Causality Monitoring for MIMOS

Chengzi Huang[1], Behnam Khodabandeloo[1], Duc Anh Nguyen[1], Wang Yi[1](✉), Jie An[2], and Zhenya Zhang[3]

[1] Uppsala University, Uppsala, Sweden
{chengzi.huang,behnam.khodabandeloo,ducanh.nguyen,yi}@it.uu.se
[2] Institute of Software, Chinese Academy of Sciences, Beijing, China
anjie@iscas.ac.cn
[3] Kyushu University, Fukuoka, Japan
zhang@ait.kyushu-u.ac.jp

Abstract. MIMOS is a tool environment developed based on an asynchronous and deterministic data flow model for the design and evolution of real-time systems, aiming at safety-critical applications. Currently MIMOS supports modeling, simulation, verification, scheduling, timing analysis and (multi-core) code generation as well as dynamic updates of systems after deployment. The asynchronous yet deterministic design paradigm supported by MIMOS enables developers to build robust applications that can evolve without compromising their critical safety properties. This paper presents the run-time monitoring feature of MIMOS, with a focus on the causality of specifications in Signal Temporal Logic (STL). A causality monitor of MIMOS, determines and visualizes not only whether an executing trace violates a given STL specification but also how relevant the incremental changes of the trace is to the violation, with quantitative information providing insights into the system's behavior. We present a case study to illustrate the power and usefulness of causality monitoring in MIMOS.

Keywords: MIMOS · real-time systems · monitoring · causality

1 Introduction

MIMOS is a tool environment designed for the development and evolution of embedded real-time systems, specifically tailored for safety-critical applications. It provides a platform for modeling, simulation, verification, scheduling, timing analysis and multi-core code generation as well as dynamic system updates. Unlike traditional synchronous design tools, MIMOS utilizes an asynchronous yet deterministic data-flow design model to facilitate incremental system modifications and dynamic updates after deployment without compromising safety guarantees.

With this work, we congratulate Christel on her outstanding scientific achievements and thank her for inspiring so many in our community. Wang would also like to thank Christel for her many years of friendship.

In this paper, we present the *monitoring* feature of MIMOS. Online monitoring (a.k.a. runtime verification) [1] is an effective approach for checking whether a trace of system execution satisfies a given specification at runtime. By extending Temporal Logic [10] with real-time constraints and real-valued predicates, Signal Temporal Logic (STL) [9] has been widely utilized to specify properties of real-time systems. In STL monitoring, an online monitor analyzes partial execution traces at runtime, evaluating the satisfaction of an STL formula φ based on the partial trace observed up to each time instant. Typically, the monitor reports results according to the STL robust semantics. However, since the trace is incomplete, a standard online monitor—such as the *online robust monitor* described in [3]—outputs a robustness interval. This interval represents the range of possible robustness values that could be achieved under any future suffix trace. From this interval, the satisfaction of φ can be inferred, for example, φ is violated if the upper bound of the robustness is negative, as Robustness values represent how much the given trace satisfies or violates the STL specification.

A main limitation arises from the definition of robust semantics is that the upper and lower bounds are monotonically decreasing and increasing; this has the consequence that the robustness interval at a given step is "masked" by the history of previous robustness intervals, and, e.g., it is not possible to detect mitigation of the violation severity. This issue has been acknowledged in the literature [11,15] as a significant drawback of these monitoring approaches. To address the problem of information masking, Zhang et al. [13,14] recently introduced the *causality monitoring* of STL. Instead of relying directly on robustness, it considers the causality of violation or satisfaction. This approach not only determines whether an executing trace violates the specification but also evaluates how relevant each incremental update of the trace is to the violation. By doing so, causality monitoring avoids monotonicity and provides richer insights into the system's behavior. It can identify the specific time intervals relevant to the violation or satisfaction of the specification and thus even count the number of violations, a feature particularly valuable for system engineers in practical applications. Furthermore, the causality monitor serves as a refinement of the classic robust monitor, meaning the latter can be straightforwardly derived from the former.

We shall introduce the causality monitoring of STL to the MIMOS platform as one of the tool features, and illustrate the power and usefulness of causality monitor through a case study using MIMOS.

The paper is organized as follows: Sect. 2 recalls the MIMOS model and tool features. Section 3 introduces Signal Temporal Logic, including syntax, robust semantics, online robust monitoring, and causality monitoring of STL. Section 4 presents a case study of an autopilot control system for marine vessels, demonstrating how causality monitoring can detect and localize specification violations. Finally, Sect. 5 concludes the paper and discusses potential future directions for MIMOS and causality monitoring in safety-critical applications.

2 An Overview of MIMOS

This section presents an overview of the semantic of MIMOS [12], and the tool chain developed based on the model.

2.1 MIMOS Model

MIMOS extends the Kahn Process Network (KPN) model [7] by integrating timing semantics, achieved through modeling its processes as periodic real-time tasks. This enables MIMOS to represent embedded systems with precise timing requirements while ensuring deterministic behavior, guaranteeing that for any given set of timed input streams, the model produces a unique set of timed output streams.

A MIMOS system may comprise a collection of process networks. Each network with the system can be represented as a simple directed graph $G = (N, L)$, where

- Nodes (N): A set of real-time tasks. Each task N_i is characterized by:
  - A local state and a set of input and output ports.
  - A function mapping inputs and its current state to outputs.
  - A release pattern dictating its activation pattern.
  - A relative deadline D_i, specifying its latest allowable completion time after release.
- Channels (L): A set of directed edges, where each channel $L_{i,j}$ connects node N_i to node N_j, enabling inter-task communication. Two types of data exchange mechanisms are supported:
  - FIFO channels: Provide ordered buffering, ensuring tokens are retrieved in a first-in-first-out manner.
  - Register channels: Used for sampling time-dependent data sources (like sensors). They store only the most recently written value, overwriting previous ones.
- Two sets of nonnegative integers, W and R, which define the token flow between nodes:
  - $W_{i,j}$ specifies the number of tokens produced by node N_i and added to channel $L_{i,j}$ at the end of each (absolute) deadline.
  - $R_{i,j}$ specifies the number of tokens consumed from channel $L_{i,j}$ when node N_j starts execution.

A node N_i is eligible to start execution at its activation time if and only if all its incoming channels $L_{j,i}$ satisfy their specific enablement rules:

- FIFO Channels (Blocking Mode): If an incoming FIFO channel $L_{j,i}$ is configured in blocking mode, then this channel is enabled if it stores at least $R_{j,i}$. Upon activation, exactly $R_{j,i}$ tokens are removed from this channel.
- FIFO Channels (Non-Blocking Mode – Upto-N): If an incoming FIFO channel $L_{j,i}$ operates in "Upto-N" mode, then this channel is always enabled. At runtime, the task may consume anywhere between 0 and $R_{j,i}$ tokens from this channel based on its availability.

