Formal Methods in System Design (2021) 59:44–76
https://doi.org/10.1007/s10703-022-00395-7

Bridging the gap between single- and multi-model predictive runtime verification

Angelo Ferrando[1] · Rafael C. Cardoso[2] · Marie Farrell[4] · Matt Luckcuck[4] · Fabio Papacchini[5] · Michael Fisher[3] · Viviana Mascardi[1]

Received: 21 June 2021 / Accepted: 5 July 2022 / Published online: 18 August 2022

Abstract

This paper presents an extension of the Predictive Runtime Verification (PRV) paradigm to consider multiple models of the System Under Analysis (SUA). We call this extension *Multi-Model* PRV. Typically, PRV attempts to predict the satisfaction or violation of a property based on a trace and a (single) formal model of the SUA. However, contemporary node- or component-based systems (e.g. robotic systems) may benefit from monitoring based on a model of each component. We show how a Multi-Model PRV approach can be applied in either a *centralised* or a *compositional* way (where the property is compositional), as best suits the SUA. Crucially, our approach is formalism-agnostic. We demonstrate our approach using an illustrative example of a Mars Curiosity rover simulation and evaluate our contribution via a prototype implementation.

✉ Angelo Ferrando
Angelo.Ferrando@unige.it

Rafael C. Cardoso
rafael.cardoso@abdn.ac.uk

Marie Farrell
Marie.Farrell@mu.ie

Matt Luckcuck
Matt.Luckcuck@mu.ie

Fabio Papacchini
f.papacchini@lancaster.ac.uk

Michael Fisher
michael.fisher@manchester.ac.uk

Viviana Mascardi
Viviana.Mascardi@unige.it

[1] Department of Computer Science, Bioengineering, Robotics and Systems Engineering (DIBRIS), University of Genova, Genova, Italy

[2] Department of Computing Science, University of Aberdeen, Aberdeen, UK

[3] Department of Computer Science, The University of Manchester, Manchester, UK

[4] Department of Computer Science, Maynooth University, Co. Kildare, Ireland

[5] School of Computing and Communications, Lancaster University in Leipzig, Leipzig, Germany

Keywords Runtime verification · Predictive runtime verification · Multi-model · Robotics

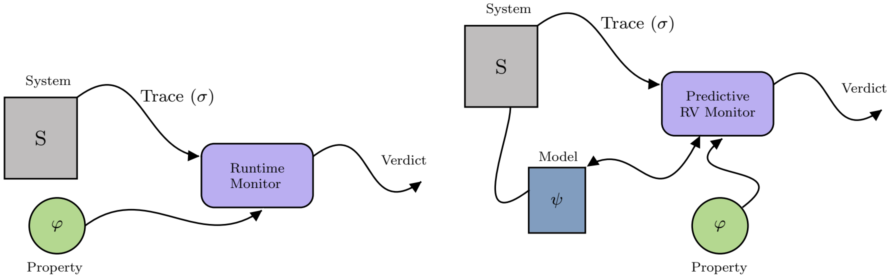
1 Introduction

Runtime Verification (RV) [8, 9, 23] is a formal technique for verifying the runtime behaviour of software systems. Figure 1a illustrates a RV monitor, which consumes a trace (a sequence of events) from the system being monitored, and concludes whether the trace satisfies or violates a property of interest. One or more monitor(s) can be used to analyse system traces, and properties of interest are usually expressed in a formal modelling language such as Linear-time Temporal Logic (LTL) [5, 8, 16].

In PRV [39], the monitor attempts to predict the satisfaction or violation of the property being monitored. If the monitor predicts the property's violation, then the system may be able to recover from the failure or even prevent the violation in advance. Conversely, if the monitor predicts the property's satisfaction, then the monitor can be removed to reduce CPU and memory overheads. Figure 1b illustrates a predictive approach, as an extension of standard RV (Fig. 1a). The system is described by a model, ψ, which can be used to predict future continuations of a generated trace. To generate a verdict, a *predictive* runtime monitor takes both the property, φ, to be verified and the model, ψ, of the system as input.

Previous approaches to PRV have represented the SUA using a single model [31, 39]. However, when the SUA is composed of multiple components—which may be written in heterogeneous languages or paradigms – it may be difficult to define a single and complete representation of the system. For example, for a robotic system, we might be interested in checking that when a sensor perception is received, then a particular action is executed. These events may belong to different components (and therefore different models) but all of the events must be taken into account in order to accurately predict the verdict. In scenarios like this, we may have (formal or non-formal) descriptions of some or all of the system components. Our approach works in a bottom-up fashion to make use of these models of individual components of the SUA, rather than assuming a monolithic model of the entire system. Thus, we seek to answer the following research question:

> RQ: *How can a predictive monitor be applied when the SUA is composed of multiple components, and each component is described by its own model?*



(a) The monitor checks whether a trace, σ, satisfies or violates property φ.

(b) An extension of Fig. 1a, the monitor uses a model, ψ, to predict future continuations of the trace, σ.

Fig. 1 The standard (left) and predictive (right) RV approaches

We consider two styles of PRV: *compositional* and *centralised*. If the monitored property concerns several components and it can be split into sub-properties with each only concerning one component, then we use a *compositional* approach: one monitor for each component, sub-property, and model. This is similar to replicating Single-Model RV [31] across several models. However, if the monitored property cannot be split in this way, then we show how a *centralised* approach can be utilised.

The remainder of this paper is structured as follows. Section 2 provides the prerequisite definitions of single- and multi-model PRV that are used throughout the paper. In Sect. 3 we describe the steps that are required to move from single-model to multi-model PRV. Section 4 introduces our two contributions to multi-model PRV; the centralised and compositional approaches. Section 5 demonstrates the theory, using RV for the Mars Curiosity rover as an illustrative example. In Sect. 6 we evaluate our contribution. Specifically, in Sect. 6.1 we present an overview of the prototype tool developed for this work, and in Sect. 6.2 we report the results of the experiments obtained by applying our tool to the Mars Curiosity example. Section 7 discusses related work and, finally, Sect. 8 concludes with a brief summary and outlines future research directions.

2 Preliminaries

This section introduces the notation and basic definitions used throughout the paper. Our approach is formalism-agnostic with respect to the models and properties meaning that it is not tied to any specific language or logic for the model and/or to specify properties. In practice, implementations of our approach will rely on formalisms selected by the monitors' developers, and these will need to respect particular conditions that we introduce later. These conditions are satisfied by many common formalisms, such as LTL and Finite-State Automata (FSA).

A system is denoted by S, and its *alphabet* (all of its observable events) is denoted by Σ_S (or Σ where there is no confusion). Given an alphabet, Σ, then a *trace*, σ, is a sequence of events in Σ, and $tr(\Sigma)$ is the *set of all possible traces* (the language) over Σ.

Properties are denoted by φ, potentially subscripted for clarity, and $\overline{\varphi}$ denotes their negation. Given an alphabet Σ, we denote the alphabet of a property φ by $\Sigma_\varphi \subseteq \Sigma$. Given an alphabet, Σ, a property, φ, is *satisfied by a trace*, σ, over Σ, written $\sigma \models \varphi$, if and only if σ belongs to the language of traces determined by φ. Thus, a property denotes a set of traces. For instance, if the chosen formalism is LTL, then the notion of satisfaction of a formula, *i.e.* $\sigma \models \varphi$, is obtained by applying the semantics of LTL over traces [32]; if the formalism is FSA, then $\sigma \models \varphi$ is obtained by checking whether σ corresponds to a path from an initial state to a final state in the automaton [24], and so on. The set $[\![\varphi]\!] = \{\sigma \mid \sigma \models \varphi\}$ contains the *set of traces satisfying* φ, and we denote that a particular trace σ satisfies a property φ as $\sigma \in [\![\varphi]\!]$.

A property, φ, can be specified in any formalism such that for a given alphabet Σ, and for any trace, $\sigma \in tr(\Sigma)$, the following two conditions hold:

$$\sigma \in [\![\varphi]\!] \text{ is decidable} \tag{1}$$

$$\sigma \in [\![\varphi]\!] \iff \sigma \notin [\![\overline{\varphi}]\!] \tag{2}$$

Condition (1) states that, given a property, φ, specified in a chosen formalism, we can always assess whether a trace, σ, satisfies φ, i.e. σ belongs to the set of traces satisfying φ. This condition is mandatory since a monitor is defined upon the notion of trace acceptance.

As we will show in Definition 1, a monitor must check if a trace satisfies the property under analysis, and this can be achieved only when condition (1) holds. Condition (2) states that a trace σ, satisfies a property, φ, if, and only if, σ does not satisfy its negation $\overline{\varphi}$. The negation of properties will be used in Definition 2 to define monitors with a predictive flavour, where we will combine a model, ψ, with the negation of a property, φ, to check for traces satisfying ψ but not φ.

Definition 1 (*Monitor*) Let S be a system with alphabet Σ, and φ be a property. Then, a *monitor* for φ is a function $Mon_\varphi : tr(\Sigma) \rightarrow \mathbb{B}_4$, where $\mathbb{B}_4 = \{\top, \bot, ?_\top, ?_\bot\}$:

$$Mon_\varphi(\sigma) = \begin{cases} \top & \forall_{u \in tr(\Sigma)}.\sigma \bullet u \notin [\![\overline{\varphi}]\!] \\ \bot & \forall_{u \in tr(\Sigma)}.\sigma \bullet u \notin [\![\varphi]\!] \\ ?_\top & \sigma \in [\![\varphi]\!] \wedge \exists_{u \in tr(\Sigma)}.\sigma \bullet u \in [\![\overline{\varphi}]\!] \\ ?_\bot & \sigma \notin [\![\varphi]\!] \wedge \exists_{u \in tr(\Sigma)}.\sigma \bullet u \in [\![\varphi]\!] \end{cases}$$

where $\bullet$ is the standard trace concatenation operator.

In Definition 1, a monitor can be intuitively understood to be a function that, given a trace (σ), returns a verdict (either $\top$, $\bot$, $?_\top$, or $?_\bot$). If all continuations of σ satisfy φ, then the monitor returns $\top$. If all possible continuations of σ violate φ, then the monitor return $\bot$. If σ satisfies φ, but there exists at least one continuation which does not, then the monitor returns $?_\top$ (read as “possibly true”). And if σ does not satisfy φ, but there exists at least one continuation that does, then the monitor returns $?_\bot$ (read as “possibly false”).

Remark 1 Even though our contribution is general and formalism-agnostic, to help the reader understand better how a monitor works, we linger a bit longer on its description. According to [7], a monitor concludes $\top$, when it has observed enough information from the system to declare the satisfaction of the property under evaluation (φ). This means that no matter what the system does in the future, what it has done in the past is more than enough to conclude satisfaction. Symmetrically, a monitor concludes $\bot$, when it has observed enough information from the system to declare that the property under evaluation (φ) has been violated. Again, this means that no matter what the system does in the future, what the system has done in the past is more than enough to conclude the violation.

The two inconclusive verdicts ($?_\top$ and $?_\bot$) are returned when a final verdict cannot be determined over the currently observed trace σ. These mean that the monitor has observed a trace σ that might satisfy φ, but that does not contain enough information to guarantee it will always do so. Thus, the monitor concludes $?_\top$ if, up to now the system seems to behave correctly (σ does satisfies φ), but in the future it could still do something that would violate the property φ. Symmetrically, $?_\bot$ is concluded where the currently observed trace σ is violating φ, but it is still possible that in the future the system will satisfy φ.

For Definition 1 we implicitly assume that the chosen formalism describes both finite and infinite traces of events. This can be observed in the third and fourth case of Definition 1, where the trace σ is finite. In the case of a formalism only accepting infinite traces, the definition of a monitor is obtained by merging the third and fourth cases, returning a single but less informative outcome (*i.e.*, ?). In this paper, we consider the more challenging scenario, when the formalism accepts both finite and infinite traces. However, the obtained results can also be ported to scenarios only considering infinite traces.

Definition 1 describes a generic monitor that does not impose constraints on the formalism used. Consequently, we collapse the definition of $tr(\Sigma)$ for representing finite and infinite traces depending on what is supported by the formalism that is used to define φ. Thus, if the formalism used to define φ supports only traces of infinite length, then $tr(\Sigma) = \Sigma^\omega$; if the formalism supports only traces of finite length, then $tr(\Sigma) = \Sigma^*$; otherwise, $tr(\Sigma) = \Sigma^* \cup \Sigma^\omega$.

Example 1 (LTL Monitor) Let $\varphi = \Box p$ be an LTL property, and $\Sigma = \{p, q\}$ be the alphabet of the system under analysis. In natural language, the property is read: "p is always true". If we consider the semantics of φ according to standard LTL semantics [32], we obtain: $[\![\varphi]\!] = \{p^\omega\}$ (where p^ω denotes an infinite trace of p). Now, according to Definition 1, we define a monitor as a function $Mon_\varphi : \Sigma^\omega \rightarrow \mathbb{B}_4$. Note that, since LTL semantics is defined on infinite traces only, we have that $tr(\Sigma) = \Sigma^\omega$. Let us consider the finite trace $\sigma = p \bullet p \bullet p$ as an example. When we apply Mon_φ to σ, we obtain[1] ? (*i.e.*, $Mon_\varphi(\sigma) = ?$). This is due to the fact that there exist continuations $u \in \Sigma^\omega$ of σ that both satisfy φ (such as the infinite trace $u = p^\omega$) and that violate φ (such as the infinite trace $u = q \bullet p^\omega$). Intuitively, this means that the monitor cannot conclude anything about the satisfaction or violation of φ because there might still be future events that would change the monitor's outcome.

