Proceedings Paper:
Chen, Z., Chen, Y., Hierons, R.M. et al. (2020) Four-valued monitorability of ω-regular languages. In: Lin, S.-W., Hou, Z. and Mahoney, B., (eds.) Formal Methods and Software Engineering : 22nd International Conference on Formal Engineering Methods, ICFEM 2020, Proceedings. 22nd International Conference on Formal Engineering Methods (ICFEM 2020), 01-03 Mar 2021, Singapore. Lecture Notes in Computer Science (12531). Springer International Publishing, pp. 198-214. ISBN: 9783030634056. ISSN: 0302-9743. EISSN: 1611-3349.

https://doi.org/10.1007/978-3-030-63406-3_12

Four-Valued Monitorability of ω-Regular Languages

Zhe Chen[1], Yunyun Chen[1], Robert M. Hierons[2], and Yifan Wu[1]

[1] College of Computer Science and Technology
Nanjing University of Aeronautics and Astronautics, China
zhechen@nuaa.edu.cn
[2] Department of Computer Science, The University of Sheffield, UK
r.hierons@sheffield.ac.uk

Abstract. The use of runtime verification has led to interest in deciding whether a property is *monitorable*: whether it is always possible for the satisfaction or violation of the property to be determined after a finite future continuation during system execution. However, classical two-valued monitorability suffers from two inherent limitations, which eventually increase runtime overhead. First, no information is available regarding whether only one verdict (satisfaction or violation) can be detected. Second, it does not tell us whether verdicts can be detected starting from the current monitor *state* during system execution.

This paper proposes a new notion of four-valued monitorability for ω-languages and applies it at the state-level. Four-valued monitorability is more informative than two-valued monitorability as a property can be evaluated as a four-valued result, denoting that only satisfaction, only violation, or both are active for a monitorable property. We can also compute state-level weak monitorability, i.e., whether satisfaction or violation can be detected starting from a given state in a monitor, which enables state-level optimizations of monitoring algorithms. Based on a new six-valued semantics, we propose procedures for computing four-valued monitorability of ω-regular languages, both at the language-level and at the state-level. Experimental results show that our tool implementation MONIC can correctly, and quickly, report both two-valued and four-valued monitorability.

Keywords: Monitorability · ω-regular languages · Linear temporal logic · Runtime verification.

1 Introduction

Runtime Verification (RV) [32,6,29] is a lightweight formal technique in which program or system execution is monitored and analyzed. RV uses information extracted from an execution to check whether certain properties are satisfied or violated after a finite number of steps, possibly leading to online responses. In RV, properties are usually expressed using formalisms [26] such as Linear Temporal Logic (LTL) formulas [36,33,17,10], Nondeterministic Büchi Automata

(NBAs), and ω-regular expressions, which represent ω-regular languages [15,7]. RV tools automatically synthesize monitors (i.e., code fragments) from formal specifications and then weave the code into the system through instrumentation [24,28,25]. The inserted code typically maintains a set of monitor objects that can detect property satisfaction or violation during system execution. Such approaches have been extended to parametric RV, in which properties are checked over every parameter instance (i.e., a combination of parameter values) by maintaining a monitor object for every parameter instance [11,34,38,13,12,27].

Figure 1 shows a monitor specification, written in the MOVEC language [13], for the parametric RV of an event-driven system that dispatches a variety of events (e.g., sensor status, keystrokes, program loadings etc.) to components (e.g., libraries, mobile apps, microservices etc.). Similar specifications can be written for other tools such as JavaMOP [11,34] and TraceMatches [4,5]. This specification defines a parametric monitor, named `priority`, which takes two parameters: a component ID `c` and an event ID `e` that should be instantiated with the values (i.e., actual arguments) generated by system execution. The specification body begins with four actions, which extract information regarding function calls: `r` records a component being registered to an event (it also creates a monitor object by instantiating the monitor parameters with the arguments of the call), `u` records an unregister, `b` records the broadcast of an event (the argument of the call) to all components, and `n` records a certain component being notified of a specific event. This specification is used to monitor system execution to check whether the property, specified as LTL formula $\phi_1 := (r \wedge \boldsymbol{F}u) \rightarrow ((\neg b \wedge \neg u)\, \boldsymbol{U}n)\, \boldsymbol{U}u$, is satisfied or violated after a finite number of steps, i.e., any infinite future continuation makes the property satisfied or violated, respectively. The property requires that if a component `c` registers to an event `e` and unregisters later, then before the unregister, the event `e` cannot be broadcasted until `c` has been notified (i.e., `c` has a higher priority than unregistered components).

In practice, if the satisfaction or violation of a property is detected by a monitor object then an associated handler (i.e., a piece of code) is automatically triggered to perform some online response [13,11,34]. For example, Figure 1 includes two handlers for the satisfaction (i.e., validation) and violation of the LTL formula: if the property is satisfied then a message is logged; if it is violated then an alarm is signaled and this prints the IDs of the component and the event. The two handlers may also be extended to more advanced operations, e.g., profiling and error recovery.

We may also monitor the system against other properties, e.g., $\phi_2 := \boldsymbol{F}r \rightarrow \boldsymbol{GF}n$ that a component should receive notifications infinitely often after its registration, $\phi_3 := r \rightarrow \boldsymbol{F}u$ that a component unregisters after its registration, and $\phi_4 := \boldsymbol{G}(r \rightarrow \neg u\, \boldsymbol{U}n)$ that a registered component receives at least one notification before its deregistration. The developer may also write handlers for the satisfaction and violation of each property.

When specifying properties, the developer is usually concerned with their monitorability [37,10,7,16], i.e., after any number of steps, whether the satisfaction or violation of the monitored property can still be detected after a finite

```
monitor priority(c,e) {
  creation action r(c,e) after call(% reg_component(% %:c, % %:e));
  action u(c,e) after call(% unreg_component(% %:c, % %:e));
  action b(e) before execution(% broadcast(% %:e));
  action n(c,e) after execution(% notify(% %:c, % %:e));

  ltl: (r && <>u) -> ((!b && !u) U n) U u;
  @validation {
    log("Priority applied: component %lu registers to event %lu.\n",
        monitor->c, monitor->e);  }
  @violation {
    printf("Priority violated: component %lu registers to event %lu.\n",
        monitor->c, monitor->e);  }
};
```

Fig. 1: A monitor specification with an LTL formula.

future continuation. When writing handlers for these properties, the developer might consider the following question: *"Can the handlers for satisfaction and violation be triggered during system execution?"* We say that a verdict and its handler are *active* if there is some continuation that would lead to the verdict being detected and thus its handler being triggered. This question can be partly answered by deciding monitorability (with the traditional two-valued notion). For example, ϕ_2 (above) is non-monitorable, i.e., there is some finite sequence of steps after which no verdict is active. Worse, ϕ_2 is also weakly non-monitorable [14], i.e., no verdict can be detected after any number of steps. Thus writing handlers for ϕ_2 is a waste of time as they will never be triggered. More seriously, monitoring ϕ_2 at runtime adds no value but increases runtime overhead. In contrast, ϕ_1, ϕ_3 and ϕ_4 are monitorable, i.e., some verdicts are always active. Thus their handlers must be developed as they may be triggered. However, this answer is still unsatisfactory, as the existing notion of monitorability suffers from two inherent limitations: *limited informativeness* and *coarse granularity*.