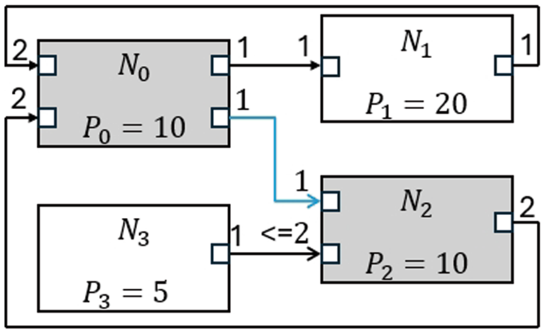


Fig. 1. An example for MIMOS

– Register Channels: If an incoming channel $L_{j,i}$ is a register channel, this channel is always enabled. N_i can always read from it at the start of execution, retrieving the latest value written by its predecessor N_j. Unlike FIFO channels, reading from a register channel does not consume tokens, and the stored value remains available for subsequent activations.

Once a task N_i is released, if it meets the activation conditions at its release time, it can execute, consuming tokens from its incoming channels according to their respective rules. However, if the activation conditions are not met, the task remains inactive and does not execute in this cycle, waiting until its next release time for another activation opportunity.

After execution begins, the task runs until completion, and at the end of its absolute deadline, it produces tokens for each outgoing channel, making them available for downstream tasks. Additionally, the MIMOS model employs read-after-write semantics for scenarios where read and write operations on the same channel logically occur simultaneously. Under these semantics, the read operation always takes place after the write operation, ensuring that the most recently written token is consistently obtained.

In this paper, we assume that each node N_i follows the release pattern of *implicit-deadline periodic tasks* [8], where P_i denotes the minimum separation between consecutive activations of N_i, and $D_i = P_i$. This assumption ensures that every task is activated periodically and must complete its execution within the same period before the next activation occurs.

Example 1. To illustrate the MIMOS model, Fig. 1 presents an example where black arrows represent FIFO channels and the blue arrow represents a register channel. In this example, node N_1 has a period of 20 time units and adds one token to channel $L_{1,0}$ at the end of each period. Meanwhile, node N_0, which has a period of 10 time units, requires at least two tokens from channel $L_{2,0}$ to begin execution and, once it completes execution, produces one token to channel $L_{0,2}$. Additionally, the notation "≤ 2" indicates that the corresponding incoming channel $L_{3,2}$ of node N_2 operates in "Upto-N" mode, meaning that this channel is always enabled. N_2 may consume between 0 and 2 tokens, depending on the number of tokens present at the time of its release.

2.2 Signals and Timed Data Streams

While the MIMOS model introduced earlier emphasizes discrete, token-based interactions at periodic task release events, it can also be beneficial to conceptualize channel states as evolving continuously over time. Adopting this continuous-time perspective naturally introduces the concepts of *signals* and *timed data streams*, which provide complementary ways of representing real-time behavior.

Definition 1 (Signal). Given a positive real number $T \in \mathbb{R}> 0$, referred to as the *time horizon*, and a positive integer $d \in \mathbb{N}> 0$ denoting the dimension, a *signal* is defined as a function:

$$\mathbf{v} : [0, T] \to \mathbb{R}^d$$

For each time $t \in [0, T]$, $\mathbf{v}(t)$ provides a d-dimensional vector describing the instantaneous state of a channel. If the condition $\mathbf{v}(t) \in \Omega \subseteq \mathbb{R}^d$ holds for every $t \in [0, T]$, the signal $\mathbf{v}$ is said to be *spatially bounded* by the region Ω.

Definition 2 (Timed Data Stream). A *timed data stream* is a finite or infinite sequence of timestamped data points:

$$\langle (x_1, t_1), (x_2, t_2), \ldots \rangle,$$

where each data item $x_i \in \mathbb{R}^d$ is paired with a timestamp $t_i \in [0, T]$. The sequence of timestamps is non-decreasing, satisfying $t_1 \leq t_2 \leq \ldots$, and typically remains bounded within the global time horizon T.

In many real-time modeling frameworks, signals and timed data streams serve as interchangeable abstractions to represent system behavior. Specifically, a continuous signal $\mathbf{v}(t)$ can be sampled at discrete time points $t_1 \leq t_2 \leq \ldots$ to yield a corresponding timed data stream:

$$\langle (\mathbf{v}(t_1), t_1), (\mathbf{v}(t_2), t_2), \ldots \rangle.$$

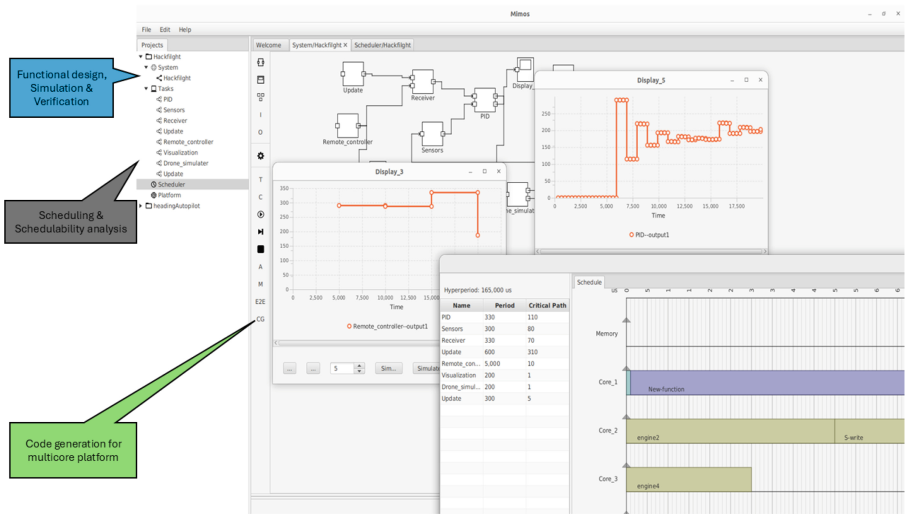
Conversely, a timed data stream can be converted into a continuous-time signal through suitable interpolation or by holding the last sampled value constant between timestamps.

From this signal-oriented viewpoint, each MIMOS channel $L_{i,j}$ is naturally described by a function

$$\gamma_{i,j} : [0, T] \to \mathbb{R}^d$$

that characterizes the evolution of the channel state or token count continuously over the interval $[0, T]$. Traditional discrete operations, such as token reads and writes, become operations of sampling from or updating the function $\gamma_{i,j}$. This continuous-time interpretation, bolstered by our assumption of *holding the last* *sampled value constant*, enhances the analytical capabilities and simulation accuracy of the MIMOS model, providing a cohesive framework that unifies discrete token interactions and continuous-time semantics.