Let us consider instead another finite trace $\sigma = p \bullet p \bullet q$ as an example. In this case, when we apply Mon_φ to σ, we obtain $\bot$ (*i.e.*, $Mon_\varphi(\sigma) = \bot$). This is due to the fact that, as specified in Definition 1, the monitor concludes the violation of φ by knowing that all possible continuations $u \in \Sigma^\omega$ violate φ. In fact, by observing q as third event in σ, we already know φ cannot be satisfied by the system (φ requires p to be observed at each step in the trace). In the case of $\sigma = p \bullet p \bullet q$, because there is no infinite continuation ($u \in \Sigma^\omega$) that can satisfy the specification ($\sigma \bullet u \models \varphi$) the monitor can safely conclude that φ has been violated. This verdict is final, indeed, no future event will ever change the outcome.

Let S be a system with alphabet Σ. We denote its *model* by ψ, and use $[\![\psi]\!] \subseteq tr(\Sigma)$ to indicate the *set of traces recognised by* ψ. A model, ψ, can be specified in any formalism such that for a given alphabet, Σ, for any trace, $\sigma \in tr(\Sigma)$, and for any property, φ, the following holds:

$$\sigma \in [\![\psi]\!] \text{ is decidable} \tag{3}$$

$$[\![\varphi]\!] \cap [\![\psi]\!] \text{ is computable} \tag{4}$$

We denote (4) via the use of a binary relation $\otimes$, that is, $[\![\varphi \otimes \psi]\!] = [\![\varphi]\!] \cap [\![\psi]\!]$.

Often, PRV frameworks express their properties in LTL (for example [28, 39]); however, we took our inspiration from a PRV framework where both the SUA and property are defined using Timed Automata (TA) [31]. The reason is that in works such as [28, 39], the predictive aspect is not formalised through a model of the system, but as a set of finite suffixes. These suffixes are then concatenated to the given trace, σ, allowing the monitor to predict the initial part of the possible continuations, u. Instead, in our work, as in [31], we explicitly represent the model without focusing only on the first events after σ, but by applying the prediction to the entire possible continuation. This can be obtained by using

[1] We remind the reader that in case of $tr(\Sigma) = \Sigma^\omega$, the two inconclusive outcomes $?_\top$ and $?_\bot$ are merged into a single inconclusive and less informative ?.

a model of the system as input to the monitor, alongside the property to be verified. Informally, the model generates the set of event traces that can be observed by executing the system. We follow the definition of a predictive monitor from [31], however, in this paper we remain formalism-agnostic.

Definition 2 (*Predictive Monitor*) Let S be a system with alphabet Σ, model ψ and let φ be a property. A *predictive monitor* for φ given ψ is a function, $Mon_{\varphi,\psi} : tr(\Sigma) \rightarrow \mathbb{B}_5$, where $\mathbb{B}_5 = \{\top, \bot, ?_\top, ?_\bot, ?\}$:

$$Mon_{\varphi,\psi}(\sigma) = \begin{cases} \top & \forall_{u \in tr(\Sigma)}.\sigma \bullet u \notin [\![\overline{\varphi} \otimes \psi]\!] \\ \bot & \forall_{u \in tr(\Sigma)}.\sigma \bullet u \notin [\![\varphi \otimes \psi]\!] \\ ?_\top & \sigma \in [\![\varphi \otimes \psi]\!] \wedge \exists_{u \in tr(\Sigma)}.\sigma \bullet u \in [\![\overline{\varphi} \otimes \psi]\!] \\ ?_\bot & \sigma \in [\![\overline{\varphi} \otimes \psi]\!] \wedge \exists_{u \in tr(\Sigma)}.\sigma \bullet u \in [\![\varphi \otimes \psi]\!] \\ ? & otherwise \end{cases}$$

The intuitive meaning of the return values is the same as in the non-predictive case (Definition 1). However, the introduction of the model requires the addition of the inconclusive value, ?, to cover the case when σ does not belong to the model in question. Note the use of $\otimes$ in the definition. For instance, the case for $\top$ requires that all traces $\sigma \bullet u$ are not in $[\![\overline{\varphi}]\!] \cap [\![\psi]\!]$ where $\overline{\varphi}$ represents the negation of φ. Definition 2, like Definition 1, assumes the most expressive kind of formalism, where both finite and infinite traces of events are considered. Nonetheless, Definition 2 can be straightforwardly modified to consider only infinite traces, by collapsing the last three cases into a single ? case.

To help the reader to better understand how this works, we again linger longer on its definition. Specifically, in the case of a predictive monitor we do not only have a property to verify φ, but a model of the system under analysis ψ, as well. The model of the system is used by the predictive monitor to reduce the number of possible continuations of a given trace σ. Since a monitor can conclude a final verdict (such as $\top$ or $\bot$) only when it is certain the system will never in the future produce anything to change its mind; without knowing how the system behaves (which is what happens in the standard scenario), the monitor has to consider any possible continuation. In fact, the monitor only stops when it knows that the property φ is certainly satisfied or violated. For a predictive monitor, the presence of a model of the system helps the monitor to constrain the future continuations to only those that are realistically observable. Thanks to this, the predictive monitor is capable of concluding a final verdict in advance of its standard counterpart. This is simply due to the fact that the standard monitor could consider a continuation that the system would never produce (that is, a continuation that is not in the model), but that satisfies or violates the property φ. This could be a continuation that would stop the monitor from concluding a final verdict (the first two cases in Definition 2).

Example 2 (Predictive Monitor) Let φ be the same LTL property used in Example 1 (i.e. $\varphi = \Box p$), with $\Sigma = \{p, q\}$ the alphabet of the system under analysis. Let $\psi = (p \wedge \bigcirc p) \rightarrow \Box p$ be the model of the system, also expressed as an LTL property[2]. In natural language, the model says: "if the first two observed events are p, then the system will always do p" (where $\bigcirc$ is the LTL next operator). The semantics of the model

[2] Note that, the formalisms used to describe properties and models can be different in general.

corresponds to $[\![\psi]\!] = \{q \bullet \{p,q\}^{\omega}, p \bullet q \bullet \{p,q\}^{\omega}, p^{\omega}\}$. Differently from Example 1, the presence of the model ψ helps reduce the traces to consider as possible continuations of a given finite trace σ. Consider, as in Example 1, the finite trace $\sigma = p \bullet p \bullet p$. The standard monitor would conclude ? ($Mon_{\varphi}(\sigma) = ?$); because after observing p three times, we do not have any assurance of the satisfaction or violation of φ (we might keep observing p forever or a q could arrive and violate φ). However, the predictive monitor (Definition 2) $Mon_{\varphi,\psi}$ has access to additional information about the system. In fact, according to Definition 2, we cannot find any continuation ($u \in \Sigma^{\omega}$) that is in the intersection of the negation of the property and the model ($\sigma \bullet u \in [\![\overline{\varphi} \otimes \psi]\!]$). This is determined by the fact that $[\![\overline{\varphi}]\!] = \{p^{*} \bullet q \bullet \{p,q\}^{\omega}\}$, and $[\![\psi]\!] = \{q \bullet \{p,q\}^{\omega}, p \bullet q \bullet \{p,q\}^{\omega}, p^{\omega}\}$. Thus, $[\![\overline{\varphi} \otimes \psi]\!] = \{p \bullet q \bullet \{p,q\}^{\omega}, q \bullet \{p,q\}^{\omega}\}$. And, consequently, when $\sigma = p \bullet p \bullet p$, we cannot find any $u \in \Sigma^{\omega}$ such that $\sigma \bullet u \in [\![\overline{\varphi} \otimes \psi]\!]$ and we obtain $Mon_{\varphi,\psi}(\sigma) = \top$; since no trace containing two initial p is contained into $[\![\overline{\varphi} \otimes \psi]\!]$.

The above definitions (Definitions 1 and 2) take into consideration the case where a system is represented by a single model.

Definition 3 (*Multi-model*) Let S be a system composed of a *set of components* $\mathcal{C} = \{\mathcal{C}_1, \dots, \mathcal{C}_n\}$ and alphabet $\Sigma_S = \Sigma_1 \cup \dots \cup \Sigma_n$ where Σ_i is the alphabet of the component $\mathcal{C}_i$. Then a *multi-model*, $\langle \Psi, \mathcal{A} \rangle$, of S is a pair where $\Psi = \{\psi_1, \dots, \psi_n\}$ is the *set of models of the components* in $\mathcal{C}$, and $\mathcal{A} : \mathcal{C} \rightarrow \Psi$ is a bijective function associating each component with its model in Ψ.

Where we only have a single model ψ, we can consider this a special case of a multi-model $\langle \Psi, \mathcal{A} \rangle$, where $\mathcal{C} = \{C\}$.

In the remainder of the paper, components' alphabets Σ_i are assumed to be disjoint. Given a multi-model $\langle \Psi, \mathcal{A} \rangle$, $\mathcal{A}^{-1}$ indicates the inverse of the bijective function $\mathcal{A}$.

3 Engineering multi-model predictive RV

This section describes the process to port a given single-model predictive monitor to a multi-model predictive monitor for a component-based system. When RV is applied to component-based systems, there might be issues concerning the ordering of event traces that are produced by distinct system components. As remarked in [22], in the absence of a global clock, composing two remote[3] traces produced by two separate components does not necessarily yield a total ordering among the composite trace, but instead gives a partial ordering. As we focus on the difficulties associated with applying RV using multiple models for prediction, in this paper we only consider systems with a shared global clock. The presence of a global clock simplifies the monitor definition, because it allows monitors to assume the existence of a total ordering over the local traces.

In this section, we describe how to extract the set of models to predict the events for a given property, and how to combine multiple models into a single-model.

[3] Traces that are generated by distributed components.

3.1 Contextualising a property

We begin by identifying to which component(s) a given property refers. In some cases the property could refer to the entire system, or it might refer to a specific subset of components. It is important to understand, first, which component(s) the corresponding monitor must watch; and second, which model(s) are required to predict the events deriving from the components of interest.

To identify which models are required to monitor a given property, we define a specific function (Definition 4) to extract these models, whose domain is the set of properties $\Phi = \bigcup_{\mathcal{C}_i \in \mathcal{C}} \Phi_{\mathcal{C}_i}$ where $\Phi_{\mathcal{C}_i}$ is the set of properties defined over the alphabet of the component $\mathcal{C}_i \in \mathcal{C}$. Here, Φ denotes the collection of properties that can be defined for all components in S.

Definition 4 (*Contextualise Function*) Let S be a system with components $\mathcal{C} = \{\mathcal{C}_1, \dots, \mathcal{C}_n\}$, alphabet $\Sigma = \Sigma_{\mathcal{C}_1} \cup \dots \cup \Sigma_{\mathcal{C}_n}$, and a multi-model $\langle \Psi, \mathcal{A} \rangle$. Then $\kappa_{\langle \Psi, \mathcal{A} \rangle} : \Phi \to \mathbb{P}(\Psi)$ is the *contextualise* function, which given a property φ with alphabet Σ_φ, returns the minimal set of models $\Psi' = \{\psi_{i_1}, \dots, \psi_{i_m}\}$ with $\Psi' \subseteq \Psi$ such that $\forall_{\mathcal{C}_i \in \mathcal{C}}.(\Sigma_{\mathcal{C}_i} \cap \Sigma_\varphi \neq \emptyset \implies \mathcal{A}(\mathcal{C}_i) \in \Psi')$.

The resulting set of models returned by $\kappa_{\langle \Psi, \mathcal{A} \rangle}$ is minimal because, as mentioned in Sect. 2, the alphabets of the components are disjoint. Since an event can be produced by only one component in S, only the events that are relevant for checking the property's satisfaction are considered. If we consider a proper subset of the models returned by $\kappa_{\langle \Psi, \mathcal{A} \rangle}$, we can always find an event that is relevant for the property, but its model is not in the set.

Once we have extracted the context for a property, we know which models can be used to predict future events. That is, given the multi-model $\langle \Psi, \mathcal{A} \rangle$ of S, we know that κ returns the smallest subset of Ψ needed to predict future events.

Observation 1 (*Contextualisation of the negation*) It is important to note that the contextualisation of a property and its negation are the same. This can be seen intuitively by the fact that Definition 4 is based on the alphabet Σ_φ of the property φ, and it follows directly that $\Sigma_\varphi = \Sigma_{\overline{\varphi}}$. Thus, passing the negation of the property (*i.e.*, $\neg\varphi$) as input to the contextualise function would produce the same result.

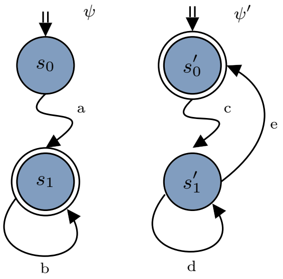
3.2 Combining multiple models

After obtaining the above set of models, we use them to construct the predictive monitor. The predictive monitor requires a property to verify, φ, (which we already have) and a single model, ψ, to be used to predict future events (which we do not yet have). Thus, we must construct ψ based on what we know about the multi-model $\langle \Psi, \mathcal{A} \rangle$.

When we combine multiple models into a global model, we need to refer to traces belonging to both the single models and to the global model. In order to simplify our presentation, we present a notion of trace projection, which we will later use to define the combination of models.

Definition 5 (*Projection*) Let S be a system with components $\mathcal{C} = \{\mathcal{C}_1, \dots, \mathcal{C}_n\}$ and alphabet $\Sigma = \Sigma_{\mathcal{C}_1} \cup \dots \cup \Sigma_{\mathcal{C}_n}$. Then $\pi_{\mathcal{C}'} : tr(\Sigma) \to tr(\Sigma_{\mathcal{C}'})$ is the projection of a trace over the language of $\mathcal{C}' \subseteq \mathcal{C}$, which is recursively defined as follows.