Limited informativeness. The existing notion of monitorability is not sufficiently informative, as it is two-valued, i.e., a property can only be evaluated as monitorable or non-monitorable. This means, for a monitorable property, we only know that some verdicts are active, but no information is available regarding whether only one verdict (satisfaction or violation) is active. As a result, the developer may still write unnecessary handlers for inactive verdicts. For example, ϕ_1, ϕ_3 and ϕ_4 are monitorable. We only know that at least one of satisfaction and violation is active, but this does not tell us which ones are active and thus which handlers are required. As a result, the developer may waste time in handling inactive verdicts, e.g., the violation of ϕ_3 and the satisfaction of ϕ_4. Thus, the existing answer is far from satisfactory.

Limited informativeness also weakens the support for property debugging. For example, when writing a property the developer may expect that both verdicts are active but a mistake may lead to only one verdict being active. The converse is also the case. Unfortunately, these kinds of errors cannot be revealed by two-valued monitorability, as the expected property and the written (erro-

neous) property are both monitorable. For example, the developer may write formula ϕ_4 while having in mind another one $\phi_5 := r \rightarrow \neg u \, \boldsymbol{U} n$, i.e., what she/he really wants is wrongly prefixed by one $\boldsymbol{G}$. These two formulas cannot be discriminated by deciding two-valued monitorability as both are monitorable.

Coarse granularity. The existing notion of monitorability is defined at the language-level, i.e., a property can only be evaluated as monitorable or not as a whole, rather than a notion for (more fine-grained) states in a monitor. This means that we do not know whether satisfaction or violation can be detected *starting from the current state* during system execution. As a result, every monitor object must be maintained during the entire execution, again increasing runtime overhead. For example, $\phi_6 := \boldsymbol{GF} r \vee (\neg n \rightarrow \boldsymbol{X} \neg b)$ is weakly monitorable, thus all its monitor objects (i.e., instances of the Finite State Machine (FSM) in Figure 2), created for every pair of component and event, are maintained.

Note that parametric runtime verification is NP-complete for detecting violations and coNP-complete for ensuring satisfaction [12]. This high complexity primarily comes from the large number of monitor objects maintained for all parameter instances [34,13,12]. For state-level optimizations of monitoring algorithms, if no verdict can be detected starting from the current state of a monitor object, then the object can be switched off and safely removed to improve runtime performance. For example, in Figure 2, only satisfaction can be detected starting from states `P1`, `P2` and `T`, whereas no verdict can be detected starting from state `N`. Thus a monitor object can be safely removed when it enters `N`. Unfortunately, the existing notion does not support such optimizations.

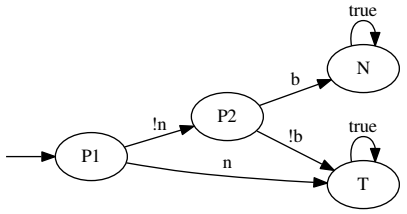

Fig. 2: A monitor for LTL formula $\phi_6 := \boldsymbol{GF} r \vee (\neg n \rightarrow \boldsymbol{X} \neg b)$. Each transition is labeled with a propositional formula denoting a set of satisfying states. For example, "!n" denotes $\{\emptyset, \{r\}, \{b\}, \{r, b\}\}$ and "true" denotes all states.

Our Solution. In this paper, we propose a new notion of four-valued monitorability for ω-languages, and apply it at the state-level, overcoming the two limitations discussed above. First, the proposed approach is more informative than two-valued monitorability. Indeed, a property can be evaluated as a four-valued result, denoting that *only satisfaction, only violation, or both are active for a monitorable property*. Thus, if satisfaction (resp. violation) is inactive, then writing handlers for satisfaction (resp. violation) is not required. This can also enhance property debugging. For example, ϕ_4 and ϕ_5 can now be discriminated by their different monitorability results, as ϕ_4 can never be satisfied but ϕ_5 can be satisfied and can also be violated. Thus, additional developer mistakes can be revealed. Second, we can compute state-level weak monitorability, i.e., whether satisfaction or violation can be detected starting from a given state in a monitor. For example, in Figure 2, state `N` is weakly non-monitorable, thus a monitor object can be safely removed when it enters `N`, which achieves a state-level optimization.

In summary, we make the following contributions.[3]

- We propose a new notion of four-valued monitorability for ω-languages (Section 3), which provides more informative answers as to which verdicts are active. This notion is defined using six types of prefixes, which complete the classification of finite sequences.
- We propose a procedure for computing four-valued monitorability of ω-regular languages, given in terms of LTL formulas, NBAs or ω-regular expressions (Section 4), based on a new six-valued semantics.
- We propose a new notion of state-level four-valued weak monitorability and its computation procedure for ω-regular languages (Section 5), which describes which verdicts are active for a state. This can enable state-level optimizations of monitoring algorithms.
- We have developed a new tool, MONIC, that implements the proposed procedure for computing monitorability of LTL formulas. We evaluated its effectiveness using a set of 97 LTL patterns and formulas ϕ_1 to ϕ_6 (above). Experimental results show that MONIC can correctly report both two-valued and four-valued monitorability (Section 6).

2 Preliminaries

Let AP be a non-empty finite set of *atomic propositions*. A *state* is a complete assignment of truth values to the propositions in AP. Let $\Sigma = 2^{AP}$ be a finite *alphabet*, i.e., the set of all states. Σ^* is the set of finite words (i.e., sequences of states in Σ), including the empty word ϵ, and Σ^ω is the set of infinite words. We denote atomic propositions by p, q, r, finite words by u, v, and infinite words by w, unless explicitly specified. We write a finite or infinite word in the form $\{p,q\}\{p\}\{q,r\}\cdots$, where a proposition appears in a state iff it is assigned true. We drop the brackets around singletons, i.e., $\{p,q\}p\{q,r\}\cdots$.

