Runtime Verification of Fixpoint Logic: Synthesis of Optimal Monitors

M.K. Lehtinen

University of Kiel

Abstract. Runtime monitoring is a light-weight verification technique which analyses the execution traces that a system produces rather than its state-space. In contrast to prior work, which focuses either on monitoring linear time specifications or restricted branching-time specifications, here we argue that arbitrary specifications can have meaningful runtime monitors. We propose a definition of monitor optimality for specifications expressed in Hennessy-Milner logic with recursion, μHML, and show that every μHML formula has an optimal deterministic monitor which can be synthesised in 2-Exptime.

Runtime monitoring [18] is a light-weight verification technique which, in contrast to model-checking, analyses executions of a system rather than the system itself. It treats the system as a black box that produces traces, and is therefore particularly suitable when the system under scrutiny is not itself observable or fully understood. This may be the case because of confidentiality issues, legacy code, or because of the system's high complexity, for example. As a low-overhead verification method that is complementary to other techniques [8] – model-checking, theorem proving, and testing for instance – it can also be used as part of a hybrid verification strategy.

While runtime monitoring evades the state explosion problem associated with modelling the state-space of a system, this comes at the cost of expressivity: it is more restricted in the kind of specifications it can fully verify. This leads to the classification of specifications according to whether they are amendable to runtime monitoring: we speak of *monitorability*. So far runtime monitoring and monitorability have mainly been studied for the verification of trace properties [4, 11, 15, 16], expressed for example as formulas of LTL [19], or of variants designed specifically for monitoring [3, 9, 10]. However, typically verification is also concerned with properties of whole processes, rather than just their traces. The potential of runtime monitoring for branching-time specifications has only recently been considered [14, 13]. So far, it has only been argued to be a sound verification approach for a subset of branching-time properties. In this paper we consider the general case, and study what runtime monitoring can achieve for the verification of arbitrary specifications expressed in the Hennessy–Milner logic with recursion, μHML.

We consider the general setting originally proposed by Francalanza *et al.* [14, 13] to study monitorability in the branching-time setting: a formula of μHML

describes a specification, while a CCS process describes the system being monitored. A monitor is then defined as a special CCS process which, when instrumenting the process of interest, executes in tandem with it without interfering, and terminates in one of three special states: **yes**, **no** and **end**. The monitor then either validates the property, flags a violation thereof, or remains indecisive.

In this setting a monitor is sound for a formula ψ if it only accepts processes satisfying ψ, and dually only rejects processes violating ψ; a monitor is satisfaction-complete if it accepts all processes that satisfy ψ, and dually violation-complete if it rejects all processes which violate ψ. The fragment of μHML without the operators $\vee$, μ, nor $\Diamond$ – the safety fragment – is exactly the fragment of μHML that has violation-complete monitors, while its dual, the co-safety fragment has satisfaction-complete monitors [14]. The union of these fragments is then the sublogic of μHML with sound and (satisfaction- or violation-) complete monitors.

This fragment of μHML is rather restrictive, and excludes many interesting properties. This should be expected, since monitoring is necessarily much weaker than model-checking. However, here we argue that properties outside of this fragment can also benefit from monitors: a monitor that rejects nearly all processes violating a property may still be useful, despite not being complete. We therefore study a weaker form of monitorability and consider monitors that are sound, but not necessarily complete. A formula can have several sound monitors, in which case one may ask for the *best* monitor, *i.e.* the one coming closest to completeness for a property. For instance, the monitor which is always indecisive is a particularly poor – albeit universally sound – monitor, while a complete monitor is obviously optimal when it exists. Here we show how to compute, given a μHML formula, a monitor that is optimal, in the sense that it gives a verdict for any process for which some sound monitor gives a verdict.

The core insight of this paper is that runtime verification, which has so far almost exclusively been considered in the linear-time domain, can also be used for the partial verification of arbitrary branching time specifications, which have previously been dismissed as unmonitorable. This work is particularly relevant to the design of hybrid verification strategies which combine different verification methods: the optimal monitor computed here is an obvious candidate for maximising the use of runtime verification.

Our technical contributions are:

- We show that the sound rejection monitors of a formula ψ are in one-to-one correspondence with its safety consequences, *i.e.,* safety formulas which are implied by ψ, in Section 2. Then, finding optimal sound rejection monitors reduces to the problem of finding, given an arbitrary formula, its strongest consequence in the safety fragment – Theorem 2.
- Our main technical contribution is to show that this strongest safety consequence, and therefore the optimal rejection-monitor, can be computed in 2-Exptime, in Section 3. The optimal acceptance-monitor is obtained by duality.

– We then show, in Section 4, that the obtained optimal monitors are deterministic, and justify why our synthesis is double exponential.

In brief, we solve the μHML problem consisting of finding, given an arbitrary formula, its strongest consequence in the safety fragment, and show that it can be used in the monitoring setting to generate optimal deterministic monitors for arbitrary formulas.

1 Background

Our set-up is in line with the presentation from [13]. In addition, we recall some concepts familiar from fixpoint model-checking. We refer the interested reader to Bradfield and Stirling's handbook chapters [7, 6] for an introduction to the propositional modal μ calculus, of which μHML [17] is an almost complete fragment, to Francalanza *et al.* [13] for more background on monitorability, and to Leucker and Schallhart [18] for a brief introduction to Runtime Verification.

1.1 Processes and their properties

Fix a finite set $Act = \{\alpha, \beta, ...\}$ of *actions*, and countable sets $PVar = \{x, y, ...\}$ of process variables, and $FVar = \{X, Y, ...\}$ of fixpoint variables.

Definition 1 (Process).

$$p, q \in Proc := \texttt{nil} \mid \alpha.p \mid p + q \mid \texttt{rec}x.p \mid x$$

Definition 2 (Computation tree).
First we define the transition relation $\xrightarrow{\alpha} \in Proc \times Proc$ *(where* $\alpha \in Act$*) of of processes as follows:*

– $\alpha.p \xrightarrow{\alpha} p$;
– *If* $p \xrightarrow{\alpha} p'$ *then* $p + q \xrightarrow{\alpha} p'$ *and* $q + p \xrightarrow{\alpha} p'$;
– *If* $p \xrightarrow{\alpha} p'$ *then* $\texttt{rec}x.p \xrightarrow{\alpha} p'$; *and*
– *If* $\texttt{rec}x.p \xrightarrow{\alpha} p'$ *then* $x \xrightarrow{\alpha} p'$.

Then a computation tree $T = (S, L, E)$ *is a potentially infinite labelled tree consisting of a set of states* S *rooted at some initial state* $s_\iota \in S$*, a labelling* $L : S \to Proc$*, and a successor relation* $E \subseteq S \times Act \times S$ *such that if* $(s, \alpha, s') \in E$*, then* $L(s) \xrightarrow{\alpha} L(s')$.

