Acta Informatica (2022) 59:183–209
https://doi.org/10.1007/s00236-021-00403-z

ORIGINAL ARTICLE

An automated system repair framework with signal temporal logic

Mert Ergurtuna[1] · **Beyazit Yalcinkaya**[1] · **Ebru Aydin Gol**[1]

Received: 29 September 2020 / Accepted: 18 May 2021 / Published online: 17 June 2021

Abstract
We present an automated system repair framework for cyber-physical systems. The proposed framework consists of three main steps: (1) system simulation and fault detection to generate a labeled dataset, (2) identification of the repairable temporal properties leading to the faulty behavior and (3) repairing the system to avoid the occurrence of the cause identified in the second step. We express the cause as a past time signal temporal logic (ptSTL) formula and present an efficient monotonicity-based method to synthesize a ptSTL formula from a labeled dataset. Then, in the third step, we modify the faulty system by removing all behaviors that satisfy the ptSTL formula representing the cause of the fault. We apply the framework to two rich modeling formalisms: discrete-time dynamical systems and timed automata. For both of them, we define repairable formulae, the corresponding repair procedures, and illustrate them over case studies.

1 Introduction

From autonomous vehicles, to smart agriculture systems, medical devices and robotics, cyber-physical systems (CPSs) are in use in a very wide range of areas. A common approach in development of CPSs is using model-based development techniques and prototyping to verify the correctness of the design via simulation, and if possible, formal techniques before the development of the actual system. Automated testing including model-based testing has proven to be cost-effective for fault detection [29] and supported in various CPS modeling tools such as Simulink [30]. Furthermore, robustness guided falsification techniques for CPSs can significantly reduce the fault localization time [8,18,39]. When a faulty behavior

This work has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie Grant Agreement No. 798482.

✉ Ebru Aydin Gol
ebrugol@metu.edu.tr

Mert Ergurtuna
mert.ergurtuna@metu.edu.tr

Beyazit Yalcinkaya
beyazit.yalcinkaya@metu.edu.tr

[1] Department of Computer Engineering, Middle East Technical University, Ankara, Turkey

is detected, first the model is analyzed to identify the root cause, and then the system is improved ("repaired") to eliminate the cause. Both cause identification and system repair steps are challenging, and they are, in general, performed manually. As the system gets more complex, identifying the causes and modifying the system manually become even more challenging and time-consuming for the system designers. In this work, we propose a repair framework to automate these steps with formal guarantees.

The proposed framework consists of three mains steps: (1) generation of a labeled dataset via simulation and testing, (2) synthesis of a "repairable" past time signal temporal logic (ptSTL) formula that describes the labeled events and (3) performing the associated repair process for the identified formula. Repairable formula and the associated repair process are the core concepts of the framework. A formula is called repairable if there exists a $Repairer$ that can modify the system to guarantee that the formula will not be satisfied, i.e., the root cause will not occur, along the modified system's traces. Furthermore, we require that the repair process does not introduce any new behavior. We formalize these requirements over the system traces and parametric ptSTL formulae.

We illustrate the framework over two rich modeling formalisms, namely, discrete-time dynamical systems and timed automata [4,5]. For dynamical systems, we define a repairable formula as a conjunction of a control formula and system formula and define $Repairer$ as a controller-refinement procedure. On the other hand, for timed automata, we identify formula templates that address absence of timing constraints and define associated $Repairer$ procedures that introduce new clocks and timing constraints. Hence, we present a fully automated framework to find the causes of faulty behaviors and repair the system to avoid these causes for discrete-time dynamical systems and timed automata.

As a part of the proposed framework, we present an efficient method to synthesize a ptSTL formula from a given set of parametric formulae and a labeled dataset of system traces such that the evaluation of the resulting formula matches the labels. Past time fragment of temporal logics includes only past operators, thus when evaluating a ptSTL formula ϕ at time t, only events occurred prior to time t are considered, which is essential in cause-effect relation. Considering that the faulty behavior can have multiple causes, our synthesis method iteratively generates a formula as a disjunction of optimized formulae from the given set. At each iteration, a candidate set of parametric formulae are optimized, and the best formula is added to the final formula via disjunction until it is not possible to further improve the final formula.

1.1 Related work

Following the success in automated fault detection methods [8,18,29,39], automated repair problem is getting more attention. In [3], machine learning and verification techniques are combined to repair system specifications. Similarly, in [13], a machine learning-based approach is developed to automatically repair system models written in B formal specification language. Automatic repair has also been studied for software, see [21] for a recent survey. The automated software repair methods include fixing an existing code, e.g., a genetic programming-based approach in [37], introducing new expressions as in [32]. In this work, we present a repair framework using ptSTL for CPSs.

Signal temporal logic (STL) is a rich specification language used for describing temporal properties of real-time signals [16]. Due to its expressivity and efficient algorithms for checking continuous signals against STL formulae, it is used in different areas including monitoring [10,11], fault detection via falsification [18] and formal control [33]. Synthe-

sizing an STL formula from a dataset is studied in the literature in different forms such as finding a formula that is satisfied by all system traces to identify the system requirements [24,25], finding a formula that differentiates the given sets of good and bad signals [31], signal clustering [36] or finding a formula that would identify the bad events as they occur [10,17]. The proposed formula synthesis method as a part of the repair framework extends [17] by performing parameter synthesis in each iteration and eliminating formulae that cannot be part of the result for efficiency. In a recent work [12], STL formula synthesis is used as a part of a fault explanation framework for CPSs, where the authors synthesized a formula describing the "good" behaviors and checked it against the faulty behaviors to find a fault explanation. Their results also support the use of STL formulae for cause explanation.

In this work, we find explanatory formulae in a special form, which we call repairable, and define automated procedures to fix the system to avoid the satisfaction of the formula. Thus, while we limit the synthesis to specific formula types, we present auto-repair procedures, which was not possible in [12]. Identification of the temporal pattern leading to the violation of a temporal logic formula over a signal is studied in [20]. Different from our work and [12], a particular signal (execution) is analyzed in [20]. As in [12], fault localization for Simulink models is studied in [28], where a test case selection process is defined to identify the Simulink blocks causing the observed fault. A fault localization method accompanied with a repair approach for Simulink models is presented in [35], where the fault is localized by applying a matrix decomposition approach over the internal signals leading to the fault and the system is repaired by tuning the parameters of the blocks identified in the localization step. In [40], fault localization for autonomous mobile systems composed of a set of sub-systems is considered. The authors of [40] define a library of parametric STL formulae for specifying system requirements and develop a localization technique that identifies the likeliest subsystem that has a fault leading to the violation of the requirement. In this work, we propose an STL-based fault localization and repair framework and apply it to discrete-time dynamical systems and timed automata.

The repair for dynamical systems is defined as a controller-refinement procedure guaranteeing satisfaction (or violation) of a ptSTL formula. Control strategies from STL specifications are synthesized by solving mixed integer linear programs [33]. A main difference of the developed refinement method is that it only restricts possible control choices to guarantee that new behavior is not formed via the repair mechanism. The repair framework for discrete-time dynamical systems improves the controller-synthesis approach presented in [34] by generalizing the constraints defined over the template formula and control set definitions.

Repair of timed automata (TA) models has not been studied until recently [6,26,27]. In [26,27], a repair suggestion is generated by analyzing a faulty timed trace of a TA. The analysis is based on running an SMT solver on a linear arithmetic encoding of the trace. The repair suggestions include changing the clock bounds in constraints and modifying clock resets. As such modifications can significantly change the TA behavior, they perform an additional step to check the equivalence of the resulting and original models. In our case, instead of modifying existing constraints or clocks, we introduce new clocks and constraints over the new clocks, which allows us to prove that the traces of the repaired TA $\mathscr{A}^R$ is a subset of the traces of the original TA $\mathscr{A}$. Thus, if $\mathscr{A}$ satisfies a universal property such as a metric temporal logic formula, then $\mathscr{A}^R$ is also guaranteed to satisfy the same property. In [6], the authors assume that the causes of the faults are the incorrect timing constraints. In order to repair the system, they parametrize these constraints and generate parameters by analyzing the traces via an oracle that can decide whether a trace belongs a system (i.e., good) or not. While the procedure from [6] only modifies existing constraints, we propose to identify fault

causes as a ptSTL formula and repair the system in an automated way by introducing new clocks and constraints.

The contribution of this work is fourfold. First, it defines an automated system repair framework based on ptSTL. The framework includes dataset generation, synthesis of repairable causes as ptSTL formula and system repair steps. The second contribution is the efficient formula synthesis method employed in the second step. Finally, the application of the framework to the discrete-time dynamical systems and timed automata together with the repairable formula sets and automated repair procedures constitute the third and the fourth contributions, respectively. We implemented the developed methods in a proof-of-concept tool [1].

The paper is organized as follows. Preliminary information on signal temporal logic is given in Sect. 2. In Sect. 3, the proposed system repair framework is presented in detail. Repairable cause identification method is explained in Sect. 4. In Sects. 5 and 6, application of the proposed method for dynamical systems and timed automata is presented, respectively. Finally, the paper is concluded with closing remarks and future research directions in Sect. 7.

2 Signal temporal logic

2.1 Signals

An n-dimensional continuous signal $\mathbf{x}$ is defined as a mapping from time domain $\mathbb{R}_{\geq 0}$ to the real numbers $\mathbb{R}^n$. For any given time t from the time domain of a signal, the value of the signal is denoted by $x(t)$, and the prefix of the signal from 0 to t is denoted by $\mathbf{x}_{\leq t}$, i.e., $\mathbf{x}_{\leq t} = \{x(t') \mid t' \in [0, t]\}$. The projection of the state on the ith dimension at time t is denoted by $x^i(t)$.

2.2 Past time signal temporal logic

A past time signal temporal logic (ptSTL) formula is:

$$\phi = \mathbf{T} \mid x^i \sim c \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \phi_1 \mathbf{S}_I \phi_2 \tag{1}$$

where x^i is a signal variable, $\sim \in \{>, \geq, <, \leq, =\}$, and c is a constant. $\mathbf{T}$ is the Boolean constant $true$, $\neg$ and $\wedge$ are the Boolean operators negation and conjunction, respectively. $\mathbf{S}_I$ is the temporal operator $since$ with time interval I that can be any open or closed interval from the time domain $\mathbb{R}_{\geq 0}$.

For a ptSTL formula ϕ, a continuous signal $\mathbf{x}$, and a time value t, the satisfaction relation $\models$ is defined by (where $F(t, [a, b]) = [t-b, t-a] \cap [0, t]$, $F(t, [a, b)) = (t-b, t-a] \cap [0, t]$, $F(t, (a, b]) = [t-b, t-a) \cap [0, t]$, and $F(t, (a, b)) = (t-b, t-a) \cap [0, t]$):

$$\begin{aligned}
&(\mathbf{x}, t) \models \mathbf{T} \\
&(\mathbf{x}, t) \models x^i \sim c && \text{iff } x^i(t) \sim c, \sim \in \{>, \geq, <, \leq, =\} \\
&(\mathbf{x}, t) \models \neg\phi && \text{iff not } (\mathbf{x}, t) \models \phi \\
&(\mathbf{x}, t) \models \phi_1 \wedge \phi_2 && \text{iff } (\mathbf{x}, t) \models \phi_1 \text{ and } (\mathbf{x}, t) \models \phi_2 \\
&(\mathbf{x}, t) \models \phi_1 \mathbf{S}_I \phi_2 && \text{iff } \exists t' \in F(t, I), (\mathbf{x}, t') \models \phi_2, \forall t'' \in [t', t](\mathbf{x}, t'') \models \phi_1
\end{aligned} \tag{2}$$

Two additional temporal operators $\mathbf{F}_I^-$ $(previously)$ and $\mathbf{G}_I^-$ $(always)$ are defined as $\mathbf{F}_I^- \phi := \mathbf{T}\, \mathbf{S}_I \phi$ and $\mathbf{G}_I^- \phi := \neg \mathbf{F}_I^- \neg\phi$.

In case of a discrete-time signal $\mathbf{x} = x_0 x_1 \ldots$, that is defined as a mapping from $\mathbb{N}$ to $\mathbb{R}^n$, ptSTL semantics are interpreted over the piece-wise constant continuous signal $\mathbf{x}_{PWC}$ derived from $\mathbf{x}$ as $\mathbf{x}_{PWC}(t) = x_i$ when $t \in [i, i+1)$. With a slight abuse of notation, we write $(\mathbf{x}, t) \models \phi$ when $(\mathbf{x}_{PWC}, t) \models \phi$.

Parametric past time signal temporal logic is an extension of ptSTL [9]. In a parametric ptSTL formula, parameters can be used in place of numerical constants (c in (1)) or time bounds (interval bounds in (1)). For a parametric formula ϕ and a suitable parameter valuation, v, $\phi(v)$ denotes the ptSTL formula obtained by replacing each parameter with the corresponding value from v. As an example, consider the parametric formula $\phi = \mathbf{F}^-_{[p_1,p_2]} x < p_3$ with parameters p_1, p_2 and p_3. ptSTL formula $\phi(v) = \mathbf{F}^-_{[3,5]} x < 10.2$ is obtained with valuation $v = [p_1 \to 3, p_2 \to 5, p_3 \to 10.2]$.

3 System repair framework

In this work, our goal is to repair a system such that the resulting system does not generate a faulty behavior. The overall framework that includes system $\mathcal{S}$, fault detection mechanism *IsFaulty*, template formula set $\mathcal{F}$, and the repair mechanism $Repairer$ is introduced in this section.