2.3 MIMOS Tool Features



Hyperperiod: 165,000 us

Name	Period	Critical Path
PID	330	110
Sensors	300	80
Receiver	330	70
Update	600	310
Remote_con...	5,000	10
Visualization	200	1
Drone_simul...	200	1
Update	300	5

Fig. 2. Main interface of the MIMOS tool.

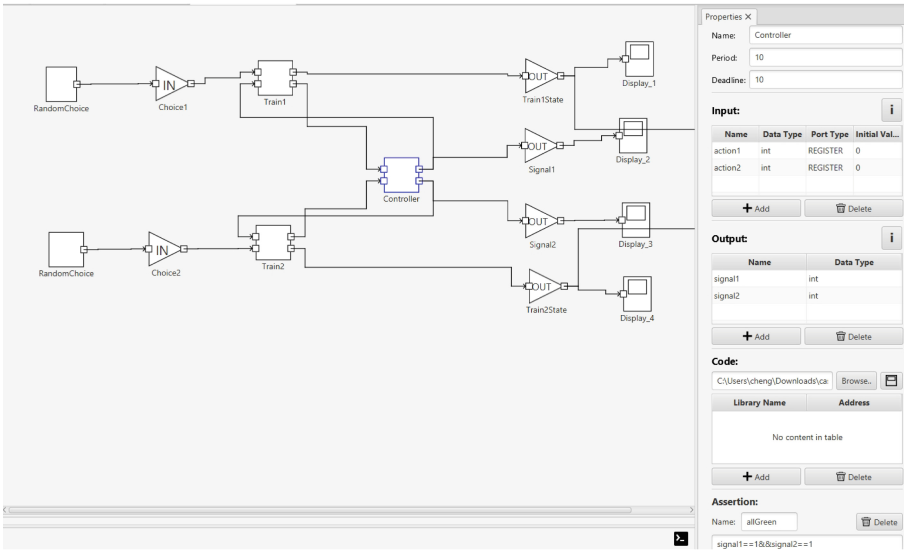
This subsection outlines the key features, architecture, and core components of the MIMOS tool. Figure 2 illustrates the primary user interface, highlighting the functionality and user interactions supported by the tool.

The design and analysis capabilities of MIMOS are structured into three distinct layers:

– *Functional Layer*: Provides a graphical interface for designing MIMOS models as networks of computational nodes, each performing periodic computations defined by user-specified periods and deadlines. This platform-independent layer supports for modeling of functional behaviors, functional simulation and verification, and also monitoring of time-variant properties.
– *Software Layer*: Abstracts the functional designs into real-time tasks represented as periodic Directed Acyclic Graphs (DAGs), supporting rigorous schedulability and timing analysis, including estimation of end-to-end latency bounds.
– *Hardware Layer*: Allows precise specification of multi-core processor platforms, including details such as cores, caches, and memory. Integration with the Gem5 simulator enables Worst-Case Execution Time (WCET) analysis through cycle-accurate simulation.

Currently the tool supports the following features:

- **Function Editor (see Fig. 3)** Provides a graphical interface to model a system as a network of nodes connected via unidirectional channels, following the MIMOS model (described in Sect. 2). Each node represents a periodic real-time task characterized by input/output ports and mapping functions. Users can configure node parameters, define mapping functions in supported languages (C/C++, Java, and Python), and set assertions on ports for verification.



Name: Controller
Period: 10
Deadline: 10

Input:

Name	Data Type	Port Type	Initial Val...
action1	int	REGISTER	0
action2	int	REGISTER	0

Output:

Name	Data Type
signal1	int
signal2	int

Code:

C:\Users\cheng\Downloads\ca:

Library Name	Address
No content in table	

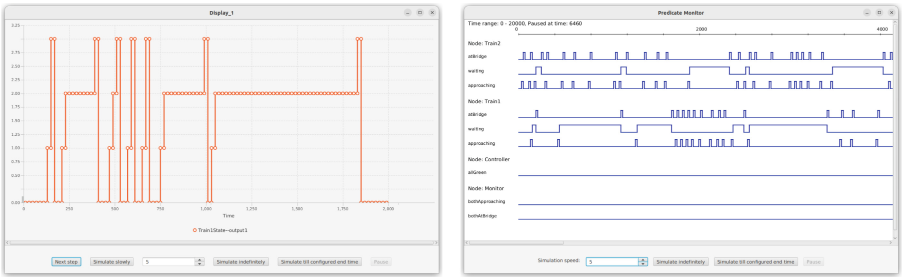
Assertion:

Name: allGreen

signal1==1&&signal2==1

Fig. 3. System modeling in MIMOS.

- **Discrete Event Simulator (Fig. 4a)**: Dynamically visualizes execution behaviors and timing properties. Simulation traces can be displayed step-by-step, continuously up to a predefined step, or indefinitely. Visualization speed is adjustable on a scale from 1 to 10. This simulator also includes monitoring functionalities to track and validate system conditions at each signal (see Fig. 4b).
- **Analyzer (Fig. 5)** Checks whether the tasks associated with a system model satisfy their timing requirements. The analysis suite includes schedulability analysis under fixed priority, earliest deadline first (EDF) scheduling, and partitioned or semi-partitioned scheduling strategies. Execution can also be visualized as Gantt charts.
- **Code Generator**: The Multi-core Code Generator in MIMOS enables the automatic generation of executable C code from system models, supporting multi-core deployment. It translates the system model into a set of communicating tasks connected via one-to-one channels, ensuring efficient execution on

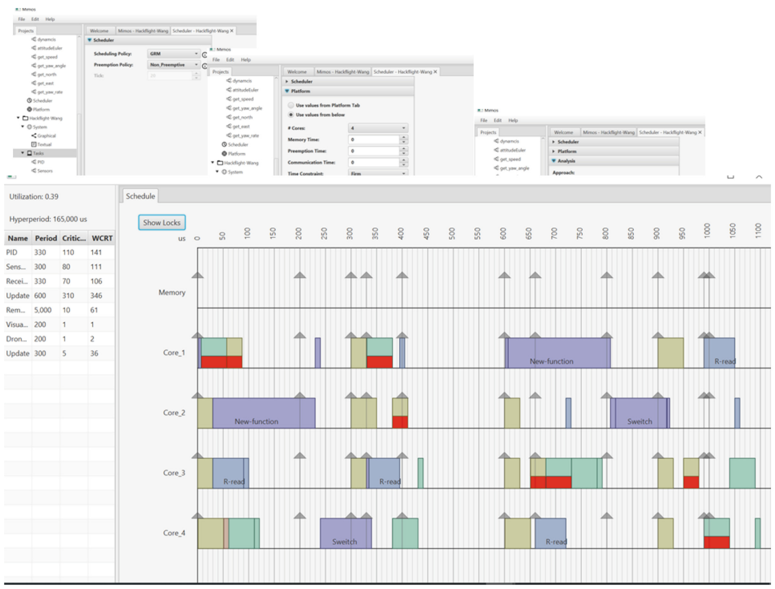


(a) Signal Value Trace in MIMOS. (b) Boolean State Monitoring in MIMOS.