Fig. 2 Two models, ψ and ψ', captured as Büchi Automata



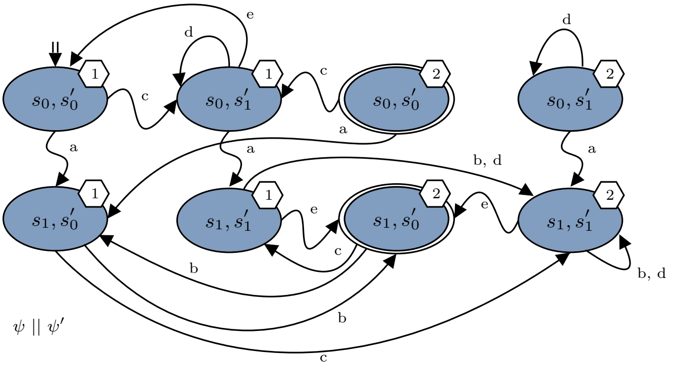


Fig. 3 The corresponding Büchi Automaton representing the combination of ψ and ψ', namely ψ_c

$$
\begin{array}{lll}
(1) & \pi_{\mathcal{C}'}(\epsilon) = \epsilon & \\
(2) & \pi_{\mathcal{C}'}(ev \bullet \sigma) = ev \bullet \pi_{\mathcal{C}'}(\sigma) & \textit{if } ev \in \Sigma_{\mathcal{C}'} \\
(3) & \pi_{\mathcal{C}'}(ev \bullet \sigma) = \pi_{\mathcal{C}'}(\sigma) & \textit{otherwise}
\end{array}
$$

where ϵ is the empty trace, and $\bullet$ is the standard concatenation operator.

For readability, when the set of components contains only one element, i.e. $\mathcal{C} = \{\mathcal{C}_i\}$, we write $\pi_{\mathcal{C}_i}(\sigma)$, meaning that we project the trace, σ, on the single component, $\mathcal{C}_i$.

Definition 6 (*Combination function*) Let S be a system and $\langle \Psi, \mathcal{A} \rangle$ its multi-model. Then $\chi : \mathbb{P}(\Psi) \rightarrow \Psi$ is the combination function which given a set of models $\Psi' \subseteq \Psi$, returns a model ψ_c representing their combination, such that:

$$
\begin{array}{ll}
(1) & \forall_{\sigma \in [\![\psi_c]\!]}.\forall_{\psi_i \in \Psi'}.(\mathcal{A}^{-1}(\psi_i) = \mathcal{C}_i \implies \pi_{\mathcal{C}_i}(\sigma) \in [\![\psi_i]\!]) \\
(2) & \forall_{\psi_i \in \Psi'}.\forall_{\sigma \in [\![\psi_i]\!]}.\exists_{\sigma' \in [\![\psi_c]\!]}.(\mathcal{A}^{-1}(\psi_i) = \mathcal{C}_i \implies \pi_{\mathcal{C}_i}(\sigma') = \sigma)
\end{array}
$$

Note that (1) requires ψ_c to not produce global traces which cannot be observed by the combination of the models, and (2) requires ψ_c to recognise all of the global traces deriving from the composition of the local traces recognised by the models.

The combination function, χ, abstracts the notion of interleaving of models. Figure 2 and Fig. 3 illustrate a small example that we use to demonstrate this similarity. For simplicity, we use Büchi Automata (BA) to describe the models. This choice helps us to describe the function, but it does not limit the theory presented. For instance, other formalisms used in the RV scenario could be used, such as Trace Expressions [2], Timed Automata [1], FSA, and so on. Naturally, even though the theory behind Definition 6 would not change (the requirements (1) and (2) would still be necessary), the instantiation of how to obtain the combination of models would depend on the formalism of choice. For instance, in case of Timed Automata[4], the combination function could be obtained by a standard parallel composition; while in case of Trace Expressions, the combination function could be obtained by using the corresponding built-in interleaving operator.

Given a system, S, which is composed of two components, $\mathcal{C}$ and $\mathcal{C}'$, with alphabets $\Sigma = \{a, b\}$, and $\Sigma' = \{c, d, e\}$, respectively, let $\langle \Psi, \mathcal{A} \rangle$ be the multi-model of S, such that $\Psi = \{\psi, \psi'\}$ (Fig. 2), and $\mathcal{A}(\psi) = \mathcal{C}$, $\mathcal{A}(\psi') = \mathcal{C}'$. We define the χ function to create the global model ψ_c such that conditions (1) and (2) in Definition 6 hold. Since our models are BA, χ can be obtained through a standard parallel composition of automata, which is a more relaxed version of product of automata where for moving among states in the product it is enough to have the transition enabled in one of the two automata[5]. The resulting automaton's behaviour (Fig. 3) is an interleaving of the two automata. In Fig. 3, we show all of the states generated in the process. As is standard in the product of BA, the states in ψ_c are labeled 1 and 2 (hexagons in Fig. 3), and ψ_c must ensure that the visit to final states occurs infinitely often.

Lemma 1 (Over-approximation of χ) *Let S be a system, ψ its single model, and $\langle \Psi, \mathcal{A} \rangle$ its multi-model. Then, $[\![\psi]\!] \subseteq [\![\chi(\Psi)]\!]$.*

Proof The lemma follows directly from Definition 6. Since χ abstracts the notion of interleaving of models, it always returns an over-approximation of S; where no constraint on the order amongst the different models is enforced. Thus, if a single model of the system ψ actually existed, then it would always be at least as restrictive as the combined model returned by χ. □

4 Multi-model predictive RV

Now that we have formally presented the function to contextualise a property, φ, and the function to combine multiple models, $\{\psi_1, \dots, \psi_n\}$ into one, ψ_c, we can demonstrate how to use these functions to adapt the single-model predictive monitoring approach to use multiple models. We assume that the models of the SUA have been constructed during earlier phases of development, although they could be built specifically for RV.

Definition 7 (*Multi-Model Predictive Monitor*) Let S be a system with components $\mathcal{C} = \{\mathcal{C}_1, \dots, \mathcal{C}_n\}$, alphabet $\Sigma = \Sigma_{\mathcal{C}_1} \cup \dots \cup \Sigma_{\mathcal{C}_n}$, multi-model $\langle \Psi, \mathcal{A} \rangle$, and let φ be a property. A multi-model predictive monitor for φ given $\langle \Psi, \mathcal{A} \rangle$ is a function, $Mon_{\varphi, \langle \Psi, \mathcal{A} \rangle} : tr(\Sigma') \to \mathbb{B}_5$, where $\mathbb{B}_5 = \{\top, \bot, ?_\top, ?_\bot, ?\}$, and $\Sigma' = \bigcup_{\psi_i \in \kappa_{\langle \Psi, \mathcal{A} \rangle}(\varphi)} \Sigma_{\mathcal{A}^{-1}(\psi_i)}$, and is defined as follows:

[4] In such case, there would be a different notion of traces w.r.t. what we present here, but the general idea of how to use such traces to predict future continuations would be the same.

[5] In the standard product of automata, a transition has to be enabled in both automata, since the product denotes the intersection of the two languages.

$$Mon_{\varphi,\langle\Psi,\mathcal{A}\rangle}(\sigma) = Mon_{\varphi,\chi(\kappa_{\langle\Psi,\mathcal{A}\rangle}(\varphi))}(\sigma)$$

The functions, $\kappa_{\langle\Psi,\mathcal{A}\rangle}$ and χ, enable us to define a multi-model predictive monitor on top of a single-model monitor. Our approach provides a general definition and requires minimal constraints, thus allowing existing implementations of predictive monitors to be reused.

For a given property, we extract the context (the models necessary for predicting a verdict for the property), and then we create a single model by merging the models. This enables us to use a standard predictive monitor that expects a single-model as input, as described by Definition 7.

This approach considers the monitored property, φ, to be a single global specification which refers to the system, S, as a whole. However, S is composed of distinct components. Consequently, we expect the property to refer to different parts of the system, so there might be cases where the property can be checked by multiple monitors, each focusing only on a part of the property that concerns only a subset of the components. Such parallelism could improve monitoring performance and better exploit the intrinsic distribution of the system. However, it would also complicate prediction. Since each monitor would check only a subset of the system, it would need the corresponding subset of the models to predict future events.

Theorem 1 (Soundness) *Let S be a system with components $\mathcal{C} = \{\mathcal{C}_1, \dots, \mathcal{C}_n\}$, and alphabet $\Sigma = \Sigma_{\mathcal{C}_1} \cup \dots \cup \Sigma_{\mathcal{C}_n}$. Let ψ be its single model, and let $\langle\Psi, \mathcal{A}\rangle$ be its multi-model. Then, for any property φ the following implications hold:*

$$\begin{array}{ll} (1) & \forall_{\sigma\in tr(\Sigma)}.Mon_{\varphi,\langle\Psi,\mathcal{A}\rangle}(\sigma) = \top \;\Rightarrow\; Mon_{\varphi,\psi}(\sigma) = \top \\ (2) & \forall_{\sigma\in tr(\Sigma)}.Mon_{\varphi,\langle\Psi,\mathcal{A}\rangle}(\sigma) = \bot \;\Rightarrow\; Mon_{\varphi,\psi}(\sigma) = \bot \end{array}$$

Proof The theorem follows from Definition 7 and Lemma 1. By Definition 7, we know that $Mon_{\varphi,\langle\Psi,\mathcal{A}\rangle}(\sigma) = Mon_{\varphi,\chi(\kappa_{\langle\Psi,\mathcal{A}\rangle}(\varphi))}(\sigma)$, and by Lemma 1, we know that χ generates an over-approximation of ψ. Thus, the resulting monitor $Mon_{\varphi,\langle\Psi,\mathcal{A}\rangle}$ has access to a model which is at most as restrictive as ψ (*i.e.*, $[\![\psi]\!] \subseteq [\![\chi(\kappa_{\langle\Psi,\mathcal{A}\rangle}(\varphi))]\!]$). Let us prove (1), and (2) can be proved analogously. By Definition 2, $Mon_{\varphi,\langle\Psi,\mathcal{A}\rangle}(\sigma) = \top$ if, and only if, for all possible continuations $u \in tr(\Sigma)$ we have $\sigma \bullet u \notin [\![\overline{\varphi} \otimes \chi(\kappa_{\langle\Psi,\mathcal{A}\rangle}(\varphi))]\!]$. Since $[\![\psi]\!] \subseteq [\![\chi(\kappa_{\langle\Psi,\mathcal{A}\rangle}(\varphi))]\!]$, it follows that $\sigma \bullet u \notin [\![\overline{\varphi} \otimes \psi]\!]$. Thus, $Mon_{\varphi,\psi}(\sigma) = \top$. □

Observation 2 (*No completeness*) The opposite direction of the implications in Theorem 1 does not hold. This can be observed by the fact that the over-approximation can denote more traces than needed (since all possible interleaving of the components are considered). Thus, it is possible that although predictive monitor using this over-approximation does not conclude a final outcome ($\top$ or $\bot$) this is because of the presence of continuations belonging to the over-approximation (satisfying and violating φ), but not belonging to the actual model ψ.

From Theorem 1, we know that a multi-model predictive monitor generated using Definition 7 preserves its results with respect to a standard, single-model monitor. If the multi-model predictive monitor returns a final outcome, then we are assured that the system actually satisfies or violates the property (in the current run σ).

In the next section we describe how, using the notion introduced previously, we can define a general and modular approach for using a multi-model to predict future events in an heterogeneous environment where multiple monitors are used.

4.1 Compositional properties

So far we have focused on a centralised approach where, given a property and a (multi-) model, a single predictive monitor is used. In the remainder of this paper we describe a compositional approach that uses several monitors to predict the satisfaction or violation of a complex property. One benefit of this approach is that it can enable a monitor to terminate earlier. This section shows how properties can be split (if certain compositional requirements are met) to allow the generated sub-properties to be verified independently.

Once properties are split into sub-properties they can be verified separately, alongside the subset of the system models that correspond to the components to which each sub-property refers. This reduces the number of possible continuations that have to checked, which can enable the predictive monitor to terminate earlier than a non-predictive monitor. If the monitor can conclude that the (sub-)property has been violated, then this could lead to swifter mitigation. If the monitor can conclude that the (sub-)property has been satisfied, then it can be terminated to save memory and CPU time.

We assume that the same formalism is used to specify the properties of different components, and that this formalism contains binary operators, which are used to build complex properties from simpler sub-properties. In this scenario, it is natural to think that the verdicts of different monitors, monitoring sub-properties, can be aggregated into a single verdict for the more complex property. This idea, however, does not translate immediately into practice as not all sub-properties can be monitored independently. This problem can be partly circumvented by partitioning the operators into those which are commutative (implying independence) and those which are not. Naturally, there might be operators which can be derived from other operators, in these cases, we extend the notion of commutativity to include operators which are not commutative *per se*, but that can be derived using commutative operators. An example is the *implication* operator ($\implies$), which is not commutative, indeed $A \implies B$ is not the same as $B \implies A$, but can be derived using the disjunction operator ($\vee$) as $\neg A \vee B$ (resp., $\neg B \vee A$), which is commutative.