The system is denoted by $\mathcal{S}$. A trace of $\mathcal{S}$ is a finite n-dimensional signal $\mathbf{x}$, and the set of all traces of $\mathcal{S}$ is denoted by $Traces(S)$. *IsFaulty* is a fault detection mechanism that takes a trace (or a partial trace) as input and generates a label indicating whether a faulty behavior is encountered at the end of the given trace (t_e denotes the last time point):

$$IsFaulty(\mathbf{x}) = \begin{cases} 1 \text{ if fault at } t_e \\ 0 \text{ otherwise} \end{cases} . \tag{3}$$

Here, we do not make any additional assumptions on the fault detection mechanism. As illustrated in the examples, it can be a safety specification checking a property of the last state (e.g., $x^i \leq c$), a temporal logic formula (e.g., $\mathbf{F}^-_{[0,3]} x^i > 0$—the value of x_i should be positive at least once within the last 3 time units), or it can compute a function that cannot be encoded as temporal formula (e.g., $\sum_{j=0}^{k} x^i_{t-j} \leq 0$).

We assume that $\mathcal{F}$ consists of a set of *repairable* parametric ptSTL formulae over the system $\mathcal{S}$ such that the $Repairer$ can modify the system $\mathcal{S}$ to avoid the satisfaction of an instance of a formula from the set $\mathcal{F}$ as stated in Assumption 1.

Assumption 1 For a parametric ptSTL formula $\phi \in \mathcal{F}$ and a valid parameter valuation v for ϕ, the repaired system $\mathcal{S}' = Repairer(\phi(v), \mathcal{S})$ satisfies the following conditions:

1. $\neg\phi(v)$ is satisfied along each trace from $Traces(\mathcal{S}')$
2. $Traces(\mathcal{S}') \subset Traces(\mathcal{S})$

The first condition states that the formula $\phi(v)$ will not be satisfied at any time step of a trace of the repaired system. In particular for each template ϕ, $Repairer$ has a mechanism to avoid ϕ in $\mathcal{S}$. Thus, when the cause of the faulty behavior is expressed as a formula $\phi(v)$, the repaired system will not generate this cause. The second condition guarantees that the changes performed by the $Repairer$ will not introduce any new behavior.

An important step of the proposed system repair framework is the identification of the fault cause in the form of a ptSTL formula. To achieve this, first a dataset of labeled signals is produced by simulating system $\mathcal{S}$ and labeling the generated traces with *IsFaulty* where t_e

is the last time point of trace $\mathbf{x}$ and $Simulation(\mathcal{S}) \subseteq Traces(\mathcal{S})$ is a set of traces generated by simulating $\mathcal{S}$:

$$\mathcal{D}(\mathcal{S}) = \{(\mathbf{x}, \mathbf{l}) \mid \mathbf{x} \in Simulation(\mathcal{S}), l(t) = IsFaulty(\mathbf{x}_{\leq t}) \text{ for } t \leq t_e\}. \tag{4}$$

Then, the optimal formula representing $\mathcal{D}(\mathcal{S})$ (4) from $\mathcal{F}$ is computed. Here, we assume that the faulty behavior can have multiple causes, for this reason we synthesize a formula in a disjunctive form (5) such that $\phi_i \in \mathcal{F}$ for each i.

$$\Phi = \phi_1(v_1) \vee \phi_2(v_2) \vee \ldots \phi_p(v_p). \tag{5}$$

The process of finding a ptSTL formula Φ (5) that would identify the cause(s) of the faulty behavior boils down to identifying a set of sub-formulae (ϕ_i) from $\mathcal{F}$ and finding a valuation v_i for the parameters in each sub-formula ϕ_i such that, the label generated by evaluating Φ mimics the label given in the dataset $\mathcal{D}(\mathcal{S})$. As the dataset labels are generated according to the fault identification process (3), if the formula evaluation along the traces matches the labels of the dataset, we can modify the system for each $\phi_i(v_i)$ from Φ according to Assumption 1 such that the traces of the modified system will satisfy $\neg\phi_i(v_i)$ for each $\phi_i(v_i)$. Thus, the modified system will not generate the identified causes.

The proposed system repair framework is summarized in Fig. 1. It requires a system $\mathcal{S}$, a fault detection mechanism *IsFaulty*, a set of repairable formulae $\mathcal{F}$ and the corresponding repairer $Repairer$. The first step is the computation of a labeled dataset as in $\mathcal{D}(\mathcal{S})$ (4). In the second step, a ptSTL formula Φ (5) explaining the causes of the faulty points in $\mathcal{D}(\mathcal{S})$ is computed from $\mathcal{F}$. Finally, $Repairer$ is called for each sub-formula $\phi_i(v_i)$ of Φ (5) iteratively.

The framework can remove the causes of the faults observed in the dataset $\mathcal{D}(\mathcal{S})$ (4), i.e., it guarantees that the repaired system will not generate the identified causes. However, the repaired system $\mathcal{S}'$ can still have faults as the synthesized formula represents a set of causes and these might mask others. Furthermore, due to the particular dataset generation (simulation) process, some of the faults might not be observed in the considered dataset $\mathcal{D}(\mathcal{S})$. To gain confidence that the repaired system has no faults, falsification techniques [8,19] can be used during the dataset generation step. In addition, if $\mathcal{S}'$ has faults, the overall process can be repeated to further refine the system.

Example 1 We illustrate the proposed framework with a toy example. Consider a two-dimensional discrete-time switched system $\mathcal{S}$:

$$x(t+1) = A_{u(t)}x(t), \quad u(t) \in \{1, 2\} \tag{6}$$

$$A_1 = \begin{bmatrix} 1.2 & 0 \\ 0 & 1.3 \end{bmatrix}, A_2 = \begin{bmatrix} 0.8 & 0 \\ 0 & 0.7 \end{bmatrix}. \tag{7}$$

The signal variables are x^0, x^1 and u. Thus, we have a three-dimensional signal that contains the state x and the control input u. The system is assumed to operate under normal conditions

Fig. 1 Proposed system repair framework

if both x^0 and x^1 are in the range $[0.1, 0.9]$. Our goal here is to find state-based constraints on u to keep the system in normal conditions. To achieve this, we apply our framework as follows. First, we generate a labeled dataset by simulating $\mathcal{S}$. Each trace is initialized randomly, at each time step t, $u_t \in \{1, 2\}$ is picked randomly and the labels are assigned according to the fault detection mechanism:

$$IsFaulty(\mathbf{x}) = \begin{cases} 0 \text{ if } x^i(t_e) \in [0.1, 0.9] \text{ for } i = 0, 1 \\ 1 \text{ otherwise} \end{cases} \tag{8}$$

where t_e is the last time point of signal $\mathbf{x}$. The dataset $\mathcal{D}(\mathcal{S})$ consists of 200 traces of length 50. A trace from $\mathcal{D}(\mathcal{S})$ is shown in Fig. 2. As seen in Fig. 2, the trace leaves the "safe" set $[0.1, 0.9] \times [0.1, 0.9]$ several times that we aim to avoid.

We define four parametric ptSTL formula to form $\mathcal{F} = \{\phi_{(0,<)}, \phi_{(1,<)}, \phi_{(0,>)}, \phi_{(1,>)}\}$, where $\phi_{(i,\sim)}$ for $\sim \in \{<, >\}$ and $i = 0, 1$ is

$$\phi_{(i,\sim)} = \mathbf{F}^-_{[1,1]}(x^i \sim p_x) \wedge \mathbf{F}^-_{[1,1]}(u = p_u). \tag{9}$$

Each formula has two parameters $p_x \in [0, 1]$ and $p_u \in \{1, 2\}$. Let $g : \mathbb{R}^2 \to 2^{\{1,2\}}$ be a set valued feedback control strategy for $\mathcal{S}$. The system is controlled in closed loop with $g(\cdot)$ when $u(t) \in g(x(t))$ for each t. Given a formula $\phi_{(i,\sim)} \in \mathcal{F}$, a parameter valuation v, and a control strategy $g(\cdot)$ of $\mathcal{S}$, *Repair* procedure generates a new strategy g^R defined by:

$$g^R([x^0, x^1]) = g([x^0, x^1]) \setminus \begin{cases} \{v(p_u)\} & \text{if } x^i \sim v(p_x) \\ \emptyset & \text{otherwise} \end{cases}. \tag{10}$$

The resulting system $\mathcal{S}'$ violates $(x^i \sim p_x) \wedge (u = p_u)$ at each time step. Furthermore, as only possible control choices are reduced, $Traces(\mathcal{S}') \subseteq Traces(\mathcal{S})$. Thus, both conditions from Assumption 1 are satisfied. The formula synthesis method described in Sect. 4 generates the repairable cause formula Φ^{toy} as in (5) by combining optimized formulae from $\mathcal{F}$. This result is found in 85 s on a PowerEdge T430 machine with Intel Xeon E5-2650 24C/48T processor.

$$\begin{aligned} \Phi^{toy} &= \phi_1 \vee \phi_2 \vee \phi_3 \vee \phi_4 \\ \phi_1 &= \mathbf{F}^-_{[1,1]}(x^1 < 0.14) \wedge \mathbf{F}^-_{[1,1]}(x^2 = 1) \\ \phi_2 &= \mathbf{F}^-_{[1,1]}(x^0 > 0.75) \wedge \mathbf{F}^-_{[1,1]}(x^2 = 0) \\ \phi_3 &= \mathbf{F}^-_{[1,1]}(x^1 > 0.69) \wedge \mathbf{F}^-_{[1,1]}(x^2 = 0) \\ \phi_4 &= \mathbf{F}^-_{[1,1]}(x^0 < 0.12) \wedge \mathbf{F}^-_{[1,1]}(x^2 = 1). \end{aligned} \tag{11}$$

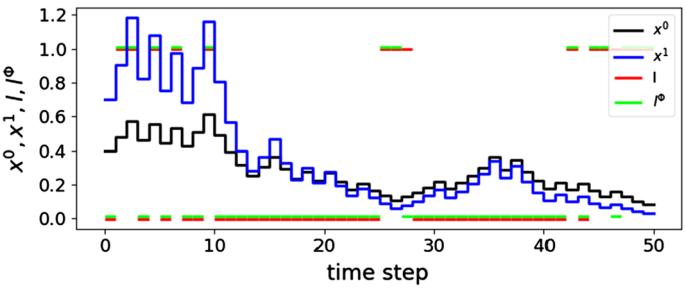


Fig. 2 A trace of system (6), its label according to (8), and the evaluation of Φ^{toy} along the trace (in green)

We iteratively apply the repair procedure (10) for each sub-formula ϕ_i from Φ^{toy} and obtain the repaired system $\mathcal{S}'$. In Fig. 3, to visualize the modification we plot arrows from x to $A_u x, u \in g(x)$ for both systems, that mimics a vector field representation. The subsystems are shown with different colors (red for A_1 and blue for A_2). The normal operating conditions are shown with green bounds, and the restrictions imposed by ϕ_i are highlighted. As it is clearly seen in Fig. 3 (right plot), the system stays in the green box.

Example 1 explains the proposed framework over a toy example by defining $\mathcal{S}$, *IsFaulty*, $\mathcal{F}$ and *Repairer* for it. While the system is quite simple, it illustrates how the framework can be used in an automated way once the required components are defined. The idea of restricting the controller as a repair mechanism is generalized in Sect. 5.

4 Repairable cause identification

In this section, the proposed formula generation method is explained in detail. The goal is to generate a ptSTL formula Φ in the form of (5) from a set of parametric ptSTL formulae $\mathcal{F}$ and a set of labeled traces $\mathcal{D}(\mathcal{S})$ such that evaluation of Φ along the traces matches the labels. To generate such a formula (5), an iterative approach is designed. At each iteration, a set of candidate parametric ptSTL sub-formulae are optimized and the optimized ptSTL formula $(\phi_i(v_i))$ with the highest evaluation score over the given dataset is added to the final formula Φ via disjunction.

First, the formula evaluation metrics that are used in the formula synthesis method are explained. For a finite signal $\mathbf{x}$, the binary label signal $l^{\phi(v)}$ for a parametric ptSTL formula ϕ and a valuation v is defined as follows:

$$l^{\phi(v)}(t) = \mathbf{1}((\mathbf{x}, t) \models \phi(v)), \tag{12}$$

where $\mathbf{1}$ maps the Boolean evaluation result to a binary value. The total duration of correctly identified positives (*True Positives*) (13) and the total duration of incorrect positive results (*False Positives*) (14) by a formula $\phi(v)$ over the given dataset $\mathcal{D}(\mathcal{S})$ are defined with respect to the labels generated by the formula $\phi(v)$ and the dataset labels:

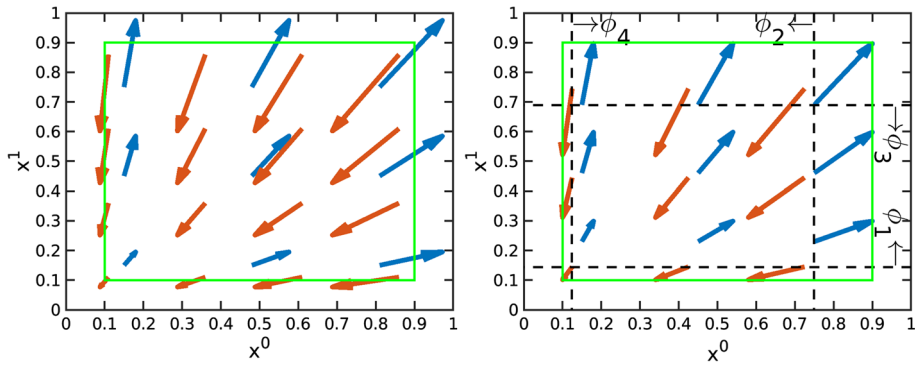


Fig. 3 Arrows from x to $A_u x, u \in g(x)$ for the original (on the left) and the repaired system (on the right)

$$TP(\phi(v), \mathcal{D}(\mathcal{S})) = \sum_{(\mathbf{x},\mathbf{l})\in\mathcal{D}(\mathcal{S})} \int_{t\in[0,t_e]} l(t) \cdot l^{\phi(v)}(t) \tag{13}$$

$$FP(\phi(v), \mathcal{D}(\mathcal{S})) = \sum_{(\mathbf{x},\mathbf{l})\in\mathcal{D}(\mathcal{S})} \int_{t\in[0,t_e]} (1 - l(t)) \cdot l^{\phi(v)}(t). \tag{14}$$

$TP(\phi(v))$ is used instead of $TP(\phi(v), \mathcal{D}(\mathcal{S}))$ when the dataset is clear in the context. In formula synthesis, the goal is to find a formula Φ that identifies all positive labels in the dataset, which maps to maximization of the total TP (13). In addition, as in the subsequent steps of the repair framework, the system will be modified to avoid the satisfaction of the generated formula, it is important to minimize FP (14) to limit unnecessary restrictions. An optimal formula would match all labels. However, due to the particular labeling process or non-determinism of the underlying system, it might not be possible to find such a formula. Here, our goal is to maximize TP (13) of the resulting formula, while bounding FP (14). The proposed synthesis method starts from $\Phi := False$, and iteratively finds a formula $\phi(v)$ with $\phi \in \mathcal{F}$ that maximizes TP for $\Phi \vee \phi(v)$ and updates Φ as $\Phi \vee \phi(v)$. Within the repair framework, $\mathcal{F}$ is a set of repairable parametric ptSTL formulae satisfying Assumption 1 and it depends on the considered system. We provide how $\mathcal{F}$ is generated for dynamical systems and timed automata in Sects. 5 and 6, respectively. Since the disjunction operator ($\vee$) carries error (FP) to the resulting formula, the error is bounded in the parameter optimization step to bound the error in the resulting formula Φ.