Fig. 4. Discrete event simulation in MIMOS

multi-core platforms. The compiler applies deterministic semantic refinement, guaranteeing that the generated code preserves both functional correctness and timing constraints when deployed on the target architecture.

In the following section, we present a framework for monitoring the time-variant properties of the input and output signals and timed streams of an MIMOS model.



Scheduling Policy: GRM
Preemption Policy: Non_Preemptive

Use values from below
Cores: 4
Memory Time: 0
Preemption Time: 0
Communication Time: 0
Time Constraint: Firm

Utilization: 0.39

Hyperperiod: 165,000 us

Name	Period	Critic...	WCRT
PID	330	110	141
Sens...	300	80	111
Recei...	330	70	106
Update	600	310	346
Rem...	5,000	10	61
Visua...	200	1	1
Dron...	200	1	2
Update	300	5	36

Fig. 5. Schedulability analysis in MIMOS.

3 Causality Monitoring for STL Specifications

We shall use the *Signal temporal logic (STL)* [9] to specify desired time-variant properties over MIMOS signals and timed data streams.

3.1 Signal Temporal Logic

In this section, we review the syntax and robust semantics [4,5] of STL.

Definition 3 (STL syntax). In STL, the *atomic propositions* α and the *formulas* φ are respectively defined as follows:

$$\alpha ::\equiv f(w_1, \ldots, w_K) > 0 \qquad \varphi ::\equiv \alpha \mid \bot \mid \neg\varphi \mid \varphi \wedge \varphi \mid \Box_I \varphi \mid \Diamond_I \varphi \mid \varphi\, \mathcal{U}_I\, \varphi$$

Here f is a K-ary function $f : \mathbb{R}^K \to \mathbb{R}$, $w_1, \ldots, w_K \in \mathbf{Var}$, and I is a closed non-singular interval in $\mathbb{R}_{\geq 0}$, i.e., $I = [l, u]$, where $l, u \in \mathbb{R}$ and $l < u$. $\Box, \Diamond$ and $\mathcal{U}$ are temporal operators, which are known as *always*, *eventually* and *until*, respectively. The always operator $\Box$ and eventually operator $\Diamond$ are two special cases of the until operator $\mathcal{U}$, where $\Diamond_I \varphi \equiv \top\, \mathcal{U}_I\, \varphi$ and $\Box_I \varphi \equiv \neg\Diamond_I \neg\varphi$. Other common connectives such as $\vee, \to$ are introduced as syntactic sugar: $\varphi_1 \vee \varphi_2 \equiv \neg(\neg\varphi_1 \wedge \neg\varphi_2)$, $\varphi_1 \to \varphi_2 \equiv \neg\varphi_1 \vee \varphi_2$.

Definition 4 (STL robust semantics). Let $\mathbf{v}$ be a signal, φ be an STL formula and $\tau \in \mathbb{R}_+$ be an instant. The *robustness* $\mathrm{R}(\mathbf{v}, \varphi, \tau) \in \mathbb{R} \cup \{+\infty, -\infty\}$ of $\mathbf{v}$ w.r.t. φ at τ is defined by induction on the construction of formulas, as follows,

$$\begin{aligned}
&\mathrm{R}(\mathbf{v}, \alpha, \tau) \;:=\; f(\mathbf{v}(\tau)) \qquad \mathrm{R}(\mathbf{v}, \neg\varphi, \tau) \;:=\; -\mathrm{R}(\mathbf{v}, \varphi, \tau)\\
&\mathrm{R}(\mathbf{v}, \varphi_1 \wedge \varphi_2, \tau) \;:=\; \min\left(\mathrm{R}(\mathbf{v}, \varphi_1, \tau), \mathrm{R}(\mathbf{v}, \varphi_2, \tau)\right)\\
&\mathrm{R}(\mathbf{v}, \Box_I \varphi, \tau) \;:=\; \inf_{t \in \tau + I} \mathrm{R}(\mathbf{v}, \varphi, t) \qquad \mathrm{R}(\mathbf{v}, \Diamond_I \varphi, \tau) \;:=\; \sup_{t \in \tau + I} \mathrm{R}(\mathbf{v}, \varphi, t)\\
&\mathrm{R}(\mathbf{v}, \varphi_1\, \mathcal{U}_I\, \varphi_2, \tau) \;:=\; \sup_{t \in \tau + I} \min\left(\mathrm{R}(\mathbf{v}, \varphi_2, t), \inf_{t' \in [\tau, t)} \mathrm{R}(\mathbf{v}, \varphi_1, t')\right)
\end{aligned}$$

where $\tau + [l, u]$ denotes the shifted interval $[l + \tau, u + \tau]$.

The Boolean semantics of STL, which determines whether $(\mathbf{v}, \tau) \models \varphi$, can be derived from the quantitative robust semantics defined in Definition 4. Specifically, if $\mathrm{R}(\mathbf{v}, \varphi, \tau) > 0$, then $(\mathbf{v}, \tau) \models \varphi$; conversely, if $\mathrm{R}(\mathbf{v}, \varphi, \tau) < 0$, then $(\mathbf{v}, \tau) \not\models \varphi$.

3.2 Online Robust Monitoring of STL

Online monitoring concerns the satisfaction of a *partial signal* $\mathbf{v}_{0:b} : [0, b] \to \mathbb{R}^d$ w.r.t. an STL formula φ. We define a *completion* of $\mathbf{v}_{0:b}$ as a signal $\mathbf{v} : [0, T] \to \mathbb{R}^d$ $(b \leq T)$ such that $\forall t \in [0, b], \mathbf{v}(t) = \mathbf{v}_{0:b}(t)$. A completion $\mathbf{v}$ can be written as the concatenation of $\mathbf{v}_{0:b}$ with a *suffix signal* $\mathbf{v}_{b:T}$, i.e., $\mathbf{v} = \mathbf{v}_{0:b} \cdot \mathbf{v}_{b:T}$.