Formally, let $\diamond$ be the set of binary operators of the formalism, then $\diamond_c \subseteq \diamond$ denotes the set of commutative operators, and $\diamond_{nc} = \diamond \setminus \diamond_c$ denotes the set of non-commutative ones. We say that a property $\varphi_1 \diamond_1 \varphi_2$ is compositional if $\diamond_1 \in \diamond_c$, and non-compositional otherwise. If a property is compositional, then different monitors can be used for the different components (sub-properties). For example, the property $(\varphi_1 \diamond_1 \varphi_2) \diamond_2 \varphi_3$ where $\diamond_1 \in \diamond_c$ and $\diamond_2 \in \diamond_{nc}$ is not compositional and requires a single monitor, but the same property where $\diamond_2 \in \diamond_c$ and $\diamond_1 \in \diamond_{nc}$, can be monitored using two monitors (one for $\varphi_1 \diamond_1 \varphi_2$ and one for φ_3). Naturally, here we are assuming that $\diamond_1$ cannot be distributed over $\diamond_2$, and vice versa. If that was the case, then in the first example, even though $\diamond_2 \in \diamond_{nc}$, we could still distribute it over $\diamond_1 \in \diamond_c$ obtaining $(\varphi_1 \diamond_2 \varphi_3) \diamond_1 (\varphi_2 \diamond_2 \varphi_3)$. Thus producing a compositional property with two sub-properties. As a last step, we define how the results of different monitors can be aggregated into a single verdict in Definition 8.

Definition 8 (*Composition Setting*) Let S be a system with components $\mathcal{C} = \{\mathcal{C}_1, \dots, \mathcal{C}_n\}$, alphabet $\Sigma = \Sigma_{\mathcal{C}_1} \cup \ldots \cup \Sigma_{\mathcal{C}_n}$, multi-model $\langle \Psi, \mathcal{A} \rangle$, and let $\diamond_c$ be the set of commutative binary operators over properties. $\mathcal{ST} = \{\langle \diamond_1, \circ_1 \rangle, \dots, \langle \diamond_n, \circ_n \rangle\}$ is a composition setting

Table 1 One possible composition setting for conjunction and disjunction

$\circ^\wedge$	$\top$	$\bot$	$?_\top$	$?_\bot$	$?$	$\circ^\vee$	$\top$	$\bot$	$?_\top$	$?_\bot$	$?$
$\top$	$\top$	$\bot$	$?_\top$	$?_\bot$	$?$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$
$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\top$	$\bot$	$?_\top$	$?_\bot$	$?$
$?_\top$	$?_\top$	$\bot$	$?_\top$	$?_\bot$	$?$	$?_\top$	$\top$	$?_\top$	$?_\top$	$?_\top$	$?_\top$
$?_\bot$	$?_\bot$	$\bot$	$?_\bot$	$?_\bot$	$?_\bot$	$?_\bot$	$\top$	$?_\bot$	$?_\top$	$?_\bot$	$?$
$?$	$?$	$\bot$	$?$	$?_\bot$	$?$	$?$	$\top$	$?$	$?_\top$	$?$	$?$

where $\diamond_i \in \diamond_c$, and each $\circ_i : \mathbb{B}_5 \times \mathbb{B}_5 \to \mathbb{B}_5$ is an aggregation function such that the following hold:

$$\begin{array}{ll} (1) & \forall_{\sigma \in tr(\Sigma)} . \forall_{\langle \diamond_i, \circ_i \rangle \in \mathcal{ST}} . \\ & \quad Mon_{\varphi_1 \diamond_i \varphi_2, \langle \Psi, \mathcal{A} \rangle}(\sigma) = Mon_{\varphi_1, \langle \Psi, \mathcal{A} \rangle}(\pi_{\mathcal{C}_s^1}(\sigma)) \circ_i Mon_{\varphi_2, \langle \Psi, \mathcal{A} \rangle}(\pi_{\mathcal{C}_s^2}(\sigma)) \\ (2) & \forall_{\langle \diamond_i, \circ_i \rangle \in \mathcal{ST}} . (X \circ_i Y = \top) \Rightarrow (Z \circ_i Y = \top)(resp., for \bot) \end{array}$$

where $\mathcal{C}_s^1 = \{\mathcal{A}^{-1}(\psi) \mid \psi \in (\kappa_{\langle \Psi, \mathcal{A} \rangle}(\varphi_1))\}, \mathcal{C}_s^2 = \{\mathcal{A}^{-1}(\psi) \mid \psi \in (\kappa_{\langle \Psi, \mathcal{A} \rangle}(\varphi_2))\}, X \in \{?, ?_\top, ?_\bot\}$, $Y \in \{\top, \bot\}$ and $Z \in \{\top, \bot, ?_\top, ?_\bot, ?\}$.

Definition 8 presents the notion of a *composition setting*, which is a set of tuples $\langle \diamond, \circ \rangle$, where each $\diamond$ operator is commutative and the two requirements hold. The first requirement states the relation between $\diamond$ and its counterpart $\circ$. Hence, to verify a composed property $\varphi_i \diamond \varphi_j$ we can verify φ_i and φ_j separately, and then we aggregate the results obtained from the verification with the corresponding $\circ$ operator. The second requirement is more complicated, as it concerns the finality of the outcomes obtained through the compose operator ($\diamond$). This second requirement states that, if a final outcome ($\top$ or $\bot$) can be concluded by combining an inconclusive verdict (such as $?_\top$, $?_\bot$, or $?$) with another possible verdict (any verdict in $\mathbb{B}_5$), then the same outcome can be concluded by replacing the inconclusive verdict with a final one. In brief, if a final outcome can be concluded with less information (inconclusive verdict), then it can definitely be concluded with more information (final verdict). This is also related to the finality of the verdict; once a final outcome is obtained, it should never change in the future (only the inconclusive verdicts can change, and eventually become final verdicts).

For example, consider the commutative Boolean operators $\wedge$ and $\vee$. The composition setting $\mathcal{ST} = \{\langle \wedge, \circ^\wedge \rangle, \langle \vee, \circ^\vee \rangle\}$ assigns functions to $\wedge$ and $\vee$ that could be defined as in Table 1.

On the left, we have a possible aggregation function for the $\wedge$ operator; while on the right, a possible aggregation for the $\vee$ operator. Except for the standard cases ($\top$ and $\bot$), the other cases represent a possible aggregation of the Boolean verdicts. Specifically, all the cases involving $?_\top$, $?_\bot$, or $?$ can be defined differently depending on the context.

It is important to note that the choices made in Table 1 are subjective (but preserve the second requirement of Definition 8), and for different scenarios we can have different definitions for the same commutative operators. For instance, $?_\top \circ^\wedge ?_\bot$ is mapped to $?_\bot$, because we decided to give more importance to negative results for the $\circ^\wedge$ operator.

Nonetheless, we might as well define $\circ^\wedge$ to map $?_\top \circ^\wedge ?_\bot$ to $?_\top$, if the scenario would gain from a more optimistic approach.

4.2 Compositional multi-model predictive monitor

Using the composition setting (Definition 8), we know which operators ($\diamond$) can be used to guide the property decomposition into multiple monitors and how we can aggregate ($\circ$) their corresponding verdicts. The derived composition monitor, presented in Definition 9, shows how such a compositional evaluation of properties can be obtained using the same assumptions and engineering steps required by the centralised approach (Definition 7).

Definition 9 (*Composition Monitor*) Let S be a system with components $\mathcal{C} = \{\mathcal{C}_1, \dots, \mathcal{C}_n\}$, alphabet $\Sigma = \Sigma_{\mathcal{C}_1} \cup \dots \cup \Sigma_{\mathcal{C}_n}$, and multi-model $\langle \Psi, \mathcal{A} \rangle$. Let φ be a property and $\mathcal{ST}$ a composition setting. Then, a composition monitor is a function $CoMon_\varphi : tr(\Sigma) \to \mathbb{B}_5$, and is defined as follows:

1. if $\varphi = \varphi_i \diamond \varphi_j$ and $\langle \diamond, \circ \rangle \in \mathcal{ST}$, then

$$CoMon_{\varphi_i \diamond \varphi_j}(\sigma) = CoMon_{\varphi_i}(\pi_{\mathcal{C}^i_s}(\sigma)) \circ CoMon_{\varphi_j}(\pi_{\mathcal{C}^j_s}(\sigma))$$

   where $\mathcal{C}^i_s = \{\mathcal{A}^{-1}(\psi) \mid \psi \in (\kappa_{\langle \Psi, \mathcal{A} \rangle}(\varphi_i))\}$ and $\mathcal{C}^j_s = \{\mathcal{A}^{-1}(\psi) \mid \psi \in (\kappa_{\langle \Psi, \mathcal{A} \rangle}(\varphi_j))\}$
2. otherwise,

$$CoMon_\varphi(\sigma) = Mon_{\varphi, \langle \Psi, \mathcal{A} \rangle}(\sigma)$$

Theorem 2 *Let S be a system with components $\mathcal{C} = \{\mathcal{C}_1, \dots, \mathcal{C}_n\}$, and alphabet $\Sigma = \Sigma_{\mathcal{C}_1} \cup \dots \cup \Sigma_{\mathcal{C}_n}$. Let ψ be its single model, let $\langle \Psi, \mathcal{A} \rangle$ be its multi-model, and let $\mathcal{ST}$ be a composition setting. Then, for any property φ the following implications hold:*

$$\begin{array}{ll} (1) & \forall_{\sigma \in tr(\Sigma)} . CoMon_\varphi(\sigma) = \top \ \Rightarrow \ Mon_{\varphi, \psi}(\sigma) = \top \\ (2) & \forall_{\sigma \in tr(\Sigma)} . CoMon_\varphi(\sigma) = \bot \ \Rightarrow \ Mon_{\varphi, \psi}(\sigma) = \bot \end{array}$$

Proof The proof is by induction over the compositional structure of φ.

Base case: Let us consider the case where φ is non-compositional. By Definition 9, the composition monitor is defined as $CoMon_\varphi(\sigma) = Mon_{\varphi, \langle \Psi, \mathcal{A} \rangle}(\sigma)$. Hence, the base case follows by Theorem 1.

Induction step: Let $\varphi = \varphi_i \diamond \varphi_j$ such that $CoMon_{\varphi_i}(\pi_{\mathcal{C}^i_s}(\sigma)) = Y$ with $Y \in \{\top, \bot\}$. By IH, it follows that $Mon_{\varphi_i, \psi}(\pi_{\mathcal{C}^i_s}(\sigma)) = Y$. By Definition 9, we have $CoMon_{\varphi_i \diamond \varphi_j}(\sigma) = CoMon_{\varphi_i}(\pi_{\mathcal{C}^i_s}(\sigma)) \circ CoMon_{\varphi_j}(\pi_{\mathcal{C}^j_s}(\sigma))$ By Definition 8, it follows that both $CoMon_{\varphi_i \diamond \varphi_j}(\sigma) = Y$ and $Mon_{\varphi_i \diamond \varphi_j, \psi}(\sigma) = Y$. Therefore,

$$CoMon_{\varphi_i \diamond \varphi_j}(\sigma) = Y \ \Rightarrow \ Mon_{\varphi_i \diamond \varphi_j, \psi}(\sigma) = Y$$

with $Y = \{\top, \bot\}$. □

Observation 3 For Composition Monitors, the opposite implications of the ones presented in Theorem 2 do not hold (similarly to Multi-Model Predictive Monitors). This can

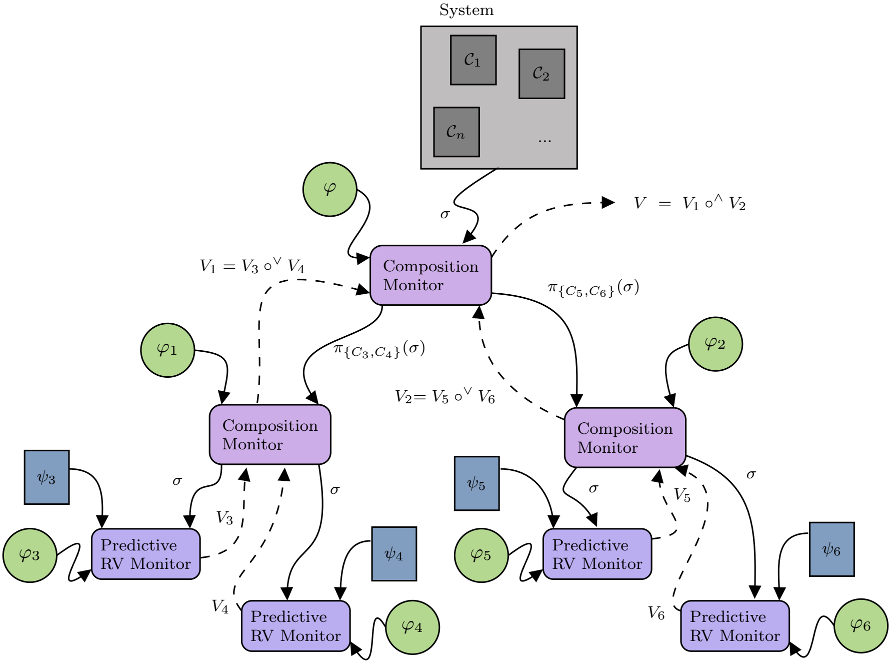


Fig. 4 Composition monitor example for property $\varphi = \varphi_1 \wedge \varphi_2$, with $\varphi_1 = \varphi_3 \vee \varphi_4$ and $\varphi_2 = \varphi_5 \vee \varphi_6$. Here, for each φ_i, we have that $\chi(\kappa_{\langle\Psi,\mathcal{A}\rangle}(\varphi_i)) = \psi_i$ and $\mathcal{A}^{-1}(\psi_i) = \mathcal{C}_i$

be observed by the fact that the over-approximation generated in the leaves of the composed property can denote more traces than needed (w.r.t. the precise single-model representation ψ). Because of this, it may be that the composition monitor using over-approximation cannot conclude a final verdict, while the corresponding single-model version can.

A composition monitor is essentially a relay, which is used to aggregate and propagate the verdicts of predictive monitors. In particular, it extracts the composed properties from the structure of φ. This definition is based on the observation that a compositional operator stops being compositional when nested inside a non-compositional one. As long as we encounter properties composed of only compositional operators (case 1) we can decompose the monitor evaluation; when we encounter a non-composed property (case 2) then we stop, and revert to the multi-model definition.