An *ω-language* (i.e., a linear-time infinitary property) L is a set of infinite words over Σ, i.e., $L \subseteq \Sigma^\omega$. Linear Temporal Logic (LTL) [36,33] is a typical representation of ω-regular languages. LTL extends propositional logic, which uses *boolean connectives* $\neg$ (not) and $\wedge$ (conjunction), by introducing *temporal connectives* such as $\boldsymbol{X}$ (next), $\boldsymbol{U}$ (until), $\boldsymbol{R}$ (release), $\boldsymbol{F}$ (future, or eventually) and $\boldsymbol{G}$ (globally, or always). Intuitively, $\boldsymbol{X}\phi$ says that ϕ holds at the next state, $\phi_1 \boldsymbol{U} \phi_2$ says that at some future state ϕ_2 holds and before that state ϕ_1 always holds. Using the temporal connectives $\boldsymbol{X}$ and $\boldsymbol{U}$, the full power of LTL is obtained. For convenience, we also use some common abbreviations: *true*, *false*, standard boolean connectives $\phi_1 \vee \phi_2 \equiv \neg(\neg\phi_1 \wedge \neg\phi_2)$ and $\phi_1 \rightarrow \phi_2 \equiv \neg\phi_1 \vee \phi_2$, and additional temporal connectives $\phi_1 \boldsymbol{R} \phi_2 \equiv \neg(\neg\phi_1 \boldsymbol{U} \neg\phi_2)$ (the dual to $\boldsymbol{U}$), $\boldsymbol{F}\phi \equiv \mathit{true}\,\boldsymbol{U}\phi$ (ϕ eventually holds), and $\boldsymbol{G}\phi \equiv \neg\boldsymbol{F}\neg\phi$ (ϕ always holds). We denote by $L(\phi)$ the ω-language accepted by a formula ϕ.

Let us recall the classification of prefixes that are used to define the three-valued semantics and two-valued monitorability of ω-languages.

[3] A longer version of this paper (with all proofs) is available at `https://arxiv.org/abs/2002.06737`.

Definition 1 (Good, bad and ugly prefixes [31,8]). *A finite word $u \in \Sigma^*$ is a* good prefix *for L if $\forall w \in \Sigma^\omega.uw \in L$, a* bad prefix *for L if $\forall w \in \Sigma^\omega.uw \notin L$, or an* ugly prefix *for L if no finite extension makes it good or bad, i.e., $\nexists v \in \Sigma^*.\forall w \in \Sigma^\omega.uvw \in L$ and $\nexists v \in \Sigma^*.\forall w \in \Sigma^\omega.uvw \notin L$.*

In other words, good and bad prefixes *satisfy* and *violate* an ω-language in some finite number of steps, respectively. We denote by $good(L)$, $bad(L)$ and $ugly(L)$ the set of good, bad and ugly prefixes for L, respectively. Note that they do not constitute a complete classification of finite words. For example, any finite word of the form $p \cdots p$ is neither a good nor a bad prefix for $p\boldsymbol{U}q$, and also is not an ugly prefix as it can be extended to a good prefix (ended with q) or a bad prefix (ended with $\emptyset$).

Definition 2 (Three-valued semantics [10]). *Let $\mathbb{B}_3$ be the set of three truth values:* true $\top$, false $\bot$ *and* inconclusive ?*. The truth value of an ω-language $L \subseteq \Sigma^\omega$ wrt. a finite word $u \in \Sigma^*$, denoted by $[u \models L]_3$, is $\top$ or $\bot$ if u is a good or bad prefix for L, respectively, and ? otherwise.*

Note that the inconclusive value does not correspond to ugly prefixes. Although an ugly prefix always leads to the inconclusive value, the converse does not hold. For example, $[p \cdots p \models L(p\boldsymbol{U}q)]_3 =$? but $p \cdots p$ is not an ugly prefix.

Bauer et al. [10] presented a monitor construction procedure that transforms an LTL formula ϕ into a three-valued monitor, i.e., a deterministic FSM that contains $\top$, $\bot$ and ? states, which output $\top$, $\bot$ and ? after reading over good, bad and other prefixes respectively. For example, in Figure 2, state `T` is a $\top$ state, whereas the remaining states are all ? states. This construction procedure requires 2ExpSpace. It has been shown that the three-valued monitor can be used to compute the truth value of an ω-language wrt. a finite word [10], which is the output of the corresponding monitor after reading over this word.

Lemma 1. *Let $M = (Q, \Sigma, \delta, q_0, \mathbb{B}_3, \lambda_3)$ be a three-valued monitor for an ω-language $L \subseteq \Sigma^\omega$, where Q is a finite set of* states*, Σ is a finite* alphabet*, $\delta : Q \times \Sigma \mapsto Q$ is a* transition function*, $q_0 \in Q$ is an* initial state*, $\mathbb{B}_3$ is an* output alphabet *and $\lambda_3 : Q \to \mathbb{B}_3$ is an* output function*. For any $u \in \Sigma^*$, $[u \models L]_3 = \lambda_3(\delta(q_0, u))$.*

Definition 3 (Two-valued monitorability [37,10,7]). *An ω-language $L \subseteq \Sigma^\omega$ is* u-monitorable *for $u \in \Sigma^*$, if $\exists v \in \Sigma^*$ s.t. uv is a good or bad prefix, and* monitorable *if it is u-monitorable for every $u \in \Sigma^*$.*

In other words, L is *u-monitorable* if u has a *good* or *bad extension*. L is *monitorable* if every finite word has a good or bad extension. Note that an ugly prefix can never be extended to a good or bad prefix. Thus, L is *non-monitorable* iff there exists an ugly prefix for L.

3 Four-valued monitorability

In this section, we propose a new notion of four-valued monitorability, to provide more informative answers to monitorability checking. As we promised, it

can indicate whether only satisfaction, only violation, or both are active for a monitorable property. Two-valued monitorability cannot achieve this because its definition only requires that all finite words (i.e., u in Definition 3) can be extended to good or bad prefixes (which witness satisfaction or violation, respectively), but does not discriminate them on the types and number of the verdicts that the extensions of each finite word can witness. To address this limitation, our approach aims to discriminate accordingly these finite words by inspecting which types of prefixes they can be extended to.

To achieve this objective, we first need to propose a new classification of prefixes, as the traditional classification (as the good, the bad and the ugly) is not satisfactory due to incompleteness, i.e., it does not include the finite words that are neither good nor bad but can be extended to good or bad prefixes. Thus we introduce the notions of positive, negative and neutral prefixes, in addition to good, bad and ugly prefixes, to complete the classification.