Definition 5 (Online robust monitor [3]). Let $\mathbf{v}_{0:b}$ be a partial signal, and let φ be an STL formula. We denote by $\mathtt{R}^{\alpha}_{\max}$ and $\mathtt{R}^{\alpha}_{\min}$ the possible *maximum* and *minimum bounds* of the robustness $\mathrm{R}(\mathbf{v}, \alpha, \tau)$[1]. Then, an *online robust monitor* returns a sub-interval $[\mathrm{R}](\mathbf{v}_{0:b}, \varphi, \tau) \subseteq [\mathtt{R}^{\alpha}_{\min}, \mathtt{R}^{\alpha}_{\max}]$ at instant b, which is defined as follows, by induction on the construction of formulas.

$$[\mathrm{R}](\mathbf{v}_{0:b}, \alpha, \tau) := \begin{cases} \left[f\left(\mathbf{v}_{0:b}(\tau)\right), f\left(\mathbf{v}_{0:b}(\tau)\right)\right] & \text{if } \tau \in [0, b] \\ \left[\mathtt{R}^{\alpha}_{\min}, \mathtt{R}^{\alpha}_{\max}\right] & \text{otherwise} \end{cases}$$

$$[\mathrm{R}](\mathbf{v}_{0:b}, \neg\varphi, \tau) := -[\mathrm{R}](\mathbf{v}_{0:b}, \varphi, \tau)$$

$$[\mathrm{R}](\mathbf{v}_{0:b}, \varphi_1 \wedge \varphi_2, \tau) := \min\Big([\mathrm{R}](\mathbf{v}_{0:b}, \varphi_1, \tau), [\mathrm{R}](\mathbf{v}_{0:b}, \varphi_2, \tau)\Big)$$

$$[\mathrm{R}](\mathbf{v}_{0:b}, \Box_I \varphi, \tau) := \inf_{t \in \tau + I}\Big([\mathrm{R}](\mathbf{v}_{0:b}, \varphi, t)\Big)$$

$$[\mathrm{R}](\mathbf{v}_{0:b}, \varphi_1 \,\mathcal{U}_I\, \varphi_2, \tau) := \sup_{t \in \tau + I} \min\big([\mathrm{R}](\mathbf{v}_{0:b}, \varphi_2, t), \inf_{t' \in [\tau, t)} [\mathrm{R}](\mathbf{v}_{0:b}, \varphi_1, t')\big)$$

Here, f is defined as in Definition 3, and the arithmetic rules over intervals $I = [l, u]$ are defined as follows: $-I := [-u, -l]$ and $\min(I_1, I_2) := [\min(l_1, l_2), \min(u_1, u_2)]$.

We denote by $[\mathrm{R}]^{\mathsf{U}}(\mathbf{v}_{0:b}, \varphi, \tau)$ and $[\mathrm{R}]^{\mathsf{L}}(\mathbf{v}_{0:b}, \varphi, \tau)$ the upper and lower bounds of $[\mathrm{R}](\mathbf{v}_{0:b}, \varphi, \tau)$ respectively. Intuitively, this interval $[\mathrm{R}](\mathbf{v}_{0:b}, \varphi, \tau)$ represents the range of possible robustness values that can be reached by the completion of $\mathbf{v}_{0:b}$ with any suffix signal $\mathbf{v}_{b:T}$. This interval enables a 3-valued verdict for a given $\mathbf{v}_{0:b}$ in relation to the specification φ: if $[\mathrm{R}]^{\mathsf{L}}(\mathbf{v}_{0:b}, \varphi, \tau) > 0$, it implies that $\mathbf{v}_{0:b}$ satisfies φ (thus returning the verdict of `true`); if $[\mathrm{R}]^{\mathsf{U}}(\mathbf{v}_{0:b}, \varphi, \tau) < 0$, it implies $\mathbf{v}_{0:b}$ violates φ (thus returning the verdict of `false`); otherwise, it does not imply either of the cases, and so it returns `unknown`.

3.3 Overview of Causality Monitoring

The information masking issue of robust monitors (as described in Definition 5) has been identified as a serious problem in [11,13,15]. This problem stems from the inherent *monotonicity* of robust monitors, i.e., as the signal $\mathbf{v}_{0:b}$ evolves, $[\mathrm{R}]^{\mathsf{U}}(\mathbf{v}_{0:b}, \varphi, \tau)$ monotonically decreases and $[\mathrm{R}]^{\mathsf{L}}(\mathbf{v}_{0:b}, \varphi, \tau)$ monotonically increases. See [13] for a formal description of the problem.

Online Causality Monitoring. Causality monitoring is proposed in [13] to address the information masking issue. Specifically, instead of monitoring robustness that indicates whether a partial trace violates the specification, this approach assesses whether each incremental update to the trace can serve as a *cause* of the violation of the specification. To determine whether an update of a signal at a moment is a cause, we follow the *trace diagnostics* technique [2,15] that returns a (violation or satisfaction) *epoch*, i.e., a set of signal segments that *sufficiently* contribute to either the violation or satisfaction of the specification. Intuitively,

[1] $\mathrm{R}(\mathbf{v}, \alpha, \tau)$ is bounded because of the bound Ω of $\mathbf{v}$. In practice, if Ω is unknown, we just need to set $\mathtt{R}^{\alpha}_{\max}$ and $\mathtt{R}^{\alpha}_{\min}$ to be ∞ and $-\infty$ respectively.

Table 1. The definitions of violation and satisfaction causation distances

$$[\mathscr{R}]^{\ominus}(\mathbf{v}_{0:b},\alpha,\tau) := \begin{cases} f(\mathbf{v}_{0:b}(\tau)) & \text{if } b=\tau \\ \mathtt{R}^{\alpha}_{\max} & \text{otherwise} \end{cases} \qquad [\mathscr{R}]^{\ominus}(\mathbf{v}_{0:b},\neg\varphi,\tau) := -[\mathscr{R}]^{\oplus}(\mathbf{v}_{0:b},\varphi,\tau)$$

$$[\mathscr{R}]^{\ominus}(\mathbf{v}_{0:b},\varphi_1\wedge\varphi_2,\tau) := \min\left([\mathscr{R}]^{\ominus}(\mathbf{v}_{0:b},\varphi_1,\tau), [\mathscr{R}]^{\ominus}(\mathbf{v}_{0:b},\varphi_2,\tau)\right)$$

$$[\mathscr{R}]^{\ominus}(\mathbf{v}_{0:b},\Box_I\varphi,\tau) \ := \ \inf_{t\in\tau+I}\left([\mathscr{R}]^{\ominus}(\mathbf{v}_{0:b},\varphi,t)\right)$$

$$[\mathscr{R}]^{\ominus}(\mathbf{v}_{0:b},\varphi_1\,\mathcal{U}_I\,\varphi_2,\tau) \ := \ \inf_{t\in\tau+I}\left(\max\left(\begin{array}{l}\min\left(\begin{array}{l}\inf_{t'\in[\tau,t)}[\mathscr{R}]^{\ominus}(\mathbf{v}_{0:b},\varphi_1,t') \\ [\mathscr{R}]^{\ominus}(\mathbf{v}_{0:b},\varphi_2,t)\end{array}\right) \\ [\mathrm{R}]^{\mathsf{U}}(\mathbf{v}_{0:b},\varphi_1\,\mathcal{U}_I\,\varphi_2,\tau)\end{array}\right)\right)$$

