Runtime Verification with Imperfect Information through Indistinguishability Relations

Angelo Ferrando[1][0000−0002−8711−4670] and Vadim Malvone[2][0000−0001−6138−4229]

[1] Department of Informatics, Bioengineering, Robotics and Systems Engineering, University of Genova, Italy
[2] LTCI, Telecom Paris, Institut Polytechnique de Paris, France
angelo.ferrando@unige.it
vadim.malvone@telecom-paris.fr

Abstract. Software systems are hard to trust, especially when autonomous. To overcome this, formal verification techniques can be deployed to verify such systems behave as expected. Runtime Verification is one of the most prominent and lightweight approaches to verify the system behaviour at execution time. However, standard Runtime Verification is built on the assumption of perfect information over the system, that is, the monitor checking the system can perceive everything. Unfortunately, this is not always the case, especially when the system under analysis contains rational/autonomous components and is deployed in real-world environments with possibly faulty sensors. In this work, we present an extension of the standard Runtime Verification of Linear Temporal Logic properties to consider scenarios with imperfect information. We present the engineering steps necessary to update the verification pipeline, and we report the corresponding implementation when applied to a case study involving robotic systems.

Keywords: Runtime Verification · Autonomous Systems · Imperfect Information

1 Introduction

Developing quality software is a very demanding task [13]. Many are the reasons, but the complexity and presence of autonomous behaviours are definitely amongst them. Techniques that were developed to approach the development of monolithic systems may not work as well for distributed and autonomous ones. This does not only represent a technological issue, but an engineering one as well. In the past decades, we all have been witnesses of technological advances in the software engineering research area, especially when focused on the actual software development. However, the need of re-engineering does not only concern software development, but its verification as well. As software changes, so the ways to verify it need to change. Runtime Verification (RV), as other verification techniques, is not free from such changes.

Runtime Verification [12, 1] is a formal verification technique that allows the verification of the runtime behaviour of a software/hardware system of interest.

Differently from other verification techniques, RV is not exhaustive, since it focuses on the actual system execution. That is, a violation of the expected behaviour is concluded only if such violation is observed in the execution trace. Nevertheless, RV is a lightweight technique, because it does not check all possible system's behaviours, and by doing this, it scales better than its static verification counterparts (which usually suffer from the state space explosion problem).

RV was born after static verification, such as model checking [7], and it inherited much from the latter; especially on how to specify the formal properties to verify. One of the most used formalisms in model checking, and by consequence in RV, is Linear Temporal Logic (LTL) [14]. We will present its syntax and semantics along the paper, but for now, we only focus on the aspect of LTL on which this work is mainly focused on, that is its implicit assumption of perfect information over the system. Indeed, LTL verification is usually performed assuming the system under analysis offers all the information needed for the verification [5]. This is translated at the verification level into the generation of atomic propositions that denote what we know about the system, and are used to verify the properties of interest. However, this is not always the case. Especially when the system to verify contains autonomous, or distributed, or even faulty components (like faulty sensors in real-world environments, e.g. any robotics scenario). In such cases, to assume all the needed information is available is too optimistic. Naturally, as we will better elaborate in related work section, other works on handling LTL RV with imperfect information exist [16, 3, 10, 2, 11]. Nevertheless, this is the first work that tackles the problem at its foundations, and without the need of creating a new verification pipeline (which in this case consists on how to synthesise the monitor to verify the LTL property). Specifically, this is the first attempt of extending the standard monitor's synthesis pipeline to explicitly take into consideration imperfect information.

In this paper, we formally define the notion of imperfect information w.r.t. the monitor's visibility over the system, and we then re-engineer the LTL monitor's synthesis pipeline to recognise such visibility information. We also present the details on the prototype that has been implemented to support our claims, and to provide the community an LTL monitoring library that natively supports imperfect information. Moreover, we show some possible uses of such prototype in a realistic case study.

The paper's structure is as follows. Section 2 reports preliminaries notions that are necessary to fully understand the paper contribution. Section 3 formally presents the notion of imperfect information, its implication at the monitoring level and the resulting re-engineering of the standard LTL monitor's synthesis pipeline. Section 4 reports the details on the prototype that has been developed as a result of the re-engineering process, along with some experiments of its use in a realistic case study. Section 5 positions the paper against the state of the art. Section 6 concludes the paper and discusses some possible future directions.

2 Preliminaries

A system S has an *alphabet* Σ with which it is possible to define the set 2^{Σ} of all its events. Given an alphabet Σ, a *trace* $\sigma = ev_0 ev_1 \ldots$, is a sequence of events in 2^{Σ}. With $\sigma(i)$ we denote the i-th element of σ (*i.e.*, ev_i), σ^i the suffix of σ starting from i (*i.e.*, $ev_i ev_{i+1} \ldots$), $(2^{\Sigma})^*$ the *set of all possible finite traces* over Σ, and $(2^{\Sigma})^{\omega}$ the *set of all possible infinite traces* over Σ.

The standard formalism to specify properties in RV is Linear Temporal Logic (LTL [14]). The relevant parts of the syntax of LTL are the following:

$$\varphi := p \mid \neg\varphi \mid (\varphi \vee \varphi) \mid \bigcirc\varphi \mid (\varphi \ \mathbf{U} \ \varphi)$$

where $p \in \Sigma$ is an atomic proposition, φ is a formula, $\bigcirc$ stands for *next-time*, and $\mathbf{U}$ stands for *until*. In the rest of the paper, we also use the standard derived operators, such as $(\varphi \rightarrow \varphi')$ instead of $(\neg\varphi \vee \varphi')$, $\varphi \ R \ \varphi'$ instead of $\neg(\neg\varphi \,\mathbf{U}\, \neg\varphi')$, $\Box\varphi$ (*always* φ) instead of $(false \ R \ \varphi)$, and $\Diamond\varphi$ (*eventually* φ) instead of $(true \,\mathbf{U}\, \varphi)$.

Let $\sigma \in (2^{\Sigma})^{\omega}$ be an infinite sequence of events over Σ, the semantics of LTL is as follows:

$$\begin{aligned}
&\sigma \models p \text{ if } p \in \sigma(0)\\
&\sigma \models \neg\varphi \text{ if } \sigma \not\models \varphi\\
&\sigma \models \varphi \vee \varphi' \text{ if } \sigma \models \varphi \text{ or } \sigma \models \varphi'\\
&\sigma \models \bigcirc\varphi \text{ if } \sigma^1 \models \varphi\\
&\sigma \models \varphi \,\mathbf{U}\, \varphi' \text{ if } \exists_{i \geq 0}.\sigma^i \models \varphi' \text{ and } \forall_{0 \leq j < i}.\sigma^j \models \varphi
\end{aligned}$$

A trace σ satisfies an atomic proposition p, if p belongs to $\sigma(0)$; which means, p holds in the initial event of the trace σ. A trace σ satisfies the negation of the LTL property φ, if σ does not satisfy φ. A trace σ satisfies the disjunction of two LTL properties, if σ satisfies at least one of them. A trace σ satisfies next-time φ, if the suffix of σ starting in the next step (σ^1) satisfies φ. Finally, a trace σ satisfies $\varphi \,\mathbf{U}\, \varphi'$, if there exists a suffix of σ s.t. φ' is satisfied, and for all suffixes before it, φ holds. Thus, given an LTL property φ, we denote $[\![\varphi]\!]$ the language of the property, *i.e.*, the set of traces which satisfy φ; namely $[\![\varphi]\!] = \{\sigma \mid \sigma \models \varphi\}$.

