mathcal{L}(T)$, for a timed trace tr and a timed automaton $T = (L, l^0, X, inv, ch, \Sigma, E)$, can be decided as follows. Let tr' be the trace obtained by complementing all synchronizations in tr (that is, substituting $c?$ with $c!$ and vice versa, for each channel $c \in ch$), and let B be the network of timed automata obtained by transforming in $\mathcal{T}(tr') \| T$ all broadcast channels into synchronous channels. Then, $tr \in \mathcal{L}(T)$ iff $B \models \exists\Diamond complete^{\mathcal{T}}(tr')$, that is, if the final state of $\mathcal{T}(tr')$ is reachable in the composition.

4 Causality Analysis

In this section we define causality of component traces for the violation of a system-level property. A system is specified by a *system signature*:

Definition 7 (System signature) *A* system signature *is a tuple $(\boldsymbol{\mathcal{S}}, \Sigma, \mathcal{P})$ where*

- *$\boldsymbol{\mathcal{S}} = (\mathcal{S}_1, ..., \mathcal{S}_n)$ is a tuple of component specifications given as timed automata;*
- *Σ is the alphabet of $\|_{i=1}^{n} \mathcal{S}_i$;*
- *$\mathcal{P}$ is a safety property such that $\mathcal{L}(\|_{i=1}^{n} \mathcal{S}_i) \subseteq \mathcal{P}$.*

The last condition means that property $\mathcal{P}$ may be violated only if at least one of the components violates its specification.

For our analysis we suppose the following inputs to be given:

- A system signature $(\boldsymbol{\mathcal{S}}, \Sigma, \mathcal{P})$ with component specifications $\boldsymbol{\mathcal{S}} = (\mathcal{S}_i, ..., \mathcal{S}_n)$ over alphabets $\Sigma_1, ..., \Sigma_n$.
- A vector of observed component traces $\vec{tr} = (tr_1, ..., tr_n)$ over Σ_i that are projections of some system-level trace. In the case where the behavior of two or more components is logged into a common trace, the trace of each component is obtained by projection.

- A set $\mathcal{I} \subseteq \{1, ..., n\}$ of component indices, indicating the set of components to be jointly analyzed for causality. Being able to reason about *group causality* is useful, for instance, in helping to determine liability of component providers that are responsible for more than one component.

The mere fact that the violation of $\mathcal{P}$ is preceded by the violation of a component specification $\mathcal{S}_k$, is not sufficient to establish that the latter is a *logical cause* of the former. Therefore, we will now turn to the formal definition and analysis of logical causality.

4.1 A Definition of Logical Causality

Intuitively, in order to verify whether the violations of $\mathcal{S}_i$ by tr_i, $i \in \mathcal{I}$, are a cause for the violation of $\mathcal{P}$ in $\vec{tr}$, we have to identify and remove the effect of these component failures on $\vec{tr}$ and replace it with behaviors that are consistent with a correct execution of the components in $\mathcal{I}$. The obtained behaviors are called *counterfactuals*. The violations of $\mathcal{S}_i$ in tr_i, $i \in \mathcal{I}$, are a cause for the violation of $\mathcal{P}$ if all counterfactual traces are in $\mathcal{P}$, that is, if without those violations, $\mathcal{P}$ would have been satisfied. In order to determine and eliminate the impact of component failures on the traces of the remaining components, we first compute the set of prefixes that are *unaffected* by the failures.

Definition 8 (Critical prefix cp**)** *Given a timed trace* $tr = \alpha_1\alpha_2 \cdots$ *over* Σ *and a timed automaton* $\mathcal{S}$ *over* Σ*, let* $cp(tr, \mathcal{S}) = \bigsqcup\{tr' \mid tr' \preceq tr \wedge tr' \in \mathcal{L}(\mathcal{S})\}$ *be the* critical prefix *of* tr *with respect to* $\mathcal{S}$*.*

The critical prefix $cp(tr, \mathcal{S})$ is the longest prefix of tr that is accepted by $\mathcal{S}$.

We can now define the unaffected prefixes of a log $\vec{tr}$ with respect to a set $\mathcal{I}$ of components as the longest prefixes of $\vec{tr}$ that could also have been observed if the components in $\mathcal{I}$ had satisfied their specifications, and the counterfactuals.

Definition 9 (Unaffected prefixes, counterfact.) *Given a system signature* $(\boldsymbol{\mathcal{S}}, \Sigma, \mathcal{P})$*, a vector* $\vec{tr}$ *of component traces, and an index set* $\mathcal{I} \subseteq \{1, ..., n\}$*, we define the* unaffected prefixes *of* $\vec{tr}$ *as follows. Let*

$$tr_i^0 = \begin{cases} cp(tr_i, \mathcal{S}_i) & \text{if } i \in \mathcal{I} \\ tr_i & \text{otherwise} \end{cases}$$

and $\forall i = 1, ..., n$*:*

$$tr_i^* = \bigsqcup \big\{ tr_i^0 \sqcap \pi_i(tr') \mid tr' \in (\Sigma \cup \mathbb{N})^* \wedge \\ \forall j \ \exists w \in extend_{\mathcal{L}(\mathcal{S}_j)}(tr_j^0) : \pi_j(tr') \preceq w \big\}$$