A computation tree T *is complete if for all* s *with label* $q = L(s)$*, and all* q' *such that* $q \xrightarrow{\alpha} q'$*, some* s' *such that* $(s, \alpha, s') \in E$ *is labelled* $L(s') = q'$*. The computation tree of a process* p *is a minimal complete computation tree of which the root is labelled with* p.

Definition 3. *Given a computation tree T, we write $T\restriction^n$ for its prefix of depth n, i.e., the finite tree of which the states are states of T within n steps from the root, and the labelling and edge-relation are those of T restricted to these states.*

Definition 4. *The logic μHML, a variation of the modal μ calculus without propositional variables, is generated by the following grammar:*

$$\phi := \bot \mid \top \mid X \mid \phi \wedge \phi \mid \phi \vee \phi \mid \langle\alpha\rangle\phi \mid [\alpha]\phi \mid \mu X.\phi \mid \nu X.\phi$$

The order of precedence is: modal operators, conjunctions, disjunctions, fixpoint operators. In a subformula $\mu X.\phi$ (or $\nu X.\phi$) the operator μ (or ν) binds all free occurrences of X in ϕ. The subformula ϕ is then said to be the binding formula of X. In a slight abuse of notation, we say that $\mu X.\phi$ is the immediate subformula of X. We assume all formulas to be guarded, i.e., every occurrence of a fixpoint variable within its binding is within the scope of a modal operator. For clarity, we will assume that every fixpoint operator binds a uniquely named fixpoint variable. A sentence is a formula without free variables. We take the size of a formula to be the number of its distinct subformulas.

A formula ψ of μHML is evaluated on a computation tree $T = (S, L, E)$ with and interpretation $\mathcal{I}$ of the fixpoint variables of the formula. We write $\| \psi \|^T_{\mathcal{I}}$ for the set of states in T which satisfy ψ, as defined by:

$$\begin{aligned}
\| X \|^T_{\mathcal{I}} &= \mathcal{I}(X) \\
\| \phi_0 \wedge \phi_1 \|^T_{\mathcal{I}} &= \| \phi_0 \|^T_{\mathcal{I}} \cap \| \phi_1 \|^T_{\mathcal{I}} \\
\| \phi_0 \vee \phi_1 \|^T_{\mathcal{I}} &= \| \phi_0 \|^T_{\mathcal{I}} \cup \| \phi_1 \|^T_{\mathcal{I}} \\
\| \langle\alpha\rangle\phi \|^T_{\mathcal{I}} &= \{s | \exists s' \in \| \phi \|^T_{\mathcal{I}} . (s, \alpha, s') \in E\} \\
\| [\alpha]\phi \|^T_{\mathcal{I}} &= \{s | \forall s' . (s, \alpha, s') \in E \implies s' \in \| \phi \|^T_{\mathcal{I}}\} \\
\| \nu X.\phi \|^T_{\mathcal{I}} &= \bigcup \{C \subseteq S | C \subseteq \| \phi \|^T_{\mathcal{I}[X:=C]}\} \\
\| \mu X.\phi \|^T_{\mathcal{I}} &= \bigcap \{C \subseteq S | \| \phi \|^T_{\mathcal{I}[X:=C]} \subseteq C\}
\end{aligned}$$

where $\mathcal{I}[X := C]$ interprets X as the set C but otherwise agrees with $\mathcal{I}$.

For a state s of T and a subformula ϕ of a sentence Ψ of L_μ, we say that ϕ holds in s, written $s \models \phi$, if $s \in \| \phi \|^T_{\mathcal{I}}$, where $\mathcal{I}$ maps each fixpoint variable X to $\| \mu X.\phi_X \|^T_{\mathcal{I}}$. Finally, write $T \models \Psi$ if $r \models \Psi$ for the root r of T. Two formulas are equivalent if they agree on all computation trees T.

We say that a process satisfies a sentence ϕ, written $p \in \phi$, if ϕ holds in the root of its computation tree.

Instead of working with the above semantics, we will use *consistent annotations* and *counter-annotations* which witness that a property holds or does not hold in a tree[1].

[1] For the playfully minded reader, consistent annotations define a winning strategy for the Verifier in the model-checking game; counter-annotations define one for Falsifier.

Definition 5 (Consistent annotation). *An annotation $A : T \to \mathcal{P}(sf(\Psi))$ is a labelling of T with sets of subformulas of a sentence Ψ of μHML. An annotation is locally consistent if for all states $s \in T$:*

- $\bot \notin A(s)$;
- *If* $\mu X.\phi \in A(s)$ *or* $\nu X.\phi \in A(s)$ *then* $\phi \in A(s)$;
- *If* $X \in A(s)$ *then* $\mu X.\phi \in A(s)$;
- *If* $\phi \wedge \psi \in A(s)$ *then* $\phi \in A(s)$ *and* $\psi \in A(s)$;
- *If* $\phi \vee \psi \in A(s)$ *then* $\phi \in A(s)$ *or* $\psi \in A(s)$;
- *If* $\langle\alpha\rangle\phi \in A(s)$ *then* $\phi \in A(s')$ *for some* α*-successor* s' *of* s;
- *If* $[\alpha]\phi \in A(s)$ *then* $\phi \in A(s')$ *for all* α*-successors* s' *of* s.

A trace in an annotation is a maximal sequence $f_0 f_1 \ldots$ of formulas such that f_{i+1} is an immediate subformula of f_i. A branch of T carries a trace π from a state s if $\pi_0 \in A(s)$ and for all i, v, if $\pi_i \in A(v)$ and π_i is not modal, then also $\pi_{i+1} \in A(v)$ and if π_i is modal then $\pi_{i+1} \in A(v')$ where v' is the successor of v on the branch.

An annotation is consistent if it is both locally consistent and on all traces carried by branches of T, the outermost fixpoint binding appearing infinitely often along the trace is a ν-binding.

It is a standard result (see for example [7] for a more thorough discussion) that for a state s of a computation tree T and a subformula ϕ of Ψ, we have that $s \in \psi$ if and only if there is a consistent annotation such that $\phi \in A(s)$. We call this a consistent ϕ-annotation of s. In particular the root s_ι of T satisfies Ψ if and only if there is a consistent annotation A such that $\Psi \in A(s_\iota)$.

For convenience, we will also define the dual, a consistent counter-annotation, which witnesses that a computation tree violates a property.

