The Best a Monitor Can Do

Luca Aceto, Antonis Achilleos, Adrian Francalanza, Anna Ingólfsdóttir, Karoliina Lehtinen

The Best a Monitor Can Do

Luca Aceto
Reykjavik University, Iceland
Gran Sasso Science Institute, L'Aquila, Italy
luca@ru.is

Antonis Achilleos
Reykjavik University, Iceland
antonios@ru.is

Adrian Francalanza
University of Malta, Malta
afra1@um.edu.mt

Anna Ingólfsdóttir
Reykjavik University, Iceland
annai@ru.is

Karoliina Lehtinen
University of Liverpool, UK
k.lehtinen@liverpool.ac.uk

Abstract

Existing notions of monitorability for branching-time properties are fairly restrictive. This, in turn, impacts the ability to incorporate prior knowledge about the system under scrutiny – which corresponds to a branching-time property – into the runtime analysis. We propose a definition of optimal monitors that verify the best monitorable under- or over-approximation of a specification, regardless of its monitorability status. Optimal monitors can be obtained for arbitrary branching-time properties by synthesising a sound and complete monitor for their *strongest monitorable consequence*. We show that the strongest monitorable consequence of specifications expressed in Hennessy-Milner logic with recursion is itself expressible in this logic, and present a procedure to find it. Our procedure enables prior knowledge to be optimally incorporated into runtime monitors.

2012 ACM Subject Classification Theory of computation → Modal and temporal logics; Software and its engineering → Formal software verification

Keywords and phrases monitorability, branching-time logics, runtime verification

Digital Object Identifier 10.4230/LIPIcs.CSL.2021.7

Funding Research supported by the Icelandic Research Fund projects "Theoretical Foundations for Monitorability" (No:163406-051) and "Epistemic Logic for Distributed Runtime Monitoring" (No:184940-051), the MIUR project PRIN 2017FTXR7S IT MATTERS, project BehAPI, funded by the EU H2020 RISE programme under the Marie Skłodowska-Curie grant agreement No:778233, and project FouCo, funded by EU H2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No:892704.

1 Introduction

Branching-time properties, as described by logics such as CTL, CTL* and the modal μ-calculus, are normally verified using well-established pre-deployment techniques like model checking [18, 10]. However, there are cases where the system model is either unavailable (*e.g.,* due to third-party intellectual property restrictions), or not fully understood (*e.g.,* when parts of the system logic is governed by machine-learning tools). In these cases, monitors

can be used effectively to observe the execution of a system (rather than its state space) for verification purposes, as demonstrated in [26, 1, 4]; this technique is broadly referred to as runtime verification (RV) [25, 11].

RV is a best-effort strategy since it is limited to the incremental analysis of a single execution. The study of monitorability [4, 5] asks what correctness guarantees RV can provide and what properties can be monitored adequately with these guarantees. A wide body of work [46, 29, 50, 21, 26, 1, 28, 2, 4] primarily considers *safety* properties [7] ("something bad never happens") as those worth monitoring for, as they correspond to properties for which violations can always be identified from some finite prefix of an execution. However, limiting monitoring to this class of properties severely restricts the utility of RV. The restriction is particularly acute for branching-time properties [26, 1], which explains, in part, why RV tools generally restrict themselves to linear-time properties. But there are cases where formal specifications (a scarce resource in verification) have already been expressed in a branching-time logic and perhaps formally verified for some subcomponents. In such cases, a systematic method to incorporate such prior knowledge about the system into the runtime analysis would be beneficial.

A number of alternatives can be used to mitigate the shortcomings of RV for branching-time properties. One method would be to increase the observational powers of the monitor or employ multiple runs of the same system [2]. Alternatively, one can weaken the *monitor guarantees* expected during RV. The latter approach is the one explored in this paper. We propose the use of *optimal* monitors, which flag *all* violations that can be determined from execution prefixes contradicting the property (and ignore the violations that cannot). Although such monitors may fail to identify all violations, they represent the *best* monitors can do, and do not impose any restrictions on the considered class of properties. We show how these optimal monitors can be obtained systematically by computing the *strongest monitorable consequence* of the property to be dynamically verified.