Using these composition monitors we obtain compositional and predictive RV; where, starting from a composed property, we can generate a set of monitors to propagate and evaluate the different sub-properties. We assume the presence of a global trace, σ, which can be propagated from the top-level composition monitors down to the predictive monitors. From a practical perspective, this implies the use of an entity which gathers the information from the components and creates a global trace. Since we are dealing with systems that have a global clock, such an entity should be relatively straightforward to obtain.

Figure 4 illustrates how the composition monitors work. Here, we have a property, φ, to verify, which is the composition of two sub-properties, φ_1 and φ_2 which, in turn,

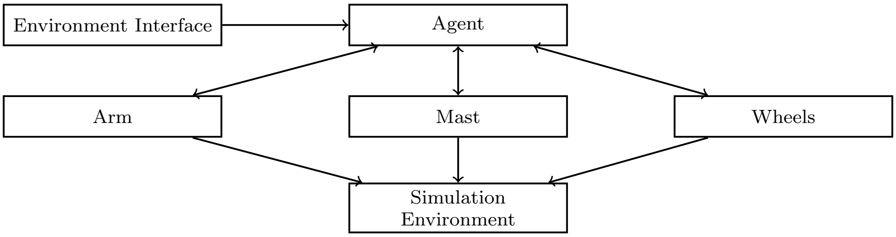


Fig. 5 Modular architecture of the Curiosity rover where arrows indicate data flow between distinct system components

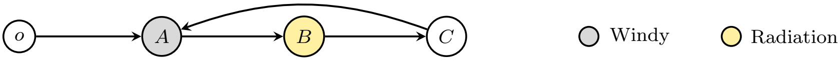


Fig. 6 The Curiosity begins at the origin, o, and then visits the waypoints A, B and C in whichever order is safe. We indicate waypoints with high levels of wind (grey) and radiation (yellow). (Color figure online)

are the composition of the sub-properties φ_3 with φ_4, and φ_5 with φ_6, respectively. This composition is obtained using the composition setting $\mathcal{ST}$ presented previously, with which we can create conjunctions and disjunctions of our properties. The verdicts obtained by the evaluation of the leaf properties (the non-composed ones), are propagated bottom-up from the predictive, to the composition monitors. Each composition monitor applies the corresponding aggregation function according to the composition settings, $\mathcal{ST}$, to generate the verdict to propagate. This process goes up to the root composition monitor, corresponding in this scenario to $CoMon_{\varphi_1 \wedge \varphi_2}$, which will propagate the result to the end user.

Thanks to the composition setting, we are able to split the verification of a property, φ, across multiple monitors. Rather than the centralised version presented at the beginning of Sect. 4, the composition approach does not require all of the models to be combined, which can be both expensive to compute and require a non-negligible amount of memory. Naturally, with the assumption of getting smaller sets of models when applying the contextualise function (Definition 4) on the sub-properties. Otherwise, the result of the composition monitor is still sound, but there might not be any gain in the verification process; since the verification of sub-properties would probably not be concluded in advance of a centralised monitor. Even though this is not relevant from a more theoretical perspective, it might have practical implications. Thus, at the implementation level, one could check whether the contextualise function returns smaller subsets of models (or not). If it does not, the composition approach could be skipped, since it would not bring any improvement over the centralised approach.

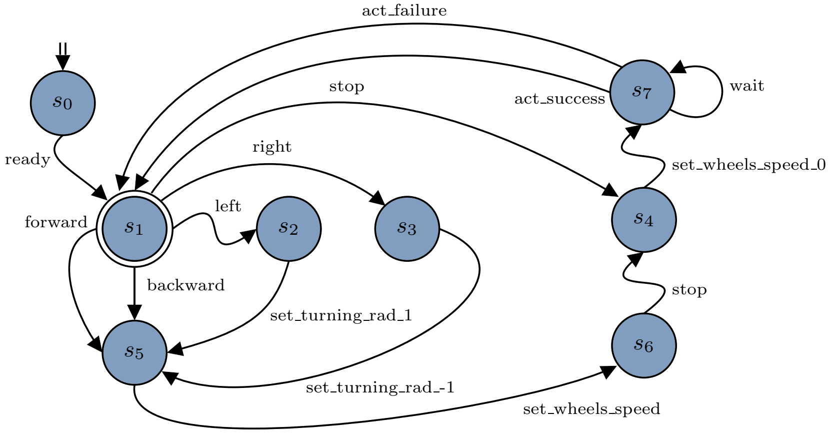


Fig. 7 BA model of the *Wheels*, ψ_{wheels}. Once it is "ready" it begins to execute the actions that it receives

5 Example: Mars Curiosity rover

In this section, we present an illustrative example of a simulation of the Mars Curiosity rover, which is an autonomous robotic system for Martian surface activities that was previously modelled in [10]. This simulation is implemented as a system of modules as shown in Fig. 5. This particular system is composed of: an *Environment Interface* that processes and sends input from sensors to an *Agent* which is responsible for decision making and instructs the *Arm*, *Mast* and *Wheels*. The output is sent to the *Simulation Environment*.

5.1 Overview

The Curiosity carries out an inspection mission where it autonomously patrols particular waypoints on a given topological map of an area on Mars. The Curiosity starts at the origin waypoint, o, and from here, the rover must patrol three other waypoints (A, B, and C).

The Curiosity autonomously navigates between the waypoints in the following order: ($o \rightarrow A \rightarrow B \rightarrow C \rightarrow A \rightarrow \ldots$), as shown in Fig. 6. However, if one of the waypoints is experiencing high levels of radiation then the rover should skip it until the radiation has reduced to a safe level. For data collection, the mast and arm should be open but it is unsafe to do so in windy conditions.

It is important to apply robust verification techniques to systems composed of multiple sub-systems, in order to improve confidence that the system is trustworthy. Particularly when these systems are to be deployed in safety- and/or mission-critical domains. Using RV here is useful because we can add one or more monitors while the system is running to check that it is behaving as expected. This is achieved by formalising the properties that the system has to preserve during its execution.

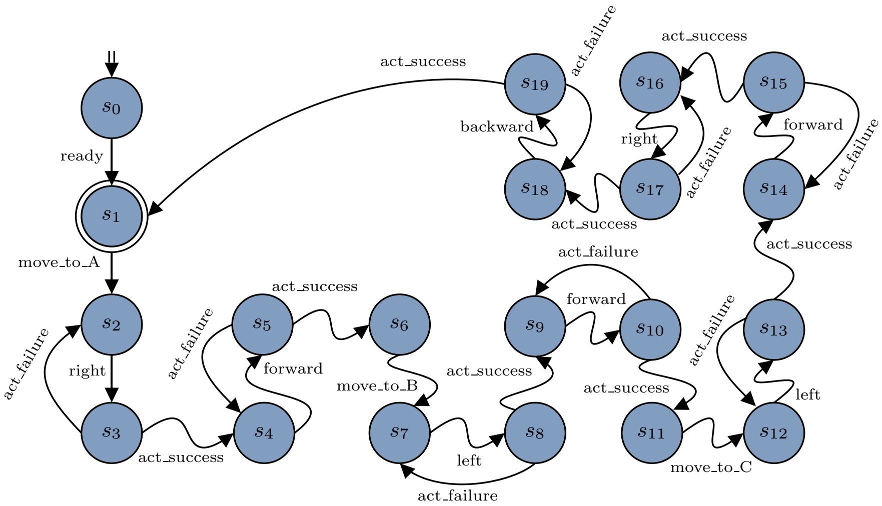


Fig. 8 BA model of the *Agent* ψ_{agent}. Once it receives “ready” it sends the action commands to be executed to the *Wheels*

5.2 Applying our approach

Following the engineering steps presented in the previous sections, we demonstrate how to apply an existing single-model predictive monitor to the Curiosity example, when we do not have a *single* model of the system but rather a collection of models, one for each component. The steps 1-4 below represent the engineering steps (Sect. 3) that are required, while steps 5(a) and 5(b) show how to instantiate the centralised and composition monitors respectively (Sect. 4).

Step 1: Formalising the models. Now we must choose a formalism to represent the models of the set of components, $\mathcal{C}$. We have chosen to represent the models as BA and so Figs. 7 and 8 depict the BA for the *Wheels* and *Agent* components, respectively. Intuitively, Fig. 7 describes how the wheels component of the rover works. It starts by producing the event *ready*, meaning that the wheels component is ready to receive instructions Then, we find different possibilities: the wheels can be asked to turn *forward*, *backward*, *left*, or *right*. Depending on the instruction that was received, some additional command may be required (like *set_turning_rad*, to change the direction of the wheels to left or right). After that, the speed of the turning wheels is set, making them move at a certain fixed rate. Finally, the wheels stop (by setting the speed to zero), and the process may restart.

The same reasoning as above also goes for Fig. 8, which defines the agent controlling the robot. Here, we also have events concerning the state of the agent, as well actions concerning the movement of the rover. Distinctly from Fig. 7, here the sequence of actions executed by the agent are checked, and in each step the agent controls the outcome of the performed action. After the sequence of different actions (which correspond to the rover's mission) has been completed, the agent may start again with the same patrolling mission.

We chose BA because they satisfy our restrictions (Sect. 2). In particular, given a BA, ψ, we can: (1) check if a trace, σ, belongs to ψ, i.e. $\sigma \in [\![\psi]\!]$ and, (2) the $\otimes$ operator exists, and can be obtained in two steps by first translating φ into its equivalent BA, B_φ, (see [36]), and then by taking the product with the model, i.e. $\varphi \otimes \psi = B_\varphi \times \psi$ (standard product of BA).

Considering the rover, with its components and models (Figs. 7 and 8), we now define its corresponding multi-model as $\langle \Psi, \mathcal{A} \rangle$, where $\Psi = \{\psi_{wheel}, \psi_{agent}\}$, $\mathcal{A}(Wheels) = \psi_{wheels}$ and $\mathcal{A}(Agent) = \psi_{agent}$.

Next, we formalise the properties that we wish to monitor.

Step 2: Formalising the properties. We use LTL as the formalism for specifying the properties because it complies with the restrictions imposed by our approach (Sect. 2). Specifically, given an LTL property, φ, we can: (1) decide whether a trace, σ, satisfies φ i.e. $\sigma \models \varphi$ and, (2) construct its negation, $\overline{\varphi}$.

In the context of our example, we are interested in specifying properties about the components of the rover, such as its *Wheels*, *Arm*, *Mast* and *Agent*. For brevity, we focus on the *Wheels* and the *Agent*. In particular, we require that after the *Agent* sends an action to the *Wheels* component then the wheels will not stop before turning left (i.e. set the turning radius to 1). We do not expect this property to be satisfied by each rover execution, but it allows us to demonstrate how the models can help to predict the monitor's final outcome. We formalise this property in LTL, using the standard "until" operator ($\mathcal{U}$), as:

$$\varphi_1 = (action \implies (\neg stop)\, \mathcal{U}\, set\_turning\_rad\_1)$$

where $action \in \{forward, left, right, backward\}$. This property refers to the *Wheels* and so only the model of the *Wheels* is required for monitoring.

The rover's high-level decision making is carried out in a single component, the *Agent*, which decides on the actions to be executed. Each action is then sent to the respective component for execution. An appropriate property to verify about the *Agent* is:

$$\varphi_2 = (\Box(move\_to\_A \implies (\neg action\_fail)\, \mathcal{U}\, move\_to\_B))$$

where $\Box$ is LTL's standard "globally" operator. This property specifies that each action requested by the agent between waypoint A and B never fails. Also in this case, the property only refers to the *Agent* so only the model of the *Agent* component is required.

Next, in order to use the properties and the models to create the predictive monitors, we must define the $\kappa_{\langle \Psi, \mathcal{A} \rangle}$ and χ functions.

Step 3: Defining the contextualise function. The $\kappa_{\langle \Psi, \mathcal{A} \rangle}$ function (Definition 4) is straightforward; given a property, φ, it extracts the set of events involved in the property (denoted Σ_φ), and returns the set of models $\Psi' \subseteq \Psi$ such that if $\exists!_{\psi \in \Psi}.\Sigma_\varphi \subseteq \Sigma_\psi$, then $\Psi' = \{\psi\}$; otherwise, $\Psi' = \{\psi_i \mid \Sigma_{\psi_i} \cap \Sigma_\varphi \neq \emptyset\}$. Where "$\exists!$" is read as "exists exactly one". With respect to the above Curiosity properties, we have that:

$$\kappa_{\langle \Psi, \mathcal{A} \rangle}(\varphi_1) = \{\psi_{wheels}\} \text{and} \kappa_{\langle \Psi, \mathcal{A} \rangle}(\varphi_2) = \{\psi_{agent}\}$$

Step 4: Defining the combination function. The χ function (Definition 6) in the context of BA composition has already been presented in Fig. 3. In automata theory, the combination of automata is often referred to as parallel composition.