Definition 6 (Consistent counter-annotation). *A counter-annotation $C : T \to P(sf(\Psi))$ is a labelling of T with sets of subformulas of a sentence Ψ of μHML. A counter-annotation is locally consistent if for all states $s \in T$:*

- $\top \notin C(s)$;
- *If* $\mu X.\phi \in C(s)$ *or* $\nu X.\phi \in C(s)$ *then* $\phi \in C(s)$;
- *If* $X \in C(s)$ *then* $\mu X.\phi \in C(s)$;
- *If* $\phi \wedge \psi \in C(s)$ *then* $\phi \in C(s)$ *or* $\psi \in C(s)$;
- *If* $\phi \vee \psi \in C(s)$ *then* $\phi \in C(s)$ *and* $\psi \in C(s)$;
- *If* $\langle\alpha\rangle\phi \in C(s)$ *then* $\phi \in C(s')$ *for all* α*-successor* s' *of* s;
- *If* $[\alpha]\phi \in C(s)$ *then* $\phi \in C(s')$ *for some* α*-successors* s' *of* s.

Traces are defined as for annotations. A counter-annotation is consistent if it is both locally consistent and all traces carried by branches of T, the outermost fixpoint binding appearing infinitely often along the trace is a μ-binding.

Then a state s of a computation tree T violates a subformula ϕ of Ψ if and only if there is a consistent counter-annotation such that $\phi \in C(s)$.

When referring to an annotations (or counter-annotations) A witnessing $s \in \phi$ (or $s \notin \phi$) for some state s and some formula ϕ, we take that annotation to be minimal, in the sense that removing any set of subformulas from the labelling renders the annotation either inconsistent or removes ϕ from $A(s)$.

1.2 Monitoring

For completeness, we briefly present the monitoring setting defined by Francalanza *et al.* [14]. However, our technical results are independent of the exact definitions of monitors, processes and instrumentations – they depend only on the concepts of soundness and completeness, and the one-to-one correspondence between rejection monitors and safety formulas, or acceptance monitors and co-safety formulas.

Definition 7 (Monitor). *A rejection monitor on Act is defined by the following process grammar:*

$$m, n, \in Mon := \texttt{end} \mid \texttt{no} \mid \alpha.m \mid m + n \mid \texttt{rec}x.m \mid x$$

where $x \in MVar$.

The `end` constant indicates an indecisive verdict, while `no` indicates rejection. Acceptance monitors are defined with an acceptance verdict `yes` instead of `no`.

The instrumentation relation defines how a monitor executes in tandem with the process it monitors.

Definition 8 (Instrumentation). *The transition relation from Definition 2 extends to monitors with the additional rule that* $\texttt{v} \xrightarrow{\alpha} \texttt{v}$ *for* $v \in \{\texttt{yes}, \texttt{no}, \texttt{end}\}$. *Thus verdicts are irrevocable: once* `yes`, `no` *or* `end` *is reached, the monitor can reach no other verdict.*

Then, the instrumentation of a monitor m *and process* p*, denoted* $m \triangleleft p$*, is a process of which the transition relation follows the rules:*

- *if* $m \xrightarrow{\alpha} m'$ *and* $p \xrightarrow{\alpha} p'$*, then* $m \triangleleft p \xrightarrow{\alpha} m' \triangleleft p'$*, and*
- *if* $p \xrightarrow{\alpha} p'$ *and* $m \not\xrightarrow{\alpha}$*, then* $m \triangleleft p \xrightarrow{\alpha} \texttt{end} \triangleleft p$.

A monitor m *accepts a process* p *if some instrumentation* $\texttt{yes} \triangleleft p'$ *is reachable via the transition relation from* $m \triangleleft p$*. Dually,* m *rejects* p *if some instrumentation* $\texttt{no} \triangleleft p'$ *is reachable from* $m \triangleleft p$*. Note that the monitors we consider are either acceptance or rejection monitors, and hence can not both accept and reject a process.*

Variation of the instrumentation presented here have also been studied to incorporate silent transitions [1]. Studying optimal monitors in that setting is outside the scope of this paper.

Definition 9. *A monitor* m *is a sound monitor for a sentence* ψ *of* μ*HML if whenever* m *accepts a process* p*, denoted* $\mathbf{acc}(p, m)$*, then* $p \in \psi$*, and conversely, if* m *rejects* p*, written* $\mathbf{rej}(p, m)$*, then* $p \notin \psi$.

Definition 10. *A monitor* m *is violation complete for a sentence* ψ *of* μ*HML if* m *rejects all processes* p *such that* $p \notin \psi$*. It is satisfaction complete for* ψ *if it accepts all processes* p *such that* $p \in \psi$.

Francalanza *et al.* [14] determined the fragment of μHML with sound and complete monitors.

Lemma 1 ([14]). *Every acceptance monitor is sound and satisfaction complete for some μHML formula. Dually, every rejection monitor is sound and violation complete for some μHML formula.*

Theorem 1 ([14]). *A formula ϕ has a sound and violation complete rejection monitor if and only if it is equivalent to a formula in the safety fragment of μHML, written $\mathtt{s}\mu$HML and generated by the following grammar:*

$$\phi := \bot \mid \top \mid X \mid \phi \wedge \phi \mid [\alpha]\phi \mid \mu X.\phi$$

Dually, a formula has a sound and satisfaction complete acceptance monitor if and only if it is equivalent to a formula in the dual co-safety fragment of μHML.

In this paper, we will relax the completeness criterion, and only require that monitors be sound. We will therefore differentiate between a monitor m being a sound and complete monitor for a formula ϕ, and m being a sound monitor for ϕ: we will write *sc-monitors* and *s-monitors* respectively, and speak about *sc-monitorability*, as in Theorem 1.

2 Optimality of monitors

In this section we define the concept of optimal monitor for a specification, and argue that finding the optimal monitor reduces to finding the strongest safety consequence of a μHML property.

Lemma 2. *If ψ_a is a consequence of ψ_b, i.e. $\psi_b \implies \psi_a$, a sound rejection monitor m for ψ_a is also a sound rejection monitor for ψ_b.*

Proof. If m rejects a process p, then since m is sound for ψ_a we have $p \not\models \psi_a$. Then $p \not\models \psi_b$, and therefore m is a sound rejection monitor for ψ_b.

Lemma 3. *If m is a sound rejection monitor for ψ_b, then there is a formula ψ_a for which m is sound and violation-complete and ψ_b implies ψ_a.*

Proof. From Lemma 1, there is a formula ψ_a for which m is a sound and violation-complete monitor. It suffices to show that ψ_b implies ψ_a. If $p \models \psi_b$, then, since m is sound for ψ_b, it does not reject p. Since m is complete for ψ_a, we get $p \models \psi_a$.

Dually, a formula has a sound acceptance monitor if and only if it has an antecedent with a sound and complete acceptance monitor, *i.e.* a formula in the co-safety fragment.