In Definition 1, we present a general and formalism-agnostic definition of a monitor. Informally, a monitor is a function that, given a trace of events in input, returns a verdict which denotes the satisfaction (resp., violation) of a formal property over the trace.

Definition 1 (Monitor). *Let S be a system with alphabet Σ, σ a finite trace, and φ be an LTL property. Then, a monitor for φ is a function $Mon_{\varphi} : (2^{\Sigma})^* \rightarrow \mathbb{B}_3$, where $\mathbb{B}_3 = \{\top, \bot, ?\}$:*

$$Mon_{\varphi}(\sigma) = \begin{cases} \top & \forall_{u \in (2^{\Sigma})^{\omega}}.\sigma \bullet u \in [\![\varphi]\!] \\ \bot & \forall_{u \in (2^{\Sigma})^{\omega}}.\sigma \bullet u \notin [\![\varphi]\!] \\ ? & otherwise \end{cases}$$

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

Intuitively, a monitor returns $\top$ if all continuations (u) of σ satisfy φ; $\bot$ if all possible continuations of σ violate φ; ? otherwise. The first two outcomes are standard representations of satisfaction and violation, while the third is specific to RV. In more detail, it denotes when the monitor cannot conclude any verdict yet. This is closely related to the fact that RV is applied while the system is still running, and future events may still change the verdict. For instance, a property might be currently satisfied (resp., violated) by the system, but violated (resp., satisfied) in the (still unknown) future. The monitor can only safely conclude any of the two final verdicts ($\top$ or $\bot$) if it is sure such verdict will never change. The addition of the third outcome symbol ? helps the monitor to represent its position of uncertainty w.r.t. the current system execution.

A monitor function is usually implemented as a Finite State Machine (FSM), specifically a Moore machine (FSM where the output value of a state is only determined by the state) [4, 5]. A Moore machine can be defined as a tuple $\langle Q, q_0, \Sigma, O, \delta, \gamma \rangle$, where Q is a finite set of states, q_0 is the initial state, Σ is the input alphabet, O is the output alphabet, $\delta : Q \times \Sigma \to Q$ is the transition function mapping a state and an event to the next state, and $\gamma : Q \to O$ is the function mapping a state to the output alphabet.

In [5], Bauer *et al.* present the sequence of steps required to generate from an LTL formula φ the corresponding Moore machine instantiating the Mon_φ function (as summarised in Figure 1).

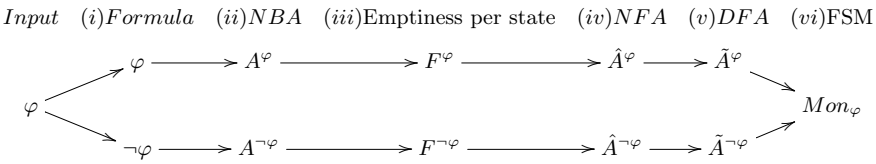


Fig. 1: Steps required to generate an FSM from an LTL formula φ. NBA is Non-deterministic Büchi Automaton, NFA is Non-deterministic Finite Automaton, and DFA is Deterministic Finite Automaton.

Given an LTL property φ, a series of transformations is performed on φ, and its negation $\neg\varphi$. Considering φ in step *(i)*, first, a corresponding NBA A^φ is generated in step *(ii)*. This can be obtained using Gerth *et al.*'s algorithm [9]. Such automaton recognises the set of infinite traces that satisfy φ (according to LTL semantics). Then, each state of A^φ is evaluated; the states that when selected as initial states in A^φ do not generate the empty language are then added to the F^φ set in step *(iii)*. With such a set, an NFA $\hat{A}^\varphi$ is obtained from A^φ by simply substituting the final states of A^φ with F^φ in step *(iv)*. $\hat{A}^\varphi$ recognises the finite traces (prefixes) that have at least one infinite continuation satisfying φ (since the prefix reaches a state in F^φ). After that, $\hat{A}^\varphi$ is transformed (Rabin–Scott powerset construction [15]) into its equivalent deterministic version $\tilde{A}^\varphi$ in step *(v)*; this is possible since deterministic and non-deterministic finite automata have the same expressive power. The exact same steps are performed on $\neg\varphi$, which bring to the generation of the $\tilde{A}^{\neg\varphi}$ counterpart. The difference between

$\tilde{A}^{\varphi}$ and $\tilde{A}^{\neg\varphi}$ is that the former recognises finite traces which have continuations satisfying φ, while the latter recognises finite traces which have continuations violating φ. Finally, a Moore machine can be generated as a standard automata product between $\tilde{A}^{\varphi}$ and $\tilde{A}^{\neg\varphi}$ in the final step *(vi)*, where the states are denoted as tuples (q, q'), with q and q' belonging to $\tilde{A}^{\varphi}$ and $\tilde{A}^{\neg\varphi}$, respectively. The outputs are then determined as: $\top$ if q' does not belong to the final states of $\tilde{A}^{\neg\varphi}$, $\bot$ if q does not belong to the final states of $\tilde{A}^{\varphi}$, and ? otherwise. This brings us to the revised monitor construction as follows.

Definition 2 (Monitor as FSM). *Given an LTL formula φ and a finite trace σ, the revised monitor is defined as follows:*

$$Mon_{\varphi}(\sigma) = \begin{cases} \top & \sigma \notin \mathcal{L}(\tilde{A}^{\neg\varphi}) \\ \bot & \sigma \notin \mathcal{L}(\tilde{A}^{\varphi}) \\ ? & \sigma \in \mathcal{L}(\tilde{A}^{\varphi}) \wedge \sigma \in \mathcal{L}(\tilde{A}^{\neg\varphi}) \end{cases}$$

where $\mathcal{L}(A)$ denotes the language recognised by automaton A.

3 Runtime Verification with Imperfect Information

Up to now, we have focused on standard RV of LTL properties. However, such standard approach, as presented in Section 2, is based upon a strong assumption:

The absence of an atomic proposition is the same as the negation of the latter.

This might be true when we apply formal verification to systems with perfect information (*i.e.*, systems where each involved component has a perfect understanding and vision of the entire system). Unfortunately, even though this may be the case for monolithic and traditional systems, it is not the case for autonomous systems, or in general, systems exploiting artificial intelligence. In such scenarios, it is very common to not have a complete vision over the system. Let us just think about robotics scenarios, where a robot can be deployed in an environment of which it can only access what its sensors provide. Such information can be incomplete. Moreover, since RV is based upon the notion of monitoring the system under analysis; if the verified component has no complete access over the system's information, by consequence, also the monitor does not. Thus, we may find ourselves in scenarios where our runtime monitors observe only partial information of the system. Because of this, the trace of events passed to the monitor to analyse may not contain some of the atomic propositions, and this would be erroneously classified as the negation of such atomic propositions. Instead, we need to give importance to the difference between knowing when something is not true, w.r.t knowing when something is simply not known.