Step 5(a): Instantiating a centralised multi-model monitor. Now that we have completed all of the engineering steps outlined in Sect. 3, we have all that we need to port an existing single-model predictive monitor into a multi-model scenario. Since we decided to use LTL and BA, we first instantiate the definition of single-model monitor. Due to the restrictions that we have imposed on the formalism, we know that such an instantiation exists. In our specific scenario, an instantiation of a single-model predictive monitor $Mon_{\varphi,\psi}$ (Definition 2) for LTL properties using BA as models can be derived from [39]. We follow the

Table 2 The outcomes of the various monitors for an example trace

Trace	Non-Predictive		Predictive		Centralised	Compositional
σ	Mon_{φ_1}	Mon_{φ_2}	$Mon_{\varphi_1,\psi_{wheels}}$	$Mon_{\varphi_2,\psi_{agent}}$	$Mon_{\varphi_3,\langle\Psi,\mathcal{A}\rangle}$	$CoMon_{\varphi_3}$
ϵ	$?_\top$	$?_\top$	$?_\top$	$?_\top$	$?_\top$	$?_\top \circ^\wedge ?_\top$
ready	$?_\bot$	$?_\top$	$?_\bot$	$?_\top$	$?_\bot$	$?_\top \circ^\wedge ?_\bot$
ready • *forward*	$?_\bot$	$?_\top$	$\bot$	$?_\top$	$?_\bot$	$\bot \circ^\wedge ?_\top$

work on constructing Runtime Verification Linear Temporal Logic (RV-LTL) monitors in [7] to distinguish between the ? and $\{?_\top, ?_\bot\}$ outcomes.

An example of a property that concerns more than one model at the same time is the composition $\varphi_3 = \varphi_1 \wedge \varphi_2$, which is derived from the composition of the two properties that were described above, for the *Wheels* and *Agent* respectively. Intuitively, φ_3 says that, after the *Agent* sends an action to the *Wheels* component, the wheels will not stop before turning left, **and** all of the actions sent by the agent between waypoints A and B never fail. To predictively monitor φ_3, we require a model of the two components. This model does not exist explicitly but it can be derived using the $\kappa_{\langle\Psi,\mathcal{A}\rangle}$ and χ functions. Next, we instantiate the multi-model predictive monitor (Definition 7) as: $Mon_{\varphi_3,\langle\Psi,\mathcal{A}\rangle} = Mon_{\varphi_3,\chi(\kappa_{\langle\Psi,\mathcal{A}\rangle}(\varphi_3))}$; where we extract the context from φ_3, which is $\kappa_{\langle\Psi,\mathcal{A}\rangle}(\varphi_3) = \{\psi_{wheels}, \psi_{agent}\}$, and we create the resulting parallel combination automaton $\psi_c = \psi_{wheels}||\psi_{agent}$. Intuitively, ψ_c corresponds to the decision-making model ψ_{agent}, where states $s_3, s_5, s_8, s_{10}, s_{13}, s_{15}, s_{17}, s_{19}$ are expanded into ψ_{wheels}'s states for each specific action. Since the two models share the majority of events, ψ_c does not offer a high level of parallelism. The *Agent*, after instructing the *Wheels* to execute an action, waits for the corresponding outcome (success or failure).

Step 5(b): Instantiating composition monitors. As shown in Fig. 3, the composition of automata may suffer from state space explosion. It is thus preferable to have, when possible, a more compositional approach, where we may use the models without having to combine them.

Recall Definition 9, which shows how we can define the predictive monitor for φ_3 as the composition of the two predictive monitors for φ_1 and φ_2. Since φ_1 and φ_2 each only refer to a single model then the corresponding predictive monitors do not require the combination of models. This brings us to the following instantiation:

$$CoMon_{\varphi_3}(\sigma) = Mon_{\varphi_1,\psi_{wheels}}(\pi_{Wheels}(\sigma))\circ^\wedge Mon_{\varphi_2,\psi_{agent}}(\pi_{Agent}(\sigma))$$

5.3 Discussion

Table 2 contains potential outcomes for each of the monitor functions instantiated in this section. From left to right, the columns contain the prefix of an observed trace, σ, the non-predictive LTL monitors for φ_1 and φ_2, the predictive monitors for φ_1 and φ_2 using ψ_{wheels} and ψ_{agent} for prediction respectively, the centralised multi-model monitor for φ_3 and finally, the composition monitor for φ_3. The rows capture how the monitors behave according to the respective prefix evolution, first when the trace is empty, ϵ, then when it contains only the *ready* event, and finally when it also contains *forward*. We can see how $Mon_{\varphi_1,\psi_{wheels}}$ concludes the violation of its non-predictive counterpart; such anticipation is obtained thanks to ψ_{wheels}, which informs the monitor that there is no continuation after

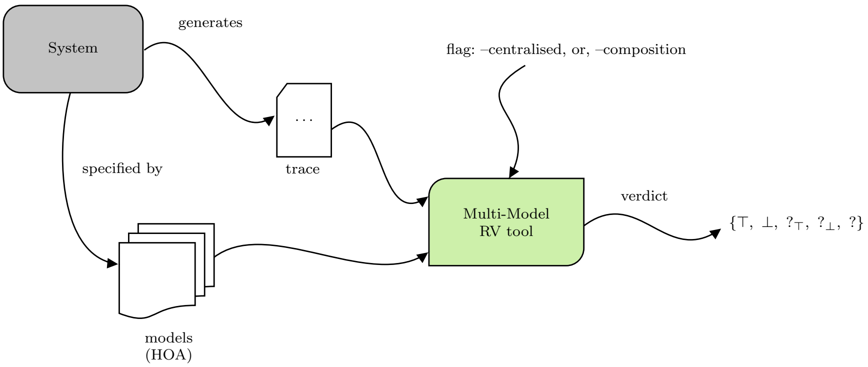

System
generates
flag: –centralised, or, –composition
...
trace
specified by
Multi-Model
RV tool
verdict
$\{\top, \bot, ?_\top, ?_\bot, ?\}$
models
(HOA)

Fig. 9 Overview of the implemented tool

forward where φ_1 can be satisfied, because after ψ_{wheels} consumes *forward*, there is no way to observe *set_turning_rad*_1 without stopping the wheels first (Fig. 7). This conclusive outcome is not obtained inside the centralised multi-model monitor, where we use the global model of the system, since φ_3 requires the combination of both ψ_{wheels} and ψ_{agent} due to it having events belonging to both components.

More specifically, the centralised monitor cannot conclude a final outcome because there exists a continuation after *forward* that satisfies φ_1. Such a trace corresponds to the execution of the *Agent*, while the *Wheels* component remains idle. This is obtained under the assumption of an absence of fairness in the system, which means that a component can be indefinitely delayed. In such case, the resulting infinite trace belonging to the ψ_{agent} model trivially satisfies φ_1. Note that, Definition 6 does not require fairness in general, but, if needed, such a requirement can be added. Finally, following its definition, we obtain the conclusive outcome with the composition monitor, since according to Table 1 the result of such aggregation is $\bot$.

6 Evaluation

To evaluate our solution, we have developed a prototype which implements all of the engineering steps presented in Sect. 3, as well as the single- and multi-model predictive runtime monitors (see Sect. 4).

6.1 Implementation

Figure 9 presents and an overview of our prototype tool[6], which implements the entire engineering process presented in this paper. The tool provides a proof of concept by applying our approach to LTL properties and Büchi automata. However, the theory presented in

[6] Prototype: `github.com/AngeloFerrando/MultiModelPredictiveRuntimeVerification`

this paper is general and the resulting engineering process can be ported to other formalisms for both properties and models.

Our tool is implemented in Python, and uses the Spot library[7] [15], which is a C++14 library for LTL, ω-automata manipulation and model checking. Thanks to this library, a predictive monitor (see Definition 2) is relatively straightforward to implement. As shown in [39], it is enough to transform an LTL property (or its negation) given as input into an equivalent Büchi automaton, and then to compute the product of the latter with the model of the system. Both LTL properties and Büchi automata are supported natively by Spot. Specifically, Spot expects a Hanoi Omega-Automata (HOA)[8] file, one of the most widely used formats for representing automata-like structures, which makes our tool highly reusable. If an alternative to Spot is needed, then any of the algorithms presented in [12, 13, 18, 35] can be used.

In our tool, the transformation to a Büchi automaton and its product are completely handled by Spot, which uses state of the art algorithms. Moreover, thanks to the Python bindings to C++, the performance is not compromised, since the most time-demanding computations are directly executed on the machine (not interpreted by Python). Once the product is obtained, a predictive monitor can be implemented directly by checking if the language recognised by the product is empty (or not). If the product of the property with the model does not recognise any trace (its language is empty), then it means that there are no continuations which belong to the model and satisfy the property. Thus, we can conclude that the property has been violated (the second case of Definition 2 holds).

Symmetrically, the same reasoning can be followed with the negation of the property; if the product of the negation of the property with the model recognises the empty language, then it means we cannot find any continuation which belongs to the model and violates the property. Thus, we can conclude the property is satisfied (the first case of Definition 2 holds). If both the products recognise at least one trace (*i.e.*, are not empty), it means we have at least one continuation which belongs to the model that satisfies (resp., violates) the property, and we find ourselves in an inconclusive case (?). To discretise ? into $?_\top$ or $?_\bot$, it is enough to consider whether the current observed prefix is accepted (resp., not accepted) by the Finite-State Machine (FSM) generated considering LTL with a finite semantics (fLTL [29]). Exactly as it has been done with RVLTL [6], which corresponds to the implementation of a standard 4-valued monitor (see Definition 1) for LTL properties.

The remaining engineering steps have been implemented directly in Python. Specifically, the contextualisation (see Definition 4), the projection (see Definition 5), and the combination (see Definition 6) functions have been implemented as Python functions. The first two are a trivial porting of the corresponding definitions. In fact, the contextualise function simply looks for the events used in an LTL property, and returns the set of models that contain at least one such event (i.e. the models of interest for predicting the events considered in the property). The projection function is simply a filter, which given a trace of events and a model as input, returns the trace deprived of the events not belonging to the model. The combination function is less trivial, but as we showed in Fig. 3, it can be obtained by interleaving the automata. In particular, this has been obtained by extending the notion of a product in Spot, where instead of considering only

[7] Spot Library: https://spot.lrde.epita.fr/

[8] http://adl.github.io/hoaf/

the events belonging to the intersection of the automata (as the product does), all transitions are considered (and combined).

By applying the engineering steps from Sect. 3, we implemented the multi-model predictive monitor. Specifically, this has been achieved by creating a Python class that handles both Definition 7 and 9. In fact, depending on the user's choice (via command line arguments), the resulting monitor is constructed following one of these two definitions. If the user wants to apply centralised (*i.e.*, single-model) predictive runtime verification, then the Python object will first combine all of the models given as input (using the combination function), and will then create a predictive monitor.

Algorithm 1 reports, in pseudo-code, the implementation of Definition 7. Most of the instructions in this algorithm correspond to notions presented previously in this section. On lines 1 and 2, the LTL property φ and its negation $\neg\varphi$ are transformed into Büchi Automata (using the Spot library). Then, since we are in a multi-model scenario, we need to contextualise the property φ w.r.t. its models. This is achieved on line 3, by applying the contextualise function (as presented in Definition 4). Then, the resulting extracted models are combined on line 4 using the combination function (as presented in Definition 6). The resulting Büchi Automaton denotes the over-approximation of the system. This step is, again, achieved using the Spot library. After that, we loop over the trace of events.

Algorithm 1: Centralised Multi-Model Predictive Monitor

```
   Data: a property $\varphi$, a multi-model $\langle \Psi, \mathcal{A} \rangle$, and a trace $\sigma$
   Result: a Boolean verdict according to Definition 7
1  $B_{\varphi} = translate(\varphi)$;                                   /* Spot */
2  $B_{\neg\varphi} = translate(\neg\varphi)$;                           /* Spot */
3  $ctxt = \kappa_{\langle \Psi, \mathcal{A} \rangle}(\varphi)$;                   /* Definition 4 */
4  $B_{\psi} = \chi(ctxt)$;                                     /* Definition 6 */
5  while $\sigma \neq \epsilon$ do
6      $ev = pop(\sigma)$;                /* extracts first event from $\sigma$ */
7      $B_{\varphi} \overset{ev}{\rightarrow} B'_{\varphi}$;              /* move inside $B_{\varphi}$ consuming $ev$ */
8      $B_{\varphi} = B'_{\varphi}$;                   /* store new result into $B_{\varphi}$ */
9      $B_{\neg\varphi} \overset{ev}{\rightarrow} B'_{\neg\varphi}$;          /* move inside $B_{\neg\varphi}$ consuming $ev$ */
10     $B_{\neg\varphi} = B'_{\neg\varphi}$;               /* store new result into $B_{\neg\varphi}$ */
11     $B_{\psi} \overset{ev}{\rightarrow} B'_{\psi}$;              /* move inside $B_{\psi}$ consuming $ev$ */
12     $B_{\psi} = B'_{\psi}$;                   /* store new result into $B_{\psi}$ */
13     if $B_{\varphi} \times B_{\psi} = \emptyset$;          /* product of Büchi Automata */
14      then
15          return $\bot$
16     if $B_{\neg\varphi} \times B_{\psi} = \emptyset$;         /* product of Büchi Automata */
17      then
18          return $\top$
19 if $\sigma \in FSM_{\varphi}$;                  /* FSM for fLTL semantics [29] */
20  then
21      return $?_{\top}$
22 else
23     return $?_{\bot}$
```

For each iteration of the loop (starting on line 5), the head of the trace (*ev*) is removed and used to update the Büchi Automata. This update is performed by using the transition functions. Starting from the current initial state, the transition that expects the *ev* is consumed, and the state reached through such transition is assigned as the new initial state. This is done for all automata, and it is necessary to keep track of the current trace σ that has been analysed. Since the loop considers only one event per iteration, the automata need to be updated to remember which events have already been observed. Without this update phase, the automata would always consider the current *ev* as the initial event of σ. By moving inside the automata and keeping track of which events have already been observed, we can incrementally check σ over the automata.