where

$$extend_L(tr) = \begin{cases} \text{p-}refine_L(tr) & \text{if } tr \in L \\ \{tr\} & \text{otherwise} \end{cases}$$

and $\text{p-}refine_L(tr) = \{tr' \in L \mid tr \preceq tr'\}$*.*

Let $\mathsf{UP}_{\boldsymbol{\mathcal{S}}}(\vec{tr}, \mathcal{I}) = (tr_1^*, ..., tr_n^*)$ *be the vector of prefixes of* $\vec{tr}$ *that are unaffected by the failures of components in* $\mathcal{I}$*. Let*

$$\mathsf{C}_{\boldsymbol{\mathcal{S}}}(\vec{tr}, \mathcal{I}) = \{tr' \in (\Sigma \cup \mathbb{N})^* \mid \forall i : \pi_i(tr') \in extend_{\mathcal{L}(\mathcal{S}_i)}(tr_i^*)\}$$

be the set of counterfactuals *to* $\vec{tr}$*.*

Intuitively, the vector of unaffected prefixes tr_i^* is computed by first removing the incorrect suffixes from tr_i, $i \in \mathcal{I}$, and then removing suffixes that are infeasible in any system-level trace where each component i starts with tr_i^0. More precisely, tr_i^* is the longest prefix of tr_i^0 that is the projection of some system-level trace tr' whose projections are prefixes of the trace extensions. The set of extensions $extend_L(tr)$ of a trace tr in $\mathcal{L}(\mathcal{S}_j)$ is the sub-language of the component specification that starts with tr, or $\{tr\}$ if tr is not accepted by any run of $\mathcal{S}_j$.

P_1	1	p!	1	v!	1		3		1	
P_2	1		1		1	p!	3	v!	1	
P_3	1		1		1		3	p!	1	v!
resource	1	p?	1	v?	1	p?	3	p? v?	1	v?

Table 1: Second scenario violating mutual exclusion: both $\{P_2\}$ and $\{P_3\}$ are causes.

The unaffected prefixes $(tr_1^*, ..., tr_n^*)$ are thus constructed as maximal prefixes that could also have been observed if all components in $\mathcal{I}$ had behaved correctly, whereas the suffixes $s_1, ..., s_n$ — such that $tr_i = tr_i^*.s_i$ — are impacted by the failures of components in $\mathcal{I}$. The counterfactuals reflect the impact of "undoing" the failures of traces in $\mathcal{I}$. The set of counterfactuals is the set of system-level traces whose projections on the components extend the unaffected prefixes with correct behaviors unless the unaffected prefix is itself incorrect. In particular, $\mathsf{C}_{\boldsymbol{\mathcal{S}}}(\vec{tr}, \mathcal{I})$ contains all system-level traces with projections $tr_1^*, ..., tr_n^*$.

Example 1 *The unaffected prefixes of the traces $(tr_1, ..., tr_4)$ in Figure 2(a) with respect to the failures of component P_2 are $\mathsf{UP}_{\boldsymbol{\mathcal{S}}}(\vec{tr}, \{P_2\}) = (tr_1, tr_2', tr_3, tr_4')$ with $tr_2' = 2$ and $tr_4' = 1.p?.1$. On the other hand, for component P_3 we get $\mathsf{UP}_{\boldsymbol{\mathcal{S}}}(\vec{tr}, \{P_3\}) = (tr_1, tr_2, tr_3', tr_4'')$ with $tr_3' = 6$ and $tr_4'' = 1.p?.1.p?.v?.v?.4$.*

Definition 10 (Causality) *Consider a system signature $(\boldsymbol{\mathcal{S}}, \Sigma, \mathcal{P})$ and traces $\vec{tr} = (tr_1, ..., tr_n)$ such that $\exists w \in (\Sigma \cup \mathbb{N})^* \setminus \mathcal{P}\ \forall i : \pi_i(w) = tr_i$, and a subset $\mathcal{I} \subseteq \{1, ..., n\}$ of component indices. The incorrect suffixes of the traces indexed by $\mathcal{I}$ are a cause of the violation of the property $\mathcal{P}$ if*

$$\mathsf{C}_{\boldsymbol{\mathcal{S}}}(\vec{tr}, \mathcal{I}) \subseteq \mathcal{P}$$

That is, the set of incorrect suffixes in $\mathcal{I}$ is a cause for the violation of $\mathcal{P}$ by $\vec{tr}$ if for any counterfactual trace w — sharing with $\vec{tr}$ the prefixes that are unaffected by the failures of components in $\mathcal{I}$, and after which no more failures occur —, $\mathcal{P}$ would have been satisfied. In other words, the failures of components in $\mathcal{I}$ are necessary for the violation of $\mathcal{P}$. Note that $\mathcal{P}$ may be satisfied by the unaffected prefixes although its violation may be unavoidable in the sequel (and thus, occur in the counterfactuals). By abuse of language we say that the set of components indexed by $\mathcal{I}$ is a cause when their incorrect suffixes are a cause.