Definition 4 (Positive, negative and neutral prefixes). *A finite word $u \in \Sigma^*$ is a*

- positive prefix *for L if it is not good, but some finite extension makes it good but never bad, i.e., $\exists w \in \Sigma^\omega.uw \notin L$, $\exists v \in \Sigma^*.\forall w \in \Sigma^\omega.uvw \in L$, and $\nexists v \in \Sigma^*.\forall w \in \Sigma^\omega.uvw \notin L$,*
- negative prefix *for L if it is not bad, but some finite extension makes it bad but never good, i.e., $\exists w \in \Sigma^\omega.uw \in L$, $\exists v \in \Sigma^*.\forall w \in \Sigma^\omega.uvw \notin L$, and $\nexists v \in \Sigma^*.\forall w \in \Sigma^\omega.uvw \in L$, or*
- neutral prefix *for L if some finite extension makes it good and some makes it bad, i.e., $\exists v \in \Sigma^*.\forall w \in \Sigma^\omega.uvw \in L$ and $\exists v \in \Sigma^*.\forall w \in \Sigma^\omega.uvw \notin L$.*

We denote by $posi(L)$, $nega(L)$ and $neut(L)$ the set of positive, negative and neutral prefixes for L, respectively. It is easy to see that the three new sets of prefixes and the three traditional sets of good, bad and ugly prefixes are mutually disjoint. An interesting fact, as shown by the following theorem, is that the six sets of prefixes exactly constitute the complete set of finite words. Furthermore, the six types of prefixes directly correspond to the six-valued semantics (cf. Definition 5). This completes the classification of prefixes.

Theorem 1. $good(L) \cup bad(L) \cup posi(L) \cup nega(L) \cup neut(L) \cup ugly(L) = \Sigma^*$.

The traditional three-valued semantics can identify only good and bad prefixes with the truth values $\top$ and $\bot$ respectively, whereas all the prefixes of the other four types are given the same value ?. To discriminate them, we further divide the value ? into four truth values.

Definition 5 (Six-valued semantics). *Let $\mathbb{B}_6$ be the set of six truth values:* true $\top$, false $\bot$, possibly true $\mp$, possibly false $\pm$, possibly conclusive $+$ *and* inconclusive $\times$. *The truth value of an ω-language $L \subseteq \Sigma^*$ wrt. a finite word $u \in \Sigma^*$, denoted by $[u \models L]_6$, is $\top$, $\bot$, $\mp$, $\pm$, $+$ or $\times$ if u is a good, bad, positive, negative, neutral or ugly prefix for L, respectively.*

Note that the six-valued semantics models a rigorous correspondence between truth values and prefix types. Unlike the three-valued semantics, the inconclusive value now exactly corresponds to ugly prefixes.

The definition of four-valued monitorability is built on the following notion of four-valued u-monitorability which is used to discriminate finite words by inspecting which types of prefixes they can be extended to.

Definition 6 (Four-valued u-monitorability). *An ω-language $L \subseteq \Sigma^\omega$ is*

- weakly positively u-monitorable *for $u \in \Sigma^*$, if $\exists v \in \Sigma^*$, s.t. uv is a good prefix.*
- weakly negatively u-monitorable *for $u \in \Sigma^*$, if $\exists v \in \Sigma^*$, s.t. uv is a bad prefix.*
- positively u-monitorable *if it is weakly positively, but not weakly negatively, u-monitorable. (u has only good extensions, thus u is a good/positive prefix.)*
- negatively u-monitorable *if it is weakly negatively, but not weakly positively, u-monitorable. (u has only bad extensions, thus u is a bad/negative prefix.)*
- neutrally u-monitorable *if it is both weakly positively and weakly negatively u-monitorable. (u has both good and bad extensions, thus u is a neutral prefix.)*
- not u-monitorable *if it is neither weakly positively nor weakly negatively u-monitorable. (u has neither good nor bad extension, thus u is an ugly prefix.)*

In other words, the traditional u-monitorability is split into two parts, i.e., weakly positive and weakly negative u-monitorability. As a result, L is u-monitorable iff L is positively, negatively or neutrally u-monitorable.

Definition 7 (Four-valued monitorability). *An ω-language $L \subseteq \Sigma^\omega$ is*

- positively monitorable *if it is positively u-monitorable for every $u \in \Sigma^*$.*
- negatively monitorable *if it is negatively u-monitorable for every $u \in \Sigma^*$.*
- neutrally monitorable *if it is u-monitorable for every $u \in \Sigma^*$, and is neutrally ϵ-monitorable for the empty word ϵ.*
- non-monitorable *if it is not u-monitorable for some $u \in \Sigma^*$.*

In other words, the set of monitorable ω-languages is divided into three classes, i.e., positively, negatively and neutrally monitorable ones. Note that the definition of neutral monitorability consists of two conditions, of which the first ensures that L is monitorable while the second ensures that both of satisfaction and violation can be detected after some finite sequences of steps. We denote the four truth values (positively, negatively, neutrally and non-monitorable) by $M_\top$, $M_\bot$, M_+ and $M_\times$, respectively.

We can validate that four-valued monitorability indeed provides the informativeness we require, as described in Section 1, by showing the following theorem, that the truth values $M_\top$, $M_\bot$, and M_+ indicate that only satisfaction, only violation, and both can be detected after some finite sequences of steps, respectively. This theorem can be proved by Definitions 7 and 6, in which u is substituted by the empty word ϵ.

Theorem 2. *If an ω-language $L \subseteq \Sigma^\omega$ is*

- $M_\top$ *then* $\exists u \in \Sigma^*.\forall w \in \Sigma^\omega.uw \in L$ *and* $\nexists u \in \Sigma^*.\forall w \in \Sigma^\omega.uw \notin L$.
- $M_\bot$ *then* $\exists u \in \Sigma^*.\forall w \in \Sigma^\omega.uw \notin L$ *and* $\nexists u \in \Sigma^*.\forall w \in \Sigma^\omega.uw \in L$.
- M_+ *then* $\exists u \in \Sigma^*.\forall w \in \Sigma^\omega.uw \in L$ *and* $\exists u \in \Sigma^*.\forall w \in \Sigma^\omega.uw \notin L$.

Let us consider some simple but essential examples regarding basic temporal connectives. More examples, such as the formulas used in Section 1, will be considered in Section 6.

- Formula $\boldsymbol{F}p$ is positively monitorable, as any finite word can be extended to a good prefix (ended with p) but never a bad prefix. This means that only satisfaction, but no violation, of the property can be detected after some finite sequences of steps.
- Formula $\boldsymbol{G}p$ is negatively monitorable, as any finite word can be extended to a bad prefix (ended with $\emptyset$) but never a good prefix. This means that only violation, but no satisfaction, of the property can be detected after some finite sequences of steps.
- Formula $p\boldsymbol{U}q$ is neutrally monitorable, as it is monitorable and ϵ (more generally, any finite word of the form $p \cdots p$) can be extended to both a good prefix (ended with q) and a bad prefix (ended with $\emptyset$). This means that both of satisfaction and violation of the property can be detected after some finite sequences of steps.
- Formula $\boldsymbol{GF}p$ is non-monitorable, as any finite word can never be extended to a good or bad prefix, due to the infinite continuations $\emptyset\emptyset\cdots$ and $pp\cdots$ respectively. This means that neither satisfaction nor violation of the property can be detected.