3.1 How can we formally represent the imperfect information?

As recognised previously in the paper, the problem of using LTL when the system has imperfect information is in confusing the absence of an atomic proposition,

with its negation. Since in case of imperfect information, the trace may not contain atomic propositions which are not known (*i.e.*, cannot be observed), we need a way to characterise such absence of information, explicitly. To do this, we follow an approach similar to [6], where atomic propositions are duplicated.

One possible way to represent imperfect information is by allowing indistinguishability on atomic propositions Σ. To do this we introduce an equivalence relation $\sim$ over Σ. Intuitively, given two atomic propositions $p, q \in \Sigma$, we say that they are indistinguishable if and only if $p \sim q$. The relation $\sim$ gives us the information available to the monitor. Moreover, given an equivalence relation $\sim$ we define a witness for each equivalence class. That is, given an equivalence class γ, we define the witness of γ with the symbol $[\gamma]$.

To handle the verification process in the imperfect information context, we need to do some extensions. First of all, we can not simply use the set of atomic propositions Σ. In particular, we need to replace Σ with a new set $\bar{\Sigma}$ that is defined as follows: for each $p \in \Sigma$ we have $p_\top \in \bar{\Sigma}$ and $p_\bot \in \bar{\Sigma}$. That is, we duplicate the set of atomic proposition to make the truth value explicit.

Without losing generality, we only consider LTL formulas in Negation Normal Form (NNF). An LTL in NNF has only negations at the atom levels (*i.e.*, we only have $\neg p$). Given an LTL formula, its NNF can be easily obtained by propagating all negations to the atoms. For instance, if we had $\neg \bigcirc p$, we would rewrite it as $\bigcirc \neg p$. The same goes for the other operators.

First, we present how to generate the explicit version of an LTL formula.

Definition 3. *Given an LTL formula φ in NNF and the set of equivalence classes Γ, we define the explicit version of φ as follows:*

$$\begin{aligned} \epsilon(p) &= [\gamma]_\top \\ \epsilon(\neg p) &= [\gamma]_\bot \\ \epsilon(\varphi \vee \varphi') &= \epsilon(\varphi) \vee \epsilon(\varphi') \\ \epsilon(\bigcirc \varphi) &= \bigcirc \epsilon(\varphi) \\ \epsilon(\varphi \,\mathbf{U}\, \varphi') &= \epsilon(\varphi) \,\mathbf{U}\, \epsilon(\varphi') \end{aligned}$$

where $\gamma \in \Gamma$ and $p \in \gamma$.

We now present how to construct the explicit and visible versions of a trace.

Definition 4. *Given a trace σ and a set Σ, we define the explicit version of σ as σ_e, for each element $\sigma(i)$ as follows:*

- *for all $p \in \sigma(i)$, $p_\top \in \sigma_e(i)$;*
- *for all $p \in \Sigma \setminus \sigma(i)$, $p_\bot \in \sigma_e(i)$.*

Definition 5. *Given an explicit trace σ_e and the set of equivalence classes Γ, we define the visible version of σ_e as σ_v, for each $\sigma(i)$ and $\gamma \in \Gamma$ as follows:*

- *$[\gamma]_\top \in \sigma_v(i)$ if and only if for all $p \in \gamma$, $p_\top \in \sigma_e(i)$;*
- *$[\gamma]_\bot \in \sigma_v(i)$ if and only if for all $p \in \gamma$, $p_\bot \in \sigma_e(i)$.*

Given the above elements, we define a three-valued semantics for LTL:

$(\sigma \models p) = \top$ if $p_\top \in \sigma(0)$
$(\sigma \models p) = \bot$ if $p_\bot \in \sigma(0)$
$(\sigma \models \neg\varphi) = \top$ if $(\sigma \not\models \varphi) = \top$
$(\sigma \models \neg\varphi) = \bot$ if $(\sigma \not\models \varphi) = \bot$
$(\sigma \models \varphi \vee \varphi') = \top$ if $(\sigma \models \varphi) = \top$ or $(\sigma \models \varphi') = \top$
$(\sigma \models \varphi \vee \varphi') = \bot$ if $(\sigma \models \varphi) = \bot$ and $(\sigma \models \varphi') = \bot$
$(\sigma \models \bigcirc\varphi) = \top$ if $(\sigma^1 \models \varphi) = \top$
$(\sigma \models \bigcirc\varphi) = \bot$ if $(\sigma^1 \models \varphi) = \bot$
$(\sigma \models \varphi \,\mathbf{U}\, \varphi') = \top$ if $\exists_{i\geq 0}.(\sigma^i \models \varphi') = \top$ and $\forall_{0\leq j<i}.(\sigma^j \models \varphi) = \top$
$(\sigma \models \varphi \,\mathbf{U}\, \varphi') = \bot$ if $\forall_{i\geq 0}.(\sigma^i \models \varphi') = \bot$ or $\exists_{0\leq j<i}.(\sigma^j \models \varphi) = \bot$

In all the other cases the truth value is undefined (uu).
To help the reader, we conclude the section with the following example.

Example 1. Consider the set $\Sigma = \{p, q, r\}$, the formula $\phi = \bigcirc r$, and a trace σ where $\sigma(1) = \{p, q\}$. Furthermore, assume $p \sim r$, this means that the monitor cannot distinguish between the atomic propositions p and r. In the context of imperfect information, we have $\bar{\Sigma} = \{p_\top, q_\top, r_\top, p_\bot, q_\bot, r_\bot\}$. By Definition 3, we have the explicit LTL version $\epsilon(\phi) = \bigcirc[\gamma_\top]$, where $\gamma = \{p, r\}$ is the equivalence class defined over $\sim$. By Definition 4-5, we generate the explicit trace σ_e where $\sigma_e(1) = \{p_\top, q_\top, r_\bot\}$ and visible trace σ_v where $\sigma_v(1) = \{q_\top\}$. Thus, given the three-valued LTL semantics, $\epsilon(\phi)$ is undefined. Indeed, to satisfy (resp., falsify) the original formula φ, the monitor has to check that both $p_\top$ and $r_\top$ (resp., $p_\bot$ and $r_\bot$) are verified since they belong to the same equivalence class γ.

3.2 Re-engineering Monitor with imperfect information

Given an LTL formula and a visible trace for the monitor, we need a way to use them to perform RV. This can be obtained by extending the standard pipeline for generating LTL monitors (see Figure 1). Such extension is based on two specific modifications: (i) we use the explicit version of the LTL formula, following Definition 3; (ii) we modify the product between $\tilde{A}^\varphi$ and $\tilde{A}^{\neg\varphi}$ to generate the Moore machine denoting the monitor. The resulting extension is reported in Figure 2, where the explicit version of the LTL formula is generated in step (ii). While the updated product between the automata is obtained in step (vii). The rest of the steps are left unchanged w.r.t. Figure 1.

The pipeline presented in Figure 2 is identical to the one presented in Figure 1, but the atomic propositions in the formula are duplicated before using the formula to generate the corresponding NBA, and an additional automaton has been added. The former aspect is important, because by duplicating the atomic propositions, we completely change the semantics of the following steps in the monitor synthesis pipeline. Specifically, it is not true that for any given visible trace σ_v, we have $\sigma_v \notin \mathcal{L}(\hat{A}^\varphi) \Rightarrow \sigma_v \in \mathcal{L}(\hat{A}^{\neg\varphi})$, nor $\sigma_v \notin \mathcal{L}(\hat{A}^{\neg\varphi}) \Rightarrow \sigma_v \in \mathcal{L}(\hat{A}^\varphi)$.