Example 1. The formula $\nu X.[\alpha]X \wedge \mu Y.[\beta]Y \wedge [\gamma]\bot$, not itself sc-monitorable, has the sc-monitorable consequence $\nu X.[\alpha]X \wedge \nu Y.[\beta]Y \wedge [\gamma]\bot$.

Monitorable consequences are by no means unique: $[\beta][\beta][\gamma]\bot$ and $[\alpha][\beta][\gamma]\bot$ are two additional examples of the infinitely many monitorable consequences of this formula. Intuitively, they are less interesting: their sc-monitors only reach a verdict on exactly one prefix, $\beta\beta\gamma$ and $\alpha\beta\gamma$ respectively. The sc-monitor of $\nu X.[\alpha]X \wedge \nu Y.[\beta]Y \wedge [\gamma]\bot$ on the other hand produces a rejecting verdict on infinitely many different trace-prefixes.

We now define a partial order, already considered by Francalanza [12], on rejection monitors. The optimal rejection-monitor for a formula is the sound monitor which rejects any process rejected by *some* sound monitor.

Definition 11. *We say that $m \leq m'$ if for all p, if $\mathbf{rej}(m, p)$, then $\mathbf{rej}(m', p)$ and if if $\mathbf{acc}(m, p)$, then $\mathbf{acc}(m', p)$. That is to say, m' rejects (accepts) all the processes that m rejects (accepts).*

A monitor m is optimal for ψ if it is sound for ψ and for all other m' which is sound for ψ, it is the case that $m' \leq m$.

Proposition 1. *If m and m' are sound and complete rejection monitors for ψ and ψ' respectively, then $m \leq m'$ if and only if ψ' implies ψ.*

Proof. If $m \leq m'$, then for any process $p \notin \psi$, by violation-completeness of m with respect to ψ, we have that $\mathbf{rej}(m, p)$. Then, since $m \leq m'$, we also have that $\mathbf{rej}(m', p)$ which in turn, through soundness of m' as a monitor for ψ', implies $p \notin \psi'$.

Conversely, from Lemma 2, if ψ' implies ψ, then m is also sound for ψ'. Therefore, if $\mathbf{rej}(m, p)$, then $p \notin \psi'$ and by completeness of m', we also have $\mathbf{rej}(m', p)$ – that is to say, $m \leq m'$.

Definition 12. *Given a μHML formula ψ, its strongest monitorable consequence is a formula ϕ such that:*

- *ϕ is in the safety fragment of μHML,*
- *ψ implies ϕ, and*
- *For any ϕ' in the safety fragment of μHML, if $\psi \implies \phi'$ then $\phi \implies \phi'$.*

Theorem 2. *A rejection monitor m is optimal for ψ if and only if it is sound and complete for the strongest monitorable consequence of ψ.*

Proof. Let ϕ be the strongest monitorable consequence of ψ, and m its sound and violation-complete monitor. We show that m is optimal for ψ. First, from Lemma 2, m is sound for ψ. Now assume some other monitor m' is also sound for ψ. From Lemma 3, m' is sound and complete for a consequence ϕ' of ψ in the safety fragment. Since ϕ is the strongest monitorable consequence, $\phi \implies \phi'$, and therefore, from Proposition 1, $m' \leq m$.

Conversely, let m be an optimal monitor for ψ, and ϕ a safety consequence of ψ such that m is sound and complete for ϕ. Assume ϕ' is another consequence

of ψ in the safety fragment, and m' is its sound and complete monitor. Since m is optimal, $m' \leq m$ and therefore ϕ implies ϕ'. Then ϕ is indeed the strongest safety consequence of ψ.

Therefore, finding the optimal rejection monitor for a formula amounts to finding its strongest consequence in the safety fragment of μHML. Dually, the optimal acceptance monitor of a formula is the sound and complete monitor of its weakest co-safety antecedent.

3 Synthesis of the strongest safety consequence

This section presents the paper's main technical contribution: We construct, for any μHML formula Ψ, its strongest safety consequence.

We synthesize the strongest safety consequence by applying successive syntactic simplification steps to Ψ. Each step eliminates a syntactic feature that is not part of the syntax of safety formulas, while preserving the set of safety consequence: We first eliminate existential modalities $\langle\alpha\rangle$, then least fixpoints μ, and finally disjunctions $\vee$. The final step then leaves us with a safety formula with the same set of safety consequences as Ψ, *i.e.,* the strongest safety consequence of Ψ. To execute these syntactic simplifications, we take the input formula in disjuntive normal form [21], a syntactic restriction on L_μ intended to mimic non-determinism.

3.1 Preliminaries

Definition 13 (Disjunctive Form). *The set of disjunctive formulas of μHML is the smallest set $\mathcal{F}$ satisfying:*

- $\top, \bot \in \mathcal{F}$;
- *If $\psi \in \mathcal{F}$ and $\phi \in \mathcal{F}$, then $\psi \vee \phi \in \mathcal{F}$;*
- *If for each action α in $A \subseteq Act$ the set $\mathcal{B}_\alpha \subseteq \mathcal{F}$ is a finite set of formulas, then*

$$\bigwedge_{\alpha \in A} ((\bigwedge_{\psi \in \mathcal{B}_\alpha} \langle a \rangle \psi) \wedge [\alpha] \bigvee_{\psi \in \mathcal{B}_\alpha} \psi)$$

  is in $\mathcal{F}$. These subformulas are called special conjunctions.
- *$\mu X.\psi$ and $\nu X.\psi$ are in $\mathcal{F}$ as long as $\psi \in \mathcal{F}$.*

That is to say, conjunctions only occur to express that for each $\alpha \in A$ every successor satisfies some formula in $\mathcal{B}_\alpha$ and every formula of $\mathcal{B}_\alpha$ is satisfied by some successor.

Lemma 4 ([21]). *Every μHML formula is equivalent to a formula in disjunctive normal form.*

We now establish a simple but fundamental property of safety properties: if a finite tree T does not satisfy a safety property ψ, then no tree T' obtained from T by adding successors satisfies ψ.

Definition 14 (Completion of a computation tree). *Let T and T' be computation trees. T' is a completion of T if T and T' share the same root, and T' contains T in the sense that if $s \in T$ then $s \in T'$, and if s' is a successor of s in T, then s' is a successor of s in T'.*

Lemma 5. *Let T be a computation tree and T' its completion. Let θ be a safety μHML formula. If $T \not\models \theta$, then $T' \not\models \theta$.*

Proof. Let $\bar{\theta}$ be the co-safety negation of θ. If $T \not\models \theta$ then there is a consistent annotation $A : T \to \mathcal{P}(sf(\bar{\theta}))$ to witness $\bar{\theta}$ in T. Since $\bar{\theta}$ contains no universal modalities, A' which copies A onto T' and assigns the empty set to all states of $T' \setminus T$ is a consistent annotation proving that $T' \in \bar{\theta}$, and therefore $T' \not\models \theta$.