$$[\mathscr{R}]^{\oplus}(\mathbf{v}_{0:b},\alpha,\tau) := \begin{cases} f(\mathbf{v}_{0:b}(\tau)) & \text{if } b=\tau \\ \mathtt{R}^{\alpha}_{\min} & \text{otherwise} \end{cases} \qquad [\mathscr{R}]^{\oplus}(\mathbf{v}_{0:b},\neg\varphi,\tau) := -[\mathscr{R}]^{\ominus}(\mathbf{v}_{0:b},\varphi,\tau)$$

$$[\mathscr{R}]^{\oplus}(\mathbf{v}_{0:b},\varphi_1\wedge\varphi_2,\tau) := \max\left(\begin{array}{l}\min\left([\mathscr{R}]^{\oplus}(\mathbf{v}_{0:b},\varphi_1,\tau), [\mathrm{R}]^{\mathsf{L}}(\mathbf{v}_{0:b},\varphi_2,\tau)\right) \\ \min\left([\mathrm{R}]^{\mathsf{L}}(\mathbf{v}_{0:b},\varphi_1,\tau), [\mathscr{R}]^{\oplus}(\mathbf{v}_{0:b},\varphi_2,\tau)\right)\end{array}\right)$$

$$[\mathscr{R}]^{\oplus}(\mathbf{v}_{0:b},\Box_I\varphi,\tau) \ := \ \sup_{t\in\tau+I}\left(\min\left([\mathscr{R}]^{\oplus}(\mathbf{v}_{0:b},\varphi,t), [\mathrm{R}]^{\mathsf{L}}(\mathbf{v}_{0:b},\Box_I\varphi,\tau)\right)\right)$$

$$[\mathscr{R}]^{\oplus}(\mathbf{v}_{0:b},\varphi_1\,\mathcal{U}_I\,\varphi_2,\tau) := \sup_{t\in\tau+I}\left(\max\left(\begin{array}{l}\min\left(\begin{array}{l}\sup_{t'\in[\tau,t)}[\mathscr{R}]^{\oplus}(\mathbf{v}_{0:b},\varphi_1,t') \\ \inf_{t'\in[\tau,t)}[\mathrm{R}]^{\mathsf{L}}(\mathbf{v}_{0:b},\varphi_1,t') \\ [\mathrm{R}]^{\mathsf{L}}(\mathbf{v}_{0:b},\varphi_2,t)\end{array}\right) \\ \min\left(\begin{array}{l}\inf_{t'\in[\tau,t)}[\mathrm{R}]^{\mathsf{L}}(\mathbf{v}_{0:b},\varphi_1,t') \\ [\mathscr{R}]^{\oplus}(\mathbf{v}_{0:b},\varphi_2,t)\end{array}\right)\end{array}\right)\right)$$

an epoch can be considered as an explanation of a violation or satisfaction; the formal definition of epoch can be found in [2,15]. Given access to trace diagnostic results at each instant, causation monitoring classifies the current time step b of $\mathbf{v}_{0:b}$ as follows: if the current instant b of $\mathbf{v}_{0:b}$ is included in the violation epoch, it is considered as a *violation causation*; if b is included in the satisfaction epoch, it is considered as a *satisfaction causation*; otherwise, it is *irrelevant*.

The causality monitor proposed in [13] achieves this as follows: at each instant, it computes two quantitative metrics $[\mathscr{R}]^{\ominus}(\mathbf{v}_{0:b},\varphi,\tau)$ and $[\mathscr{R}]^{\oplus}(\mathbf{v}_{0:b},\varphi,\tau)$, that respectively represent the distances of the current instant b from being a violation causation and a satisfaction causation. The formal definition of causation distances $[\mathscr{R}]^{\ominus}(\mathbf{v}_{0:b},\varphi,\tau)$ and $[\mathscr{R}]^{\oplus}(\mathbf{v}_{0:b},\varphi,\tau)$ are presented in Definition 6.

Definition 6 (Online causality monitor [13]**).** Let $\mathbf{v}_{0:b}$ be a partial signal and φ be an STL formula. At an instant b, an online causality monitor returns a *violation causation distance* $[\mathscr{R}]^{\ominus}(\mathbf{v}_{0:b},\varphi,\tau)$ and a *satisfaction causation distance* $[\mathscr{R}]^{\oplus}(\mathbf{v}_{0:b},\varphi,\tau)$, as defined in Table 1.

The causation verdict, regarding whether b is a causation or not, can be inferred by the results of the online causality monitor in Definition 6, as follows:

- if $[\mathscr{R}]^{\ominus}(\mathbf{v}_{0:b},\varphi,\tau) < 0$, then b is a violation causation;
- if $[\mathscr{R}]^{\oplus}(\mathbf{v}_{0:b},\varphi,\tau) > 0$, then b is a satisfaction causation;
- otherwise, i.e., $[\mathscr{R}]^{\ominus}(\mathbf{v}_{0:b},\varphi,\tau) > 0$ and $[\mathscr{R}]^{\oplus}(\mathbf{v}_{0:b},\varphi,\tau) < 0$, b is irrelevant.

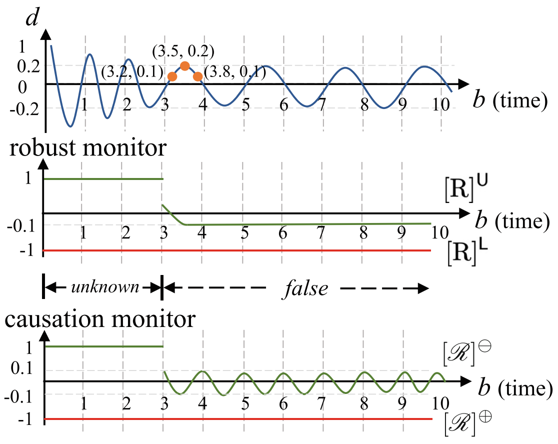


Fig. 6. Monitoring results for a trace of d and an STL formula $\varphi : \square_{[0,10]}(\lozenge_{[0,1]}|d| < 0.1)$, by using robust monitoring and causality monitoring, respectively.

Below, we use an example to illustrate how online causality monitor works.

Example 2. We use the example in Fig. 6 to illustrate how causality monitoring works. According to the robust monitor, the specification is violated by the signal after $b = 3.2$. If we apply the trace diagnostics technique in [15] at the moment $b = 3.5$, we can find that the specification violation of the prefix of d till $b = 3.5$ is caused by the signal values of d during $[3.2, 3.5]$. Since $b = 3.5$ is included in this set, $b = 3.5$ should be considered as a causation of the violation. The causality monitor can directly report whether a moment is a causation of violation: by Fig. 6, we can see that the violation causation distance $[\mathscr{R}]^{\ominus}(\mathbf{v}_{0:b}, \varphi, 0) = -0.1 < 0$, which implies that $b = 3.5$ is indeed a violation causation.