Which means, it is not true that when a visible trace of events σ_v is not a good prefix for φ (*i.e.*, a prefix that can be extended to an infinite trace satisfying φ), it has to be then a bad prefix for φ (*i.e.*, a prefix that cannot be extended to an infinite trace satisfying φ). This aspect is closely related to the reason why a third formula (*i.e.*, $\otimes\varphi$) has been introduced in Figure 2. Since by duplicating the atomic propositions in the formula we break the duality between φ and $\neg\varphi$, we need a third automaton (*i.e.*, $\tilde{A}^{\otimes\varphi}$) to recognise all the traces that do not satisfy, nor violate, φ. For this reason, we extended the pipeline by adding $\otimes\varphi$, which is an abbreviation for $\neg\epsilon(\varphi) \wedge \neg\epsilon(\neg\varphi)$. The automaton $\tilde{A}^{\otimes\varphi}$, obtained following the same steps as for the positive $\tilde{A}^{\epsilon(\varphi)}$ and negative $\tilde{A}^{\epsilon(\neg\varphi)}$ automata, recognises all prefixes for which no continuation satisfying or violating φ exist.

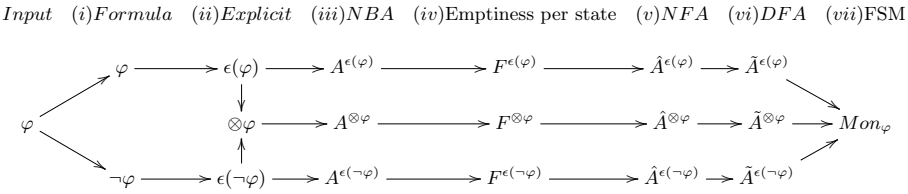


Fig. 2: Extended pipeline to consider imperfect information.

Now, we formalize the above reasoning with the following lemma.

Lemma 1. *Given a visible finite trace σ_v and an LTL formula φ, we have:*

$$\sigma_v \notin \mathcal{L}(\hat{A}^{\epsilon(\varphi)}) \not\Rightarrow \sigma_v \in \mathcal{L}(\hat{A}^{\epsilon(\neg\varphi)})$$
$$\sigma_v \notin \mathcal{L}(\hat{A}^{\epsilon(\neg\varphi)}) \not\Rightarrow \sigma_v \in \mathcal{L}(\hat{A}^{\epsilon(\varphi)})$$

Proof. Assume we have a visible trace σ_v and it is not included in the NFA $\hat{A}^{\epsilon(\varphi)}$. To prove our result, we just need to show that σ_v is also not included in $\hat{A}^{\epsilon(\neg\varphi)}$. To do the latter, suppose $\Sigma = \{p, q, r\}$, $\varphi = \bigcirc p$, $p \sim q$, and σ where $\sigma(1) = \{p\}$. Now, given Definition 4-5, we can conclude that $\sigma_v(1) = \{r_\perp\}$. So, σ_v does not satisfy φ and by consequence it is not included in the NFA $\hat{A}^{\epsilon(\varphi)}$. However, it is not included neither in $\hat{A}^{\epsilon(\neg\varphi)}$. This is because $p_\top$ and $p_\perp$ are not included in $\sigma_v(1)$. This concludes the first relation. For the second one, we can use a variant of the above reasoning.

By adding the third automaton, the corresponding FSM synthesis needs also to change. In more detail, the revised version is reported in Definition 6. In such definition, we can see how the addition of a third automaton in the equation allows us to synthesise a finer monitor, in the sense of the number of possible outcomes it returns. Indeed, w.r.t. Definition 2, we have three additional outcomes. Specifically, given a visible trace σ_v, the monitor returns $\top$ if there is no continuation of σ_v which either violates $\epsilon(\varphi)$ or makes it undefined. On the other hand, it returns $\perp$ if there is no continuation which either satisfies $\epsilon(\varphi)$ or makes it undefined. Since now we have three automata, there is an additional

final outcome to consider, which is uu. So, the monitor returns uu if there is no continuation which either satisfies or violates $\epsilon(\varphi)$. These first three outcomes are all deriving by the three-values semantics for LTL. Then, we may find $?_{\not\perp}$, which is read "unknown, but it will never be violated from the monitor's point of view". Such outcome is returned by the monitor when the visible trace σ_v does not have any continuation which will eventually violate $\epsilon(\varphi)$, but there are continuations that satisfy $\epsilon(\varphi)$ and make it undefined. Symmetrically, we may find $?_{\not\top}$, which is read "unknown, but it will never be satisfied from the monitor's point a view". This outcome is the dual of the previous one, where no continuations satisfying $\epsilon(\varphi)$ can be found, but continuations that violate $\epsilon(\varphi)$ and make it undefined exist. Last but not least, we may find ? denoting the completely unknown case. As before, this outcome concerns the case where the monitor cannot conclude anything yet, because there exist continuations satisfying $\epsilon(\varphi)$, continuations violating $\epsilon(\varphi)$, and continuations that make it undefined.

Definition 6 (Monitor with imperfect information). *Given an LTL formula φ and a visible trace σ_v, a monitor with imperfect information is so defined:*

$$Mon^v_\varphi(\sigma_v) = \begin{cases} \top & \sigma_v \in \mathcal{L}(\tilde{A}^{\epsilon(\varphi)}) \wedge \sigma_v \notin \mathcal{L}(\tilde{A}^{\epsilon(\neg\varphi)}) \wedge \sigma_v \notin \mathcal{L}(\tilde{A}^{\otimes\varphi}) \\ \bot & \sigma_v \notin \mathcal{L}(\tilde{A}^{\epsilon(\varphi)}) \wedge \sigma_v \in \mathcal{L}(\tilde{A}^{\epsilon(\neg\varphi)}) \wedge \sigma_v \notin \mathcal{L}(\tilde{A}^{\otimes\varphi}) \\ uu & \sigma_v \notin \mathcal{L}(\tilde{A}^{\epsilon(\varphi)}) \wedge \sigma_v \notin \mathcal{L}(\tilde{A}^{\epsilon(\neg\varphi)}) \wedge \sigma_v \in \mathcal{L}(\tilde{A}^{\otimes\varphi}) \\ ?_{\not\perp} & \sigma_v \in \mathcal{L}(\tilde{A}^{\epsilon(\varphi)}) \wedge \sigma_v \notin \mathcal{L}(\tilde{A}^{\epsilon(\neg\varphi)}) \wedge \sigma_v \in \mathcal{L}(\tilde{A}^{\otimes\varphi}) \\ ?_{\not\top} & \sigma_v \notin \mathcal{L}(\tilde{A}^{\epsilon(\varphi)}) \wedge \sigma_v \in \mathcal{L}(\tilde{A}^{\epsilon(\neg\varphi)}) \wedge \sigma_v \in \mathcal{L}(\tilde{A}^{\otimes\varphi}) \\ ? & \sigma_v \in \mathcal{L}(\tilde{A}^{\epsilon(\varphi)}) \wedge \sigma_v \in \mathcal{L}(\tilde{A}^{\epsilon(\neg\varphi)}) \wedge \sigma_v \in \mathcal{L}(\tilde{A}^{\otimes\varphi}) \end{cases}$$

Note that, in the above definition, not all the possible combination are included. In particular, it is not possible to have $\sigma_v \notin \mathcal{L}(\tilde{A}^{\epsilon(\varphi)}) \wedge \sigma_v \notin \mathcal{L}(\tilde{A}^{\epsilon(\neg\varphi)}) \wedge \sigma_v \notin \mathcal{L}(\tilde{A}^{\otimes\varphi})$ and $\sigma_v \in \mathcal{L}(\tilde{A}^{\epsilon(\varphi)}) \wedge \sigma_v \in \mathcal{L}(\tilde{A}^{\epsilon(\neg\varphi)}) \wedge \sigma_v \notin \mathcal{L}(\tilde{A}^{\otimes\varphi})$. In particular, the former is not possible because there exists at least one automaton that includes the trace by following the definition of the three-valued semantics for LTL. The latter follows by the fact that it is unfeasible given the nature of a visible trace that a formula will be true or false but not undefined in the future.