3.2 Eliminating existential modalities

Lemma 6. *Every sentence Ψ of μHML has a consequence Ψ' such that:*

- *Ψ' has no existential modalities;*
- *If θ is a safety formula and Ψ implies θ, then Ψ' also implies θ.*

Proof. Without loss of generality, assume Ψ to be in disjunctive normal form in which all subformulas except $\bot$ are satisfiable, and disjunctions with $\bot$ are simplified. Note that for disjunctive μHML, satisfiability can be decided in linear time. Obtain Ψ' from Ψ by replacing every existential modality $\langle\alpha\rangle\psi$ with $\top$.

Observe that Ψ implies Ψ'. Let θ be a safety μHML formula such that Ψ implies θ. We now show that Ψ' also implies θ.

Assume otherwise: let T be a computation tree such that $T \models \Psi' \wedge \neg\theta$. Let $A : T \to \mathcal{P}(sf(\Psi'))$ be an annotation that witnesses $T \models \Psi'$. Since the only conjunction in Ψ' are over over universal modalities via different actions, A annotates every state s of T with at most one (not necessarily binary) conjunction.

Let T' be the computation tree T where every state with annotation $A(s)$ containing $\bigwedge_{\alpha\in A}[\alpha]\bigvee\mathcal{B}_\alpha$ has, for each $\alpha \in A$ and $\phi \in \mathcal{B}_\alpha$, an additional disjoint α-successors s_ϕ such that $s_\phi \models \phi$. Note that such a successor exists since all subformulas of Ψ except $\bot$ are required to be satisfiable. Let $A' : T' \to \Psi$ be an annotation similar to A except that:

- If for a state s, its annotation $A(s)$ contains $[\alpha]\bigvee\mathcal{B}_\alpha$ then $A'(s)$ also contains $\bigwedge_{\psi\in\mathcal{B}_\alpha}\langle\alpha\rangle\psi \wedge [\alpha]\bigvee\mathcal{B}_\alpha$.
- If $A'(s)$ contains $\bigwedge_{\psi\in\mathcal{B}_\alpha}\langle\alpha\rangle\psi \wedge [\alpha]\bigvee\mathcal{B}_\alpha$, it also contains $\bigwedge_{\psi\in\mathcal{B}_\alpha}\langle\alpha\rangle\psi$ and $\langle\alpha\rangle\psi$ for each $\psi \in \mathcal{B}_\alpha$.
- A' restricted to a new state s_ϕ is a consistent ϕ-annotation of s.

A' is a consistent Ψ-annotation which witnesses that $T' \models \Psi$. However, since θ is a safety condition, $T \models \neg\theta$ implies that θ does not hold in any completion of T, from Lemma 5: in particular $T' \models \neg\theta$. This contradicts that Ψ implies θ. Therefore Ψ' implies θ, as required.

Example 2. The necessity of disjunctive form can be seen from the following example: $(\langle\alpha\rangle[\beta]\bot) \wedge ([\alpha]\langle\beta\rangle\top \vee [\alpha][\gamma]\bot)$.

Here replacing existential modalities with $\top$ would yield a formula itself equivalent to $\top$. However, in disjunctive form we can extract its strongest safety consequence $[\alpha][\gamma]\bot$ (rather than $\top$). For $F = \{[\beta]\bot \wedge [\gamma]\bot, [\gamma]\bot\}$, the disjunctive formula is:

$$\bigwedge_{\phi \in F} \langle\alpha\rangle\phi \wedge [\alpha] \bigvee F.$$

3.3 Eliminating least fixpoints

Let Ψ be a sentence of μHML without existential modalities. In this section we replace all minimal fixpoint operators with maximal ones. As in the previous step, this transformation does not preserve semantics, but we show that it preserves safety consequences.

Lemma 7. *Every sentence Ψ of μHML without existential modalities has a consequence Ψ' such that:*

- *Ψ' has no least-fixpoint operators;*
- *If θ is a safety formula and Ψ implies θ, then Ψ' also implies θ.*

Proof. Obtain Ψ' by replacing every μ-operator with a ν operator. Observe that Ψ implies Ψ'. Let θ be a safety formula such that Ψ implies θ. Since θ is a safety formula, it only contains ν-fixpoint operators. We will show that $\neg\theta$ implies $\neg\Psi'$.

We first show that for all M such that $M \not\models \theta$, there is an n such that θ does not holds in the prefix $M \restriction^n$ of depth n of M. To this end, consider a minimal consistent annotation A that witnesses $M \models \bar{\theta}$ for $\bar{\theta}$ the co-safety negation of θ. Since $\bar{\theta}$ does not contain ν, all traces of A are finite. Then, since it is minimal, A assigns a non-empty annotation only to a finite prefix of M. Let n be the length of the longest trace of A. Then A is also a consistent annotation on $M \restriction^n$, therefore $M \restriction^n \models \bar{\theta}$.

Hence, if $M \not\models \theta$, then for some n, we have $M \restriction^n \not\models \theta$ and therefore also $M \restriction^n \models \bar{\Psi}$, where $\bar{\Psi}$ is the syntactic negation of Ψ. Since an annotation of a guarded formula on a finite tree has no infinite traces, Ψ and Ψ' agree on finite trees and therefore $M \restriction^n \not\models \Psi'$. Since $\bar{\Psi}$ is the syntactic negation of Ψ, it has no $[a]$ modalities, and therefore a witness annotation for $\bar{\Psi}'$ in $M \restriction^n$ is also a witness annotation for $\bar{\Psi}'$ in any completion of $M \restriction^n$, in particular in M. Then $M \not\models \Psi'$.

3.4 Eliminating disjunctions

The final step turns a sentence without existential modalities or least fixpoints into its strongest safety consequence. This step is similar to the transformation into disjunctive form, although simpler: We will use a set of tableau rules to distribute disjunctions while simplifying modalities. As a result, this transformation will not preserve semantics, only safety consequences. The intuition is to

turn $[\alpha]\phi \vee [\alpha]\phi'$ into $[\alpha](\phi \vee \phi')$: we can only monitor a violation if both ϕ and ϕ' are violated on the same branch.