Example 2 (Two causes) *Applied to our running example and the unaffected prefixes computed in Example 1, Definition 10 determines P_2 to be a cause, whereas P_3 is not. Now consider the modified scenario (tr_1, tr_2, tr_3, tr_4) shown in Table 1. P_2 releases the resource late, and P_3 accesses it early. The unaffected prefixes $\mathsf{UP}_{\boldsymbol{\mathcal{S}}}(\vec{tr}, \{P_2\})$ are $(tr_1, tr_2', tr_3, tr_4')$ with $tr_2' = 3.p!.1$ and $tr_4' = 1.p?.1.v?.1.p?.1$. On the other hand, for component P_3 we get $\mathsf{UP}_{\boldsymbol{\mathcal{S}}}(\vec{tr}, \{P_3\}) = (tr_1, tr_2, tr_3', tr_4'')$ with $tr_3' = 6$ and $tr_4'' = 1.p?.1.v?.1.p?.3$. According to Definition 10, both $\{P_2\}$ and $\{P_3\}$ are causes: a correct behavior of any of them would have resulted in the satisfaction of $\mathcal{P}$.*

Our blame assignment is sound and complete:

Theorem 1 (Soundness) *Each cause contains an incorrect trace.*

Proof 1 *Consider a set $\mathcal{I} \subseteq \{i \mid tr_i \in \mathcal{L}(\mathcal{S}_i)\}$. We show that the set of traces indexed by $\mathcal{I}$ is not a cause for the violation of $\mathcal{P}$ by $\vec{tr} = (tr_1, ..., tr_n)$. If all components in $\mathcal{I}$ satisfy their specifications, then $(tr_1^*, ..., tr_n^*) = \mathsf{UP}_{\boldsymbol{\mathcal{S}}}(\vec{tr}, \mathcal{I}) = \vec{tr}$. By the hypothesis of Definition 10 there exists $w \in (\Sigma \cup \mathbb{N})^*$ such that $w \notin \mathcal{P} \wedge \forall i : \pi_i(w) = tr_i = tr_i^*$. Thus $\mathcal{I}$ is not a cause according to Definition 10.*

Theorem 2 (Completeness) *Each violation of $\mathcal{P}$ has a cause.*

Proof 2 *Let $\mathcal{I} = \{i \mid tr_i \notin \mathcal{L}(\mathcal{S}_i)\}$, $\vec{tr} = (tr_1, ..., tr_n)$, and $(tr_1^*, ..., tr_n^*) = \mathsf{UP}_{\boldsymbol{\mathcal{S}}}(\vec{tr}, \mathcal{I})$. By construction of $\vec{tr}^0$ — and thus of $\vec{tr}^*$ —, all traces in $\vec{tr}^*$ are prefixes of the traces in $\vec{tr}$ and satisfy the component specifications. Since $(\boldsymbol{\mathcal{S}}, \Sigma, \mathcal{P})$ is a system signature and hence $\mathcal{L}(\|_{i=1}^n \mathcal{S}_i) \subseteq \mathcal{P}$, and $\mathcal{P}$ is prefix-closed, all counterfactual traces satisfying the condition of Definition 10 are in $\mathcal{P}$. Hence, $\mathcal{I}$ is a cause for the violation of $\mathcal{P}$ by $\vec{tr}$.*

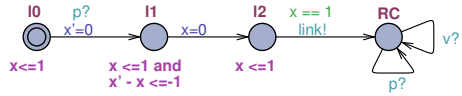


Figure 4: *p-refine*$_{resource}(1.p?.1)$.

4.2 Symbolic Analysis

In general there may be an infinite number of counterfactual traces. The next definitions will be useful to symbolically construct counterfactual scenarios and analyze causality.

4.2.1 Computing the unaffected prefixes.

First we define *p-refine* in terms of timed automata. For the sake of simplicity we assume that the component specifications $\mathcal{S}_i$ are such that for all observable actions $a \in \Sigma_i \setminus \{\tau\}$ and edges $\big(l_1, (a, g_1, r_1), l_1'\big)$ and $\big(l_2, (a, g_2, r_2), l_2'\big)$, $r_1 = r_2$. That is, all clock resets are uniquely defined by the action. We thus denote them with $r(a)$.

Definition 11 (*p-refine*$_{TA}$) *Consider a timed trace $tr = d_1 a_1 \cdots d_n \in (\Sigma \cup \mathbb{N}_{\geq 0})^*$ with $\mathcal{T}(tr) = (L_1, l_1^0, X, inv_1, ch, \Sigma, E_1)$ and a timed automaton $B = (L_2, l_2^0, X, inv_2, ch, \Sigma, E_2)$ such that $tr \in \mathcal{L}(B)$. Let $l_1^f \in L_1$ be the final location of $\mathcal{T}(tr)$ and $l_2^* \in L_2$ be the location reached in B after accepting tr. Let $\textit{p-refine}_B(tr) = (L, l_1^0, X, inv, ch \cup \{\mathsf{link}\}, \Sigma, E)$ where $L = L_1 \cup L_2$,*

$$\forall l \in L : inv(l) = \begin{cases} inv_1(l) & \text{if } l \in L_1 \\ inv_2(l) & \text{if } l \in L_2 \end{cases},$$

link is a fresh broadcast channel, $E = E_1' \cup E_2 \cup \{(l_1^f, (\mathsf{link}!, x = d_n, \emptyset), l_2^*)\}$, and $E_1' = \{(l, (a, g, r \cup r(a)), l') \mid (l, (a, g, r), l') \in E_1\}$.