Once this update phase is completed, as we mentioned earlier in the section, it is sufficient to calculate the product of the automata. If the product of the Büchi Automaton B_{φ} with the Büchi Automaton B_{ψ} is empty (line 13), then, it means no future continuation belonging to ψ satisfies φ. Thus, we can return $\bot$ (*i.e.*, violation). Correspondingly, if the product with the Büchi Automaton $B_{\neg\varphi}$ is empty, then, it means no continuation belonging to the model violates φ. Thus, we can return $\top$ (*i.e.*, satisfaction). Otherwise, the loop continues with the following event in the trace σ, until the trace is empty. If that happens, then the only thing left to check is if the current trace σ at least satisfies the property φ considering a finite semantics. If that is the case, then we can conclude

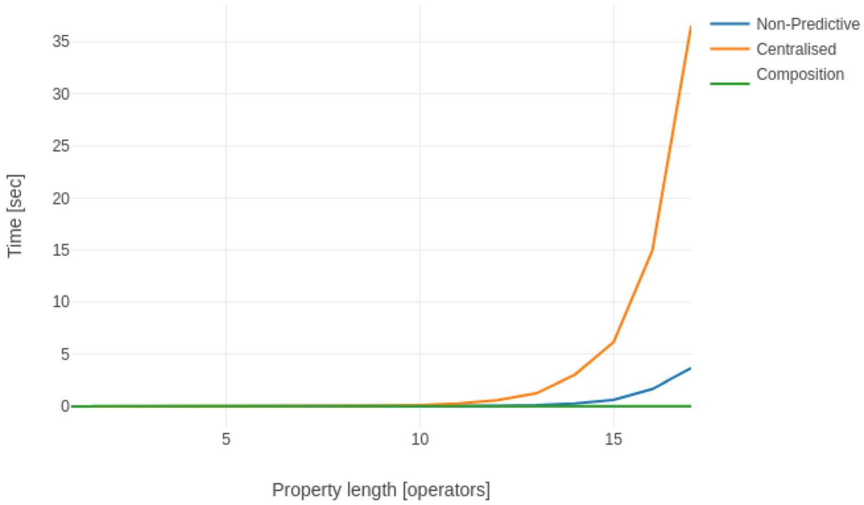


Fig. 10 Time required for synthesising monitors. This chart plots the relationship between time required (y-axis) and property length (x-axis) for each of the three kinds of monitor synthesis. It shows that the time taken for non-predictive and centralised monitors (blue and orange) grows exponentially as the size of the property (*i.e.* its number of operators) increases. Instead, the time taken for composition monitor (green) remains fairly constant as property length increases. Also, non-predictive was faster than centralised. (Color figure online)

$?_\top$; since the trace is currently satisfying the property. Otherwise, we conclude $?_\perp$; since the trace is currently violating the property. This last step is obtained by generating a FSM for LTL considering its finite semantics (as presented in [29]).

Otherwise, if the user wants to apply compositional (*i.e.*, multi-model) predictive runtime verification, then the Python object will decompose the property (case 1 in Definition 9) until it encounters a non-composed property; when this happens, the object combines the contextualised models and creates the resulting predictive monitor (case 2 in Definition 9). Since this has been implemented recursively, the links amongst the different generated monitors are then easily created. These links are used to propagate the partial outcomes of the different monitors (as shown in Fig. 4) and are encapsulated inside the Python object (whose pseudo-code is reported in Algorithm 2).

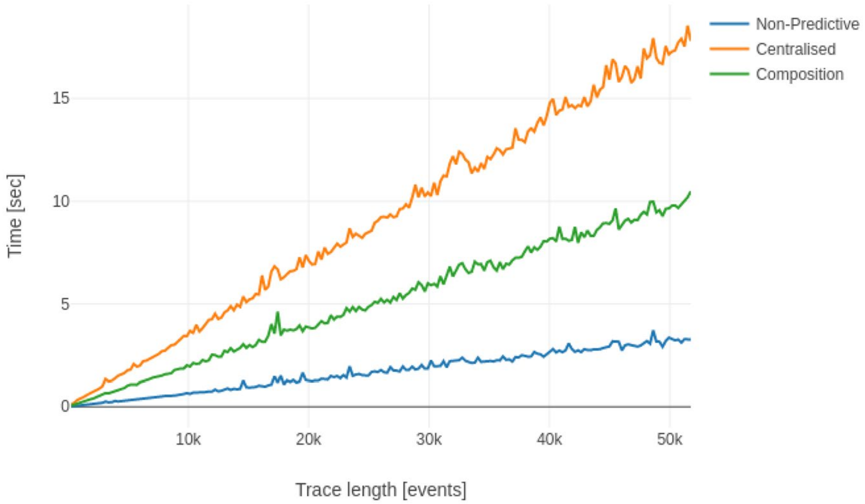


Fig. 11 Time required for performing runtime verification with the monitors synthesised in Fig 10. This chart plots the relationship between time required (y-axis) and trace length (x-axis) for each of the three kinds of monitor. It shows that the time taken for the three monitors is linear with respect to the trace length. Also, non-predictive was faster than the two predictive ones, and the execution time of the composition and centralised approaches is similar

```
Algorithm 2: Composition Monitor
  Data: a property $\varphi$, a multi-model $\langle \Psi, \mathcal{A} \rangle$, and a trace $\sigma$
  Result: a Boolean verdict according to Definition 9
1 if $\varphi = \varphi_i \diamond \varphi_j \ \wedge \ \diamond \in \diamond_c$ then
2     $C_s^i = \{\mathcal{A}^{-1}(\psi) \mid \psi \in (\kappa_{\langle \Psi, \mathcal{A} \rangle}(\varphi_i))\}$;
3     $C_s^j = \{\mathcal{A}^{-1}(\psi) \mid \psi \in (\kappa_{\langle \Psi, \mathcal{A} \rangle}(\varphi_j))\}$;
4     $res_i$ = Algorithm 2 $(\varphi_i, \langle \Psi, \mathcal{A} \rangle, \pi_{C_s^i}(\sigma))$;
5     $res_j$ = Algorithm 2 $(\varphi_j, \langle \Psi, \mathcal{A} \rangle, \pi_{C_s^j}(\sigma))$;
6     return $res_i \circ res_j$
7 else
8     return Algorithm 1 $(\varphi, \langle \Psi, \mathcal{A} \rangle, \sigma)$
```

Algorithm 2 is a straightforward implementation of Definition 9, that takes the LTL property to verify φ, the multi-model denoting the system $\langle \Psi, \mathcal{A} \rangle$, and the trace generated by the system execution to analyse σ. It returns the Boolean verdict according to Definition 9. It starts by checking whether the LTL property φ is compositional (line 1). Recall that a binary operator $\diamond$ is compositional if, and only if, it is commutative (*i.e.*, $\diamond \in \diamond_c$). If the property φ is compositional, then the algorithm proceeds by first extracting the components referred to by the two sub-properties φ_i and φ_j of φ (resp., C_s^i and C_s^j in lines 2-3). This is obtained by retrieving models belonging to the context of φ, and applying $\mathcal{A}^{-1}$ to obtain the corresponding components. Intuitively, in line 2 (resp., 3), we extract the set of components C_s^i (resp., C_s^j) which generate events considered by property φ_i (resp., φ_j).

After the components of interest have been extracted, the algorithm recursively calls itself on the two sub-properties (lines 4-5). This recursive call is performed on the projection of σ over the selected components of interest, which is obtained using the project function introduced in Definition 5.

Once the verdicts for the two sub-properties are returned, the algorithm concludes with their combination using $\circ$; which is the operator corresponding to $\diamond$, in the composition setting. Otherwise, if Algorithm 2 reaches a leaf (w.r.t. the structure of the property), meaning φ cannot be decomposed any further, the algorithm concludes by calling Algorithm 1 (line 8). In this way, the algorithm decomposes the verification as far as possible, and when it cannot decompose any more, it performs the multi-model verification.

In both the single- and multi-model scenarios, our prototype tool generates a predictive monitor; be it single or compound. The monitor is then used to verify a trace given as input (as a file). Further implementation details can be found in our GitHub repository[6], along with a short example of use. In the next section, we report the results of the experiments that we obtained by applying our tool to our running example (see Sect. 5).

6.2 Experiments

In our experiments, we focus on two aspects: the monitor synthesis time, and the runtime verification execution time. The first aspect is concerned with how long it takes to synthesise a monitor, given an LTL property and some models as input. The second aspect instead focusses on how long it takes to verify a given trace using a previously synthesised monitor. For both cases, we consider standard (non-predictive), centralised (single-model), and compositional (multi-model) monitors.

In Fig. 10, we report the time required to synthesise the monitors for the Mars Curiosity example with respect to a given input property. As expected, the monitor synthesis time is influenced by the length of the property. The standard explicit algorithm to synthesise a Büchi Automaton from an LTL property is double exponential w.r.t. the length of the property [36]. In our implementation we follow the monitor construction in [8], so we need to translate LTL properties into their equivalent Büchi Automata (as we show in Sect. 6.1, specifically, lines 1-2 in Algorithm 1). Naturally, on the fly algorithms to improve the LTL translation can be used, but this is left for future developments.

In Fig. 10, the exponential nature of the techniques can be seen. Specifically, both the non-predictive and the centralised approaches behave exponentially w.r.t. the length of the property. In our experiments, the length of the property was changed by by adding conjunctions or disjunctions with new LTL properties. For instance, if the property was $\varphi = \Box p$, of length 1, we could generate a new LTL property $\varphi' = \varphi \wedge \Diamond q$, of length 2.

Another important result to note in Fig. 10 is the synthesis time for the composition monitor. Unlike the previous two monitors, it does not grow exponentially w.r.t. the length of the property. Instead, it remains constant. This depends on the decomposition of the property carried out by the composition monitor. According to Definition 9, the composition monitor splits the property and it is recursively defined in terms of its sub-properties (w.r.t. compositional operators). Because of this, when we synthesise a composition monitor from the generated properties, we actually push the automaton translation to only the leaves (*i.e.* the operands). Thus, even though the property length grows, the translation is always performed on smaller properties. Considering the previous example, in the case of non-predictive and centralised monitors, the automaton is synthesised using the entire property $\Box p \wedge \Diamond q$ (of length 2), while in the case of composition monitor, two automata are synthesised, one for $\Box p$ and another for $\Diamond q$ (both of length 1). Naturally, if the properties could not be decomposed (*i.e.*, the main operator is not compositional), then the performance of the composition monitor would be exponential, like the centralised monitor.

Figure 11 shows the time required to verify at runtime a trace of a certain length, by using the monitors synthesised in the previous step. For both non-predictive and predictive monitors we observe linear behaviour. As the length of the trace increases, we find a proportional increment in the time required to verify the traces. However, even though linear, the slope of the times for predictive monitoring is steeper than for non-predictive monitoring. Nonetheless, the total amount of time for verifying a trace of ~50k events is less than ~20 seconds.

In the case of the predictive monitors, we observe different execution times. The composition monitor, performs better than its centralised counterpart. This is because the models used by the centralised monitor become larger (because of the composition of the models of the components) than the models used by the composition monitor. The composition monitor performs better because of these small models.

Another important aspect to keep in mind is that the results reported in Fig. 11 concern a complete analysis of the traces. This means that in all three monitors, all the events in the traces have been analysed. This was enforced by checking properties that could not be determined, neither positively nor negatively, at runtime (*i.e.*, the verdict was always ?). We decided to apply our approach to such worst case scenarios to show how the implementation actually behaves w.r.t. the trace length. However, in a less stressed scenario, it would be possible for the predictive monitors to conclude the verdict in advance, and to perform much better.

In general, the prototype allows us to show the feasibility of our approach, and its efficacy. We do not lose linearity in the predictive case, and the amount of time required to verify our robotic case study is very promising.

7 Related work

As far as we know, our work is the first to tackle the problem of PRV applied to non-monolithic systems. Specifically, it is the first work to re-engineer the standard PRV process, which uses only a single model to predict events; into a multi-model PRV process, where multiple models are used to predict events. In this section, we report in this area.

A survey [22] on RV for distributed and decentralised systems identifies several architectures for monitoring concurrent systems. The survey, defines a *decentralised* system as synchronous and controlled by a global (shared) clock, while a *distributed* system is asynchronous and has no shared clock.

The approach in [17] allows both the monitors and the specified properties to be decentralised. This approach facilitates the monitoring of separate specifications for individual components of the SUA but, crucially, does not tackle the *predictive* element of the work presented here.

In [26], the authors use Signal Temporal Logic (STL) with a semantics that enables monitored behaviour to be compared with its specification, with deviations quantified in both space and time. This comparison provides a measure of how robust a system is, with respect to its specification, instead of merely indicating if it satisfies or violates the specification. This provides a finer-grained measure of how far the system is from obeying its specification. Their approach calculates the deviation "between a signal and a set of signals, defined by the specification" (in STL); whereas our work operates on multiple traces, each from different components of a system. Deshmukh et al. [14] use a similar approach (but with a different measure of distance between system and specification) to check the conformance of Simulink diagrams to a given specification. This, again, takes a single system model, whereas our work caters to multiple models. Both of these approaches are also more closely coupled with a particular formalism than our work, whereas we present formalism-agnostic approach and use LTL as an example instantiation.

Qin and Deshmukh [34] use statistical time-series analysis to predict future satisfaction of an STL property, based on the existing time-series data gathered from the monitored system. This is similar in intent to our work; both approaches attempt to side-step the problem that online monitoring only evaluates events as they happen. Our work requires models of (parts of) the system, whereas theirs constructs a model from observed traces. In both our work and theirs, it is assumed that the system correctly implements the model; but [34] assumes that enough of the trace has been observed to construct a model of the system. The downside of their assumption is that if some events have not been observed yet (events that are crucial to the prediction of the property's satisfaction) the model will be incomplete and thus the prediction will be further away from reality.