Algorithm 1 *FormulaSynthesis*

Require: $\mathcal{F}$: a set of parametric ptSTL formulae. $\mathcal{D}(\mathcal{S})$: a dataset of labeled traces.
Ensure: Φ: a ptSTL formula representing $\mathcal{D}(\mathcal{S})$.
1: $\phi(v^\star) = Optimize(\phi)$ and $TP^\star(\phi) = TP(\phi(v^\star))$ for each $\phi \in \mathcal{F}$
2: $\Phi_0 = False, \mathcal{F}_0 = \mathcal{F}, i = 0$
3: $\Phi_{i+1}, \mathcal{F}_{i+1} = Iterate(\Phi_i, \mathcal{F}_i, TP^\star(\phi))$
4: **while** $TP(\Phi_{i+1}) > TP(\Phi_i)$ **do**
5: $\quad i = i + 1$
6: $\quad \Phi_{i+1}, \mathcal{F}_{i+1} = Iterate(\Phi_i, \mathcal{F}_i, TP^\star(\phi))$
7: **end while**
8: **return** Φ_{i+1}

The iterative synthesis method is summarized in Algorithm 1. Initially, the optimal parameter valuation $v^\star$ and the corresponding optimal $TP(\phi(v^\star))$ (13) values, denoted as $TP^\star(\phi)$, are computed via $Optimize(\phi)$ method for each parametric ptSTL formula $\phi \in \mathcal{F}$. $Optimize(\phi)$ finds the parameter valuation $v^\star$ that maximize $TP(\phi(v))$ (13) while guaranteeing that $FP(\phi(v))$ (14) is less than a predefined bound B. The diagonal parameter synthesis method based on monotonicity properties from [17] is used in this step. Then, starting from $i = 0$, $\Phi_0 = False$ and $\mathcal{F}_0 = \mathcal{F}$; Φ_{i+1} and $\mathcal{F}_{i+1}$ are computed iteratively via Algorithm 2. Here, Φ_i is in the form of (5) and it represents the optimal formula found in the i-th iteration, and $\mathcal{F}_i$ is the set of parametric ptSTL formulae to be used in the following iteration. Algorithm 2 finds $\phi_i \in \mathcal{F}_i$ and valuation v_i, that maximize the number of true positives for $\Phi_{i+1} = \Phi_i \vee \phi_i(v_i)$ while bounding the cumulative error. Furthermore, it computes $\mathcal{F}_{i+1} \subseteq \mathcal{F}_i$ for the next iteration with a guarantee that no formula from $\mathcal{F}_i \setminus \mathcal{F}_{i+1}$ can be selected in the subsequent iterations. The iteration continues until there is no more increase in $TP(\Phi)$ (line 4).

Algorithm 2 *Iterate*

Require: $\mathcal{F}_i$: a set of parametric ptSTL formulae, Φ_i: a ptSTL formula, $TP^\star(\phi)$ optimal TP values for each parametric formula $\phi \in \mathcal{F}_i$
Ensure: $\Phi_{i+1} = \arg\max_{\phi \in \mathcal{F}_i} TP(\phi(v) \vee \Phi_i)$, $\mathcal{F}_{i+1} \subseteq \mathcal{F}_i$
1: $\Phi_{i+1} = \Phi_i$, $\mathcal{F}_{i+1} = \mathcal{F}_i$
2: **for** $\phi \in \mathcal{F}_i$ **do**
3: **if** $TP^\star(\phi) + TP(\Phi_i) > TP(\Phi_{i+1})$ **then**
4: $\phi' = \phi \vee \Phi_i$
5: $\phi'(v) = Optimize(\phi')$
6: **if** $TP(\phi'(v)) > TP(\Phi_{i+1})$ **then** $\Phi_{i+1} = \phi'(v)$
7: **if** $TP(\Phi_i) == TP(\phi'(v))$ **then** $\mathcal{F}_{i+1} = \mathcal{F}_{i+1} \setminus \{\phi\}$
8: **end if**
9: **end for**
10: **return** $\Phi_{i+1}, \mathcal{F}_{i+1}$

In Algorithm 2, parameter optimization of the combined formula $\phi \vee \Phi_i$ (line 5) for each $\phi \in \mathcal{F}_i$ is performed in a loop and the best known formula is stored in Φ_{i+1} (line 6). Note that at each iteration, only the parameters of the formula $\phi \in \mathcal{F}_i$ are optimized. While considering parametric formulae used to form Φ_i and optimizing the whole formula could potentially lead to a higher TP count, due to the computational complexity, it is not feasible. In addition, the optimization is only performed for the formulae that can have a higher score than the current best solution Φ_{i+1}, which is checked in line 3 via inequality (15) that holds for any valuation v.

$$TP(\Phi_i \vee \phi(v)) \leq TP(\Phi_i) + TP^\star(\phi). \tag{15}$$

Lastly, it might not be possible to increase the total TP count of Φ_i by optimizing parametric formula ϕ (line 7). In particular, it might be the case that all positive labels that can be identified by ϕ are already identified in previous iterations and integrated to Φ_i. Thus, it is no longer necessary to perform parameter optimization for ϕ in the subsequent iterations (line 7). Note that both $\mathcal{F}_i$ reduction in line 7 and TP check in line 3 reduce the total number of parameter optimizations that is the computationally expensive step of the overall algorithm.

Algorithm 1 iteratively selects parametric formulae from $\mathcal{F}$ and optimizes their parameters to maximize the total TP while bounding FP. If the set $\mathcal{F}$ is not sufficiently general, i.e., if the cause is not expressible with the formulae from $\mathcal{F}$, Algorithm 1 generates an over approximation of the actual cause. The approximation error is captured in FP and it can be adjusted via B.

5 Application to dynamical systems

In this section, we describe how the proposed repair framework is applied to dynamical systems with a finite control set via controller refinement.

We consider a discrete-time control system $\mathcal{S}$:

$$x(t+1) = f(x(t), u(t), w(k)), \quad u(t) \in g(\mathbf{x}_{[t-K,t-1]}, x(t)) \subseteq \mathbb{U} \tag{16}$$

where the state is $x(t) = [x^0(t), \ldots, x^{n-1}(t)] \in \mathbb{X} \subset \mathbb{R}^n$, the control input is $u(t) = [u^0(t), \ldots, u^{m-1}(t)] \in \mathbb{U} \subset \mathbb{R}^m$ that takes value from a finite set $\mathbb{U}$ and it is determined by a set valued feedback control strategy with a finite memory $g : (\mathbb{X} \times \mathbb{U})^K \times \mathbb{X} \to 2^{\mathbb{U}}$,

and $w(t) \in \mathbb{W} \subset \mathbb{R}^l$ is the noise at time step t. Furthermore, $\mathbf{x}_{[t-K,t-1]}$ is defined as $(x(t-K'), u(t-K'), \ldots, (x(t-1), u(t-1))$, and $K' = \min(t, K)$, that is necessary to guarantee that indices are positive when $t < K$. A finite trace of system (16) is denoted by $\mathbf{x} = (x(0), u(0)), \ldots, (x(N), u(N))$ such that $x(t+1) = f(x(t), u(t), w(t))$ for some $u(t) \in g(\mathbf{x}_{[t-K,t-1]}, x(t))$ and $w(t) \in \mathbb{W}$ for each $t = 0, \ldots, N-1$.

A *repairable* parametric ptSTL formula for system $\mathcal{S}$ (16) is in the following form:

$$\phi := \mathbf{G}^{-}_{[1,b]}(u^i = c) \wedge \mathbf{F}^{-}_{[1,1]}\phi', \tag{17}$$

where b and c are parameters, u^i is a control variable and ϕ' is any parametric ptSTL formula over the state $\{x^0, \ldots, x^{n-1}\}$ and control $\{u^0, \ldots, u^{m-1}\}$ variables of system $\mathcal{S}$ (16). The set of all parametric ptSTL formulae $\mathcal{F}^{\leq oc}$ that contain up to oc operators (temporal or Boolean) over a given set of variables is defined in [10]. By using $\mathcal{F}^{\leq oc}$ over $\{x^0, \ldots, x^{n-1}\}$ and $\{u^0, \ldots, u^{m-1}\}$, a set of formulae in the form of (17) is defined as:

$$\mathcal{F} := \{\mathbf{G}^{-}_{[1,b]}(u^i = c) \wedge \mathbf{F}^{-}_{[1,1]}\phi' \mid u^i \in \{u^0, \ldots, u^{m-1}\}, \phi' \in \mathcal{F}^{\leq oc}\}. \tag{18}$$

Remark 1 Both control and system formulae are shifted by 1 time unit relative to the fault location so that the controller can avoid the fault before it occurs if the fault is state based, which is the case for the considered examples. Based on the studied system, different relative time values can be used, e.g., $\mathbf{G}^{-}_{[k_1,b]}(u^i = c) \wedge \mathbf{F}^{-}_{[k_2,k_2]}\phi'$, or it can be embedded into the $\mathcal{F}^{\leq oc}$ set. The repair method presented in this section applies to formulae in the form of $\mathbf{G}^{-}_{[k,b]}(u^i = c) \wedge \mathbf{F}^{-}_{[k,k]}\phi'$ for any $k \geq 0$. A particular k value, i.e., $k = 1$ is used to simplify the notation as it is also used in the examples. In addition, the repair method also applies to the case when $k_2 > k_1$ via formula equality $\mathbf{G}^{-}_{[k_1,b]}(u^i = c) \wedge \mathbf{F}^{-}_{[k_2,k_2]}\phi' \equiv \mathbf{G}^{-}_{[k_1,b]}(u^i = c) \wedge \mathbf{F}^{-}_{[k_1,k_1]}\mathbf{F}^{-}_{[k_2-k_1,k_2-k_1]}\phi'$.

Next, we describe a repair procedure for an instance $\phi(v)$ of a parametric formula $\phi \in \mathcal{F}$ (18). The procedure is based on the refinement of the control strategy $g(\cdot)$ (16) such that the trajectories of the resulting system are guaranteed to violate $\phi(v)$. The refined strategy $g^R : (\mathbb{X} \times \mathbb{U})^{\bar{K}} \times \mathbb{X} \to 2^{\mathbb{U}}$ is also a finite memory feedback controller, where $\bar{K} = \max(K, K_\phi)$ and K_ϕ is the oldest time relative to k that is required to evaluate $\phi(v)$ at time k. The refined strategy is defined as:

$$\begin{aligned} g^R(\mathbf{x}_{[t-\bar{K},t-1]}, x(t)) = g(\mathbf{x}_{[t-\bar{K},t-1]}, x(t)) \setminus \\ \{u \in g(\mathbf{x}_{[t-\bar{K},t-1]}, x(t)) \mid ((\mathbf{x}_{[t-K,t-1]}, (x(t), u)), \bar{K}+1) \models \phi^{-1}(v)\}, \end{aligned} \tag{19}$$

where $\phi^{-1}(v) = \mathbf{G}^{-}_{[0,b-1]}(u^i = v(c)) \wedge \phi'(v)$, i.e., shifts the evaluation by $k = 1$ time unit (see Remark 1). The refined strategy simply removes the control inputs that lead to satisfying $\phi(v)$ at the next time step. We first state an assumption to guarantee that $g^R(\mathbf{x}, x) \neq \emptyset$ for any $(\mathbf{x}, x) \in (\mathbb{X} \times \mathbb{U})^{\bar{K}} \times \mathbb{X}$, and then prove that $g^R(\cdot)$ guarantees the satisfaction of $\neg\phi(v)$ at each time step.

Assumption 2 The strategy $g(\cdot)$ (16) satisfies

$$\{u \in g(\mathbf{x}, x) \mid u^i \neq c\} \neq \emptyset$$

for each $(\mathbf{x}, x) \in (\mathbb{X} \times \mathbb{U})^K \times \mathbb{X}$, $i = 0, \ldots, m-1$, and $c \in \mathbb{U} \downarrow_i$, where $\mathbb{U} \downarrow_i = \{c^i \mid [c^0, \ldots, c^{m-1}] \in \mathbb{U}\}$ is the projection of $\mathbb{U}$ on the ith dimension.