Let $\textit{complete}^{\textit{p-refine}}(B, tr)$ be the predicate characterizing the locations in L_2.

The timed automaton $\textit{p-refine}_B(tr)$ is a refinement of B that accepts exactly those traces accepted by B that start with tr. For instance, Figure 4 shows $\textit{p-refine}_B(tr)$ for $tr = 1.p?.1$ and B being the shared resource in Figure 1(d). Labeling the link transition with a dummy broadcast rather than with an internal τ-transition helps us to ensure compositionality of the symbolic representation, as will be explained in Section 4.2.2.

Using $\textit{p-refine}_B$ we define a symbolic variant of $\textit{extend}_L$:

$$\textit{extend}_B(tr) = \begin{cases} \textit{p-refine}_B(tr) & \text{if } tr \in \mathcal{L}(B) \\ \mathcal{T}(tr) & \text{otherwise} \end{cases}$$

where B is a timed automaton. Next we define the *feasible prefix* of a component trace in a system where all components behave according to given timed automata.

Definition 12 (Feasible prefix FP) *For a trace tr of component k and a vector $\vec{B}$ of timed automata whose composition has alphabet Σ, let*

$$\mathsf{FP}_k(tr, \vec{B}) = \bigsqcup \{tr \sqcap \pi_k(tr') \mid tr' \in (\Sigma \cup \mathbb{N})^* \wedge \forall i : \pi_i(tr') \in \mathcal{L}(B_i)\}$$

$\mathsf{FP}_k(tr_k, \vec{B})$ is the longest prefix of tr_k that is the projection of some trace accepted by $\|_{i=1}^n B_i$. We can symbolically compute $\mathsf{FP}_k(tr_k, \vec{B})$ by determining the longest prefix tr_k' of tr_k for which $\mathcal{T}(tr_k') \parallel \|_{i \neq k} B_i \models \exists\Diamond \textit{complete}^{\mathcal{T}}(tr_k')$.

We are now able to symbolically compute the unaffected prefixes as follows. Let

$$(tr_i^0, B_i^0) = \begin{cases} \big(cp_i, \textit{p-refine}_{\mathcal{S}_i}(cp_i)\big) & \text{if } i \in \mathcal{I} \vee tr_i \in \mathcal{L}(\mathcal{S}_i) \\ \big(tr_i, \mathcal{T}(tr_i)\big) & \text{otherwise} \end{cases}$$

where $cp_i = cp(tr_i, \mathcal{S}_i)$, and $\forall i = 1, ..., n$:

$$tr_i^* = \mathsf{FP}_i(tr_i^0, \vec{B}^0)$$
$$B_i^* = extend_{\mathcal{S}_i}(tr_i^*)$$

The symbolic computation of tr_i^* satisfies Definition 9, and we can show that $\vec{tr}^* = \mathsf{UP}_{\boldsymbol{\mathcal{S}}}(\vec{tr}, \mathcal{I})$.

4.2.2 Computing the counterfactuals.

According to Definition 9, the set of counterfactual traces $\mathsf{C}_{\mathcal{S}}(\vec{tr}, \mathcal{I})$ is the set of traces whose projections extend the unaffected prefixes $\vec{tr}^*$. In order to symbolically represent the set of counterfactual traces, we have to ensure that the composition $B = \|_{i=1}^n B_i^*$ of the symbolic models constructed above produces (up to prefix-closure) exactly the counterfactual traces in $\mathsf{C}_{\mathcal{S}}(\vec{tr}, \mathcal{I})$. In particular, we have to avoid that some timed automaton $B_i = \textit{p-refine}_{\mathcal{S}_i}(tr_i^*)$ that has finished accepting the prefix tr_i^* is still in the final location of $\mathcal{T}(tr_i^*)$ while another component j sends on a broadcast channel to which i should be listening, according to $\mathcal{S}_i$, after accepting tr_i^*. To this end we use the possibility of Uppaal to give priority to link (see Definition 11) over all other channels.

4.2.3 Verifying causality.

In order to effectively analyze causality we assume the safety property $\mathcal{P}$ to be given as an observer $\mathcal{O}_{\mathcal{P}}$, that is, a timed automaton whose only discrete actions are receptions on broadcast channels, with a dedicated location $\perp$ indicating that $\mathcal{P}$ has been violated. Let

$$\textit{complete} = \bigwedge_i \begin{cases} \textit{complete}^{\textit{p-refine}}(B_i^*, tr_i^*) & \text{if } tr_i^* \in \mathcal{L}(B_i^*) \\ \textit{complete}^{\mathcal{T}}(tr_i^*) & \text{otherwise} \end{cases}$$

characterize the set of states of B where all prefixes $\vec{tr}^*$ have been accepted. In order to decide causality according to Definition 10 we have to tell whether every complete trace of B satisfies the property $\mathcal{P}$, which is equivalent to model-checking whether

$$B \| \mathcal{O}_{\mathcal{P}} \not\models \exists\Diamond(\textit{complete} \wedge \perp) \tag{1}$$

that is, whether no complete execution — whose projections are extensions of the unaffected prefixes, and which is thus a counterfactual trace — violates $\mathcal{P}$.