In what follows, we provide two preservation results from the monitor with imperfect information to the one with perfect information.

Lemma 2. *Given a finite trace σ, a monitor with its visibility $Mon^v_\varphi(\sigma)$, and a general monitor $Mon_\varphi(\sigma)$, we have that:*

$$\text{if } Mon^v_\varphi(\sigma_v) = \top \text{ then } Mon_\varphi(\sigma) = \top$$
$$\text{if } Mon^v_\varphi(\sigma_v) = \bot \text{ then } Mon_\varphi(\sigma) = \bot$$

Proof. Suppose $Mon^v_\varphi(\sigma_v) = \top$. This means that the visible trace σ_v satisfies the formula $\epsilon(\varphi)$. We want to prove that the original trace σ satisfies the formula φ. To do this, given σ_v, by Definition 4-5, we know that for each $\sigma_v(i)$, for all $p_\top \in \sigma_v(i)$, $p \in \sigma(i)$ and for all $p_\bot \in \sigma_v(i)$, $p \notin \sigma(i)$. Given the above reasoning, we need to provide an induction proof over the structure of the formula

$\epsilon(\varphi)$. Case: $\epsilon(\varphi) = p_\top$. So, $\varphi = p$. By hypothesis, $Mon^v_\varphi(\sigma_v) = \top$, by the semantics of three-valued LTL this means that $p_\top \in \sigma_v(0)$ and by Definition 4-5, $p \in \sigma(0)$. By the latter, $Mon_\varphi(\sigma) = \top$. Case: $\epsilon(\varphi) = p_\bot$. Thus, $\varphi = \neg p$. By hypothesis, $Mon^v_\varphi(\sigma_v) = \top$, by the semantics of three-valued LTL this means that $p_\bot \in \sigma_v(0)$ and by Definitions 4-5, $p \notin \sigma(0)$. By the latter, $Mon_\varphi(\sigma) = \top$. Since in the inductive cases the transformation of Definition 3 does not change the structure and the elements of the formula, we can conclude the proof.

Suppose $Mon^v_\varphi(\sigma_v) = \bot$. This means that the visible trace σ_v does not satisfy the formula $\epsilon(\varphi)$. We want to prove that the original trace σ does the same for the formula φ. As for the previous case, we need to prove the implication by induction over the structure of the formula $\epsilon(\varphi)$ for the base cases. Case: $\epsilon(\varphi) = p_\top$. So, $\varphi = p$. By hypothesis, $Mon^v_\varphi(\sigma_v) = \bot$, by the semantics of three-valued LTL this means that $p_\bot \in \sigma_v(0)$ and by Definition 4-5, $p \notin \sigma(0)$. By the latter, $Mon_\varphi(\sigma) = \bot$. Case: $\epsilon(\varphi) = p_\bot$. Thus, $\varphi = \neg p$. By hypothesis, $Mon^v_\varphi(\sigma_v) = \bot$, by the semantics of three-valued LTL this means that $p_\top \in \sigma_v(0)$ and by Definition 4-5, $p \in \sigma(0)$. By the latter, $Mon_\varphi(\sigma) = \bot$.

Given the above results, we can deduce the following corollary.

Corollary 1. *Given a visible finite trace σ_v and an LTL formula φ, we have:*

$$\sigma_v \notin \mathcal{L}(\hat{A}^{\epsilon(\varphi)}) \Rightarrow \sigma_v \in \mathcal{L}(\hat{A}^{\epsilon(\neg\varphi)}) \vee \sigma_v \in \mathcal{L}(\tilde{A}^{\otimes\varphi})$$
$$\sigma_v \notin \mathcal{L}(\hat{A}^{\epsilon(\neg\varphi)}) \Rightarrow \sigma_v \in \mathcal{L}(\hat{A}^{\epsilon(\varphi)}) \vee \sigma_v \in \mathcal{L}(\tilde{A}^{\otimes\varphi})$$
$$\sigma_v \notin \mathcal{L}(\tilde{A}^{\otimes\varphi}) \Rightarrow \sigma_v \in \mathcal{L}(\hat{A}^{\epsilon(\neg\varphi)}) \vee \sigma_v \in \mathcal{L}(\hat{A}^{\epsilon(\varphi)})$$

4 Implementation

The prototype implementing the theory presented in this paper is publicly available as a GitHub repository[3]. It consists in a Python script which implements the entire pipeline presented in Figure 2. The reason for choosing Python lies in the presence of a rich library for automaton manipulation, named Spot[4] [8]. In more detail, we used Spot to automatically generate an NBA, given an LTL formula. This corresponds to step (iii) in Figure 2, which is the most complicated and computationally expensive step in the pipeline. The rest of the pipeline has been directly implemented in Python.

Going a bit deeper in the implementation, the prototype consists in a Python class, named Monitor. To create a Monitor, its constructor requires: (i) an LTL formula to verify; (ii) a set of atomic propositions; (iii) an equivalence relation on atomic propositions; (iv) a trace of events to analyse.

With the previous information, a FSM representing the monitor as in Definition 6 is constructed. Then, such monitor is used to analyse the input trace, and the corresponding verdict is returned back to the user. The trace is assumed to be stored inside a file (e.g., a log file). These input parameters can be passed as command line arguments to the tool. However, since the monitor is denoted

[3] https://github.com/AngeloFerrando/RuntimeVerificationWithImperfectInformation
[4] https://spot.lrde.epita.fr/

as a single data structure, it is also possible (and quite natural) to import the script and use the monitor as preferred. This can be useful for instance if the monitor is to be used for online verification, rather than offline verification.

4.1 Remote inspection case study

We talked about the theory behind our approach, and we also briefly introduced the resulting prototype. Let us now focus on the experiments we carried out on a robotic case study, as a proof of concept.

Our case study is based on a 3D simulation of a Jackal[5], a four-wheeled unmanned ground vehicle (referred to as the 'rover' from now on), coupled with a simulated radiation sensor, that the rover uses to take radiation readings of points of interest while patrolling around a nuclear facility, and a camera, that the rover uses to inspect images of the nuclear waste barrels in the area. This simulation is based on the work presented in [20], which explains how the simulated sensor works and how radiation was simulated in the environment. In our version of the simulation the rover is autonomously controlled by a rational/intelligent agent [19]. Figure 3 reports a screenshot of the case study.