▶ **Example 1.** A system with two (enumerated) components produces the events *open*, o_i, *write*, w_i, and *close*, c_i, for $i \in \{1, 2\}$. A specification for the first component states that:

"In all executions, w_1 (write) occurs, but only after an open, o_1." (Spec 1)

According to the existing notion of branching-time monitorability [26], a specification is monitorable if there is monitor that correctly identifies all violating processes from some prefix they produce. In other words, all violating processes must produce an execution prefix such that any process that produces this prefix also violates the property. This is one way of generalising the notion of safety property to the branching-time setting. (Spec 1) is *not* monitorable because there is *no* monitor which correctly identifies *all* violations of this specification. In particular, a first component that *never* reaches w_1 violates this property, but this cannot be determined from any finite prefix of its executions. However, there are other violations of (Spec 1) than can be detected. For instance, monitors can detect executions where w_1 occurs before o_1. Consider the (weaker) specification

"In all executions, w_1 (write) does not occur before an open, o_1." (Spec 2)

Since there is a monitor that detects *all* violations for (Spec 2), it is monitorable. More importantly, this monitor also turns out to be optimal with respect to (Spec 1) since these violations are the *only* ones that can be detected in (Spec 1). Conveniently, [26] also describes a procedure to synthesise the complete monitor from the logical formula describing (Spec 2) which could, in turn, also be used as the optimal monitor for (Spec 1). ⌟

▶ **Example 2.** The previous example illustrates the difficulty of monitoring for unbounded or infinite behaviour ("In all executions, w_1 occurs ...") which applies equally to linear and branching-time properties. Branching-time properties present additional challenges relating to the branching structure of computation. Consider the following specification:

"After o_2 (opening the second component), (closing it) c_2 is reachable but always via w_2 (by writing to the second component beforehand)." (Spec 3)

This is intrinsically a branching time property as it concerns the state space of the system. In particular, no single execution can provide information about whether c_2 is reachable from all states entered via o_2. This property is therefore classified as unmonitorable. It turns out that its strongest monitorable consequence is the following property:

"After o_2, c_2 is only reachable via w_2." (Spec 4)

A sound and complete monitor for this specification flags a violation when it witnesses an execution in which o_2 is followed by c_2 without first seeing w_2. Such a monitor is also the optimal monitor for (Spec 3). Computing the strongest monitorable consequence of a property allows us to extract the part of the property amenable to runtime analysis. ⌟

Our proposed methodology allows us to address another common weakness found in existing RV approaches. Specifically, these approaches often treat the system under scrutiny as a *black box*, without leveraging any prior *partial* knowledge about the system.

▶ **Example 3.** Recall the system in Ex. 1 and consider the property:

"In all executions, c_1 (close) is never immediately followed by a write, w_1." (Spec 5)

(Spec 5) is monitorable according to [26]: a monitor can flag a violation whenever it observes $\mathsf{c}_1\mathsf{w}_1$ during an execution. On the other hand, if the monitor observes the sequence of events $\mathsf{c}_1\mathsf{w}_2\mathsf{w}_1$ in an execution, it cannot determine whether the system violates (Spec 5) or not.

But, suppose we have prior knowledge that the executions of the first and second components are completely independent. Then events such as w_2 and w_1 – coming from independent concurrent components – can be interleaved arbitrarily. A monitor that observes $\mathsf{c}_1\mathsf{w}_2\mathsf{w}_1$ can then infer that the system can also produce the sequence of events $\mathsf{c}_1\mathsf{w}_1\mathsf{w}_2$, meaning that observing $\mathsf{c}_1\mathsf{w}_2\mathsf{w}_1$, or more generally $\mathsf{c}_1\mathsf{w}_2^*\mathsf{w}_1$, provides enough evidence to flag that the system violates (Spec 5). In other words, the prior knowledge allows the monitor to infer violations from executions which by themselves wouldn't suffice to reach a verdict. ⌟