The reason why we use an observer for $\mathcal{P}$, rather than taking a property in the TCTL fragment that Uppaal model-checks [7], is that $\mathcal{P}$ has to be a property over traces — and thus, refer to occurrences of component actions —, whereas TCTL refers to propositional variables (i.e., states). Observers can, for instance, be constructed from properties in the metric interval temporal logic MITL [2, 18], or written by hand as in our case studies.

5 Case Studies

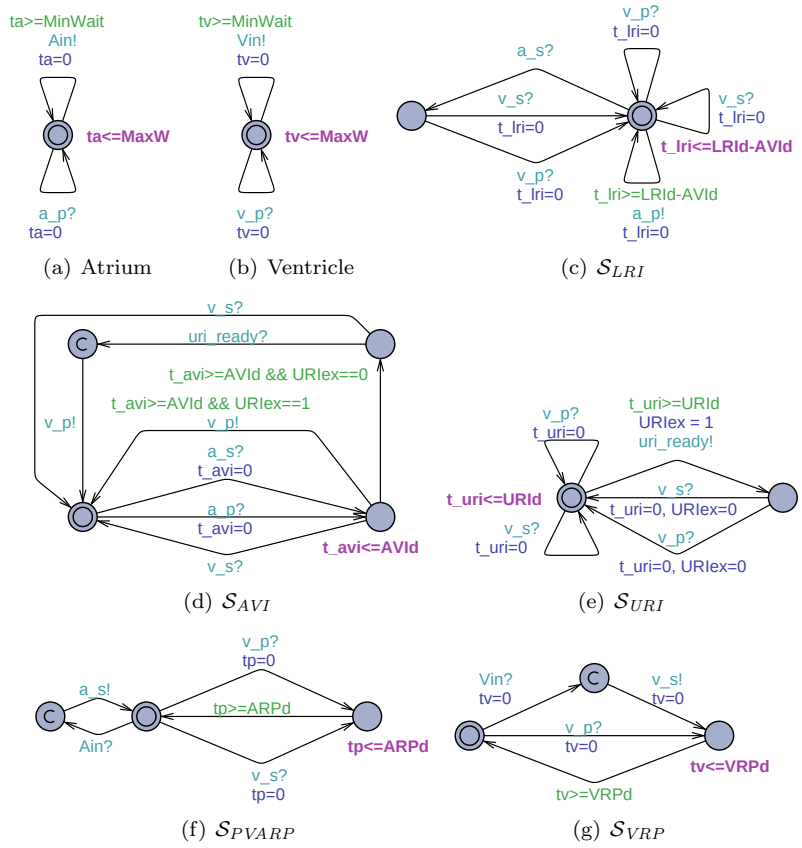
We have implemented the symbolic analysis presented in the previous section in the prototype tool LoCA *(Logical Causality Analyzer)* written in Scala, using Uppaal as a backend. In this section we describe two of our experiments with LoCA. For each experiment, the execution times for constructing the counterfactual model and verifying which component to blame are shown in Table 2. These timings were obtained on an Intel Core i7 with 16 GB RAM running Linux.

5.1 The Pacemaker Model

This model is that of a dual chamber implantable pacemaker [13, 21]. It consists of seven components whose specifications are shown in Figure 5:

- *atrium*, modeling random behavior of the atrium chamber of a heart. The model assumes an interval $[MinW, MaxW] = [200ms, 1200ms]$ between two consecutive events $Ain!$.
- *ventricle*, modeling the ventricle chamber, with the same timing assumption. *Atrium* and *ventricle* are used as a model of the environment of the pacemaker.
- LRI, for *lower rate interval*, ensures a maximal interval $LRI_d = 1000ms$ between a sensed or paced ventricular event and the next atrial event. Past this delay, an atrial pacing event $a_p!$ is delivered.
- AVI, a component ensuring an upper bound of $AVI_d = 150ms$ between an atrial event (sensed or paced) and a ventricular event. After an elapsed time of AVI_d, a ventricular pacing event $v_p!$ is delivered.
- URI, for *upper rate interval*, ensures a minimal delay of $URI_d = 400ms$ between a ventricular event and the next ventricular pacing event by delaying the delivery of $v_p!$ by AVI. A broadcast on *uri_ready* signals that the minimal delay has elapsed.
- $PVARP$ (*post ventricular atrial refractory period*) and VRP (*ventricular refractory period*) filter out noise and early events.

All channels are broadcast channels. The locations labeled with C are *committed*, that is, they must be left immediately [7].



(a) Atrium (b) Ventricle (c) $\mathcal{S}_{LRI}$

(d) $\mathcal{S}_{AVI}$ (e) $\mathcal{S}_{URI}$

(f) $\mathcal{S}_{PVARP}$ (g) $\mathcal{S}_{VRP}$

Figure 5: The Pacemaker model.