Definition 15 (Tableau). *Given a sentence Ψ without μ operators nor existential modalities $\langle\alpha\rangle$, we build a tableau $\mathcal{T}(\Psi)$ consisting of a tree with back edges, where each node n is labelled with a set $L_\Psi(n)$ of subformulas of Ψ, such that:*

- *The root is labelled $\{\Psi\}$,*
- *For each node n and its children, there is a tableau rule such that n is labelled with the conclusion and its children are labelled with its premises,*
- *This tableau rule is the rule $[\alpha]$ only if $L_\Psi(n)$ matches the conclusion of no other tableau-rule.*

$$\frac{\{\Gamma, \phi, \psi\}}{\{\Gamma, \psi \vee \phi\}}\,(\vee) \qquad \frac{\{\Gamma, \phi\} \qquad \{\Gamma, \psi\}}{\{\Gamma, \psi \wedge \phi\}}\,(\wedge) \qquad \frac{\{\Gamma, \phi\}}{\{\Gamma, \nu X.\phi\}}\,(\nu)$$

$$\frac{\{\Gamma, \phi_X\}}{\{\Gamma, X\}}\,(X) \text{ where } \phi_X \text{ is the binding formula for } X$$

$$\frac{\{\top\}}{\{\Gamma, \top\}}\,(\top) \qquad \frac{\top}{\{\Gamma, [\alpha]\psi, [\beta]\phi\}}\,([\alpha, \beta]) \text{ where } \alpha \neq \beta$$

$$\frac{\{\psi | [\alpha]\psi \in \Gamma\}}{\{\Gamma\}}\,([\alpha])$$

The disjunction-free sentence Ψ' is then retrieved from $\mathcal{T}(\Psi_\mu)$ by defining the following Φ'-labelling $L_{\Phi'}$:

- *A leaf with a back-edge to an inner node n is labelled X_n;*
- *A leaf without back-edge is labelled with the disjunction of its contents: either $\bot$ or $\top$.*

For inner nodes that are not the target of a back-edge:

- *A node with child m via the rules $\vee, \top, [\alpha, \beta], X, \nu$ has the same label as m;*
- *A node with children m, m' with labels f and f', via rule $\wedge$ has label $f \wedge f'$;*
- *A node with one successor m with label f via $[a]$ has label $[a]f$.*

If n is the target of back-edges, then its label is $\nu X_n.f$ where f is the label defined as above.

Let Ψ' be the L'_Ψ-label of the root of $\mathcal{T}(\Psi)$.

The key to the following proofs is to observe that a Ψ' or Ψ-counter annotation of a computation tree corresponds to a single path in $\mathcal{T}(\Psi)$. We can then use the two labellings of $\mathcal{T}(\Psi)$ to move between counter-annotations of Ψ' and Ψ.

Example 3. Figures 1 and 2, show both labellings of the tableau for $\nu X.[\alpha]([\alpha]X \wedge [\beta]\bot) \vee [\alpha]([\alpha]\bot \wedge [\beta]X)$. Its strongest safety consequence is semantically trivial. The formula $\nu X.[\alpha]([\alpha]X \wedge [\beta]\bot \wedge [\gamma]\bot) \vee [\alpha]([\alpha]X \wedge [\gamma]\bot \wedge [\delta]\bot)$ has an unwieldy tableau (not presented here) but its strongest safety consequence is not semantically trivial: $\nu X.[\alpha]([\alpha]X \wedge [\gamma]\bot)$.

$$\dfrac{\dfrac{\dfrac{\dfrac{X}{[\alpha]X,[\alpha]\bot}([\alpha]) \quad \dfrac{\top}{[\alpha]X,[\beta]X}([\alpha,\beta])}{[\alpha]X,[\alpha]\bot\wedge[\beta]X}(\wedge) \quad \dfrac{\dfrac{\top}{[\beta]\bot,[\alpha]\bot}([\alpha/\beta]) \quad \dfrac{X}{[\beta]\bot,[\beta]X}([\beta])}{[\beta]\bot,[\alpha]\bot\wedge[\beta]X}(\wedge)}{[\alpha]X\wedge[\beta]\bot,[\alpha]\bot\wedge[\beta]X}(\wedge)}{[\alpha]([\alpha]X\wedge[\beta]\bot),[\alpha]([\alpha]\bot\wedge[\beta]X)}([\alpha])}{[\alpha]([\alpha]X\wedge[\beta]\bot)\vee[\alpha]([\alpha]\bot\wedge[\beta]X)}(\vee)}{\nu X.[\alpha]([\alpha]X\wedge[\beta]\bot)\vee[\alpha]([\alpha]\bot\wedge[\beta]X)}(\nu)$$

Fig. 1. The tableau of a formula, labelled with subsets of subformulas.

$$\dfrac{\dfrac{\dfrac{\dfrac{X}{[\alpha]X}([\alpha]) \quad \dfrac{\top}{\top}([\alpha/\beta])}{[\alpha]X\wedge\top}(\wedge) \quad \dfrac{\dfrac{\top}{\top}([\alpha/\beta]) \quad \dfrac{X}{[\beta]X}([\beta])}{\top\wedge[\beta]X}(\wedge)}{[\alpha]X\wedge\top\wedge[\beta]X}(\wedge)}{[\alpha]([\alpha]X\wedge\top\wedge[\beta]X)}([\alpha])}{[\alpha]([\alpha]X\wedge\top\wedge[\beta]X)}(\vee)}{\nu X.[\alpha]([\alpha]X\wedge\top\wedge[\beta]X)}(\nu)$$

Fig. 2. Same tableau, labelled with the subformulas of the strongest safety consequence

Lemma 8. *Every μHML sentence Ψ without μ-operators nor existential modalities has a consequence Ψ' such that:*

- *Ψ' is a safety formula;*
- *If θ is a safety formula and Ψ implies θ, then Ψ' also implies θ.*

Proof. Let Ψ' be the $L_{\Psi'}$-label of the root of the tableau $\mathcal{T}(\Psi)$ as described in Definition 15. By construction Ψ' is a sentence in the safety fragment of μHML. We show that that Ψ' is a consequence of Ψ, *i.e.,* $\neg\Psi'$ implies $\neg\Psi$.

Let T be a computation tree such that $T \not\models \Psi'$. First we note that since Ψ' is a safety formula, there is a Ψ'-counter-annotation $C : T \to \mathcal{P}(sf(\Psi'))$ which consists of an annotation of a branch b of T with a single finite trace τ ending in $\bot$. This is because Ψ' contains no disjunctions or existential modalities which could cause there to be more than one trace; since the all fixpoints are ν-bound, the unique trace must be finite and end in $\bot$.

There is a finite path π in $\mathcal{T}(\Psi)$ of which the $L_{\Psi'}$-labelling, ignoring successive repetitions of the same label, is the trace τ: more precisely, the subformulas appearing in τ after exactly m modalities are exactly those in the union of labels of nodes n in π occurring after exactly m applications of the modal tableau rule.