When synthesising monitors for a property P, such as (Spec 5), we would like to systematically incorporate any prior knowledge on the system, such as the independence of components or state-reachability, that can be expressed as a branching-time property K. To do this, we build a monitor based on the conjunction $K \wedge P$ rather than just P. Then, if an execution of a system known to satisfy K is inconsistent with $K \wedge P$, we can deduce that the system violates P. However, as $K \wedge P$ can be an arbitrary branching-time property, it might not itself be monitorable, even if P is monitorable, and the known monitor synthesis procedures might not apply. Again, we can adopt the procedure discussed earlier to obtain an *optimal* monitor for $K \wedge P$ instead. Note that while P might be designed to be a monitorable property, or even a linear-time property, K typically cannot be restricted in this way. In particular, properties such as those in Ex. 3 describing the possible interleavings of concurrent components, and those in Ex. 2 describing the system state-space, are inherently unmonitorable branching-time properties. Yet, so far, approaches to incorporate prior knowledge into runtime monitoring, referred to as grey-box monitoring or monitoring with assumptions, has restricted itself to knowledge representable as a linear-time property [30, 49, 17, 39].

Our contribution is twofold. First, we propose a general procedure to obtain optimal monitors for arbitrary branching-time properties (Sec. 3): following the intuition of Ex. 1 and 2, we find the *strongest monitorable consequence*, *e.g.,* (Spec 2), of an arbitrary branching-time property, *e.g.,* (Spec 1), which allows us to use existing synthesis procedures (*e.g.,* those in [9, 8]) to produce the sound and complete monitor from this monitorable consequence. We show that the resulting monitor is *optimal* for the original specification. This approach allows arbitrary branching-time specifications, for instance those originally designed for model checking, or those combining a monitorable property with prior knowledge, to be verified at runtime. We show that this is indeed the best a monitor can do with prior knowledge. Note that although we use an existing definition of branching-time monitorability to define the strongest monitorable consequence, our optimality result proves that using a different definition cannot improve the procedure. Our result can be seen as the generalisation of the notion of *bad prefixes* [35], i.e. prefixes that monitors can use to reach a negative verdict, to the branching-time setting. Although the set of bad prefixes appears frequently in various works in RV, its generalisation to the branching-time setting and the proposed disciplined methodology for obtaining optimal monitors from non-monitorable properties via the strongest monitorable consequence is, to the best of our knowledge, new.

Our second contribution is technical: we show in Sec. 5 how to compute the strongest monitorable consequence of an arbitrary property expressed in the Hennessy–Milner logic with Recursion, a variant of the modal μ-calculus. This is a popular verification logic that captures all regular tree languages, embeds other popular modal and temporal logics, such as LTL, CTL and CTL*, and corresponds to the bisimulation invariant fragment of monadic second order logic [31]. The size of the strongest monitorable consequence that we compute is bounded by a double exponential in the size of the original formula. This matches the bound on the size of a deterministic automaton that recognises the bad prefixes of an LTL formula [35]. In contrast, the transformation from an LTL formula to its strongest monitorable consequence, also expressed in LTL, is non-elementary (see Sec. 5.5).

We discuss related work, and in particular how this work compares to monitoring in the linear-time setting, in Sec. 6. Omitted proofs can be found in the appendix.

2 Preliminaries

Actions, Processes, Properties and Traces. Fix a finite set $\textsc{Act}$ of *actions* where $a, b \in \textsc{Act}$, a set of *process states* $p, q, r, \ldots \in \textsc{Prc}$ (sometimes called processes), and a transition relation, $\longrightarrow \subseteq (\textsc{Prc} \times \textsc{Act} \times \textsc{Prc})$. The triple $\langle \textsc{Prc}, \textsc{Act}, \longrightarrow \rangle$ forms a Labelled Transition System (LTS) [33] where the suggestive notation $p \xrightarrow{a} q$ denotes $(p, a, q) \in \longrightarrow$. For simplicity, we assume that all the processes that we refer to in this paper can be found in the same fixed infinite LTS, such as the one obtained from the set of CCS processes [43]. *Specifications*, or *properties*, are subsets of $\textsc{Prc}$, ranged over by P, Q, R. A property P is a *consequence* of property Q whenever $Q \subseteq P$. Actions may be sequenced to form *finite or infinite traces* $t, u \in (\textsc{Act}^* \cup \textsc{Act}^\omega)$; the trace prefix-ordering $t \leq u$ denotes that t is a prefix of u. We say that a process p *produces* a trace t, or that t is a trace of p, if there is a sequence of transitions $p \xrightarrow{a} q \xrightarrow{b} \cdots$, such that $t = ab\cdots$; the trace t is also referred to as an *execution* of p. Note that if t is a trace of p, then so are all of its prefixes.