Definition 1 (Repaired system) Let $\mathcal{S}$ (16) be a control system, ϕ be a parametric ptSTL formula over the state and control variables of $\mathcal{S}$ as in (17), v be a valuation for ϕ, and $g^R(\cdot)$

be a strategy as defined in (19) with respect to $\mathcal{S}$ and $\phi(v)$. Then, the repaired system $\mathcal{S}'$ is defined as

$$x(t+1) = f(x(t), u(t), w(t)), \qquad u(t) \in g^R(\mathbf{x}_{[t-K,t-1]}, x(t)) \subseteq \mathbb{U}. \tag{20}$$

Proposition 1 *Given a control system $\mathcal{S}$ (16), a ptSTL formula ϕ as in (17) over $\mathcal{S}$, a valuation v, if Assumption 2 holds, then the traces of the repaired system $\mathcal{S}'$ as given in Definition 1 satisfy $\neg\phi(v)$ at each time step.*

Proof By construction of the refined strategy $g^R(\cdot)$ (19), if $g^R(\mathbf{x}_{[t-K,t-1]}, x(t)) \neq \emptyset$, then at each time step $t \geq 0$, the resulting trace $\mathbf{x} = (x(0), u(0)), \ldots (x(t), u(t))$ with $u(t) \in g^R(\mathbf{x}_{[t-K,k-1]}, x(t))$ is guaranteed to satisfy $\neg\phi(v)$. Note that any control input u with $u^i \neq c$ is sufficient to satisfy $\neg\phi(v)$ at the next time step due to the first part $\mathbf{G}^-_{[1,b]}(u^i = c)$ of ϕ. Consequently, by Assumption 2, we conclude that $g^R(\mathbf{x}_{[t-K,t-1]}, x(t)) \neq \emptyset$ and $\neg\phi(v)$ is satisfied at each time step. □

Proposition 1 ensures that the proposed repair procedure satisfies the first condition of Assumption 1. Next, we show that the second condition holds.

Proposition 2 *Given a control system $\mathcal{S}$ (16), a ptSTL formula ϕ as in (17) over $\mathcal{S}$, its valuation v, let be $\mathcal{S}'$ the repaired system as given in Definition 1, then Traces$(\mathcal{S}') \subseteq$ Traces$(\mathcal{S})$.*

Proof By construction of $g^R(\cdot)$, $g^R(\mathbf{x}, x) \subseteq g(\mathbf{x}, x)$ holds for each $(\mathbf{x}, x) \in (\mathbb{X} \times \mathbb{U})^K \times \mathbb{X}$ for any $K \geq 0$. The proof trivially follows from this set inclusion property. □

As stated in Sect. 3, the repair procedure is iteratively applied for a set of formulae from $\mathcal{F}$. We first present a stronger version of Assumption 2, and then present a sufficient condition such that refinement $g^{R'}(\cdot)$ of a refined strategy $g^R(\cdot)$ satisfies Proposition 1 under the stronger assumption.

Assumption 3 The strategy $g(\cdot)$ (16) satisfies

$$\{u \in g(\mathbf{x}, x) \mid \bigwedge_{i \in \mathcal{M}} u^i \neq c_i\} \neq \emptyset$$

for each $(\mathbf{x}, x) \in (\mathbb{X} \times \mathbb{U})^K \times \mathbb{X}$, $\mathcal{M} \subseteq \{0, \ldots, m-1\}$, and $c_i \in \mathbb{U} \downarrow_i$.

Assumption 3 states that when $g(\mathbf{x}, x)$ is filtered with an inequality (up to) each control dimension, the resulting set is not empty.

Proposition 3 *For a system $\mathcal{S}$ (16) with $g(\cdot)$ satisfying Assumption 3, let $\phi_j := \mathbf{G}^-_{[1,b_j]}(u^{j,i} = c_j) \wedge \mathbf{F}^-_{[1,1]}\phi'_j$, for $j = 1, \ldots, M$ be instances of the parametric ptSTL template given in (17) with $b_j \geq 2$ for each j, and let $g^{R_j}(\cdot)$ (19) be the refinement of $g^{R_{j-1}}(\cdot)$ w.r. to ϕ_j for $j \geq 1$ and $g^{R_0}(\cdot) = g(\cdot)$. Then, $g^{R_M}(\boldsymbol{x}, x) \neq \emptyset$ for any $(\boldsymbol{x}, x) \in (\mathbb{X} \times \mathbb{U})^K \times \mathbb{X}$.*

Proof First, observe that applying the refinement procedure iteratively maps to applying the procedure for $\phi_1 \vee \ldots \vee \phi_M$, i.e., the control strategy $g^{R_M}(\cdot)$ found at the end of the iterative procedure equals to the refined strategy w.r.to $\phi_1 \vee \ldots \vee \phi_M$. Consider an arbitrary control dimension $\alpha \in \{0, \ldots, m-1\}$, and let $\alpha^M \subseteq \{1, \ldots, M\}$ be the indices of the formulae restricting u^α, e.g., if $j \in \alpha^M$ then $i = \alpha$ for $u^{j,i} = c_j$ part of the formula. For any $\alpha_1, \alpha_2 \in \alpha^M$, $u \in \mathbb{U}$ and $(\mathbf{x}, x) \in (\mathbb{X} \times \mathbb{U})^K \times \mathbb{X}$ if $(\mathbf{x}, (x, u)) \models \phi^{-1}_{\alpha,1}$ then either (1) $(\mathbf{x}, (x, u)) \not\models \phi^{-1}_{\alpha,2}$ or (2) $c_{\alpha_1} = c_{\alpha_2}$ since $b_j \geq 2$. As it holds for an arbitrary control dimension α, and two arbitrary formulae along this dimension, we reach that at most one c_j is eliminated for each $j = 0, \ldots, m-1$. By, Assumption 3, we conclude that $g^{R_M}(\cdot)$ is not empty for any $(\mathbf{x}, x)$. □

By Propositions 1, 2 and 3, we reach that for a control system $\mathcal{S}$ (16), the repair framework outlined in Sect. 3 can be applied with the set of *repairable* formulae $\mathcal{F}$ as in (18), and *repairer* that implements controller refinement as given in Definition 1.

5.1 Case study: traffic system

As a case study, we consider a traffic system that consists of 6 links and 2 traffic signals shown in Fig. 4. The state variables are x^i (number of vehicles on link i) for each link i and the control variables are u^0 and u^1 (configuration of traffic signals). Thus, a trace of the system is an eight-dimensional signal over the state and the control variables. The dynamics of the traffic network is modeled as piece-wise affine system:

$$x^i(t+1) = x^i(t) + w^i(t) - f^i(x^i(t), s^i(t)) + \sum_{j=0,\ldots,n-1} \beta_{ji} f^j(x^j(t), s^j(t)) \text{where}$$

$$f^l(x^l(t), s^l(t)) = s^l(t) \min\{x^l(t), c^l, \min_{p=0,\ldots,n-1, \beta_{lp}\neq 0} \left\{ \frac{\alpha_{lp}}{\beta_{lp}} (x^p_{cap} - x^p(t)) \right\}\}. \tag{21}$$

In (21), the number of vehicles that leave link i at time step t is computed as $f^i(x^i(t), s^i(t))$, where $s^i(t) \in \{0, 1\}$ is computed w.r.to the control input $u(t)$. $s^i(t)$ is 1 if flow from i is allowed, otherwise it is 0. A traffic signal (u^0, u^1) can be 0 or 1, where $u^j = 0$ means that the flow is allowed in the horizontal direction and $u^j = 1$ means that the flow is allowed in the vertical direction. The capacity x^i_{cap} of a horizontal link ($i \in \{0, 1, 2\}$) and a vertical link ($i \in \{3, 4, 5\}$) are 40 and 20, respectively. The saturation flow c^i is 20 for $i \in \{0, 1, 2\}$ and 10 for $i \in \{3, 4, 5\}$. The vehicles that flow from links outside of the network are modeled via the noise w. The following bounds are used for the links: $w^i \in [4, 8]$ for $i = 0$, $w^i \in [0, 4]$ for $i = 3, 4$ and $w_i = 0$ for $i = 1, 2, 5$. The ratio of the free space in link j that is reserved for link i is denoted as α_{ij} (when the flow from i to j is allowed — determined w.r.to $u(t)$), the ratio of vehicles in link i that flow to j is denoted by β_{ij}, and the following values are used to define the network: $\beta_{ij} = 0.75$ for $i - j \in \{0-1, 1-2, 3-5\}$, $\beta_{ij} = 0.25$ for $i - j \in \{0-5, 3-1, 4-2\}$, $\alpha_{ij} = 1$ for $i - j \in \{0-1, 1-2, 0-5, 3-5\}$, $\alpha_{ij} = 0.5$ for $i - j \in \{3-1, 4-2\}$ and the other ratio parameters are 0. We refer the interested reader to [14] for more information on the system dynamics.

A faulty behavior is defined as a traffic congestion on link-1, and it is assumed that congestion occurs when the number of vehicles is greater than 75% of the link's capacity. Thus, we define the fault detection mechanism as:

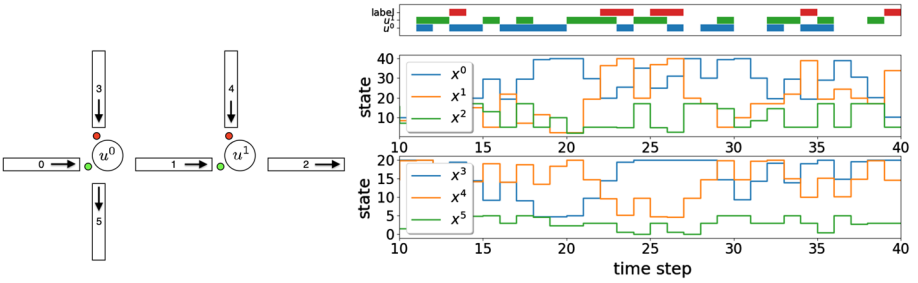


Fig. 4 Traffic network with 2 traffic signals and 6 links (on the left). A sample trace of the traffic network is given on the right. In the top plot, red, green and blue lines show that the label, control u^1 and control u^0 are 1, respectively

$$IsFaulty(\mathbf{x}) = \begin{cases} 1 \text{ if } (x^1(t_e) > 30) \\ 0 \text{ otherwise} \end{cases} . \tag{22}$$

A dataset $\mathcal{D}(\mathcal{S})$ of 20 labeled traces as in (4) is generated with (22) by simulating the system from random initial conditions for 100 time steps with $g(x) = \{[u^0, u^1] \mid u^0, u^1 \in \{0, 1\}\}$ for each x. Out of 2000 data points, 217 of them are labeled with 1, which means that link-1 is congested 10.85% of the time. A sample trace of the system is given in Fig. 4. Our aim in this example is to modify the faulty system to avoid the congestion on link-1.

We generate the set $\mathcal{F}^{\leq 1}$ of all parametric ptSTL formulae with at most 1 operator over the system variables as in [10] which contains 133 parametric ptSTL formulae, and define $\mathcal{F}$ w.r. to $\mathcal{F}^{\leq 1}$ as in (18). The parameter domains are defined as: $p_a, p_b \in \{i \mid i = 1, \ldots, 4\}$ for $\mathbf{G}^-_{[p_a,p_b]}$,$\mathbf{F}^-_{[p_a,p_b]}$ from $\mathcal{F}^{\leq 1}$, $p_c \in \{2i + 1 \mid i = 0, \ldots, 14\}$ for $c \in \{0, 1, 2\}$, $p_c \in \{2i + 1 \mid i = 0, \ldots, 7\}$ for $c \in \{3, 4, 5\}$. Algorithm 1 generates Φ^{tn} (23) when run over the dataset $\mathcal{D}(S)$, $\mathcal{F}$ and these parameter domains with a bound $B = 30$.

$$\begin{aligned} &\Phi^{tn} = \phi_1 \vee \phi_2, \text{ where} \\ &\phi_1 = (\mathbf{G}^-_{[1,2]}(u^1 = 1)) \wedge (\mathbf{F}^-_{[1,1]}(x^1 > 23)) \\ &\phi_2 = (\mathbf{G}^-_{[1,1]}(u^1 = 1)) \wedge (\mathbf{F}^-_{[1,1]}((x^1 > 15) \wedge (u^0 = 0))). \end{aligned} \tag{23}$$

Each sub-formula explains a condition that leads to congestion on link-1. The sub-formulae read as there will be a congestion (ϕ_1) when there are more than 23 vehicles on x^1 and u^1 does not allow the traffic flow from link-1 at the current and previous time steps, or (ϕ_2) when the flow from link-0 to link-1 is allowed, link-1 to link-2 is blocked and there are more than 15 vehicles on link-1. $TP(\Phi^{tn}, \mathcal{D}(\mathcal{S}))$ and $FP(\Phi^{tn}, \mathcal{D}(\mathcal{S}))$ are 217 and 59, respectively, and there are no false negatives. Thus, Φ^{tn} identifies all conditions leading to a congestion on link-1.

For each iteration of Algorithm 2, the number of parametric formulae ($|\mathcal{F}_i|$), the number of optimized formulae (i.e., number of formulae that satisfies the condition from line 3), the resulting formula Φ_i and its $TP(\Phi_i, \mathcal{D}(\mathcal{S}))$, and $FP(\Phi_i, \mathcal{D}(\mathcal{S}))$ values are given in Table 1. As seen in Table 1, the number of parameter optimizations performed in the second iteration is dropped drastically to 101 from 266 thanks to the formula analysis.