Let $C' : T \to \mathcal{P}(sf(\Psi))$ be the counter-annotation which assigns the L_Ψ-labels of π to b as follows: $C'(s)$ consists of the union of the L_Ψ labels of the nodes of π occurring after exactly m applications of the modal tableau rule. In

other words, we replace the Ψ'-counter-annotation with a Ψ-counter-annotation by using the tableau $\mathcal{T}(\Psi)$ as a mapping between subformulas of Ψ' and sets of subformulas of Ψ.

We now argue that the counter-annotation C' is then a consistent counter-annotation of Ψ on T. First of all, it does not contain $\top$ since a node $n \in \pi$ has $\top$ in its Ψ-label exactly when its Ψ'-label is $\top$. As a result, we know that π does not see any occurrence of the $(\top)$ or $([\alpha, \beta])$ rules. The remaining tableau rules guarantee C' to be locally consistent. Finally, like C, the counter-annotation C' only has finite traces, so it is also globally consistent. C' then witnesses that $T \not\models \Psi$. We conclude that Ψ implies Ψ'.

We now show that if Ψ implies a safety formula θ, then Ψ' also implies θ.

Assume θ to be a safety formula such that Ψ implies θ. Let T be a computation tree such that $T \not\models \theta$. Since θ is a safety formula, there is a branch B of T such that if B is treated as a computation tree then $B \not\models \theta$ and therefore $B \not\models \Psi$. Let C be a consistent counter-annotation of Ψ on B. There is a path π in the tableau $\mathcal{T}(\Psi)$ such that $C(s)$, the annotation of a state s at modal depth m consists exactly of the union of Ψ-labels in $\mathcal{T}(\Psi)$ occurring on π after exactly m modal rules. To find this path π, it suffices to choose at conjunctions the child corresponding to the conjunct chosen by the counter annotation C.

Let C' be the Ψ'-counter-annotation that labels a node s of B at modal depth m to the union of the Ψ'-annotations which occur on π exactly after m modal rules. Note that since C is a counter-annotation, is does not contain $\top$ and therefore C' does not assign $\top$ to any node either. By construction C' is locally consistent; since it doesn't have any infinite traces, and does not contain $\top$, it is also globally consistent. Therefore $B \not\models \Psi'$. Since Ψ' is a safety formula, no completion of B satisfies Ψ'. In particular, $T \not\models \Psi'$. Hence $\neg\theta$ implies $\neg\Psi'$, as required.

Theorem 3. *Every formula of μHML has a computable strongest safety consequence.*

Proof. Let Ψ be a μHML formula. Using Lemma 6, we can compute a consequence Ψ_1 of Ψ without existential modalities which has the same safety consequences as Ψ. Then, applying Lemma 7 to Ψ_1, we obtain the consequence Ψ_2 of Ψ without μ-operators but with the same set of safety consequences. Finally, using Lemma 8 we construct a safety consequence Ψ_3 of Ψ, again with the same set of safety consequences. Ψ_3 is then the strongest safety consequence of Ψ.

The transformation into disjunctive form can incur one exponential blow-up, and the elimination of disjunctions can incur another: the synthesis of optimal monitors is in 2-Exptime. In the next section, we argue that this double exponential complexity pays for obtaining a deterministic optimal monitor. Whether a more compact non-deterministic monitor could be synthesised instead remains open.

4 On determinism

Monitors as defined in this set-up can be non-deterministic: a monitor $(\alpha.\alpha.\mathtt{no})+(\alpha.\beta.\mathtt{no})$, for example, when instrumenting a process $\alpha.p$, has to make a non-deterministic choice of whether to transition to $\alpha.\mathtt{no}$ or to $\beta.\mathtt{no}$. Implementing this non-determinism, for example by forking parallel monitors, may then come at exponential cost.

One can instead restrict oneself to non-deterministic monitors, and use the monitor $\alpha.(\alpha.\mathtt{no}+\beta.\mathtt{no})$. Although the monitor still has the sum $+$, its transition will be determined by whether the instrumented process transitions a second time with α or β. For such deterministic monitors, $m \triangleleft p$ has a single outcome which determines the verdict of the monitor.

We now check that the monitors generated by the strongest safety consequences synthesised here are in fact already deterministic. This justifies the double-exponential complexity of our synthesis: going from arbitrary (alternating) μHML formulas to disjunctive ones, and then from disjunctive ones to deterministic ones can each incur an exponential blow-up.

We use the definition of determinism presented by Aceto *et al.* [2]. This syntactic definition is equivalent to a semantic one, which postulates that for any finite sequence of actions $\pi \in Act^*$ there is a unique monitor m' such that $m \xrightarrow{\pi} m'$.

Definition 16 (Determinism). *A monitor m is deterministic if and only if every sum m is of the form $\Sigma_{\alpha\in A}\alpha.m_\alpha$ for some $A \subseteq Act$.*

Definition 17. *We now recall the monitor synthesis function from Francalanza et al. [14], restricted to rejection monitors synthesised from safety formulas:*

$$
\begin{array}{rlrl}
m(\bot) &= \mathtt{no} & m(X) &= x \\
m(\top) &= \mathtt{end} & m(\psi \wedge \psi') &= \mathit{no}\ \mathit{if}\ m(\psi) = \mathtt{no} \\
m([\alpha]\psi) &= \alpha.m(\psi)\ \mathit{if}\ m(\psi) \neq \mathtt{end} & m(\psi \wedge \psi') &= \mathit{no}\ \mathit{if}\ m(\psi') = \mathtt{no} \\
m([\alpha]\psi) &= \mathtt{end}\ \mathit{otherwise.} & m(\psi \wedge \psi') &= m(\psi)\ \mathit{if}\ m(\psi') = \mathtt{end} \\
m(\nu x.\psi) &= \mathtt{rec}x.m(\psi)\ \mathit{if}\ m(\psi) \neq \mathtt{end} & m(\psi \wedge \psi') &= m(\psi')\ \mathit{if}\ m(\psi) = \mathtt{end} \\
m(\nu x.\psi) &= \mathtt{end}\ \mathit{otherwise.} & m(\psi \wedge \psi') &= m(\psi) + m(\psi')\ \mathit{otherwise.}
\end{array}
$$

Lemma 9. *The strongest safety consequences synthesised in Theorem 3 generate deterministic monitors.*

Proof. This monitor synthesis only generates non-determinism from formulas which contain conjunctions $\psi \wedge \psi'$ where the next modal subformulas reachable from ψ and ψ' are over different actions. This restriction to conjunctions over universal modalities with different actions – call these deterministic conjunctions – is implied by the restriction imposed by disjunctive normal form in the form of special conjunctions. The synthesis of the strongest safety consequence in Theorem 3 begins by taking a formula in disjunctive normal form,

and then eliminating its existential modalities; The resulting formula still only has deterministic conjunctions, which additionally no longer contain existential modalities. The elimination of least fixpoints does not affect conjunctions. We now check that the tableau construction which removes disjunctions preserves deterministic conjunctions.