Some RV approaches have been developed specifically for particular robotic software systems: for example, RV-BIP [19] for the Behaviour Interaction Priority (BIP) framework; and ROSRV [25] and ROSMonitoring [20] for the Robot Operating System (ROS) middleware. Other approaches are designed to complement the development process for robotic systems. For example, [21], which uses RV to highlight when environmental assumptions used in previous formal verification for an autonomous robotic system is invalidated by real environmental interactions.

In [37], the authors propose a predictive runtime monitoring approach for linear systems with stochastic disturbance. Their approach is based on the construction of a data-driven linear model of the SUA. In [4], the authors present Adaptive RV, which uses multiple monitors but a single probabilistic model of the SUA to perform PRV. The probability of property violation is calculated and used to control the framework's overheads. In principle, their technique should be applicable to multiple models; however, their framework would need to be altered to accommodate this change. A similar approach is presented in [3], where the model is represented as an Hidden Markov Model (HMM) to extend the partially observable paths of the system. All of the previously mentioned works, along with the ones which inspired our work [31, 39], are based on a single-model representation of the SUA; consequently, they are suitable candidates to be engineered using our approach for use with non-monolithic systems.

In order to side-step the problem of creating a model of the SUA the work in [38] generates a call-flow graph for the SUA and uses it to do "speculative execution" to predict the satisfaction or violation of the property. It is claimed that this approach is easier and more consistent than modelling the system by hand. However, call-flow graphs do not provide the same rigour as our approach, which formalises the model of the SUA.

The work in [33] introduces the *notion of temporal testers* (of properties expressed in LTL, Property Specification Language (PSL) and Metric Interval Temporal Logic (MITL) , for example) and show how to combine them compositionally. Even though in principle our contribution and theirs are similar, we accept properties expressed in any formalism that meets our requirements (not just temporal logic properties). We also consider how the system models affect the compositional verification. Nonetheless, it could be useful to analyse the relation between our compositional monitors and the temporal testers in [33] when our contribution is instantiated in for temporal properties.

Colombo and Falcone [11] present a choreographed approach for the decentralised verification of LTL properties. Like we do, they also assume the presence of a global clock to support communication amongst the different monitors. However, they focus explicitly on LTL, and do not consider any form of prediction. It might be possible to extend our approach, taking into account some of the ideas in [11], specifically when our approach is instantiated in temporal logic. This could provide a more decentralised and choreography-oriented approach, which may improve both the performance and distribution of the composition of properties.

8 Conclusions and future work

This paper investigated the research question: *How can we apply a predictive monitor when the SUA is composed of multiple components, and each component is described by its own model?* In answer to this, we described the application of predictive RV to modular systems, where instead of a single model to foresee future events, we have a set of models. We also presented the engineering steps that are necessary to bridge the gap between single- and multi-model predictive RV. This process resulted in two different approaches: (1) centralised multi-model monitoring, where we combine the models of the components into a global model, and we verify a property over these components; and (2) compositional multi-model monitoring, where we use composition monitors for each of the models and then compose these monitors in a bottom-up fashion, arriving at a root composition monitor.

The theory behind these two approaches remains formalism-agnostic in specifying both the monitored properties and the models; making them highly adaptable to existing monitor implementations. We also present the restrictions that a chosen formalism must satisfy. To demonstrate how the theory can be applied, we use an example of a simulation of the Mars Curiosity rover to describe the engineering steps in practice. Finally, we close the gap with the theory by developing a tool in Python that implements all of the engineering steps that we present in this paper. Moreover, we show an overview of the tool and the results of the experiments that we obtain by applying it to verify the Mars Curiosity rover example.

Future work involves extending the current implementation to support additional formalisms. Since the theory behind our contribution is general, there is no reason for supporting only a single formalism. Moreover, the extension to additional temporal formalisms, such as MTL [27], pLTL [29], STL [30], and so on, is natural. Indeed, they should be fairly straightforward to implement in the tool, and would bring more advantages to the approach. Other than extending the tool, we are also planning to apply our implementation to case studies with more complex and bigger models; in particular, where the decomposition of the monitors can help distributing the computational workload.

With respect to composition properties and their verification using composition settings, an interesting aspect is the choice of a more optimistic or pessimistic semantics for the $\circ$ operators. In this paper, we have not focussed on the possible implications and reasons for choosing either option, since they are out of the scope of our contribution. In general, this choice could be guided by additional information about the system under analysis. For instance, if the system guaranteed fair execution, this detail could be used to guide a more optimistic semantics, and vice versa. Naturally, this is only one possibility, and other system's features could still be used to enhance the choice of more suitable semantics for the $\circ$ operators.

Funding Open access funding provided by Università degli Studi di Genova within the CRUI-CARE Agreement. Research at Manchester was supported by both the Royal Academy of Engineering, under the Chairs in Emerging Technologies scheme, and the UKRI's TAS Node in Verifiability (EP/V026801).

Data availability Not applicable.

Code availability The GitHub repository containing the implementation of the tool is publicly available at https://github.com/AngeloFerrando/MultiModelPredictiveRuntimeVerification.

Declarations

Conflict of interest Not applicable.

References

1. Alur R, Dill DL (1994) A theory of timed automata. Theor Comput Sci 126:183–235
2. Ancona D, Ferrando A, Mascardi V (2016) Comparing trace expressions and linear temporal logic for runtime verification. In: Theory and practice of formal methods-essays dedicated to Frank de Boer on the Occasion of His 60th Birthday, LNCS, vol 9660, pp 47–64. Springer
3. Babaee R, Gurfinkel A, Fischmeister S (2018) Prevent : a predictive run-time verification framework using statistical learning. In: Software engineering and formal methods, LNCS, vol 10886, pp 205–220. Springer
4. Bartocci E, Grosu R, Karmarkar A, Smolka SA, Stoller SD, Zadok E, Seyster J (2013) Adaptive runtime verification. In: Runtime verification, LNCS, vol 7687, pp 168–182. Springer
5. Bauer A, Leucker M, Schallhart C (2006) Monitoring of real-time properties. In: Foundations of software technology and theoretical computer science, LNCS, vol 4337, pp 260–272. Springer
6. Bauer A, Leucker M, Schallhart C (2007) The good, the bad, and the ugly, but how ugly is ugly? In: Runtime verification, LNCS, vol 4839, pp 126–138. Springer
7. Bauer A, Leucker M, Schallhart C (2010) Comparing LTL semantics for runtime verification. J Logic Comput 20(3):651–674
8. Bauer A, Leucker M, Schallhart C (2011) Runtime verification for LTL and TLTL. ACM Trans Softw Eng Methodol 20(4):1–14
9. Blech JO, Falcone Y, Becker K (2012) Towards certified runtime verification. In: Formal methods and software engineering, LNCS, vol 7635, pp 494–509. Springer
10. Cardoso RC, Farrell M, Luckcuck M, Ferrando A, Fisher M (2020) Heterogeneous verification of an autonomous curiosity rover. In: NASA formal methods symposium, LNCS, vol 12229, pp 353–360. Springer
11. Colombo C, Falcone Y (2016) Organising LTL monitors over distributed systems with a global clock. Formal Methods Syst. Des. 49(1–2):109–158. https://doi.org/10.1007/s10703-016-0251-x
12. Couvreur J (1999) On-the-fly verification of linear temporal logic. In: JM Wing, J Woodcock, J Davies (eds.) FM'99-Formal Methods, World Congress on Formal Methods in the Development of Computing

Systems, Toulouse, France, September 20–24, 1999, Proceedings, Volume I, Lecture Notes in Computer Science, vol 1708, pp 253–271. Springer . https://doi.org/10.1007/3-540-48119-2_16

13. Daniele M, Giunchiglia F, Vardi MY (1999) Improved automata generation for linear temporal logic. In: N Halbwachs, DA Peled (eds.) Computer aided verification, 11th International Conference, CAV ’99, Trento, Italy, July 6–10, 1999, Proceedings, Lecture Notes in Computer Science, vol 1633, pp 249–260. Springer. https://doi.org/10.1007/3-540-48683-6_23
14. Deshmukh JV, Majumdar R, Prabhu VS (2017) Quantifying conformance using the Skorokhod metric. Formal Methods Syst Des 50(2–3):168–206. https://doi.org/10.1007/s10703-016-0261-8
15. Duret-Lutz A, Poitrenaud D (2004) SPOT: an extensible model checking library using transition-based generalized büchi automata. In: D DeGroot, PG Harrison, HAG Wijshoff, Z Segall (eds.) 12th International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS 2004), 4–8 October 2004, Vollendam, pp 76–83. IEEE Computer Society. https://doi.org/10.1109/MASCOT.2004.1348184
16. Eisner C, Fisman D, Havlicek J, Lustig Y, McIsaac A, Campenhout DV (2003) Reasoning with temporal logic on truncated paths. In: Computer aided verification, LNCS, vol 2725, pp 27–39. Springer
17. El-Hokayem A, Falcone Y (2020) On the monitoring of decentralized specifications: semantics, properties, analysis, and simulation. ACM Trans Softw Eng Methodol 29(1):1–57
18. Etessami K, Holzmann GJ (2000) Optimizing büchi automata. In: C Palamidessi (ed.) CONCUR 2000-concurrency theory, 11th International Conference, University Park, PA, USA, August 22–25, 2000, Proceedings, Lecture Notes in Computer Science, vol 1877, pp 153–167. Springer. https://doi.org/10.1007/3-540-44618-4_13
19. Falcone Y, Jaber M, Nguyen TH, Bozga M, Bensalem S (2011) Runtime verification of component-based systems. In: Software Engineering and Formal Methods, LNCS, vol 7041, pp 204–220. Springer
20. Ferrando A, Cardoso RC, Fisher M, Ancona D, Franceschini L, Mascardi V (2020) ROSMonitoring: a runtime verification framework for ROS. In: towards autonomous robotic systems conference, LNCS, vol 12228, pp 387–399. Springer
21. Ferrando A, Dennis LA, Ancona D, Fisher M, Mascardi V (2018) Recognising assumption violations in autonomous systems verificaion. In: Autonomous agents and multiagent systems, pp 1933–1935. IFAAMAS/ACM
22. Francalanza A, Pérez JA, Sánchez C (2018) Runtime verification for decentralised and distributed systems. In: Lectures on runtime verification, LNCS, vol 10457, pp 176–210. Springer
23. Havelund K, Goldberg A (2005) Verify your runs. In: Verified software: theories, tools, experiments, LNCS, vol 4171, pp 374–383. Springer
24. Hopcroft JE, Ullman JD (1979) Introduction to automata. Theory Addison–Wesley, languages and computation. Longman, London
25. Huang J, Erdogan C, Zhang Y, Moore B, Luo Q, Sundaresan A, Rosu G (2014) ROSRV: runtime verification for robots. In: Runtime verification, LNCS, vol 8734, pp 247–254. Springer
26. Jakšić S, Bartocci E, Grosu R, Nguyen T, Ničković D (2018) Quantitative monitoring of STL with edit distance. Formal Methods Syst Des 53(1):83–112. https://doi.org/10.1007/s10703-018-0319-x
27. Koymans R (1990) Specifying real-time properties with metric temporal logic. Real Time Syst 2(4):255–299
28. Leucker M (2012) Sliding between model checking and runtime verification. In: Runtime verification, LNCS, vol 7687, pp 82–87. Springer
29. Lichtenstein O, Pnueli A, Zuck LD (1985) The glory of the past. In: Logics of programs, LNCS, vol 193, pp 196–218. Springer
30. Maler O, Nickovic D (2004) Monitoring temporal properties of continuous signals. In: Formal techniques, modelling and analysis of timed and fault-tolerant systems, LNCS, vol 3253, pp 152–166. Springer
31. Pinisetty S, Jéron T, Tripakis S, Falcone Y, Marchand H, Preoteasa V (2017) Predictive runtime verification of timed properties. J Syst Softw 132:353–365
32. Pnueli A (1977) The temporal logic of programs. In: 18th annual symposium on foundations of computer science, Providence, Rhode Island, USA, 31 October-1 November 1977, pp 46–57. IEEE Computer Society. https://doi.org/10.1109/SFCS.1977.32
33. Pnueli A, Zaks A (2008) On the merits of temporal testers. In: O Grumberg, H Veith (eds.) 25 Years of model checking-history, achievements, perspectives, Lecture Notes in Computer Science, vol 5000, pp 172–195. Springer. https://doi.org/10.1007/978-3-540-69850-0_11
34. Qin X, Deshmukh JV (2020) Clairvoyant monitoring for signal temporal logic. In: N Bertrand, N Jansen (eds.) Formal modeling and analysis of timed systems, vol 12288, pp 178–195. Springer International Publishing. https://doi.org/10.1007/978-3-030-57628-8_11. Lecture Notes in Computer Science

35. Thirioux X (2002) Simple and efficient translation from LTL formulas to Buchi automata. Electron Notes Theor Comput Sci 66(2):145–159. https://doi.org/10.1016/S1571-0661(04)80409-2
36. Vardi MY, Wolper P (1986) An automata-theoretic approach to automatic program verification (preliminary report). In: Proceedings of the symposium on logic in computer science, pp 332–344. IEEE Computer Society
37. Yoon H, Chou Y, Chen X, Frew EW, Sankaranarayanan S (2019) Predictive runtime monitoring for linear stochastic systems and applications to geofence enforcement for UAVs. In: Runtime verification, LNCS, vol 11757, pp 349–367. Springer
38. Yu K, Chen Z, Dong W (2014) A predictive runtime verification framework for cyber-physical systems. In: Software security and reliability-companion, pp. 223–227. IEEE
39. Zhang X, Leucker M, Dong W (2012) Runtime verification with predictive semantics. In: NASA formal methods, LNCS, vol 7226, pp 418–432. Springer

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.