Similarly, at $b = 4$, we obtain the causal interval for the violation of the specification, which is $[3.2, 3.8]$ that does not include $b = 4$, therefore, $b = 4$ is irrelevant to the violation. This is also shown by computing the causation distances $[\mathscr{R}]^{\ominus}(\mathbf{v}_{0:4}, \varphi, 0) = 0.1 > 0$ and $[\mathscr{R}]^{\oplus}(\mathbf{v}_{0:4}, \varphi, 0) = \mathtt{R}^{\alpha}_{\min} < 0$.

Relationship with Robust Monitors. By Example 2, we can see that the causality monitor is not monotonic, and does not suffer from the information masking problem. Lemma 1 states the relationship between causality monitoring and robust monitoring, i.e., for a given signal and specification, the monitoring results by using causality monitors in Definition 6 refine the results by using robust monitors in Definition 5. In other words, the information delivered by causality monitors is a superset of that can be delivered by classic robust monitors.

Lemma 1. The causality monitor in Definition 6 refines the classic online robust monitor in Definition 5, in the sense that the monitoring results of the robust monitor can be inferred from the results of the causation monitor, as follows:

$$[\mathrm{R}]^{\mathsf{U}}(\mathbf{v}_{0:b}, \varphi, \tau) = \inf_{t\in[0,b]} [\mathscr{R}]^{\ominus}(\mathbf{v}_{0:t}, \varphi, \tau)\,, \; [\mathrm{R}]^{\mathsf{L}}(\mathbf{v}_{0:b}, \varphi, \tau) = \sup_{t\in[0,b]} [\mathscr{R}]^{\oplus}(\mathbf{v}_{0:t}, \varphi, \tau)$$

4 A Case Study

To demonstrate the modeling, simulation, and monitoring capabilities of MIMOS, we present a case study based on an autopilot for autonomous marine vessels.

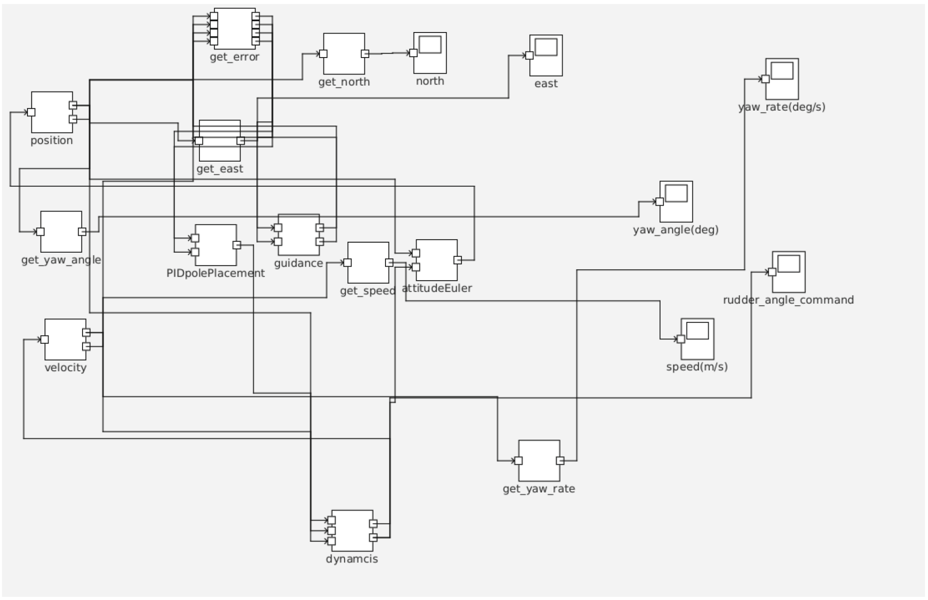


Fig. 7. Modeling a heading autopilot system in MIMOS.

4.1 A Heading Autopilot for Marine Vessels

An autopilot for autonomous marine vessels is a control system designed to regulate a vessel's position and orientation in three-dimensional space. This involves managing six degrees of freedom: three translational movements (surge, sway, heave) and three rotational movements (roll, pitch, yaw). While navigating a vessel from one point to another theoretically requires adjusting all six parameters, practical systems often prioritize specific aspects, such as position and heading, depending on the vessel and its operational needs. These autopilots are complex, integrating multiple control loops to account for dynamic and environmental conditions. In this study, we will examine the control system of a heading autopilot, which specifically adjusts the vessel's yaw angle, as detailed in [6].

Figure 7 illustrates the system model for the heading autopilot, designed to regulate the vessel's yaw angle through real-time Euler angle computations and yaw dynamics modeled using the Clarke approach. The vessel has specific parameters: a length of 70 m, a beam of 8 m, and a draft of 6 m. A PID controller, tuned using pole-placement, forms the system's core, adjusting the rudder to maintain the desired yaw angle. The system includes the following components:

- Guidance module: Generates smooth yaw angle and rate trajectories using a reference model.
- Position module: Extracts the current yaw angle from the vessel's state.
- Error calculation: Computes deviations between desired and actual yaw angle and rate.
- PID controller: Processes these errors to determine the rudder control input.
- Dynamics module: Simulates vessel motion using the Clarke model in response to rudder adjustments.
- Attitude module: Updates the Euler angles to reflect the vessel's orientation.

These components are interconnected, with feedback enabling continuous correction, forming a closed-loop control system that ensures precise and robust yaw control under varying dynamic and environmental conditions.

4.2 STL Specifications and Evaluation Results

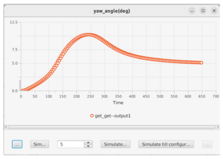
Signal Temporal Logic (STL) is employed to formalize safety-critical requirements of the autopilot system. Specifically, the following STL specifications define the primary safety constraints:

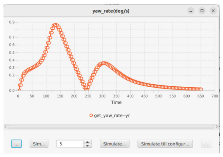
$$\varphi_1 = \Box\,(|yaw\_rate(t)| < 5.0)$$
$$\varphi_2 = \Box\,(|yaw\_angle(t)| < 12.0)$$

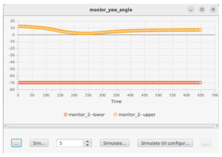
Specification φ_1 ensures that the yaw rate remains within limits to prevent rapid rotational movements, mitigating severe rolling and protecting the crew, cargo, and vessel integrity. Similarly, specification φ_2 ensures that yaw angle deviations remain within acceptable boundaries, guaranteeing stable and controlled vessel maneuvering.