We iteratively refine the strategy $g(\cdot)$ to repair the system as in the repair procedure defined in Definition 1 for ϕ_1 and ϕ_2 from (23). Let $g^{R_1}(\cdot)$ and $g^{R_2}(\cdot)$ denote the strategies obtained after the first and the second refinement. Note that even though Φ^{tn} does not satisfy the condition from Proposition 3 ($b = 1$), the resulting strategy $g^{R_2}(\cdot)$ is not empty for any x since $g^{R_1}(\cdot)$ satisfies Assumption 2. The congestion rate drops to 3.7% when the system is run in closed loop with $g^{R_1}(\cdot)$, and it drops to 0% when the system is run in closed loop with $g^{R_2}(\cdot)$. Thus, we are able to identify the cause of the congestion on link-1 and repair the system to avoid it in a fully automated way. The computation took 332 s on the same machine as Example 1.

Table 1 Numerical results of Algorithm 2 over the traffic example for each iteration

Iteration i	$\lvert\mathcal{F}_i\rvert$	Optimized	Φ_i	$TP(\Phi_i, \mathcal{D}(\mathcal{S}))$	$FP(\Phi_i, \mathcal{D}(\mathcal{S}))$
1	266	266	ϕ_1	153	25
2	247	101	$\phi_1 \vee \phi_2$	217	59

Over the same traffic network, we apply our framework to avoid congestion on any link. For this purpose, we define the fault detection mechanism as:

$$IsFaulty(\mathbf{x}) = \begin{cases} 0 \text{ if } x^i(t_e) \leq 30 \text{ for each } i \in \{0,1,2\} \\ \quad \text{and } x^i(t_e) \leq 15 \text{ for each } i \in \{3,4,5\} \\ 1 \text{ otherwise} \end{cases} . \tag{24}$$

A dataset $\mathcal{D}'(\mathcal{S})$ of 20 labeled traces as in (4) is generated with (24) by simulating the system from random initial conditions for 100 time steps with the same control strategy $g(\cdot)$. Out of 2000 data points, 1093 of them are labeled with 1, which means that the traffic network is congested 54.65% of the time. Algorithm 1 generates $\Phi^{tn'}$ when run with $\mathcal{D}'(\mathcal{S})$, the formula set $\mathcal{F}$ and parameter domains introduced for the first case (with the exception of $p_b \in \{2,3,4\}$), and bound $B = 150$.

$$\begin{aligned} &\Phi^{tn'} = \phi_1 \vee \phi_2 \vee \phi_3 \vee \phi_4 \\ &\phi_1 = \mathbf{G}^-_{[1,2]}(u^0 = 0), \qquad \phi_2 = \mathbf{G}^-_{[1,2]}(u^0 = 1) \\ &\phi_3 = \mathbf{G}^-_{[1,2]}(u^1 = 0), \qquad \phi_4 = \mathbf{G}^-_{[1,2]}(u^1 = 1). \end{aligned} \tag{25}$$

$TP(\Phi^{tn'}, \mathcal{D}(\mathcal{S}))$ and $FP(\Phi^{tn'}, \mathcal{D}(\mathcal{S}))$ are 982 and 111, respectively. This formula states that if a traffic signal (u^0 or u^1) is kept the same for two consecutive time steps, there will be a traffic congestion. As in the previous case, we obtain a strategy $g^{R_4}(\cdot)$ from $\Phi^{tn'}$ by iteratively refining $g(\cdot)$ with respect to each sub-formula. We simulate the system in closed loop $g^{R_4}(\cdot)$ and there are no time points labeled with 1, thus the congestion is avoided. The computation took 586 s on the same machine as Example 1. In addition, the analysis of the system dynamics reveals that a congested state is unreachable. Thus, the proposed data-driven repair framework is able to avoid the unsafe (congested) states without explicitly considering the system dynamics.

The size of the set $\mathcal{F}$ (18) increases with the number of system variables and the number of operators (oc). The traffic system has 6 states and 2 control variables, and each formula $\phi \in \mathcal{F}_1$ has up to 6 parameters. Formula Φ^{tn} (23) includes 8 operators and 7 parameters and formula $\Phi^{tn'}$ (25) includes 3 operators and 8 parameters. An alternative approach to Algorithm 1 is to perform a parameter optimization for each parametric ptSTL formula in the form of (5). However, due to the complexity of the parameter optimization step, only 4% of the formulae from $\Phi \in \{\phi_1 \vee \phi_2 \mid \phi_1, \phi_2 \in \mathcal{F}_1\}$ are optimized over the dataset $\mathcal{D}(\mathcal{S})$ (congestion on link-1) in 10 hours on the same machine. In addition, we run the synthesis method from [17] on $\mathcal{D}(\mathcal{S})$. The computation takes 306s, the resulting formula Φ^{tn-c} includes 18 operators, $TP(\Phi^{tn-c}, \mathcal{D}(\mathcal{S})) = 201$ and $FP(\Phi^{tn-c}, \mathcal{D}(\mathcal{S})) = 101$. These results show that the proposed method achieves higher TP with more compact formulae compared to [17]. Finally, the resulting formulae show that the proposed method can generate complex formulae in an efficient way. Even though the size of $\mathcal{F}$ increases with the number of system variables, the formula synthesis is parallelizable.

6 Application to timed automata

In this section, we first introduce the timed automata formalism and then define repairable formulae together with the corresponding repair mechanisms.

A timed automaton (TA) [4,5] is a finite-state machine extended with a finite set of real-valued clocks progressing monotonically and measuring the time spent after their latest resets.

$\Phi(C)$ is a set of clock constraints over a set of clocks C. A *clock constraint* $\varphi \in \Phi(C)$ is given by the grammar:

$$\varphi := c \sim n \mid c_1 - c_2 \sim n \mid \varphi \wedge \varphi,$$

where $c, c_1, c_2 \in C$, $n \in \mathbb{N}$, and $\sim \in \{<, \leq, >, \geq, =\}$. A *clock interpretation* ν for a set of clocks C is a mapping from C to $\mathbb{R}_{\geq 0}$, i.e., it assigns a nonnegative real value to each clock in C. ν satisfies a clock constraint φ (shown as $\nu \models \varphi$) if and only if that constraint evaluates to true when ν is used. Two operations are defined for clock interpretations: delay and reset. For ν and $\delta \in \mathbb{R}_{\geq 0}$, the delay operation $\nu' := \nu + \delta$ increments each clock by δ, i.e., $\nu'(c) = \nu(c) + \delta$ for each $c \in C$. For ν and $\lambda \subseteq C$, the reset $\nu[\lambda]$ operation assigns 0 to each $c \in \lambda$ and agrees with ν for each $c' \in C \setminus \lambda$.

Definition 2 (Timed Automata) A *timed automaton*[1] is a tuple $\mathscr{A} = (L, l_0, C, Inv, T)$, where (i) L is a finite set of locations, (ii) $l_0 \in L$ is an initial location, (iii) C is a finite set of clocks, (iv) $Inv : L \to \Phi(C)$ is an invariant function, and (v) $T \subseteq L \times L \times 2^C \times \Phi(C)$ is a finite transition relation.

The semantics of a TA $\mathscr{A}$ is given by a timed transition system (TTS) induced by $\mathscr{A}$:

Definition 3 (Timed Transition System) A *timed transition system* of a TA $\mathscr{A} = (L, l_0, C, Inv, T)$ is a tuple $\mathcal{T}(\mathscr{A}) = (Q, q_0, \to)$, where

- $Q = \{(l, \nu) \mid l \in L, \nu \in \mathbb{R}_{\geq 0}^{|C|}, \nu \models Inv(l)\}$ is the set of states,
- $q_0 = (l_0, \nu_0) \in Q$ where $\nu_0(c) = 0$ for each $c \in C$ is the initial state, and
- $\to \subseteq (Q \times \mathbb{R}_{\geq 0} \times Q) \cup (Q \times Q)$ is the transition relation defined by the following rules

  1. (delay) $(l, \nu) \xrightarrow{\delta} (l, \nu + \delta)$ if $\nu + \delta' \models Inv(l)$,
  2. (discrete) $(l, \nu) \to (l', \nu[\lambda])$ if there exists $(l, l', \lambda, \varphi) \in T$ such that $\nu \models \varphi$, and $\nu[\lambda] \models Inv(l')$.

A *run* ρ of $\mathscr{A}$ is an alternating sequence of delay and discrete transitions

$$\rho : q_0 \xrightarrow{\delta_0} q_0 \to q_1 \xrightarrow{\delta_1} q_1 \to q_2 \xrightarrow{\delta_2} \dots \tag{26}$$

Run ρ induce a time sequence $\tau_\rho = \tau_0 \tau_1 \tau_2 \dots$ such that $\tau_0 = 0$ and $\tau_{i+1} = \tau_i + \delta_i$ for $i \geq 0$. We define a one-dimensional signal $\mathbf{x}$ from an automaton run ρ (26) as follows:

$$x(t) = l_j, \text{ when } t \in [\tau_j, \tau_{j+1}), \tag{27}$$

where $q_j = (l_j, \nu_j)$ is a state from run ρ as in (26) for a clock interpretation ν_j. The set of all such signals is denoted as $Traces(\mathscr{A})$. A network of timed automata (NTA) $\mathscr{A}^1, \dots, \mathscr{A}^N$, with $\mathscr{A}^i = (L^i, l_0^i, C^i, Inv^i, T^i)$, is used to model complex systems. The behavior of the overall system is defined via the product automaton $\mathscr{A} = \mathscr{A}^1 \mid \dots \mid \mathscr{A}^N$, where $L = L^1 \times \dots \times L^N$ is the location set of $\mathscr{A}$. We refer the interested reader to [4] for a detailed product automaton definition. The product is also a timed automaton as defined in Definition 2. Thus, the same derivations (e.g., Definition 3) and the same analysis apply. For the proposed data-driven repair framework, when the automaton $\mathscr{A}$ is defined as a product of N automata, we map a run ρ (26) of $\mathscr{A}$ into an N-dimensional signal $\mathbf{x}$:

[1] Our method does not include any operation on the particular language of the automaton; therefore, we omit an alphabet in the TA definition.

$$x(t) = (l_j^1, l_j^2, \ldots, l_j^N), \text{ when } t \in [\tau_j, \tau_{j+1}), \tag{28}$$

where $q_j = ((l_j^1, l_j^2, \ldots, l_j^N), v_j)$ is a state from ρ. The projection of the state on the ith dimension, i.e., ith TA, at time t is denoted by $x^i(t)$, i.e., $x^i(t) = l_j^i$ for $t \in [\tau_j, \tau_{j+1})$. Consequently, ptSTL formulae over $\{x^1, \ldots, x^N\}$ can be interpreted over signal $\mathbf{x}$ as in (2).

We continue by defining the parametric formula set $\mathcal{F}$ for the repair framework. The set $\mathcal{F}$ contains two types of parametric formulae:

$$(x^i = l^i \wedge \mathbf{G}^-_{(0,\epsilon]} x^i \neq l^i) \wedge \mathbf{G}^-_{(0,b]} \phi_l \tag{29}$$

$$(x^i = l^i \wedge \mathbf{G}^-_{(0,\epsilon]} x^i \neq l^i) \wedge \mathbf{F}^-_{[0,b)} \phi_l \tag{30}$$

where ϕ_l is defined as:

$$\phi_l := (x^j = l^{j,1}) \vee \ldots \vee (x^j = l^{j,n}). \tag{31}$$

$x^i, x^j \in \{x^1, \ldots, x^N\}$ are signal variables, $\epsilon \in \mathbb{R}_{>0}$ is a small positive constant, l^i, $b, l^{j,1}, \ldots, l^{j,n}$ are parameters with domains $l_i \in L^i, b \in [\underline{b}, \overline{b}] \subset \mathbb{R}_{>0}, l^{j,1}, \ldots, l^{j,n} \in L^j$. First part of (29) (and (30)) is satisfied when automaton $\mathscr{A}^i$ takes a transition to l^i. Notice that, ϕ_l consists of potential locations for the same TA and j can be equal to i, i.e., ϕ_l is used to refer a set of locations on a particular TA. Given a valuation v for ϕ_l (or for (29), (30)), the set of locations is denoted as $locs(\phi_l(v)) = \{v(l^{j,1}), \ldots, v(l^{j,n})\}$. Next, we define repair procedures and prove that they satisfy Assumption 1 for (29) and (30).

We start with formula (29) which indicates that the source of the error is the time spent in a set of locations being more than or equal to a threshold before entering a target location, i.e., it addresses the absence of a constraint on a transition. The proposed repair procedure is given in Definition 4.

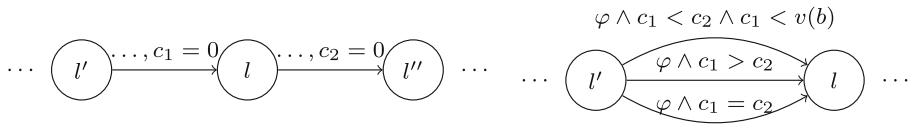
Definition 4 (TALessThanGuardRepair) Given an NTA $\mathscr{A}^1, \ldots, \mathscr{A}^N$ with $\mathscr{A}^k = (L^k, l_0^k, C^k, Inv^k, T^k)$, a parametric ptSTL formula ϕ as in (29) and a valuation v, the repaired system $\mathscr{A}^{1,r}, \ldots, \mathscr{A}^{N,r}$ is defined as follows with $\mathscr{A}^{k,r} = (L^k, l_0^k, C^{k,r}, Inv^k, T^{k,r})$.