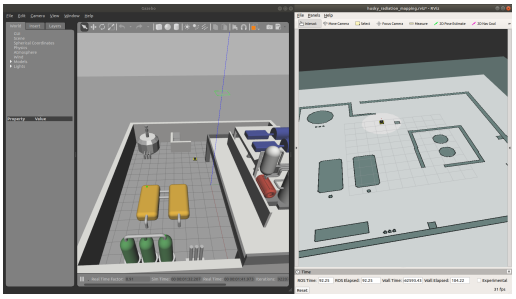


Fig. 3: Simulation in Gazebo of the remote inspection of nuclear plant.

A typical mission in our simulation starts with the rover positioned at the entrance of a nuclear facility. The goal of this mission is to inspect a number of points of interest (*i.e.*, waypoints). Inspecting a waypoint serves two purposes: taking radiation readings to check if the radiation is at an acceptable level, and using a camera to detect abnormalities such as leakage in barrels and pipes. After inspecting all of the waypoints, the rover can either return to the entrance to await for a new mission, or keep patrolling and inspecting the waypoints.

Without losing generality, we assume the image captured by the rover's camera can be represented as a grid. Each cell in such a grid can contain, or not, an abnormality (e.g., a cut in the barrel). This information is translated into propositions, that can be transmitted to the monitor to be analysed at runtime.

[5] `https://clearpathrobotics.com/jackal-small-unmanned-ground-vehicle`

Let us assume that, because of the rover's limited resources, the latter is not always capable of sending all the information to the monitor. Because of this, some times the monitor is not able to distinguish a cut from a rust stain. In such cases, from the viewpoint of the monitor analysing the scene, there is imperfect information over the atomic propositions. We assume the presence of a cut on a barrel b with c and the presence of a rust stain with s. So, the set of atomic propositions is $\Sigma = \{b, c, s\}$. Then, we have imperfect information over c and s, which is formalised as $c \sim s$ (*i.e.*, there is an equivalence class γ_{cs} between c and s). Let us now say that the property we want to verify at runtime is whether the rover will not find a cut in the barrel. This information could be used by the software agent controlling the rover to react properly (e.g., by informing a human operator about a possible leakage). Such property can be formulated as the following LTL formula: $\varphi = \Diamond(b \wedge \bigcirc\neg c)$. Nevertheless, this formula would make sense in a case of perfect information over the system, but in this case, where c and s cannot be distinguished (in general), a standard LTL monitor should not be used. To understand this, let us just assume that the trace of events σ observed by the rover is $\sigma(0) = \{\}$, $\sigma(1) = \{b\}$, and $\sigma(2) = \{c\}$. In such trace, the first event means that the rover has not observed anything relevant, the second event means that the rover observed the barrel b, and the third event denotes the presence of a cut on the barrel b. But, since the monitor has imperfect information, the truth value of c cannot be observed. Consequently, if the monitor considered $\neg c$ without caring about the imperfect information, it could report that there are no problems, i.e. the general monitor in this case returns true. But the latter is not correct. Thus, to tackle this aspect in its foundations, we can apply our extended semantics and its resulting monitor.

Since in this scenario we have an equivalence relation between c and s (*i.e.*, $c \sim s$), first we need to explicit the atomic propositions inside the formula, obtaining: $\epsilon(\varphi) = \Diamond(b_\top \wedge \bigcirc[\gamma_{cs}]_\bot)$. By using the newly updated LTL formula, we can generate the three automata as shown in Figure 2. After that, we can update the trace of events as well, first by generating its explicit version σ_e (see Definition 4), where $\sigma_e(0) = \{b_\bot, c_\bot, s_\bot\}$, $\sigma_e(1) = \{b_\top, c_\bot, s_\bot\}$, and $\sigma_e(2) = \{b_\bot, c_\top, s_\bot\}$. Then by defining its visible version according to the given equivalence class γ_{cs} (see Definition 5), we obtain σ_v, where $\sigma_v(0) = \{b_\bot, [\gamma_{cs}]_\bot\}$, $\sigma_v(1) = \{b_\top, [\gamma_{cs}]_\bot\}$, and $\sigma_v(2) = \{b_\bot\}$. Note that, as expected, the last event in σ_v does not contain information about the atomic proposition c. This is determined by the fact that the atomic propositions $c_\top$ and $s_\bot$ hold in the last event of σ_v, and according to Definition 5, since $c \sim s$, we can have $[\gamma_{cs}]_\top$ (resp., $[\gamma_{cs}]_\bot$) if and only if both $c_\top$ and $s_\top$ hold (resp., $c_\bot$ and $s_\bot$). Thus, having a mismatch between the two atomic propositions (*i.e.*, one is true while the other is false), we cannot safely add any witness for the equivalence class γ_{cs}. Instead, in the first two events of σ_v, since we have both $c_\bot$ and $s_\bot$, we can safely add the witness $[\gamma_{cs}]_\bot$ to the trace. Thanks to our three-value semantics and the presence of explicit atomic propositions, the trace σ which was erroneously classified as satisfying φ from the standard LTL monitor before, now is classified as $?_{\not\perp}$. The semantics of the two verdicts is fundamentally different, as well as the reaction

that the system should have. In the first case, by using a standard LTL monitor, the verdict returned by the monitor was $\top$. Thus, the agent controlling the rover could have used such information to continue the inspection with another barrel and not detecting a danger. In the second case, by using the extended LTL monitor that we presented in this work, the verdict returned by the monitor was $?_{\not\perp}$. Thus, the agent controlling the rover could use this information to, for instance, ask the rover to check again, maybe taking another picture. Even though this is a simple example, it allows us to show how our extension tackles the foundations of the imperfect information issue.

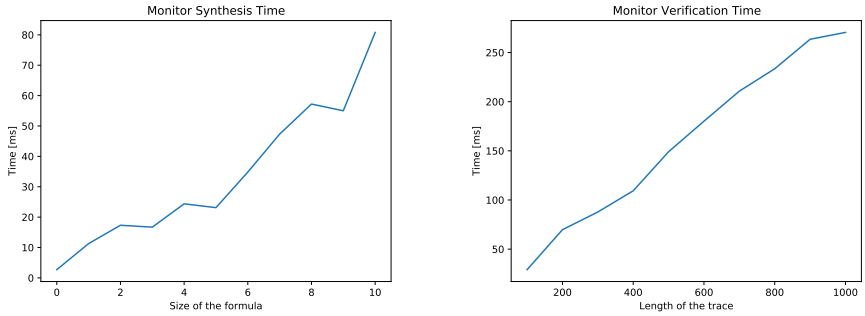
4.2 Experimental results

Other than verifying the property previously presented for the remote inspection scenario, we carried out more general experiments to study the execution time of our prototype. In more detail, we focused on two fundamental aspects, the generation and verification time. The former concerns the execution time required to synthesise a monitor given an LTL formula (according to Definition 6). While the latter concerns the execution time required to analyse a given trace of events with the so synthesised monitor. It is important to separate the two experimental evaluations since the monitor's generation is not usually performed online, but ahead of the system execution. Thus, the most critical aspect to consider when evaluating runtime verification techniques is the verification time, since it is the only one which is performed online. Consequently, it is the only part that influences the execution; this is also referred to as the monitor's overhead.