Let us consider the failure scenario shown in Fig. 7(a). Two of the traces violate their specifications: LRI due to an early atrial pacing event $a_p!$, and URI due to a missing *uri_ready*! event

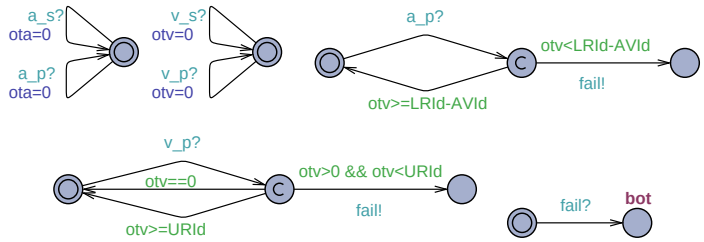


Figure 6: The observer.

after $400ms$. The safety property $\mathcal{P}$ we are interested in is part of the pacemaker specification cited in [21]:

1. atrial pacing cannot occur during the interval $t_v \in [0, LRI_d - AVI_d)$;
2. ventricular pacing cannot occur during the interval $t_v \in (0, URI_d)$

where t_v measures the time elapsed since the last ventricular event. The observer for $\mathcal{P}$ is shown in Figure 6. The execution of Fig. 7(a) violates the first condition of $\mathcal{P}$ since at the moment of $a_p!$, the time elapsed since the last ventricular event is $200ms < LRI_d - AVI_d = 850ms$. Our goal is to find out whether to blame the violation of $\mathcal{P}$ on LRI, or whether the violation of $\mathcal{P}$ by LRI was indirectly caused by the preceding failure of URI.

atrium	500	*Ain!*		100			200	*a_p?*	150		
ventricle	500			100	*Vin!*		200		150		*v_p?*
AVI	500		*a_s?*	100			200		150	*uri_ready?*	*v_p!*
LRI	500		*a_s?*	100		*v_s?*	200	*a_p!*	150		
URI	500			100		*v_s?*	200		150	*uri_ready!*	
PVARP	500	*Ain?*	*a_s!*	100		*v_s?*	200		150		*v_p?*
VRP	500			100	*Vin?*	*v_s!*	200		150		*v_p?*

(a) Traces $\vec{tr}$. Violations of component specifications are underlined.

atrium	500	*Ain!*		100			200	*a_p?*	150
ventricle	500			100	*Vin!*		200		150
AVI	500		*a_s?*	100			200		150
LRI	500		*a_s?*	100		*v_s?*	200	*a_p!*	150
URI	**400**								
PVARP	500	*Ain?*	*a_s!*	100		*v_s?*	200		150
VRP	500			100	*Vin?*	*v_s!*	200		150

(b) Unaffected prefixes $\mathsf{UP}_{\boldsymbol{\mathcal{S}}}(\vec{tr}, \{URI\})$.

Figure 7: The failure scenario.

Let us first analyze causality of the incorrect suffix of URI. Figure 7(b) shows the unaffected prefixes $\mathsf{UP}_{\boldsymbol{\mathcal{S}}}(\vec{tr}, \{URI\})$ for the violation of $\mathcal{S}_{URI}$. In particular, the early $a_p!$ event turns out to be independent of URI's failure. By constructing the set of counterfactual scenarios, Definition 10 establishes that the incorrect suffix of URI is a not a cause for the violation of $\mathcal{P}$. In contrast, the incorrect suffix of LRI is a cause for the violation of $\mathcal{P}$: without $a_p!$ being sent early $\mathcal{P}$ would have been satisfied. Thus, only LRI is to blame for the violation of $\mathcal{P}$.

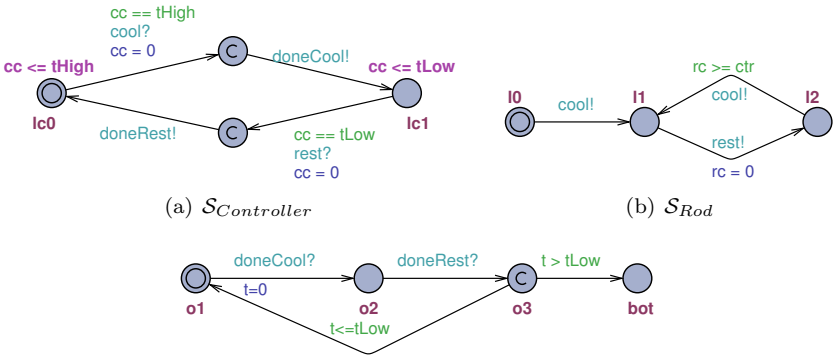
5.2 The Temperature Controller Model

This model of a chemical reactor is an adaptation from [1]. The system being modeled consists of a controller moving n identical rods in order to maintain the temperature between the bounds $tLow$

$\mathcal{I}$	construction of $\mathsf{C}_{\mathcal{S}}(\vec{tr}, \mathcal{I})$	model-checking Eq. (1)	cause?
$\{LRI\}$	7.2 s	6.0 s	yes
$\{URI\}$	4.7 s	8 ms	no
$\{controller\}$	4.0 s	108 ms	yes
$\{rod_1\}$	1.3 s	7 ms	no
$\{rod_2\}$	2.8 s	7 ms	yes

Table 2: Execution times of the queries on the failure scenario in the pacemaker model (top) and the chemical reactor model (bottom).

and *tHigh*: when the temperature in the reactor reaches *tHigh* (resp. *tLow*), a rod must be used to cool (resp. heat) the reactor. Once moved, a rod can be moved again only after $ctr = tHigh \times n$ units of time. The controller is designed such that the time interval between a cooling and heating is of at most *tLow* time units. Any violations of this safety property leads to shutdown. Our analysis concentrates precisely on this property. To implement an observer which monitors it, the controller is modeled such that it sends a broadcast *doneCool*! (resp. *doneRest*!) to signal when a cooling action (resp. resting) just took place. The interactions between the controller and the rods are implemented by means of synchronous communication on the channels *cool* and *rest*. All timed automata are depicted in Figure 8.