- For $k \neq i$ and $k \neq j$, $\mathscr{A}^{k,r} = \mathscr{A}^k$.
- Introduce new clocks c_1 and c_2 shared by $\mathscr{A}^{i,r}$ and $\mathscr{A}^{j,r}$.
- For $k = j$, $\mathscr{A}^{j,r} = (L^j, l_0^j, C^{j,r}, Inv^j, T^{j,r})$, where $C^{j,r} = C^j \cup \{c_1, c_2\}$ and $T^{j,r} = T_{enter} \cup T_{leave} \cup T_{rest}$, where

$$T_{rest} = \{(l_s, l_t, \lambda, \varphi) \mid (l_s, l_t, \lambda, \varphi) \in T^j, \text{ and } \{l_s, l_t\} \subseteq locs(\phi_l(v)) \text{ or } \{l_s, l_t\} \cap locs(\phi_l(v)) = \emptyset\} \tag{32}$$

$$T_{enter} = \{(l_s, l_t, \lambda \cup \{c_1\}, \varphi) \mid (l_s, l_t, \lambda, \varphi) \in T^j, \; l_s \notin locs(\phi_l(v)), \text{ and } l_t \in locs(\phi_l(v))\} \tag{33}$$

$$T_{leave} = \{(l_s, l_t, \lambda \cup \{c_2\}, \varphi) \mid (l_s, l_t, \lambda, \varphi) \in T^j, \; l_s \in locs(\phi_l(v)), \text{ and } l_t \notin locs(\phi_l(v))\} \tag{34}$$



(a) A portion of $\mathscr{A}^{j,r}$ for $l, l', l'' \in L^j$, $l \in locs(\phi_l(v))$, and $l', l'' \notin locs(\phi_l(v))$.

(b) A portion of $\mathscr{A}^{i,r}$ for $l, l' \in L^i$, $l = v(l^i)$, and $l_0^j \notin locs(\phi_l(v))$.

Fig. 5 Two partial TA demonstrating the repair method of Definition 4

– For $k = i$, $\mathscr{A}^{i,r} = (L^i, l_0^i, C^{i,r}, Inv^i, T^{i,r})$, where $C^{i,r} = C^i \cup \{c_1, c_2\}$ and $T^{i,r} = T_{\not\to l^i} \cup T_{\to l^i}$, where

$$
\begin{aligned}
T_{\not\to l^i} =& \{(l_s, l_t, \lambda, \varphi) \mid (l_s, l_t, \lambda, \varphi) \in T^i \text{ and } l_t \neq v(l^i)\} \\
T_{\to l^i} =& \{(l_s, l_t, \lambda, \varphi \wedge c_1 < c_2 \wedge c_1 < v(b)), \\
& (l_s, l_t, \lambda, \varphi \wedge c_1 > c_2), \\
& (l_s, l_t, \lambda, \varphi \wedge \varphi') \mid \quad (l_s, l_t, \lambda, \varphi) \in T^i \text{ and } l_t = v(l^i)\}, \text{ and} \qquad (35)
\end{aligned}
$$

$$
\varphi' = \begin{cases} \neg \mathbf{T} & \text{if } l_0^j \in locs(\phi_l(v)) \\ c_1 = c_2 & \text{otherwise} \end{cases}. \qquad (36)
$$

The repair procedure given in Definition 4 creates two clocks c_1 and c_2 shared by $\mathscr{A}^i$ and $\mathscr{A}^j$, resets c_1 on each transition from $L^j \setminus locs(\phi_l(v))$ to $locs(\phi_l(v))$, resets c_2 on each transition from $locs(\phi_l(v))$ to $L^j \setminus locs(\phi_l(v))$, and checks these clocks on transitions end in l^i in $\mathscr{A}^i$. The clock resets imply that when $c_1 < c_2$, $\mathscr{A}^j$ is in a location from $locs(\phi_l(v))$ and c_1 measures the time spent in $locs(\phi_l(v))$. On the other hand, when $c_1 > c_2$, $\mathscr{A}^j$ is not in a location from $locs(\phi_l(v))$ and c_2 measures the time passed since $locs(\phi_l(v))$ is left. For each transition that ends in $v(l^i)$ on $\mathscr{A}^i$, three transitions are added to $\mathscr{A}^{i,r}$ in (35) to handle different cases with respect to the clocks c_1 and c_2[2]. Fig. 5 visualizes the given procedure over two partial TA by demonstrating placements of c_1 and c_2.

Now, we prove that the repair procedure given in Definition 4 satisfies Assumption 1, i.e., after the repair procedure is executed, ptSTL formula $\phi(v)$ is never satisfied and no new behavior is introduced.

Proposition 4 *Given an NTA $\mathscr{A}^1, \ldots, \mathscr{A}^N$ with $\mathscr{A}^k = (L^k, l_0^k, C^k, Inv^k, T^k)$, a parametric ptSTL formula ϕ as in (29) and a valuation v, let $\mathscr{A}^{1,r}, \ldots, \mathscr{A}^{N,r}$ be the repaired network of TA as defined in Definition 4, then each $\mathbf{x} \in Traces(\mathscr{A}^{1,r} \mid \ldots \mid \mathscr{A}^{N,r})$ always satisfies $\neg\phi(v)$.*

Proof Assume by contradiction that a trace $\mathbf{x} \in Traces(\mathscr{A}^r)$ satisfies $\phi(v)$ at time t, i.e., $x(t) \models \phi(v)$. Thus, $x^i(t) = v(l^i)$, $x^i(t') \neq v(l^i)$ for $t' \in [t-\epsilon, t)$, and $x^j(t') \in locs(\phi_l(v))$ for all $t' \in [0, t) \cap [t - v(b), t)$ by (29) and (2). As x^i is changed to $v(l^i)$ at time t, a discrete transition $(l_s, v(l^i), \lambda, \varphi^r) \in T^{i,r}$ is taken at t. Since $x^j(t') \in locs(\phi_l(v))$ for $t' \in [0, t) \cap [t - v(b), t)$, either (1) $x^j(t'') \in locs(\phi_l(v))$ for each $t'' \in [0, t)$, or (2) a transition from $L^j \setminus locs(\phi_l(v))$ to $locs(\phi_l(v))$ is taken at some time in $(0, t - v(b))$. We first analyze case (1). The condition implies that $l_0^j \in locs(\phi_l(v))$ since $x^j(0) \in locs(\phi_l(v))$.

[2] An edge case is passing $locs(\phi_l(v))$ within 0 time units. If such a run exists, the transition with the constraint $\varphi \wedge \varphi'$ (35) should be omitted. The existence of such a run can be checked by verifying the original NTA against formula $\mathbf{G}_{[0,\infty]}\left(\phi_l(v) \implies \mathbf{F}_{[0,0]}\neg\phi_l(v)\right)$.

By (32), at time t, $c_1 = t$ and $c_2 = t$. By construction of $T^{i,r}$ (35), each transition to $v(l^i)$ includes $c_1 < c_2 \wedge c_1 < v(b)$, $c_1 > c_2$ or $\neg\mathbf{T}$ (i.e., *false*) in its guard when $l_0^j \in locs(\phi_l(v))$, and each guard evaluates to false at $x(t)$ since $c_1 = c_2$.

Now consider case (2). Let t'' be the time of the last transition from $L^j \setminus locs(\phi_l(v))$ to $locs(\phi_l(v))$ along $\mathbf{x}$ prior t , i.e., $t'' < t - v(b)$ and $x^j(t''') \in locs(\phi_l(v))$ for $t''' \in (t'', t)$. By (33), c_1 is reset at t'', and by (34) c_2 is not reset during (t'', t). Thus, $c_2 > c_1$, and c_1 is $t - t''$ at time t, which implies $c_1 \geq v(b)$ As in the previous case, none of the constraints (e.g., $c_1 < c_2 \wedge c_1 < v(b)$, $c_1 > c_2$, or $c_1 = c_2$) introduced on transitions that end in $v(l^i)$ is satisfied at $x(t)$, thus we reached a contradiction. As we considered all cases, we conclude that each trace of the repaired system always satisfies $\neg\phi(v)$. □

Proposition 5 *Given an NTA $\mathscr{A}^1, \ldots, \mathscr{A}^N$ with $\mathscr{A}^k = (L^k, l_0^k, C^k, Inv^k, T^k)$, a parametric ptSTL formula ϕ as in (29), and a valuation v, let $\mathscr{A}^{1,r}, \ldots, \mathscr{A}^{N,r}$ be the repaired network of TA as defined in Definition 4, then*

$$Traces(\mathscr{A}^{1,r} \mid \ldots \mid \mathscr{A}^{N,r}) \subseteq Traces(\mathscr{A}^1 \mid \ldots \mid \mathscr{A}^N).$$

Proof For each $k \notin \{i, j\}$, $\mathscr{A}^k$ and $\mathscr{A}^{k,r}$ are the same. For j (when $i \neq j$), the traces of $\mathscr{A}^j$ and $\mathscr{A}^{j,r}$ are the same since the changes only concern the new clocks and no new constraint is introduced (see (32),(33),(34)). For i, the changes only restrict the behavior via new transition constraints. Essentially, for each transition of $(l_s, l_t, \lambda, \varphi^r)$ of $\mathscr{A}^{i,r}$, there exists a transition $(l_s, l_t, \lambda, \varphi)$ of $\mathscr{A}^i$ (see (35)) such that if a clock valuation $\nu \models \varphi^r$, then $\nu \models \varphi$. Hence, each trace of the product of the repaired system is also a trace of the product of the original system. □

We continue with formula (30) which indicates that the source of error is the time spent out of a set of locations being less than or equal to a threshold before entering a target location. In particular, it states that a location from $locs(\phi_l)$ should not be visited within the last b time units before entering l^i in $\mathscr{A}^i$. The proposed repair procedure is given in Definition 5.

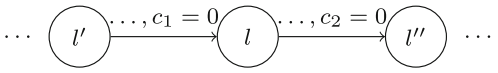
Definition 5 (TAMoreThanGuardRepair) Given an NTA $\mathscr{A}^1, \ldots, \mathscr{A}^N$ with $\mathscr{A}^k = (L^k, l_0^k, C^k, Inv^k, T^k)$, a parametric ptSTL formula ϕ as in (30) and a valuation v, the repaired system $\mathscr{A}^{1,r}, \ldots, \mathscr{A}^{N,r}$ is defined as follows with $\mathscr{A}^{k,r} = (L^k, l_0^k, C^{k,r}, Inv^k, T^{k,r})$.

- For $k \neq i$ and $k \neq j$, $\mathscr{A}^{k,r} = \mathscr{A}^k$.
- Introduce new clocks c_1 and c_2 shared by $\mathscr{A}^{i,r}$ and $\mathscr{A}^{j,r}$.
- For $k = j$, $\mathscr{A}^{j,r} = (L^j, l_0^j, C^{j,r}, Inv^j, T^{j,r})$, where $C^{j,r} = C^j \cup \{c_1, c_2\}$ and $T^{j,r} = T_{enter} \cup T_{leave} \cup T_{rest}$, where T_{rest}, T_{enter} and T_{leave} are as defined in (32), (33) and (34), respectively.
- For $k = i$, $\mathscr{A}^{i,r} = (L^i, l_0^i, C^{i,r}, Inv^i, T^{i,r})$, where $C^{i,r} = C^i \cup \{c_1, c_2\}$ and $T^{i,r} = T_{\not\to l^i} \cup T_{\to l^i}$, where

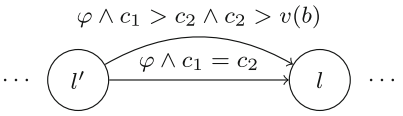
$$\begin{aligned} T_{\not\to l^i} =& \{(l_s, l_t, \lambda, \varphi) \mid (l_s, l_t, \lambda, \varphi) \in T^i \text{ and } l_t \neq v(l^i)\} \\ T_{\to l^i} =& \{(l_s, l_t, \lambda, \varphi \wedge c_1 > c_2 \wedge c_2 > v(b)), (l_s, l_t, \lambda, \varphi \wedge \varphi') \\ & \mid (l_s, l_t, \lambda, \varphi) \in T^i \text{ and } l_t = v(l^i)\}, \end{aligned} \tag{37}$$

  where φ' is defined as in (36).

The repair procedure given in Defn 5 is similar to the one given in Defn 4. Again $c_1 > c_2$ implies that $\mathscr{A}^j$ is not in a location from $locs(\phi_l(v))$, $c_2 > c_1$ implies that $\mathscr{A}^j$ is in a location from $locs(\phi_l(v))$ and c_2 measures the time passed since $locs(\phi_l(v))$ is left if it was entered



(a) A portion of $\mathscr{A}^{j,r}$ for $l, l', l'' \in L^j$, $l \in ocs(\phi_l(v))$, and $l', l'' \notin locs(\phi_l(v))$.



(b) A portion of $\mathscr{A}^{i,r}$ for $l, l' \in L^i$, $l = v(l^i)$, and $l_0^j \notin locs(\phi_l(v))$.

Fig. 6 Two partial TA demonstrating the repair method of Definition 5

from $L \setminus locs(\phi_l(v))$. Consequently, $c_1 > c_2$ and $c_2 > v(b)$ implies that $\mathscr{A}^j$ was not in a location from $locs(\phi_l(v))$ within the last $v(b)$ time units. A second transition is added to handle the special case of $c_1 = c_2$ according to the initial location l_0^j[3]. In particular, $c_1 = c_2$ means that $locs(\phi_l(v))$ is never entered from a location $L^j \setminus locs(\phi_l(v))$. Thus, it is safe to take a transition to $v(l^i)$ when $l_0^j \notin locs(\phi_l(v))$[2]. However, it should be avoided when $l_0^j \in locs(\phi_l(v))$ as $\mathscr{A}^j$ is still in $locs(\phi_l(v))$ (36). Fig. 6 demonstrates the repair method by showing placements of c_1 and c_2.

Now, we prove that the repair procedure given in Definition 5 satisfies Assumption 1, i.e., after the repair procedure is executed, ptSTL formula $\phi(v)$ is never satisfied and no new behavior is introduced.