We carried out experiments for both aspects. Specifically, for the monitor's synthesis, we did experiments varying the size of the LTL formula; where the size of the formula consists in the number of operators inside the formula. We picked the size of the formula as target of our experiments because it is the input driving the generation of the monitor[6]. Instead, for the verification part, we carried out experiments varying the length of the trace of events to analyse. Also in this case, we picked the length of the trace because it is the only input which influences the monitor's verification time. This can be easily understood by considering the fact that once the FSM has been generated, it will not change. Thus, its size is fixed and is determined by the size of the formula. So, at runtime, the only aspect that changes is the length of the trace, which is populated by events generated through the system execution.

Figure 4 reports the results obtained with our experiments, where both LTL formulas and traces are randomly generated. Specifically, Figure 4a reports the execution time to synthesise a monitor given an LTL formula, while Figure 4b reports the execution time to analyse a given trace of events with the so synthesised monitor. In Figure 4a, we may find the size of the LTL formula on the x-axis, and the execution time on the y-axis (in milliseconds). Note that, as expected, the execution time for the monitor synthesis grows exponentially w.r.t. the size of the formula. In Figure 4b we may find the length of the trace of the

[6] Let us remember that steps (iii) and (vi) in Figure 2 are very expensive and require exponential time w.r.t. the size of the formula.



(a) Time to synthesise a monitor. (b) Time to verify a trace.

Fig. 4: Experimental results.

events on the x-axis, and the execution time on the y-axis (in milliseconds). Note that, the execution time is linear w.r.t. the length of the trace; this is crucial for using the monitor at runtime, while the system is running. Since the execution time is linear w.r.t. the length of the trace, the time required for the monitor to analyse a single event in the trace is constant. Thus, the monitor can be used to incrementally analyse events generated at runtime by the system[7].

5 Related Work

The closest work to our contribution is [18], where Past-Time LTL is verified at runtime in case of uncertainty over the observed events. In such work, the verification is carried out on abstract traces of events. An abstract trace corresponds to a trace where not all concrete events are present, but only samples taken with a certain time step. The uncertainty comes from unknown event interleaving, while in our case comes from indistinguishability relations amongst events. Differently from [18], we do not sample the events, and the uncertainty is determined by the monitor's visibility. Thus, the abstraction is not on the order amongst the events in a trace, but on the kind of events the trace contains.

In a completely different line of research, we may find [16, 3, 10, 2, 11], where the uncertainty in the verification is caused by the absence of information. In such works, the trace of events may contain gaps, which means at certain point of the system execution, the monitor is not capable of observing the system behaviour. This problem has been tackled in different ways, but in general, the solution consists in filling the gaps with events. Naturally, since there is uncertainty on what was exactly the event in the gap, these approaches depend on probabilities to guess which events to use to fill the gap. These works are different from ours in principle, because we do not assume to miss information, indeed we do not

[7] Where with incrementally, we mean the monitor analyses the events one by one (not as in offline RV where the monitor expects the entire trace all at once).

have gaps in our traces. Our uncertainty is not based on the monitor missing events, but on the monitor not being capable of recognising (discerning) some events from other events (according to a indistinguishability relation).

A recent work on RV with uncertainty can be found in [17]. There, the concept of uncertainty is abstracted by considering multi-traces, instead of uni-traces (standard traces). A multi-trace allows multiple evaluations for the same atomic proposition inside the trace. The authors present a monitor to handle such multi-traces and prove its soundness. Like for [16, 3, 10, 2, 11], also [17] is focused on missing events, even though partially missing ones are considered too.

Differently from our contribution, all the works previously mentioned explicitly represent the notion of uncertainty (e.g. through a gap). When the trace contains concrete events, the semantics is the standard one. Our approach is less invasive, since it is constructed on top of the standard RV pipeline for the verification of LTL properties. We do not require the addition of gaps. We mainly focus on how to update the standard RV technique for LTL when the monitor can have imperfect information over the system. From an engineering perspective, our approach aims at extending the standard LTL approach to be used in case of imperfect information over the system, while the other works in literature are more focused on proposing completely new techniques to handle the absence of information (usually caused by noise or technical issues).

6 Conclusions and Future Work

In this paper, we presented an extension of the standard LTL runtime verification approach. We introduce the problem of imperfect information at the monitor level, and how such lack of information can bring a standard LTL monitor to conclude a wrong verdict. We present theoretically the notion of imperfect information (through equivalence classes) and how it influences the LTL property verification. In particular, we propose how to extend the standard LTL monitor synthesis [5], we show the resulting Python prototype, and we report its use on a relevant case study along with additional experiments to stress test it.

As future work, we are planning to further extend our approach by considering a post-processing function to add additional information to the monitor's verdict. Such function would depend on the trace of events, the LTL property and the monitor's verdict to establish a level of confidence on the final outcome. Up to now, we mainly focused on how to tackle the problem of imperfect information at the foundations of LTL runtime verification, however, once we obtain the final outcome from the monitor, we can still refine it more. In more detail, when the outcome concluded by the monitor is uu, we could elaborate it further and assign a probability value. For instance, instead of saying uu, we could say that the property is undefined w.r.t. the trace, but according to some probability distribution over the involved equivalence classes, we can claim the property would be satisfied (resp., violated) with a certain probability threshold.