(a) $\mathcal{S}_{Controller}$ (b) $\mathcal{S}_{Rod}$

(c) Observer

Figure 8: The temperature controller model and its observer.

Compared to the pacemaker model, this model is more regular: there are only two different specifications (one for the controller and one for each of the rods), and their only possible interactions are basically synchronizations on *cool* and *rest*. The difficulty here comes more from the possibly great number of rods the controller must interact with. Figure 9(a) shows an excerpt from a log recording the execution of a controller in parallel with 7 rods. The prefix corresponds to observations during the first 14 time units for the values of 6, resp. 3 for *tHigh* and resp. *tLow*. Two of the traces of the rods, as well as that of the controller have local violations: the rods send their second *cool* too early and the controller breaks the timing requirements in its specification. Especially the second delay of 4 time units before a *rest* action entails the invalidation of the global safety property. Among the three candidates, only the controller and the second rod are causes for the global violation: rod_1 cannot be blamed because in any counterfactual scenario constructed from the unaffected prefixes $\mathsf{UP}_{\mathcal{S}}(\vec{tr}, \{rod_1\})$ in Figure 9(b), the global violation due to the late *rest* between rod_2 and the controller occurs nevertheless. On the contrary, had either one of rod_2 or the controller behaved correctly, the safety property would have been satisfied.

rod_1	11.cool!.3.rest!.cool!.rest!
rod_2	7.cool!.rest!.cool!.4.rest!.3
rod_3	14.cool!.rest!
rod_4	14.cool!.rest!
rod_5	14.cool!.3.rest!.7
rod_6	14.cool!.rest!
rod_7	14.cool!.rest!
controller	4.cool?.doneCool!.3.rest?.doneRest!.cool?.doneCool!.rest?.doneRest!.cool?.doneCool!.4.rest?.doneRest!. cool?.doneCool!.3.rest?.doneRest!.cool?.doneCool!.rest?.doneRest!.cool?.doneCool!.rest?.doneRest!. cool?.doneCool!.rest?.doneRest!.cool?.doneCool!.rest?.doneRest!.cool?.doneCool!.rest?.doneRest!

(a) Traces $\vec{tr}$. The first violations of component specifications are underlined. The double underlined delay in the trace of the controller indicates the position where the global violation happens.

rod_1	11.cool!.3.rest!
rod_2	7.cool!.rest!.cool!.4.rest!.3
rod_3	14.cool!.rest!
rod_4	14.cool!.rest!
rod_5	14.cool!.3.rest!.7
rod_6	14.cool!.rest!
rod_7	14.cool!.rest!
controller	4.cool?.doneCool!.3.rest?.doneRest!.cool?.doneCool!.rest?.doneRest!.cool?.doneCool!.4.rest?.doneRest!. cool?.doneCool!.3.rest?.doneRest!.cool?.doneCool!.rest?.doneRest!.cool?.doneCool!.rest?.doneRest!. cool?.doneCool!.rest?.doneRest!

(b) Unaffected prefixes $\mathsf{UP}_{\boldsymbol{S}}(\vec{tr}, \{rod_1\})$.

Figure 9: The failure scenario for the chemical reactor model.

6 Conclusion

We have given a formalization of blaming in real-time systems composed of black-box components, and provided a symbolic approach for effective blame assignment. The approach has been implemented and successfully applied to the models of a dual chamber implantable pacemaker and that of a chemical reactor, where the described approach could be used to read out logs periodically and identify the components causing safety hazards.

Our work opens several directions for future research. First, we would like to investigate possible applications, in particular, to blame assignment in actual safety-critical systems. We believe that ensuring accountability [16] by construction, that is, the possibility of automatic blame assignment after a system failure, should become a requirement in the design of any safety-critical system. Another application of our approach is in system validation by model-checking or testing where the analysis is applied to (possibly symbolic) counterexample traces.

A first extension we will investigate is the distinction between observable and non-observable events, only the former of which are logged. In particular, whenever component failures are not observable, we have to combine fault diagnosis [22] and causality analysis.

In closed-loop systems, an alternative behavior of the control part will impact the physical process. Therefore the counterfactual behavior of the former should be propagated through a model of the latter. For the pacemaker example, more realistic models of the heart that could be used for this purpose are [25, 8], at the price of constructing and verifying counterfactuals in the more expressive framework of (stochastic) hybrid automata. We also intend to study how our observation-based approach can be combined with a model-based approach such as [11] to allow reasoning about white-box systems for which causal models are (partially) available.

Our experimental results are promising. In particular, they suggest that the cost for causality analysis of smaller problems is very reasonable. In order to tackle larger models — where the cost should be dominated by model-checking Equation (1) —, it would be of interest to investigate the use of compositional verification for timed automata [3].

Acknowledgment The first author thanks Insup Lee and Oleg Sokolsky for their feed-back on an early version of this work and for suggesting the pacemaker as a case study.