Proposition 6 *Given an NTA $\mathscr{A}^1, \ldots, \mathscr{A}^N$ with $\mathscr{A}^k = (L^k, l_0^k, C^k, Inv^k, T^k)$, a parametric ptSTL formula ϕ as in (30) and a valuation v, let $\mathscr{A}^{1,r}, \ldots, \mathscr{A}^{N,r}$ be the repaired network of TA as defined in Definition 5, then each $\mathbf{x} \in Traces(\mathscr{A}^{1,r} \mid \ldots \mid \mathscr{A}^{N,r})$ always satisfies $\neg\phi(v)$.*

Proof A similar argument to the proof of Proposition 4 applies to this proof as well. Assume by contradiction that $x(t) \models \phi(v)$ for some t. Similar to the proof of Proposition 4, the satisfaction of $\phi(v)$ at t implies that a transition $(l_s, v(l^i), \lambda, \varphi^r) \in T^{i,r}$ is taken at t. Since $x^j(t') \in locs(\phi_l(v))$ for some $t' \in [0, t) \cap (t - v(b), t)$, either (1) $x^j(t'') \in locs(\phi_l(v))$ for each $t'' \in [0, t]$, or (2) a transition from $L^j \setminus locs(\phi_l(v))$ to $locs(\phi_l(v))$ is taken at some time in $(0, t']$. Case (1) implies that $l_0^j \in locs(\phi_l(v))$ since $x^j(0) \in locs(\phi_l(v))$. By (32), at time t, $c_1 = t$ and $c_2 = t$. Construction of $T^{i,r}$ (37) implies that each transition to $v(l^i)$ includes $c_1 > c_2 \wedge c_2 > v(b)$ or $\neg\mathbf{T}$ in its guard when $x^j(0) \in locs(\phi_l(v))$, and neither is satisfied since $c_1 = c_2$. For case (2), let $\bar{t}$ be the largest time point up to t such that $x^j(\bar{t}) \in locs(\phi_l(v))$, by assumption $\bar{t} > t - v(b)$. Let $t'' \in (0, \bar{t}]$ be the time of the last transition from $L^j \setminus locs(\phi_l(v))$ to $locs(\phi_l(v))$ prior to $\bar{t}$, thus $x^j(t''') \in locs(\phi_l(v))$ for $t''' \in (t'', \bar{t}]$. By (33), c_1 is reset at t''. Now consider 2 sub-cases: (a) $\bar{t} = t$, (b) $\mathscr{A}^j$ left $locs(\phi_l(v))$ at time $t''' \in [\bar{t}, t)$. For case (a), since c_2 is not reset during $[t'', t)$ (see (34)), $c_2 > c_1$ and the condition $c_1 > c_2$ from (37) is violated. For case (b), c_2 is $t - t'''$ at time t and $t''' > \bar{t}$. By the initial assumption $t - v(b) < \bar{t} \leq t$, thus $c_2 < v(b)$. Thus, the constraint $c_2 > v(b)$ from (37) is violated, and the transition cannot be taken. Hence, none of the constraints introduced on transitions ending in $v(l^i)$ is satisfied at $x(t)$ which implies a contradiction. As we considered all cases, we conclude that each trace of the repaired system satisfies $\neg\phi(v)$ at each time step. □

[3] An edge case is a run to $locs(\phi_l(v))$ within 0 time units. If such a run exists, the second transition should be omitted. Existence of such a run can be checked by verifying the original NTA against formula $\mathbf{F}_{[0,0]}\phi_l(v)$.

Proposition 7 *Given an NTA* $\mathscr{A}^1, \ldots, \mathscr{A}^N$ *with* $\mathscr{A}^k = (L^k, l_0^k, C^k, Inv^k, T^k)$*, a parametric ptSTL formula* ϕ *as in* (30)*, and a valuation* v*, let* $\mathscr{A}^{1,r}, \ldots, \mathscr{A}^{N,r}$ *be the repaired network of TA as defined in Definition 5, then*

$$Traces(\mathscr{A}^{1,r} \mid \ldots \mid \mathscr{A}^{N,r}) \subseteq Traces(\mathscr{A}^1 \mid \ldots \mid \mathscr{A}^N).$$

Proof A similar argument to the proof Proposition 5 applies, i.e., for each automaton $\mathscr{A}^{i,r}$, and for each transition of $(l_s, l_t, \lambda, \varphi^r)$ of $\mathscr{A}^{i,r}$, there exists a transition $(l_s, l_t, \lambda, \varphi)$ of $\mathscr{A}^i$ (see (37)) such that if a clock valuation $\nu \models \varphi^r$, then $\nu \models \varphi$. Hence, no new behavior is introduced by the procedure. □

We present two types of ptSTL formulae (29) and (30) and the corresponding repair procedures for applying the proposed repair framework to TA. Both repair procedures add two new clocks and up to four unique simple clock constraints ($c_1 < c_2$, $c_1 > c_2$, $c_1 = c_2$, $c_1 < v(b)$) to the model; therefore, the number of clocks and the number of unique simple constraints in the repaired TA increase linearly with the number of sub-formulae synthesized by Algorithm 1. The increase in the number of clocks can be reduced by applying a clock reduction algorithm [22,38]. In the case studies, we run the algorithm from [38] on the repaired models and report the results.

Next, we present case studies to demonstrate our framework on TA. In our case studies, we borrow well-known UPPAAL [15] models from the literature, i.e., Fischer's protocol [23], DB from [26], SBR from [26,27], and nuclear plant and train models from [7]. To observe a faulty behavior; for Fischer's protocol and SBR, we instantiate the model with a faulty configuration; DB is already faulty; and for nuclear plant and train examples, we randomly delete guards and invariants. Our experiment setup consists of five steps: (i) trace generation using UPPAAL SMC; (ii) formula synthesis using Algorithm 1; (iii) automatic repair according to synthesized formula; (iv) verification of the repaired model using UPPAAL; and (v) running the clock reduction algorithm from [38] on the repaired model.

In Table 2, we report the results for the case studies. The second column presents the runtime of steps (ii) and (iii). Note that step (iii) takes significantly less time than (ii). The third, fourth and fifth columns present the number of clocks of the original model, the number of clocks after the repair and the number of clocks after running the clock reduction algorithm on the repaired model, respectively.

6.1 Case study: Fischer's protocol

We apply the proposed repair framework on a TA shown in Fig. 7 which models Fischer's mutual exclusion protocol [23]. The protocol provides a timed mechanism without any block-

Table 2 Performance evaluations for case studies

Name of case study	Runtime (s)	Clock count before repair	Clock count after repair	Clock count after running [38]
Fischer [23]	1.11	2	6	4
DB [26]	2.11	3	5	4
SBR [26,27]	2.68	8	14	11
Nuclear Plant [7]	4.94	2	4	4
Train [7]	3.70	2	4	4

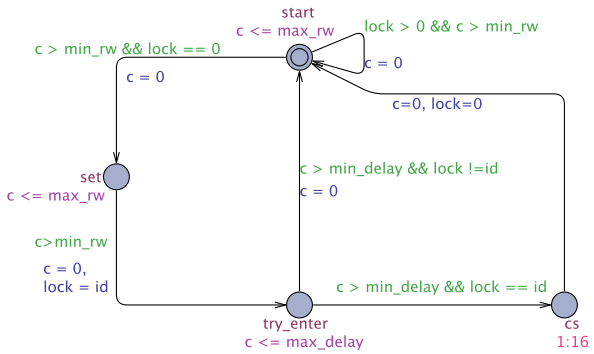


Fig. 7 TA model implementing the Fischer's protocol

ing structure for processes sharing the same resource and no two processes are allowed in the critical section simultaneously, i.e., a non-reachability property is satisfied.

Implementation of the protocol in Fig. 7 is a generic template for each process in the system, i.e., for each process, the template is instantiated with a different id to form an NTA. Each process has its own clock for the timed behavior and processes share global variables: $max\_rw, min\_rw, max\_delay, min\_delay$ and $lock$. max_rw and min_rw limits the time spent in the start and set locations. Similarly, max_delay and min_delay limits the time spent in try_enter location. Integer variable $lock$ indicates which process is currently in its critical section. The necessary condition for a correct implementation of the protocol is $max\_rw \leq min\_delay$ [23]. For demonstration, we instantiated two processes with the following configuration:

$$max\_rw = 5,\ min\_rw = 3,\ max\_delay = 6,\ min\_delay = 2. \tag{38}$$

This configuration does not satisfy the necessary condition for the protocol since both processes can simultaneously be in the critical section, i.e., $P1.cs \wedge P2.cs$ is reachable. Our goal is to repair the system so that $P1.cs \wedge P2.cs$ will be unreachable. We define the fault detection mechanism with respect to this requirement (see (3)):

$$IsFaulty(\mathbf{x}) = \begin{cases} 1 \text{ if } x^1(t_e) = \mathsf{cs} \wedge x^2(t_e) = \mathsf{cs} \wedge \\ \neg(x^1(t_e - 1) = \mathsf{cs} \wedge x^2(t_e - 1) = \mathsf{cs}) \\ 0 \text{ otherwise} \end{cases}. \tag{39}$$

In particular, we only mark the starting point of the violation (i.e., the first time step that the violation appears). We generate 100 traces with duration 100 of the model in Fig. 7 using the configuration in (38) with UPPAAL SMC toolbox [15], and label the traces according to (39). The total duration for the positive label is 29, i.e., 0.29%. As both processes share the same template TA and the model is designed to avoid the unsafe state (e.g., fault (39)) via delay parameters, we define the parametric formulae over a single TA, e.g., only use x^1 in (29) and (30) to form $\mathcal{F}$. Note that, the repair procedure will affect both TA since they share the same template. Algorithm 1 generates Φ^{ta} when run on this dataset and $\mathcal{F}$.

$$\Phi^{ta} = (x^1 = \mathsf{cs} \wedge \mathbf{G}^{-}_{(0,1]} x^1 \neq \mathsf{cs}) \wedge \mathbf{F}^{-}_{[0,5)}((x^1 = \mathsf{set})).$$

By Definition 5, formula Φ^{ta} implies the following repair procedure: create two new clocks c_1 and c_2; reset c_1 on the transition entering set and c_2 on the transition leaving set; and

control c_1 and c_2 on two new transitions replacing the transition entering cs with constraints $c > min\_delay \wedge lock = id \wedge c_1 = c_2$ and $c > min\_delay \wedge lock = id \wedge c_1 > c_2 \wedge c_2 > v(b)$ where $v(b) = 5$. Our automated implementation outputs the repaired TA as described.

Observe that c_1 is always more than c_2 since c is checked with $c > min\_rw$, where $min\_rw = 3$, on the transition from set to try_enter. Then, condition $c_1 > c_2$ is always satisfied; hence, c_1 can simply be discarded which leaves us with c_2 and a transition entering cs with the constraint $c > min\_delay \wedge lock = id \wedge c_2 > 5$. Since c is reset on the same transition as c_2, they have the same value when they are checked on the transition entering cs. On that transition, c is check with $c > min\_delay$, where $min\_delay = 3$, and c_2 is checked with $c_2 > 5$. Clearly, the constraint on c_2 dominates the constraint on c. Therefore, discarding c_2 and redefining $min\_delay = 5$ gives us the same semantic behavior as the repaired model. Notice that, redefining min_delay induces the following configuration which satisfies the necessary condition for a correct instantiation of the protocol:

$$max\_rw = 5,\ min\_rw = 3\ max\_delay = 6, min\_delay = 5. \tag{40}$$

After the repair, $P1.cs \wedge P2.cs$ is not reachable (verified by UPPAAL [15]) and the condition $max\_rw \leq min\_delay$ from [23] is satisfied. Hence, our framework is able to repair the model in a fully automated way.

First row of Table 2 reports the performance results of this case study. Our framework repairs the model in 1.11s (0.94s for Algorithm 1), increases the number of clocks to six (three for each instance of the model), and running [38] on the repaired model reduces this number to four (two for each instance of the model). Notice that, integrating expression simplification methods [22] into [38] can further reduce the number of clocks to two (one for each instance of the model) by automating the presented detailed clock analysis.

6.2 Case study: DB

After the detailed demonstration of the Fischer's protocol, we present our results on an NTA modeling the communication between a database server and a database from [26]. Due to space limitations, we cannot describe the model in detail and refer the reader to [26]. First, we make two minor modifications on the model in order to generate traces for our tool. We convert the channels to broadcast channels (a requirement of UPPAAL SMC) and introduce an error location error that is only reachable from serReceiving when the safety specification (`A[] (not dbServer.serReceiving) or (x <= 4)`) from the running example of [26] is violated. We generate 100 traces with duration 100 and feed these traces to our framework. Our framework repairs the model by introducing two new clocks c_1 and c_2. c_1 is reset on the transition entering serReceiving and c_2 is reset on the transitions leaving serReceiving. Both clocks checked on two new transitions replacing the transition entering error with constraints $x > 4 \wedge c_1 > c_2$ and $x > 4 \wedge c_1 < c_2 \wedge c_1 < 1$. By carrying a simple clock analysis similar to the previous example, one can observe that $c_1 < c_2$ is always satisfied and c_2 can be discarded which leaves us with the new clock c_1 and the new transition to error with constraint $x > 4 \wedge c_1 < 1$. In [26], the model was repaired by introducing the invariant $z < 1$ in serReceiving. For both repairs, the resulting systems satisfy the safety specification. Notice that, although our method does not suggest any invariant repairs, it accurately finds the source of the error in the model and repairs the model with the invariant-free counterpart of the repair procedure of [26].

Second row of Table 2 presents the results of the case study. Proposed framework repairs the model in 2.11s (1.90s for Algorithm 1), increases the number of clocks to five, and this number is reduced to four after running [38].