4 Computing four-valued monitorability

In this section, we propose a procedure for computing the four-valued monitorability of ω-regular languages, based on the six-valued semantics.

The first step is a monitor construction procedure that transforms an LTL formula into a six-valued monitor, i.e., a deterministic FSM which outputs $\top$, $\bot$, $\mp$, $\pm$, $+$ and $\times$ after reading over good, bad, positive, negative, neutral and ugly prefixes respectively. For example, in Figure 2, states `P1`, `P2` and `N` are all ? states under the three-valued semantics. After refining the output function with the six-valued semantics, states `P1` and `P2` become $\mp$ states, whereas state `N` becomes a $\times$ state.

The construction procedure first constructs a three-valued monitor, using the traditional approach which requires 2ExpSpace [10]. Then we refine its output function, assigning new outputs to ? states. Specifically, our procedure traverses all the states in the monitor, and for each state, starts another nested traversal to check whether a $\top$ state or a $\bot$ state is reachable. A ? state is assigned output $\mp$ if $\top$ states are reachable but no $\bot$ state is, $\pm$ if $\bot$ states are reachable but no $\top$ state is, $+$ if both $\top$ and $\bot$ states are reachable, or $\times$ if neither is reachable. This refinement step can be done in polynomial time and NLSpace (using the three-valued monitor as the input). Thus, constructing a six-valued monitor requires also 2ExpSpace. Let us formalize the above construction procedure.

Definition 8. *Let $M = (Q, \Sigma, \delta, q_0, \mathbb{B}_3, \lambda_3)$ be a three-valued monitor for an ω-language $L \subseteq \Sigma^\omega$. The corresponding six-valued monitor $M' = (Q, \Sigma, \delta, q_0, \mathbb{B}_6, \lambda)$ is obtained by refining the output function λ_3 of M as in Figure 3.*

$$\text{for any } q \in Q, \lambda(q) = \begin{cases} \top, \text{ if } \lambda_3(q) = \top \\ \bot, \text{ if } \lambda_3(q) = \bot \\ \mp, \text{ if } \begin{cases} \lambda_3(q) \neq \top \\ \exists v \in \Sigma^*.\ \delta(q,v) = q' \wedge\ \lambda_3(q') = \top, \text{ and} \\ \forall v \in \Sigma^*.\ \delta(q,v) = q' \rightarrow \lambda_3(q') \neq \bot \end{cases} \\ \pm, \text{ if } \begin{cases} \lambda_3(q) \neq \bot \\ \exists v \in \Sigma^*.\ \delta(q,v) = q' \wedge\ \lambda_3(q') = \bot, \text{ and} \\ \forall v \in \Sigma^*.\ \delta(q,v) = q' \rightarrow \lambda_3(q') \neq \top \end{cases} \\ +, \text{ if } \begin{cases} \exists v \in \Sigma^*.\ \delta(q,v) = q' \wedge\ \lambda_3(q') = \top, \text{ and} \\ \exists v \in \Sigma^*.\ \delta(q,v) = q' \wedge\ \lambda_3(q') = \bot \end{cases} \\ \times, \text{ if } \begin{cases} \forall v \in \Sigma^*.\ \delta(q,v) = q' \rightarrow \lambda_3(q') \neq \top, \text{ and} \\ \forall v \in \Sigma^*.\ \delta(q,v) = q' \rightarrow \lambda_3(q') \neq \bot \end{cases} \end{cases}$$

Fig. 3: The output function λ.

We can show the following lemma, that the six-valued monitor can be used to compute the truth value of an ω-language wrt. a finite word. This lemma can be proved by Definitions 5 and 2, Lemma 1 and Definition 8.

Lemma 2. *Let $M = (Q, \Sigma, \delta, q_0, \mathbb{B}_6, \lambda)$ be a six-valued monitor for an ω-language $L \subseteq \Sigma^\omega$. For any $u \in \Sigma^*$, $[u \models L]_6 = \lambda(\delta(q_0, u))$.*

As a property of the six-valued monitor, the following theorem shows that each state in a monitor can be reached by exactly one type of prefixes (by Lemma 2 and Definition 5).

Theorem 3. *Let $M = (Q, \Sigma, \delta, q_0, \mathbb{B}_6, \lambda)$ be a six-valued monitor for an ω-language $L \subseteq \Sigma^\omega$. For a state $q \in Q$, $\lambda(q)$ equals $\top$, $\bot$, $\mp$, $\pm$, $+$ or $\times$, iff it can be reached by good, bad, positive, negative, neutral or ugly prefixes, respectively.*

Based on the six-valued monitor, the second step determines the four-valued monitorability of an ω-language L by checking whether its monitor has some specific reachable states. The monitorability of L is $M_\top$ iff neither $\times$ nor $\bot$ states are reachable (thus neither $\pm$ nor $+$ states are reachable), $M_\bot$ iff neither $\times$ nor $\top$ states are reachable (thus neither $\mp$ nor $+$ states are reachable), M_+ iff no $\times$ state is reachable but a $+$ state is reachable (thus both $\top$ and $\bot$ states are reachable), and $M_\times$ iff a $\times$ state is reachable. These rules can be formalized:

Theorem 4. *Let $M = (Q, \Sigma, \delta, q_0, \mathbb{B}_6, \lambda)$ be a six-valued monitor for an ω-language $L \subseteq \Sigma^\omega$. The monitorability of L:*

$$\eta(L) = \begin{cases} M_\top, \textit{ iff } \forall u \in \Sigma^*.\ \delta(q_0,u) = q' \rightarrow \lambda(q') \neq \times \wedge \lambda(q') \neq \bot \\ M_\bot, \textit{ iff } \forall u \in \Sigma^*.\ \delta(q_0,u) = q' \rightarrow \lambda(q') \neq \times \wedge \lambda(q') \neq \top \\ M_+, \textit{ iff } \begin{cases} \forall u \in \Sigma^*.\ \delta(q_0,u) = q' \rightarrow \lambda(q') \neq \times, \textit{ and} \\ \exists u \in \Sigma^*.\ \delta(q_0,u) = q' \wedge\ \lambda(q') = + \end{cases} \\ M_\times, \textit{ iff } \exists u \in \Sigma^*.\ \delta(q_0,u) = q' \wedge\ \lambda(q') = \times \end{cases}$$

The above checking procedure can be done in linear time and NLSpace by traversing all the states of monitor. However, note that this procedure is performed after constructing the monitor. Thus, when an ω-regular language L is given in terms of an LTL formula, the four-valued monitorability of L can be computed in 2ExpSpace; the same complexity as for two-valued monitorability. As we will see in Section 6, the small size of standard LTL patterns means that four-valued monitorability can be computed in very little time

Now consider other representations of ω-regular languages. If L is given in terms of an NBA, we first explicitly complement the NBA, and the rest of the procedure stays the same. However, the complement operation also involves an exponential blowup. If L is given in terms of an ω-regular expression, we first build an NBA for the expression, which can be done in polynomial time, and the rest of the procedure is the same as for NBA. Hence, independent of the concrete representation, four-valued monitorability of an ω-regular language can be computed in 2ExpSpace, by using the monitor-based procedure.