References

1. Bartocci, E., Falcone, Y., Francalanza, A., Reger, G.: Introduction to runtime verification. In: Bartocci, E., Falcone, Y. (eds.) Lectures on Runtime Verification - Introductory and Advanced Topics, Lecture Notes in Computer Science, vol. 10457, pp. 1–33. Springer (2018). https://doi.org/10.1007/978-3-319-75632-5_1
2. Bartocci, E., Grosu, R.: Monitoring with uncertainty. In: Bortolussi, L., Bujorianu, M., Pola, G. (eds.) Proceedings Third International Workshop on Hybrid Autonomous Systems, HAS 2013, Rome, Italy, 17th March 2013. EPTCS, vol. 124, pp. 1–4 (2013). https://doi.org/10.4204/EPTCS.124.1, https://doi.org/10.4204/EPTCS.124.1
3. Bartocci, E., Grosu, R., Karmarkar, A., Smolka, S.A., Stoller, S.D., Zadok, E., Seyster, J.: Adaptive runtime verification. In: Qadeer, S., Tasiran, S. (eds.) Runtime Verification, Third International Conference, RV 2012, Istanbul, Turkey, September 25-28, 2012, Revised Selected Papers. Lecture Notes in Computer Science, vol. 7687, pp. 168–182. Springer (2012). https://doi.org/10.1007/978-3-642-35632-2_18, https://doi.org/10.1007/978-3-642-35632-2_18
4. Bauer, A., Leucker, M., Schallhart, C.: Monitoring of real-time properties. In: Arun-Kumar, S., Garg, N. (eds.) FSTTCS 2006: Foundations of Software Technology and Theoretical Computer Science, 26th International Conference, Kolkata, India, December 13-15, 2006, Proceedings. Lecture Notes in Computer Science, vol. 4337, pp. 260–272. Springer (2006). https://doi.org/10.1007/11944836_25
5. Bauer, A., Leucker, M., Schallhart, C.: Runtime verification for ltl and tltl. ACM Trans. Softw. Eng. Methodol. **20**(4) (Sep 2011). https://doi.org/10.1145/2000799.2000800
6. Belardinelli, F., Lomuscio, A., Malvone, V., Yu, E.: Approximating perfect recall when model checking strategic abilities: Theory and applications. J. Artif. Intell. Res. **73**, 897–932 (2022). https://doi.org/10.1613/jair.1.12539, https://doi.org/10.1613/jair.1.12539
7. Clarke, E.M.: Model checking. In: International Conference on Foundations of Software Technology and Theoretical Computer Science. pp. 54–56. Springer (1997)
8. Duret-Lutz, A., Poitrenaud, D.: SPOT: an extensible model checking library using transition-based generalized büchi automata. In: DeGroot, D., Harrison, P.G., Wijshoff, H.A.G., Segall, Z. (eds.) 12th International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS 2004), 4-8 October 2004, Vollendam, The Netherlands. pp. 76–83. IEEE Computer Society (2004). https://doi.org/10.1109/MASCOT.2004.1348184, https://doi.org/10.1109/MASCOT.2004.1348184
9. Gerth, R., Peled, D.A., Vardi, M.Y., Wolper, P.: Simple on-the-fly automatic verification of linear temporal logic. In: Dembinski, P., Sredniawa, M. (eds.) Protocol Specification, Testing and Verification XV, Proceedings of the Fifteenth IFIP WG6.1 International Symposium on Protocol Specification, Testing and Verification, Warsaw, Poland, June 1995. IFIP Conference Proceedings, vol. 38, pp. 3–18. Chapman & Hall (1995). https://doi.org/10.1007/978-0-387-34892-6_1
10. Kalajdzic, K., Bartocci, E., Smolka, S.A., Stoller, S.D., Grosu, R.: Runtime verification with particle filtering. In: Legay, A., Bensalem, S. (eds.) Runtime Verification - 4th International Conference, RV 2013, Rennes, France, September 24-27, 2013. Proceedings. Lecture Notes in Computer Science, vol. 8174, pp. 149–166. Springer (2013). https://doi.org/10.1007/978-3-642-40787-1_9, https://doi.org/10.1007/978-3-642-40787-1_9

11. Leucker, M., Sánchez, C., Scheffel, T., Schmitz, M., Thoma, D.: Runtime verification for timed event streams with partial information. In: Finkbeiner, B., Mariani, L. (eds.) Runtime Verification - 19th International Conference, RV 2019, Porto, Portugal, October 8-11, 2019, Proceedings. Lecture Notes in Computer Science, vol. 11757, pp. 273–291. Springer (2019). https://doi.org/10.1007/978-3-030-32079-9_16, https://doi.org/10.1007/978-3-030-32079-9_16
12. Leucker, M., Schallhart, C.: A brief account of runtime verification. J. Log. Algebraic Methods Program. **78**(5), 293–303 (2009). https://doi.org/10.1016/j.jlap.2008.08.004
13. Miguel, J.P., Mauricio, D., Rodriguez, G.: A review of software quality models for the evaluation of software products. CoRR **abs/1412.2977** (2014), http://arxiv.org/abs/1412.2977
14. Pnueli, A.: 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 (1977). https://doi.org/10.1109/SFCS.1977.32
15. Rabin, M.O., Scott, D.S.: Finite automata and their decision problems. IBM J. Res. Dev. **3**(2), 114–125 (1959). https://doi.org/10.1147/rd.32.0114
16. Stoller, S.D., Bartocci, E., Seyster, J., Grosu, R., Havelund, K., Smolka, S.A., Zadok, E.: Runtime verification with state estimation. In: Khurshid, S., Sen, K. (eds.) Runtime Verification - Second International Conference, RV 2011, San Francisco, CA, USA, September 27-30, 2011, Revised Selected Papers. Lecture Notes in Computer Science, vol. 7186, pp. 193–207. Springer (2011). https://doi.org/10.1007/978-3-642-29860-8_15, https://doi.org/10.1007/978-3-642-29860-8_15
17. Taleb, R., Khoury, R., Hallé, S.: Runtime verification under access restrictions. In: Bliudze, S., Gnesi, S., Plat, N., Semini, L. (eds.) 9th IEEE/ACM International Conference on Formal Methods in Software Engineering, FormaliSE@ICSE 2021, Madrid, Spain, May 17-21, 2021. pp. 31–41. IEEE (2021). https://doi.org/10.1109/FormaliSE52586.2021.00010, https://doi.org/10.1109/FormaliSE52586.2021.00010
18. Wang, S., Ayoub, A., Sokolsky, O., Lee, I.: Runtime verification of traces under recording uncertainty. In: Khurshid, S., Sen, K. (eds.) Runtime Verification - Second International Conference, RV 2011, San Francisco, CA, USA, September 27-30, 2011, Revised Selected Papers. Lecture Notes in Computer Science, vol. 7186, pp. 442–456. Springer (2011). https://doi.org/10.1007/978-3-642-29860-8_35, https://doi.org/10.1007/978-3-642-29860-8_35
19. Wooldridge, M., Rao, A. (eds.): Foundations of Rational Agency. Applied Logic Series, Kluwer Academic Publishers (1999)
20. Wright, T., West, A., Licata, M., Hawes, N., Lennox, B.: Simulating ionising radiation in gazebo for robotic nuclear inspection challenges. Robotics **10**(3) (2021). https://doi.org/10.3390/robotics10030086

7 Explanation of how we dealt with reviewer's comments

Reviewer 1

- We fixed the title as suggested by another reviewer as well.
- The property generation is completely random. The size of the property is, indeed, the number of temporal operators of the formula. We experimented thousands of properties. We did not count the nested operators, but the number of operators inside each formula. The number of nested operators might be an interesting feature to consider in future extensions of the work.
- The suggested comparison with other works is interesting. Of course, we do not have space here to carry it out, but we will definitely consider it for future versions.

Reviewer 2

- To give you the intuition about the indistinguishability, consider the following example. Suppose that the rover can send a sequence of 3 bits to the monitor and that 101 is decoded as a cut and 001 is a rust. If the monitor receives 01, it cannot determine if there is a cut or a rust. This is because the missing bit determines the event. A bit can be missed due to a connection problem or to a signal band limited (in our case limited to two bits).
- The reviewer was right about the property, we actually meant $\wedge$ instead of $\rightarrow$. In such a way the property matches our description and is also more relevant as an example.
- Since the standard notion of monitorability does not change w.r.t. to standard RV there is no point in discussing it in this paper. A monitorable (resp., non monitorable) property, remains monitorable (resp., non monitorable) in our setting. When imperfect information is considered, it is not that the property is not monitorable, but the result is biased by assuming wrong information. This is what our work is used for.
- Thanks to the reviewer, we improved and fixed a bug in the implementation. Already publicly available.

Reviewer 3

- We updated the title as suggested. Now is more precise w.r.t. the paper's topic.
- The property analysed in the case study is satisfied w.r.t. a standard monitor simply because we observe b and then we do not observe c in the following event of the trace. A monitor should not return ? because the property is a liveness property, and the requirement to its satisfaction has been met (a monitor should guarantee anticipation).
- We fixed the sentences in the case study section that were pointed out. This is to better state that the indistinguishability relations is a possible way to formalise imperfect information (naturally, not the only one).
- All typos have been fixed as suggested by the reviewer.