References

[1] R. Alur, C. Courcoubetis, N. Halbwachs, T. Henzinger, P. Ho, X. Nicollin, A. Olivero, J. Sifakis, , and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138:3–34, 1995.

[2] R. Alur, T. Feder, and T. A. Henzinger. The benefits of relaxing punctuality. *J. ACM*, 43(1):116–146, 1996.

[3] L. Astefanoaei, S. B. Rayana, S. Bensalem, M. Bozga, and J. Combaz. Compositional invariant generation for timed systems. In E. Ábrahám and K. Havelund, editors, *TACAS*, volume 8413 of *LNCS*, pages 263–278. Springer, 2014.

[4] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Trans. on Dependable and Secure Computing*, 1(1):11–33, Jan. 2004.

[5] T. Ball, M. Naik, and S. Rajamani. From symptom to cause: localizing errors in counterexample traces. In *POPL*, pages 97–105, 2003.

[6] I. Beer, S. Ben-David, H. Chockler, A. Orni, and R. Trefler. Explaining counterexamples using causality. *Formal Methods in System Design*, 40(1):20–40, 2012.

[7] G. Behrmann, A. David, and K. Larsen. A tutorial on uppaal. In M. Bernardo and F. Corradini, editors, *SFM*, volume 3185 of *LNCS*, pages 200–236. Springer, 2004.

[8] T. Chen, M. Diciolla, M. Z. Kwiatkowska, and A. Mereacre. A simulink hybrid heart model for quantitative verification of cardiac pacemakers. In C. Belta and F. Ivancic, editors, *HSCC*, pages 131–136. ACM, 2013.

[9] G. Gössler and D. Le Métayer. A general trace-based framework of logical causality. In J. L. Fiadeiro, Z. Liu, and J. Xue, editors, *FACS - 10th International Symposium on Formal Aspects of Component Software - 2013*, volume 8348 of *LNCS*, pages 157–173. Springer, 2013.

[10] A. Groce, S. Chaki, D. Kroening, and O. Strichman. Error explanation with distance metrics. *STTT*, 8(3):229–247, 2006.

[11] J. Halpern and J. Pearl. Causes and explanations: A structural-model approach. part I: Causes. *British Journal for the Philosophy of Science*, 56(4):843–887, 2005.

[12] D. Hume. *An Enquiry Concerning Human Understanding*. 1748.

[13] Z. Jiang, M. Pajic, S. Moarref, R. Alur, and R. Mangharam. Modeling and verification of a dual chamber implantable pacemaker. In C. Flanagan and B. König, editors, *TACAS*, volume 7214 of *LNCS*, pages 188–203. Springer, 2012.

[14] B. Jobstmann, S. Staber, A. Griesmayer, and R. Bloem. Finding and fixing faults. *J. Comput. Syst. Sci.*, 78(2):441–460, 2012.

[15] M. Kuntz, F. Leitner-Fischer, and S. Leue. From probabilistic counterexamples via causality to fault trees. In F. Flammini, S. Bologna, and V. Vittorini, editors, *SAFECOMP*, volume 6894 of *LNCS*, pages 71–84. Springer, 2011.

[16] R. Küsters, T. Truderung, and A. Vogt. Accountability: definition and relationship to verifiability. In *ACM Conference on Computer and Communications Security*, pages 526–535, 2010.

[17] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *CACM*, 21(7):558–565, 1978.

[18] O. Maler, D. Nickovic, and A. Pnueli. From mitl to timed automata. In E. Asarin and P. Bouyer, editors, *FORMATS*, volume 4202 of *LNCS*, pages 274–289. Springer, 2006.

[19] M. Moy and K. Altisen. Arrival curves for real-time calculus: The causality problem and its solutions. In J. Esparza and R. Majumdar, editors, *TACAS*, volume 6015 of *LNCS*, pages 358–372. Springer, 2010.

[20] I. Ober, S. Graf, and D. Lesens. Modeling and validation of a software architecture for the Ariane-5 launcher. In R. Gorrieri and H. Wehrheim, editors, *FMOODS*, volume 4037 of *LNCS*, pages 48–62. Springer, 2006.

[21] M. Pajic, Z. Jiang, I. Lee, O. Sokolsky, and R. Mangharam. From verification to implementation: A model translation tool and a pacemaker case study. In M. D. Natale, editor, *IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 173–184. IEEE, 2012.

[22] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. Teneketzis. Diagnosability of discrete-event systems. *IEEE Transactions on Automatic Control*, 40(9):1555–1575, 1995.

[23] C. Wang, Z. Yang, F. Ivancic, and A. Gupta. Whodunit? causal analysis for counterexamples. In S. Graf and W. Zhang, editors, *ATVA*, volume 4218 of *LNCS*, pages 82–95. Springer, 2006.

[24] S. Wang, A. Ayoub, B. Kim, G. Gössler, O. Sokolsky, and I. Lee. A causality analysis framework for component-based real-time systems. In A. Legay and S. Bensalem, editors, *Proc. Runtime Verification 2013*, volume 8174 of *LNCS*, pages 285–303. Springer, 2013.

[25] P. Ye, E. Entcheva, R. Grosu, and S. Smolka. Efficient modeling of excitable cells using hybrid automata. In *CMSB'05*, 2005.

[26] A. Zeller. *Why Programs Fail.* Elsevier, 2009.