5 State-level four-valued weak monitorability

In this section, we apply four-valued monitorability at the state-level, to predict whether satisfaction and violation can be detected *starting from a given state in a monitor*. Recall that the notions of monitorability (cf. Definitions 3 and 7) are defined using the extensions to good and bad prefixes. However, good and bad prefixes are defined for an ω-language, not for a state. Thus such definitions cannot be directly applied at the state-level. Instead, we define state-level monitorability using the reachability of $\top$ and $\bot$ states, which are equivalent notions to good and bad prefixes according to Theorem 3.

Another note is that the resulting state-level monitorability is too strong to meet our requirements, because it places restrictions on all the states reachable from the considered state. For example, in Figure 2, we require discriminating states `P1` and `P2` from state `N`, as satisfaction can be detected starting from `P1` and `P2`, but neither satisfaction nor violation can be detected starting from `N`. However, `P1`, `P2` and `N` are all non-monitorable as neither $\top$ states nor $\bot$ states are reachable from `N` (in turn, reachable from `P1` and `P2`). To provide the required distinction, we should use a weaker form of state-level monitorability as follows.

Definition 9 (State-level four-valued weak monitorability). *Let* $M = (Q, \Sigma, \delta, q_0, \mathbb{B}_6, \lambda)$ *be a six-valued monitor. A state* $q \in Q$ *is*

- weakly $M_\top$ *if a* $\top$ *state but no* $\bot$ *state is reachable from* q.
- weakly $M_\bot$ *if a* $\bot$ *state but no* $\top$ *state is reachable from* q.
- weakly M_+ *if both a* $\top$ *state and a* $\bot$ *state are reachable from* q.
- weakly $M_\times$ *if neither* $\top$ *states nor* $\bot$ *states are reachable from* q.

A state is *weakly monitorable*, iff it is weakly positively, negatively or neutrally monitorable. For example, in Figure 2, states `P1`, `P2` and `T` are all weakly positively monitorable as `T` is a reachable $\top$ state, while state `N` is weakly non-monitorable. Thus, states `P1` and `P2` can now be discriminated from state `N`.

We can validate that state-level four-valued weak monitorability can indeed predict whether satisfaction and violation can be detected *starting from a given state*, as anticipated in Section 1, by showing the following theorem, that the truth values $M_\top$, $M_\bot$, M_+ and $M_\times$ indicate that only satisfaction, only violation, both and neither can be detected, respectively. This theorem can be proved by Definition 9 and Theorem 3.

Theorem 5. *Let $M = (Q, \Sigma, \delta, q_0, \mathbb{B}_6, \lambda)$ be a six-valued monitor. Suppose a state $q \in Q$ can be reached from q_0 by reading $u \in \Sigma^*$, i.e., $\delta(q_0, u) = q$. If q is*

- *weakly $M_\top$ then $\exists v \in \Sigma^*.\forall w \in \Sigma^\omega.uvw \in L \wedge \nexists v \in \Sigma^*.\forall w \in \Sigma^\omega.uvw \notin L$.*
- *weakly $M_\bot$ then $\exists v \in \Sigma^*.\forall w \in \Sigma^\omega.uvw \notin L \wedge \nexists v \in \Sigma^*.\forall w \in \Sigma^\omega.uvw \in L$.*
- *weakly M_+ then $\exists v \in \Sigma^*.\forall w \in \Sigma^\omega.uvw \in L \wedge \exists v \in \Sigma^*.\forall w \in \Sigma^\omega.uvw \notin L$.*
- *weakly $M_\times$ then $\nexists v \in \Sigma^*.\forall w \in \Sigma^\omega.uvw \in L \wedge \nexists v \in \Sigma^*.\forall w \in \Sigma^\omega.uvw \notin L$.*

The four truth values can be used in state-level optimizations of monitoring algorithms:

- If a state is weakly positively (resp. negatively) monitorable, then a monitor object can be safely removed when it enters this state, provided that only violation (resp. satisfaction) handlers are specified, as no handler can be triggered.
- If a state is weakly neutrally monitorable, then a monitor object must be preserved if it is at this state as both satisfaction and violation can be detected after some continuations.
- If a state is weakly non-monitorable, then a monitor object can be safely removed when it enters this state as no verdict can be detected after any continuation.

Besides, a monitor object can also be removed when it enters a $\top$ state or a $\bot$ state, as any finite or infinite continuation yields the same verdict.

Let us consider the relationship between the language-level monitorability and the state-level weak monitorability. The following lemma shows that the monitorability of an ω-language depends on the weak monitorability of all the reachable states of its monitor. This means, if an ω-language is non-monitorable, then the state space of its monitor may consist of both weakly monitorable and weakly non-monitorable states.

Lemma 3. *Let $M = (Q, \Sigma, \delta, q_0, \mathbb{B}_6, \lambda)$ be a six-valued monitor for an ω-language $L \subseteq \Sigma^\omega$. L is monitorable iff every reachable state of M is weakly monitorable.*

Let us consider how one can compute the state-level four-valued weak monitorability for each state in a six-valued monitor. We first formalize a mapping from truth values to weak monitorability, and then show that the state-level weak monitorability can be quickly computed from the output of the state.

Definition 10 (Value-to-weak-monitorability). *Let $\mathtt{vtom} : \mathbb{B}_6 \mapsto \mathbb{M}_4$ be the* value-to-weak-monitorability operator *that converts a truth value in $\mathbb{B}_6$ into*

the corresponding result of weak monitorability in $\mathbb{M}_4 = \{M_\top, M_\bot, M_+, M_\times\}$, *defined as follows:* $\mathit{vtom}(\top) = \mathit{vtom}(\mp) = M_\top$, $\mathit{vtom}(\bot) = \mathit{vtom}(\pm) = M_\bot$, $\mathit{vtom}(+) = M_+$ *and* $\mathit{vtom}(\times) = M_\times$.

Theorem 6. *Let* $M = (Q, \Sigma, \delta, q_0, \mathbb{B}_6, \lambda)$ *be a six-valued monitor for an* ω*-language* $L \subseteq \Sigma^\omega$. *The four-valued weak monitorability of* $q \in Q$ *equals* $\mathit{vtom}(\lambda(q))$.