6.3 Case study: SBR

Our next case study is an NTA implementing three cyclic processes, a processor and a feature deployment machine [26,27]. Due to space limitations, we invite interested reader to [27] for the details of the model. The safety specification of the model is that all processes shall finish their execution before their corresponding deadlines. We instantiated the worst case execution times of each process to ten so that, in their hyper-period (which also has the duration of ten time units), at least one of the processes misses the deadline. To observe a violation of the specification, we converted the safety specification to a non-reachability property by introducing three new locations error1, error2 and error3 only reachable from processor_idle in the processor TA. Each of these error locations corresponds to a deadline miss for one of the three processes. To run our framework on the model, we generate 100 traces with duration 100. Our framework synthesizes three formulae of the form (29) (one formula for each process). Essentially, each process is repaired by introducing two new clocks c_1 and c_2 (six new clocks in total) and the corresponding constraints limit the worst case execution time of each process to two. Since the repairs are identical of all three processes, the total execution time is limited by six, which is less than the duration of the hyper-period. The semantic analysis of the repaired model shows that our framework accurately finds the cause of the error. After the repair, we also check the model against the specification using UPPAAL and observe that no violation occurs. Therefore, we conclude that the proposed framework successfully repairs the model.

SBR is a more complex example than the other case studies presented so far. Moreover, in the experiments of [26], SBR is the only example that reached their two minutes timeout limit for some of the timed diagnostic traces. Third row of Table 2 presents the overall results of this example.

6.4 Case study: nuclear plant model and train model

Finally, we present two more case studies on a nuclear plant model and a train model from the Imitator package [7]. For both examples, we run an experiment setup inspired from mutation testing [2]: we delete a guard or an invariant, if this modification causes a violation of the safety specification, we generate 100 traces with duration 100, run our framework, and finally, we verify the repaired model against the safety specification. The number of the mutated models violating the safety specification is six for the nuclear plant model and four for the train model. Interestingly, in most of the cases, suggested repairs are at the exactly same positions with the deleted ones but their content is different. Another difference is that since our framework does not suggest invariants for repairs, instead of the deleted invariants, the framework suggests guards with less than operators. The same approach is applied to generate invariant-free models in the literature. In spite of the differences between the original and suggested constraints, our framework successfully repairs the model in all cases, i.e., each repaired model satisfies its specification.

Fourth and fifth rows of Table 2 report the average computation time and the maximum number of clocks observed in the repaired models. In average, our framework repairs the models in 4.94s (4.69s for Algorithm 1) and 3.70s (3.42s for Algorithm 1).

The works from [6,26,27] also aim at repairing timed automata. In [6] and [26], bounds from the existing constraints are modified. The method from [26] is extended with additional repair operations including introducing resets, changing comparison operators and clock references in [27]. Our method adds new clocks and introduces new constraints over the new clocks, which allows us to detect errors of the model due to the missing clocks. The approaches relying on modifying existing clocks (resets/constraints) cannot capture such errors. On the other hand, while modification of the bound of an existing clock can be achieved via our approach in various cases (e.g., see Fischer's model), there exist some cases in which this is not possible. For example, if an automaton has transitions $(l', l, \lambda, c < n)$ and $(l'', l, \lambda', c < n)$, then our approach cannot modify only one of the bounds from these transitions. Finally, resetting an existing clock or increasing an upper bound in a constraint can introduce new behaviors, which is not possible with our approach. As summarized here, one approach is not more general than the others. Introducing new clocks and constraints can be advantageous in a variety of cases as illustrated with the examples.

7 Conclusion

We presented an automated system repair framework for cyber-physical systems and showed its use on discrete-time dynamical systems and timed automata. The proposed framework first constructed a dataset of labeled system traces via simulation, identified repairable temporal properties leading the faulty behavior as a ptSTL formula, and finally repaired the system to avoid the satisfaction of the formula. We developed an efficient iterative method to generate a ptSTL formula from a labeled dataset. The case studies illustrated that the proposed STL-based repair framework can successfully repair discrete-time dynamical systems and timed automata. For both modeling formalisms, we defined repairable formulae and the corresponding repair procedures. Applying the repair framework to a new class of system requires defining repairable formulae for the considered system and the corresponding repair procedures, which, in general, are not trivial processes.

Future research directions include expanding the repairable parametric formula sets. For timed automata, we plan to consider clock values, and TA extensions such as discrete variables, which will allow us to apply our framework on a larger set of benchmarks. For dynamical systems, we plan to consider automata-based control strategies. Another research direction is applying the repair framework on Simulink models, which requires defining repairable formulae and the corresponding repair procedures.

References

1. System repair toolbox. https://gitlab.com/MertErgurtuna/system_repair_toolbox

2. Aichernig, B.K., Lorber, F., Ničković, D.: Time for mutants – model-based mutation testing with timed automata. In: Veanes, M., Viganò, L. (eds.) Tests and Proofs, pp. 20–38. Springer, Berlin Heidelberg, Berlin, Heidelberg (2013)
3. Alrajeh, D., Craven, R.: Automated error-detection and repair for compositional software specifications. In: Software Engineering and Formal Methods, pp. 111–127. Springer International Publishing, Cham (2014)
4. Alur, R.: Principles of Cyber-Physical Systems. The MIT Press, Cambridge (2015)
5. Alur, R., Dill, D.L.: A theory of timed automata. Theoret. Comput. Sci. **126**(2), 183–235 (1994)
6. André, É., Arcaini, P., Gargantini, A., Radavelli, M.: Repairing Timed Automata Clock Guards Through Abstraction and Testing. In: International Conference on Tests and Proofs, pp. 129–146. Springer (2019)
7. André, É., Fribourg, L., Kühne, U., Soulat, R.: Imitator 2.5: A tool for analyzing robustness in scheduling problems. In: D. Giannakopoulou, D. Méry (eds.) FM 2012: Formal Methods, pp. 33–36. Springer Berlin Heidelberg, Berlin, Heidelberg (2012)
8. Annpureddy, Y., Liu, C., Fainekos, G., Sankaranarayanan, S.: S-taliro: A tool for temporal logic falsification for hybrid systems. In: Abdulla, P.A., Leino, K.R.M. (eds.) Tools and Algorithms for the Construction and Analysis of Systems, pp. 254–257. Springer, Berlin (2011)
9. Asarin, E., Donze, A., Maler, O., Nickovic, D.: Parametric identification of temporal properties. In: Proceedings of the Second International Conference on Runtime Verification, RV'11, pp. 147–160. Springer, Berlin (2012)
10. Aydin, S.K., Gol, E.A.: Synthesis of monitoring rules with STL. J. Circ. Syst. Comput. **29**(11), 2050177 (2020). https://doi.org/10.1142/S0218126620501777
11. Bartocci, E., Deshmukh, J., Donzé, A., Fainekos, G., Maler, O., Ničković, D., Sankaranarayanan, S.: Specification-Based Monitoring of Cyber-Physical Systems: A Survey on Theory, Tools and Applications, pp. 135–175. Springer Int. Pub., Cambridge (2018)
12. Bartocci, E., Manjunath, N., Mariani, L., Mateis, C., Ničković, D.: Automatic failure explanation in CPS models. In: Software Engineering and Formal Methods, pp. 69–86. Springer International Publishing, Cambridge (2019)
13. Cai, C.H., Sun, J., Dobbie, G.: Automatic B-model repair using model checking and machine learning. Automated Software Engineering **26**, (2019). https://doi.org/10.1007/s10515-019-00264-4
14. Coogan, S., Gol, E.A., Arcak, M., Belta, C.: Traffic network control from temporal logic specifications. IEEE Trans. Control Netw. Syst. **3**(2), 162–172 (2016)
15. David, A., Larsen, K.G., Legay, A., Mikuăionis, M., Poulsen, D.B.: Uppaal SMC tutorial. Int. J. Softw. Tools Technol. Transf. **17**(4), 397–415 (2015)
16. Donze, A.: On signal temporal logic. In: Legay, A., Bensalem, S. (eds.) RV 2013, LNCS 8174, pp. 382–383. Springer, Berlin (2013)
17. Ergurtuna, M., Gol, E.A.: An efficient formula synthesis method with past signal temporal logic. IFAC-PapersOnLine **52**(11), 43 – 48 (2019). https://doi.org/10.1016/j.ifacol.2019.09.116. 5th IFAC Conference on Intelligent Control and Automation Sciences ICONS 2019
18. Ernst, G., Arcaini, P., Donze, A., Fainekos, G., Mathesen, L., Pedrielli, G., Yaghoubi, S., Yamagata, Y., Zhang, Z.: Arch-comp 2019 category report: Falsification. In: ARCH19. 6th International Workshop on Applied Verification of Continuous and Hybrid Systems, *EPiC Series in Computing*, vol. 61, pp. 129–140. EasyChair (2019). https://doi.org/10.29007/68dk
19. Fainekos, G.E., Pappas, G.J.: Robustness of temporal logic specifications for continuous-time signals. Theoret. Comput. Sci. **410**(42), 4262–4291 (2009). https://doi.org/10.1016/j.tcs.2009.06.021
20. Ferrère, T., Maler, O., Ničković, D.: Trace diagnostics using temporal implicants. In: Automated Technology for Verification and Analysis, pp. 241–258. Springer International Publishing, Cambridge (2015)
21. Gazzola, L., Micucci, D., Mariani, L.: Automatic software repair: A survey. IEEE Trans. Software Eng. **45**(1), 34–67 (2019)
22. Guha, S., Narayan, C., Arun-Kumar, S.: Reducing clocks in timed automata while preserving bisimulation. In: International Conference on Concurrency Theory, pp. 527–543. Springer, Berlin (2014)
23. Hune, T., Romijn, J., Stoelinga, M., Vaandrager, F.: Linear parametric model checking of timed automata. In: Margaria, T., Yi, W. (eds.) Tools and Algorithms for the Construction and Analysis of Systems, pp. 189–203. Springer, Berlin (2001)
24. Jha, S., Tiwari, A., Seshia, S.A., Sahai, T., Shankar, N.: Telex: learning signal temporal logic from positive examples using tightness metric. Form. Methods Syst. Des. **54**, 364–387 (2019)
25. Jin, X., Donze, A., Deshmukh, J.V., Seshia, S.A.: Mining requirements from closed-loop control models. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **34**(11), 1704–1717 (2015)
26. Kölbl, M., Leue, S., Wies, T.: Clock bound repair for timed systems. In: International Conference on Computer Aided Verification, pp. 79–96. Springer, Berlin (2019)

27. Kölbl, M., Leue, S., Wies, T.: Tartar: A timed automata repair tool. In: Lahiri, S.K., Wang, C. (eds.) Computer Aided Verification, pp. 529–540. Springer International Publishing, Cham (2020)
28. Liu, B., Lucia, Nejati, S., Briand, L.C., Bruckmann, T.: Simulink fault localization: an iterative statistical debugging approach. Softw. Test. Verif. Reliab. **26**(6), 431–459 (2016). https://doi.org/10.1002/stvr.1605
29. Mark Utting, B.L.: Practical Model-Based Testing: A Tools Approach. Morgan Kaufmann, Burlington (2006)
30. MATLAB: version (R2016b). The MathWorks Inc., Natick, Massachusetts (2016)
31. Mohammadinejad, S., Deshmukh, J.V., Puranic, A.G., Vazquez-Chanlatte, M., Donzé, A.: Interpretable classification of time-series data using efficient enumerative techniques. In: Proceedings of the 23rd International Conference on Hybrid Systems: Computation and Control, HSCC '20. ACM, New York, NY, USA (2020). https://doi.org/10.1145/3365365.3382218
32. Nguyen, H.D.T., Qi, D., Roychoudhury, A., Chandra, S.: Semfix: Program repair via semantic analysis. In: Proceedings of the 2013 International Conference on Software Engineering, ICSE '13, pp. 772–781. IEEE Press (2013)
33. Raman, V., Donze, A., Sadigh, D., Murray, R.M., Seshia, S.A.: Reactive synthesis from signal temporal logic specifications. In: Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control, HSCC '15, pp. 239–248. ACM, New York, NY, USA (2015)
34. Saglam, I., Gol, E.A.: Cause mining and controller synthesis with STL. In: 58th IEEE Conference on Decision and Control (CDC), pp. 4589–4594 (2019)
35. Singh, N.K., Saha, I.: Specification-guided automated debugging of CPS models. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **39**(11), 4142–4153 (2020). https://doi.org/10.1109/TCAD.2020.3012862
36. Vazquez-Chanlatte, M., Deshmukh, J.V., Jin, X., Seshia, S.A.: Logical clustering and learning for time-series data. In: Majumdar, R., Kunčak, V. (eds.) Computer Aided Verification, pp. 305–325. Springer International Publishing, Cambridge (2017)
37. Weimer, W., Nguyen, T., Le Goues, C., Forrest, S.: Automatically finding patches using genetic programming. In: Proceedings of the 31st International Conference on Software Engineering, ICSE '09, p. 364–374. IEEE Computer Society, USA (2009). https://doi.org/10.1109/ICSE.2009.5070536
38. Yalcinkaya, B., Gol, E.A.: Clock reduction in timed automata while preserving design parameters. In: 2019 IEEE/ACM 7th International Conference on Formal Methods in Software Engineering (FormaliSE), pp. 31–40. IEEE (2019)
39. Yamagata, Y., Liu, S., Akazaki, T., Duan, Y., Hao, J.: Falsification of cyber-physical systems using deep reinforcement learning. IEEE Transactions on Software Engineering pp. 1–1 (2020)
40. Yamaguchi, T., Hoxha, B., Prokhorov, D., Deshmukh, J.V.: Specification-guided software fault localization for autonomous mobile systems. In: 18th ACM-IEEE International Conference on Formal Methods and Models for System Design (MEMOCODE), pp. 1–12 (2020). https://doi.org/10.1109/MEMOCODE51338.2020.9315067