This is indeed the case, as we can see from observing the $(\wedge)$ tableau rule, which is responsible for all the conjunctions in the final formula. If the input formula only has deterministic conjunctions, then when the rule $(\wedge)$ is applied, the children will each contain a formula with a different outermost modality. Since the outermost modality in the label of a tableau node always matches the outermost modalities in its contents, the child labels will have different modalities. Hence the conjunction in the label of the parent node will be a deterministic conjunction. The same argument holds for non-binary conjunctions.

The strongest safety consequence synthesised by Theorem 3 only has deterministic conjunctions. The synthesis from 17 generates deterministic monitors.

In light of the exponential[2] lower bound presented by Aceto *et al.* [2] on the determinization of monitors, the existence of a more concise deterministic optimal monitor seems unlikely. Our second exponential is necessary to eliminate alternations since we start outside of the safety and co-safety fragments. Note that the theoretical worst-case blow-up is unlikely to seriously affect monitoring efforts: in many, and seemingly most cases, the strongest safety consequence is significantly smaller than the original formula.

5 Conclusion

We have proposed a new, weaker notion of monitorability, based on relaxing the expectation that monitors must be complete. Our core insight is that the theory of runtime monitors can then be extended to the partial verification of μHML formulas that were previously dismissed as unmonitorable.

Although we argue that meaningful (if not complete) monitorability extends beyond the safety and co-safety fragments of μHML, some formulas remain thoroughly unmonitorable. These are the formulas of which both the optimal rejection and acceptance monitors are the universally indecisive monitor. We leave as future work the study of this class of thoroughly unmonitorable formulas, but conjecture that they are recognisable in ExpTime.

A further avenue for future work is to consider optimal monitors in the linear-time setting. There, the classical notions of monitorability [20, 5] are not operational in the sense that they are not defined in terms of monitors. Therefore, while the notion of strongest monitorable consequence translates immediately into the linear-time setting for any definition of monitorability, our notion of optimal monitor does not. We leave as future work the comparison in the linear setting of our notion of true unmonitorability with the Francalanza *et al.,* notion of monitorability, and the classical definitions of monitorability.

[2] Note that Aceto *et al.* measure size in the length of a formula, while here we count distinct subformulas: Their double-exponential is for us a single exponential.

References

1. Luca Aceto, Antonis Achilleos, Adrian Francalanza, and Anna Ingólfsdóttir. Monitoring for silent actions. In *37th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS*, 2017.
2. Luca Aceto, Antonis Achilleos, Adrian Francalanza, Anna Ingólfsdóttir, and Sævar Örn Kjartansson. On the complexity of determinizing monitors. In *International Conference on Implementation and Application of Automata*, pages 1–13. Springer, 2017.
3. Howard Barringer, Allen Goldberg, Klaus Havelund, and Koushik Sen. Rule-based runtime verification. In *VMCAI*, volume 2937, pages 44–57. Springer, 2004.
4. Andreas Bauer, Martin Leucker, and Christian Schallhart. Comparing ltl semantics for runtime verification. *Journal of Logic and Computation*, 20(3):651–674, 2010.
5. Andreas Bauer, Martin Leucker, and Christian Schallhart. Runtime verification for ltl and tltl. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 20(4):14, 2011.
6. J Bradfield and C Stirling. Modal logics and mu-calculi: An introduction. chapter 4 of handbook of process algebra. ja bergstra, a. ponse and sa smolka, editors, 2001.
7. Julian Bradfield and Colin Stirling. 12 modal mu-calculi. *Studies in logic and practical reasoning*, 3:721–756, 2007.
8. Guillaume Brat, Doron Drusinsky, Dimitra Giannakopoulou, Allen Goldberg, Klaus Havelund, Mike Lowry, Corina Pasareanu, Arnaud Venet, Willem Visser, and Rich Washington. Experimental evaluation of verification and validation tools on martian rover software. *Formal Methods in System Design*, 25(2):167–198, 2004.
9. Ben d'Angelo, Sriram Sankaranarayanan, César Sánchez, Will Robinson, Bernd Finkbeiner, Henny B Sipma, Sandeep Mehrotra, and Zohar Manna. Lola: Runtime monitoring of synchronous systems. In *Temporal Representation and Reasoning, 2005. TIME 2005. 12th International Symposium on*, pages 166–174. IEEE, 2005.
10. Doron Drusinsky. Monitoring temporal rules combined with time series. In *CAV*, volume 3, pages 114–118. Springer, 2003.
11. Yliès Falcone, Jean-Claude Fernandez, and Laurent Mounier. What can you verify and enforce at runtime? *International Journal on Software Tools for Technology Transfer*, 14(3):349–382, Jun 2012.
12. Adrian Francalanza. A theory of monitors. In *International Conference on Foundations of Software Science and Computation Structures*, pages 145–161. Springer, 2016.
13. Adrian Francalanza, Luca Aceto, Antonis Achilleos, Duncan Paul Attard, Ian Cassar, Dario Della Monica, and Anna Ingólfsdóttir. A foundation for runtime monitoring. In *International Conference on Runtime Verification*, pages 8–29. Springer, 2017.
14. Adrian Francalanza, Luca Aceto, and Anna Ingolfsdottir. Monitorability for the hennessy–milner logic with recursion. *Formal Methods in System Design*, pages 1–30.
15. Dimitra Giannakopoulou and Klaus Havelund. Runtime analysis of linear temporal logic specifications. In *Proceedings of the 16th IEEE International Conference on Automated Software Engineering, San Diego, California*, 2001.
16. Klaus Havelund and Grigore Rosu. Synthesizing monitors for safety properties. In *TACAS*, volume 2, pages 342–356. Springer, 2002.

17. Kim G Larsen. Proof systems for satisfiability in hennessy-milner logic with recursion. *Theoretical Computer Science*, 72(2-3):265–288, 1990.
18. Martin Leucker and Christian Schallhart. A brief account of runtime verification. *The Journal of Logic and Algebraic Programming*, 78(5):293–303, 2009.
19. Amir Pnueli. The temporal logic of programs. In *Foundations of Computer Science, 1977., 18th Annual Symposium on*, pages 46–57. IEEE, 1977.
20. Amir Pnueli and Aleksandr Zaks. Psl model checking and run-time verification via testers. In *International Symposium on Formal Methods*, pages 573–586. Springer, 2006.
21. Igor Walukiewicz. Completeness of Kozen's Axiomatisation of the Propositional Mu-Calculus. In *Proceedings of the 10th Annual IEEE Symposium on Logic in Computer Science*, LICS '95, pages 14–, Washington, DC, USA, 1995. IEEE Computer Society.