6 Implementation and experimental results

We have developed a new tool, MONIC, that implements the proposed procedure for computing four-valued monitorability of LTL formulas. MONIC also supports deciding two-valued monitorability. We have evaluated its effectiveness using a set of LTL formulas, including formulas ϕ_1 to ϕ_6 (used in Section 1) and Dwyer et al.'s 97 LTL patterns [18,10]. The tool implementation MONIC and the dataset of LTL formulas are available at `https://github.com/drzchen/monic` (Appendix A explains how to run it). The evaluation was performed on an ordinary laptop, equipped with an Intel Core i7-6500U CPU (at 2.5GHz), 4GB RAM and Ubuntu Desktop (64-bit).

The result on formulas ϕ_1 to ϕ_6 shows that: ϕ_1 is neutrally monitorable, ϕ_2 is non-monitorable, ϕ_3 is positively monitorable, ϕ_4 is negatively monitorable, ϕ_5 is neutrally monitorable, and ϕ_6 is non-monitorable (but weakly monitorable). Thus, the violation of ϕ_3 and the satisfaction of ϕ_4 can never be detected, whereas both verdicts are active for ϕ_1 and ϕ_5. Further, ϕ_4 and ϕ_5 can be discriminated by their different monitorability results.

We also ran MONIC on Dwyer et al.'s specification patterns [18,10], of which 97 are well-formed LTL formulas. The result shows that 55 formulas are monitorable and 42 are non-monitorable. For those monitorable ones, 6 are positively monitorable, 40 are negatively monitorable and 9 are neutrally monitorable. Our result disagrees with the two-valued result reported in [10] only on the 6th LTL formula listed in the Appendix of [10]. More precisely, MONIC reports negatively monitorable, whereas the result in [10] is non-monitorable. The formula is as follows (`!` for $\neg$, `&` for $\wedge$, `|` for $\vee$, `->` for $\rightarrow$, `U` for $\boldsymbol{U}$, `<>` for $\boldsymbol{F}$, `[]` for $\boldsymbol{G}$):

```
[](("call" & <>"open") ->
   ((!"atfloor" & !"open") U
    ("open" | (("atfloor" & !"open") U
               ("open" | ((!"atfloor" & !"open") U
                          ("open" | (("atfloor" & !"open") U
                          ("open" | (!"atfloor" U "open")))))))))
```

The result in [10] is unreliable as it is based on manual inspection of monitors and no tool is implemented in that work. To validate, a manual inspection of its monitor (in Figure 4) shows that our result is correct. Indeed, state `F` is a $\bot$ state, and states `N1` to `N7` are all $\pm$ states that can reach the $\bot$ state `F`.

Finally, the above results for ϕ_1 to ϕ_6 and the 97 LTL patterns were computed in 0.03 and 0.07 seconds, with 16MB and 20MB memory consumed, respectively

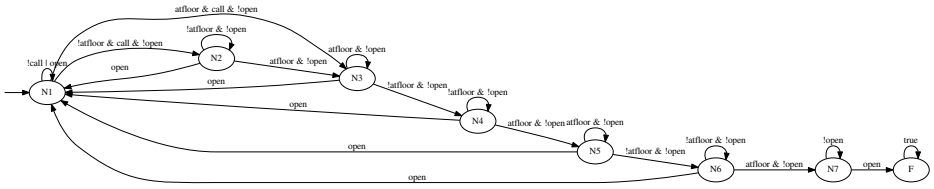

Fig. 4: The monitor of an LTL pattern.

(all reported by GNU `time`). To conclude, the results show that MONIC can correctly report both two-valued and four-valued monitorability of typical formulas in very little time.

7 Related work

Monitorability is a principal foundational question in RV because it delineates which properties can be monitored at runtime. The classical results on monitorability have been established for ω-languages, especially for LTL [37,10,7]. Francalanza and Aceto et al. have studied monitorability for the Hennessy-Milner logic with recursion, both with a branching-time semantics [21,22,23,1] and with a linear-time semantics [2]. There exist some variants of monitorability as well. For example, monitorability has been considered over unreliable communication channels which may reorder or lose events [30]. However, all of the existing works only consider two-valued notions of monitorability at the language-level.

Monitorability has been studied in other contexts. For example, a topological viewpoint [16] and the correspondence between monitorability and the classifications of properties (e.g., the safety-progress and safety-liveness classifications) [19,20,35] have been established. A hierarchy of monitorability definitions (including monitorability and weak monitorability [14]) has been defined wrt. the operational guarantees provided by monitors [3].

A four-valued semantics for LTL [8,9] has been proposed to refine the three-valued semantics [10]. It divides the inconclusive truth value ? into two values: *currently true* and *currently false*, i.e., whether the finite sequence observed so far satisfies the property based on a finite semantics for LTL. Note that it provides more information on what has already been seen, whereas our six-valued semantics describes what verdicts can be detected in the future continuation.

8 Conclusion

We have proposed four-valued monitorability and the corresponding computation procedure for ω-regular languages. Then we applied the four-valued notion at the state-level. To our knowledge, this is the first study of multi-valued monitorability, inspired by practical requirements from RV. We believe that our work

and implementation can be integrated into RV tools to provide information at the development stage and thus avoid the development of unnecessary handlers and the use of monitoring that cannot add value, enhance property debugging, and enable state-level optimizations of monitoring algorithms.