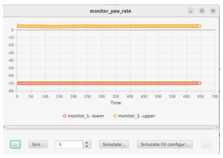
4.3 Simulation Results

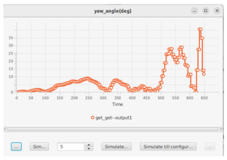
Figure 8 presents comparative simulation results for two implementations of the autopilot system. The first implementation (top row, subfigures a-d) demonstrates stability with the yaw rate consistently staying below 5.0°/s, and the yaw angle remaining within $\pm 12.0^\circ$. These results are reflected by the monitor outputs in Fig. 8c and 8d: in both subfigures, the violation causation distances (i.e., the yellow curves) are always positive, and the satisfaction causation distances (i.e., the orange curves) are negative constants (because the specifications are safety properties); these results deliver that the specifications are not violated during the simulation. This performance validates the PID controller's effectiveness in regulating rudder movements to achieve the target heading.

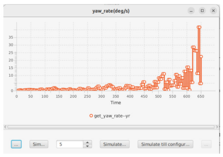

(a) Yaw angle (Impl. 1)

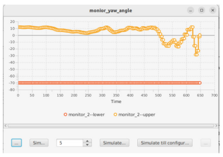

(b) Yaw rate (Impl. 1)


(c) Monitored yaw angle (Impl. 1)


(d) Monitored Yaw rate (Impl. 1)


(e) Yaw angle (Impl. 2)


(f) Yaw rate (Impl. 2)


(g) Monitored yaw angle (Impl. 2)

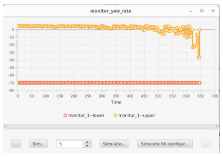

(h) Monitored Yaw rate (Impl. 2)

Fig. 8. Comparative simulation results for two implementations of the autopilot system. The top row (a-d) shows results for the first implementation, which maintains stability within safety thresholds. The bottom row (e-h) shows results for the second implementation, which exhibits pronounced oscillations and frequent violations of safety specifications.

In contrast, the second implementation (bottom row, subfigures e-h) exhibits pronounced oscillations and sharp transitions in both yaw angle and yaw rate. As demonstrated by the monitor outputs (see the yellow curves that depict the violation causation distances in Fig. 8g and 8h), this implementation frequently violates the safety thresholds specified by φ_1 and φ_2, indicating poor control stability that could compromise vessel safety and operational performance.

5 Conclusion

This paper presents the monitoring feature of MIMOS, an integrated tool environment for the design, analysis, and implementation of real-time systems. MIMOS is built on a deterministic and asynchronous data-flow model, supporting key functionalities such as modeling, simulation, verification, scheduling, timing analysis, and multi-core code generation. The tool enables a structured and modular approach to real-time system development, ensuring predictable behavior and facilitating dynamic updates without compromising system safety. We have introduced the causality monitoring plugin for Signal Temporal Logic, demonstrating its application within the MIMOS framework. A case study based on an autopilot control system for marine vessels is presented to illustrate the monitoring feature, which can be used to verify time-variant properties of MIMOS model at run-time.

References

1. Bartocci, E., Falcone, Y. (eds.): Lectures on Runtime Verification. LNCS, vol. 10457. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-75632-5
2. Bartocci, E., Ferrère, T., Manjunath, N., Ničković, D.: Localizing faults in Simulink/Stateflow models with STL. In: Proceedings of the 21st International Conference on Hybrid Systems: Computation and Control (Part of CPS Week), HSCC 2018, pp. 197–206. Association for Computing Machinery, New York (2018). https://doi.org/10.1145/3178126.3178131
3. Deshmukh, J.V., Donzé, A., Ghosh, S., Jin, X., Juniwal, G., Seshia, S.A.: Robust online monitoring of signal temporal logic. Formal Methods Syst. Des. **51**(1), 5–30 (2017). https://doi.org/10.1007/s10703-017-0286-7
4. Donzé, A., Maler, O.: Robust satisfaction of temporal logic over real-valued signals. In: Chatterjee, K., Henzinger, T.A. (eds.) FORMATS 2010. LNCS, vol. 6246, pp. 92–106. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15297-9_9
5. Fainekos, G.E., Pappas, G.J.: Robustness of temporal logic specifications for continuous-time signals. Theor. Comput. Sci. **410**(42), 4262–4291 (2009). https://doi.org/10.1016/j.tcs.2009.06.021
6. Fossen, T.I.: Handbook of Marine Craft Hydrodynamics and Motion Control. Wiley, Hoboken (2011)
7. Kahn, G.: The semantics of a simple language for parallel programming. In: Rosenfeld, J.L. (ed.) Information Processing, Proceedings of the 6th IFIP Congress 1974, Stockholm, Sweden, 5–10 August 1974, pp. 471–475. North-Holland (1974)
8. Liu, C.L., Layland, J.W.: Scheduling algorithms for multiprogramming in a hard-real-time environment. J. ACM (JACM) **20**(1), 46–61 (1973)
9. Maler, O., Nickovic, D.: Monitoring temporal properties of continuous signals. In: Lakhnech, Y., Yovine, S. (eds.) FORMATS/FTRTFT -2004. LNCS, vol. 3253, pp. 152–166. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30206-3_12
10. Pnueli, A.: The temporal logic of programs. In: FOCS 1977, pp. 46–57. IEEE (1977). https://doi.org/10.1109/SFCS.1977.32
11. Selyunin, K., et al.: Runtime monitoring with recovery of the SENT communication protocol. In: Majumdar, R., Kunčak, V. (eds.) CAV 2017. LNCS, vol. 10426, pp. 336–355. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63387-9_17
12. Yi, W., Mohaqeqi, M., Graf, S.: MIMOS: a deterministic model for the design and update of real-time systems. In: ter Beek, M.H., Sirjani, M. (eds.) In Proceedings of 24th IFIP WG 6.1 International Conference on Coordination Models and Languages , COORDINATION 2022, Held as Part of the 17th International Federated Conference on Distributed Computing Techniques, DisCoTec 2022, Lucca, Italy, 13–17 June 2022. Lecture Notes in Computer Science, vol. 13271, pp. 17–34. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-08143-9_2
13. Zhang, Z., An, J., Arcaini, P., Hasuo, I.: Online causation monitoring of signal temporal logic. In: Enea, C., Lal, A. (eds.) 35th International Conference on Computer Aided Verification, CAV 2023. Lecture Notes in Computer Science, vol. 13964, pp. 62–84. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-37706-8_4

14. Zhang, Z., An, J., Arcaini, P., Hasuo, I.: Caumon: an informative online monitor for signal temporal logic. In: Platzer, A., Rozier, K.Y., Pradella, M., Rossi, M. (eds.) Formal Methods - 26th International Symposium, FM 2024, Milan, Italy, 9–13 September 2024, Proceedings, Part II. Lecture Notes in Computer Science, vol. 14934, pp. 286–304. Springer, Cham (2024). https://doi.org/10.1007/978-3-031-71177-0_18
15. Zhang, Z., Arcaini, P., Xie, X.: Online reset for signal temporal logic monitoring. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **41**(11), 4421–4432 (2022). https://doi.org/10.1109/TCAD.2022.3197693