References

1. Aceto, L., Achilleos, A., Francalanza, A., Ingólfsdóttir, A.: A framework for parameterized monitorability. In: Proceedings of FOSSACS'18. LNCS, vol. 10803, pp. 203–220. Springer (2018)
2. Aceto, L., Achilleos, A., Francalanza, A., Ingólfsdóttir, A., Lehtinen, K.: Adventures in monitorability: from branching to linear time and back again. Proceedings of the ACM on Programming Languages **3**(POPL'19), 52:1–52:29 (2019)
3. Aceto, L., Achilleos, A., Francalanza, A., Ingólfsdóttir, A., Lehtinen, K.: An operational guide to monitorability. In: Proceedings of SEFM'19. LNCS, vol. 11724, pp. 433–453. Springer (2019)
4. Allan, C., Avgustinov, P., Christensen, A.S., Hendren, L.J., Kuzins, S., Lhoták, O., de Moor, O., Sereni, D., Sittampalam, G., Tibble, J.: Adding trace matching with free variables to AspectJ. In: Proc. of OOPSLA'05. pp. 345–364. ACM (2005)
5. Avgustinov, P., Tibble, J., de Moor, O.: Making trace monitors feasible. In: Proceedings of OOPSLA'07. pp. 589–608. ACM (2007)
6. Bartocci, E., Falcone, Y., Francalanza, A., Reger, G.: Introduction to runtime verification. In: Lectures on Runtime Verification - Introductory and Advanced Topics, LNCS, vol. 10457, pp. 1–33. Springer (2018)
7. Bauer, A.: Monitorability of ω-regular languages. CoRR **abs/1006.3638** (2010)
8. Bauer, A., Leucker, M., Schallhart, C.: The good, the bad, and the ugly, but how ugly is ugly? In: RV'07. LNCS, vol. 4839, pp. 126–138. Springer (2007)
9. Bauer, A., Leucker, M., Schallhart, C.: Comparing LTL semantics for runtime verification. J. Log. Comput. **20**(3), 651–674 (2010)
10. Bauer, A., Leucker, M., Schallhart, C.: Runtime verification for LTL and TLTL. ACM Transactions on Software Engineering and Methodology (TOSEM) **20**(4), 14 (2011)
11. Chen, F., Rosu, G.: MOP: an efficient and generic runtime verification framework. In: Proceedings of OOPSLA'07. pp. 569–588. ACM (2007)
12. Chen, Z.: Parametric runtime verification is NP-complete and coNP-complete. Information Processing Letters **123**, 14–20 (2017)
13. Chen, Z., Wang, Z., Zhu, Y., Xi, H., Yang, Z.: Parametric runtime verification of C programs. In: TACAS'16. LNCS, vol. 9636, pp. 299–315. Springer (2016)
14. Chen, Z., Wu, Y., Wei, O., Sheng, B.: Deciding weak monitorability for runtime verification. In: Proceedings of ICSE'18. pp. 163–164. ACM (2018)
15. d'Amorim, M., Rosu, G.: Efficient monitoring of ω-languages. In: Proceedings of CAV'05. LNCS, vol. 3576, pp. 364–378. Springer (2005)
16. Diekert, V., Leucker, M.: Topology, monitorable properties and runtime verification. Theor. Comput. Sci. **537**, 29–41 (2014)
17. Drusinsky, D.: The temporal rover and the ATG rover. In: Proceedings of SPIN'00. LNCS, vol. 1885, pp. 323–330. Springer (2000)
18. Dwyer, M.B., Avrunin, G.S., Corbett, J.C.: Patterns in property specifications for finite-state verification. In: Proceedings of ICSE'99. pp. 411–420. ACM (1999)

19. Falcone, Y., Fernandez, J., Mounier, L.: Runtime verification of safety-progress properties. In: Proceedings of RV'09. LNCS, vol. 5779, pp. 40–59. Springer (2009)
20. Falcone, Y., Fernandez, J.C., Mounier, L.: What can you verify and enforce at runtime? International Journal on Software Tools for Technology Transfer (STTT) **14**(3), 349–382 (2012)
21. Francalanza, A.: A theory of monitors. In: Proceedings of FOSSACS'16. LNCS, vol. 9634, pp. 145–161. Springer (2016)
22. Francalanza, A., Aceto, L., Achilleos, A., Attard, D.P., Cassar, I., Monica, D.D., Ingólfsdóttir, A.: A foundation for runtime monitoring. In: Proceedings of RV'17. LNCS, vol. 10548, pp. 8–29. Springer (2017)
23. Francalanza, A., Aceto, L., Ingólfsdóttir, A.: Monitorability for the hennessy-milner logic with recursion. Formal Methods in System Design **51**(1), 87–116 (2017)
24. Geilen, M.: On the construction of monitors for temporal logic properties. Electr. Notes Theor. Comput. Sci. **55**(2), 181–199 (2001)
25. Havelund, K.: Runtime verification of C programs. In: Proceedings of TestCom/-FATES'08. LNCS, vol. 5047, pp. 7–22. Springer (2008)
26. Havelund, K., Reger, G.: Runtime verification logics - A language design perspective. In: Models, Algorithms, Logics and Tools - Essays Dedicated to Kim Guldstrand Larsen on the Occasion of His 60th Birthday. LNCS, vol. 10460, pp. 310–338. Springer (2017)
27. Havelund, K., Reger, G., Thoma, D., Zalinescu, E.: Monitoring events that carry data. In: Lectures on Runtime Verification - Introductory and Advanced Topics, LNCS, vol. 10457, pp. 61–102. Springer (2018)
28. Havelund, K., Rosu, G.: Synthesizing monitors for safety properties. In: Proceedings of TACAS'02. LNCS, vol. 2280, pp. 342–356. Springer (2002)
29. Havelund, K., Rosu, G.: Runtime verification - 17 years later. In: Proceedings of RV'18. LNCS, vol. 11237, pp. 3–17. Springer (2018)
30. Kauffman, S., Havelund, K., Fischmeister, S.: Monitorability over unreliable channels. In: Proceedings of RV'19. LNCS, vol. 11757, pp. 256–272. Springer (2019)
31. Kupferman, O., Vardi, M.Y.: Model checking of safety properties. Formal Methods in System Design **19**(3), 291–314 (2001)
32. Leucker, M., Schallhart, C.: A brief account of runtime verification. Journal of Logic and Algebraic Programming **78**(5), 293–303 (2009)
33. Manna, Z., Pnueli, A.: The Temporal Logic of Reactive and Concurrent Systems: Specification. Springer-Verlag (1992)
34. Meredith, P.O., Jin, D., Griffith, D., Chen, F., Rosu, G.: An overview of the MOP runtime verification framework. International Journal on Software Tools for Technology Transfer (STTT) **14**(3), 249–289 (2012)
35. Peled, D., Havelund, K.: Refining the safety-liveness classification of temporal properties according to monitorability. In: Models, Mindsets, Meta: The What, the How, and the Why Not? - Essays Dedicated to Bernhard Steffen on the Occasion of His 60th Birthday. LNCS, vol. 11200, pp. 218–234. Springer (2019)
36. Pnueli, A.: The temporal logic of programs. In: Proceedings of FOCS'77. pp. 46–57. IEEE Computer Society (1977)
37. Pnueli, A., Zaks, A.: PSL model checking and run-time verification via testers. In: Proceedings of FM'06. LNCS, vol. 4085, pp. 573–586. Springer (2006)
38. Rosu, G., Chen, F.: Semantics and algorithms for parametric monitoring. Logical Methods in Computer Science **8**(1), 1–47 (2012)

A Implementation and experimental results

The tool implementation Monic and the dataset of LTL formulas are available at `https://github.com/drzchen/monic`. After unzipping the downloaded file, you can see an executable file `monic` and two textual files which contain the LTL formulas. Note that Monic can only run on Ubuntu Desktop (64-bit) or compatible systems.

To compute two-valued monitorability, one may use the following commands:

```
$ ./monic -c four_valued_examples.txt
$ ./monic -c ltl_patterns.txt
```

To compute four-valued monitorability, one may use the following commands:

```
$ ./monic -n4 -c four_valued_examples.txt
$ ./monic -n4 -c ltl_patterns.txt
```