Runtime Monitoring, Verification and Monitorability. Runtime monitors are computational entities that reach a verdict after observing a finite prefix of an execution. A verdict, once reached, is irrevocable [5]. We only consider single-verdict monitors, namely *rejection monitors*, which flag violations of a property, and *acceptance monitors*, which validate a

property. Although mixed-verdict monitors can be used in a linear-time setting [4], only single-verdict monitors make sense in a branching-time setting[1]. Rejection and acceptance monitors are dual to one another in this setting. Our technical development thus focuses on rejection monitors, and obtains results for acceptance monitors by duality.

A monitor, denoted by $m, n, \ldots$, may be abstractly described as a (possibly infinite) set of finite traces, $m \subseteq \text{Act}^*$, that satisfies the following condition: if $t \in m$, then for any $u \in \text{Act}^*$ where $t \leq u$ it holds that $u \in m$. Intuitively, the traces in m are those that witness a violation of a property. The closure condition describes the irrevocability of verdicts. The collection of upward-closed subsets of Act^*, denoted by Mon, is therefore the set of all possible monitors. Often we restrict our discussion to a *subset* of Mon, $M \subseteq \text{Mon}$.

▶ **Definition 4.** *Monitor m rejects process p, $\textbf{\textit{rej}}(m, p)$, if p produces a trace t in m.* ⌟

Earlier work [23, 26, 24, 4] provides an operational interpretation of Def. 4 via an instrumentation of the monitor m executing with process p. Soundness and completeness relate monitors to the specifications they are expected to monitor [26, 4, 5]. *Soundness* requires that a monitor give only correct verdicts, while *completeness* demands that a monitor reject whenever the specification is violated.

▶ **Definition 5** (Soundness and Completeness). *A monitor $m \in \text{Mon}$ is:*
1. sound *for specification P if for all $p \in \text{Prc}$, $\textbf{\textit{rej}}(m, p)$ implies $p \notin P$;*
2. complete *for specification P if for all $p \in \text{Prc}$, $p \notin P$ implies $\textbf{\textit{rej}}(m, p)$.* ⌟

▶ **Definition 6** (Monitorability). *A specification P is* monitorable *in a monitor set M if there exists some $m \in M$ that is sound and complete for P.* ⌟

The notion of monitorability given in Def. 6 comes from [26]; although it is one of many possible definitions [5], it is the only one that has been extensively studied in the branching-time setting [26, 3, 1, 2, 4]. It also turns out to be the right one to use in the quest for optimal monitors, as argued in Sec. 3. One important consequence of Def. 6 is that there are some properties that are *not* monitorable.

▶ **Example 7.** The monitor from Ex. 3 that rejects all traces containing the consecutive events $\mathsf{c}_1\mathsf{w}_1$ is sound and complete for Spec 5, whereas Spec 1 in Ex. 1 is not monitorable. In the sequel, we simplify our example and assume that there is only one component in the system generating events $\mathsf{o}, \mathsf{w}, \mathsf{c}$. Another property that is not monitorable is the following:

"cw never occurs and on all infinite executions, w occurs." (Spec 6)

Indeed, a process whose only maximal trace is o^ω is not in this property but there is *no* monitor that is sound for Spec 6 and rejects it. ⌟

In practice, we often have (prior) knowledge about the type of process the monitor will be analysing at runtime, and the definition of monitorability should take such information into account, *i.e.*, grey-box monitoring. For our setting, we can express this prior knowledge as a set of processes, denoted as $R \subseteq \text{Prc}$, *i.e.*, the processes satisfying that prior knowledge.

▶ **Definition 8** (Soundness/Completeness with Knowledge). *The monitor $m \in \text{Mon}$ is:*
- Sound *for specification P with knowledge R if for all $p \in R$, $\textbf{\textit{rej}}(m, p)$ implies $p \notin P$.*
- *Complete for specification P with knowledge R if for all $p \in R$, $p \notin P$ implies $\textbf{\textit{rej}}(m, p)$.* ⌟

[1] Multi-verdict monitors are necessarily unsound in the branching-time setting [26].

▶ **Definition 9** (Monitorability with Knowledge). *A specification P is monitorable in a monitor set M, with prior knowledge R, if there exists a monitor $m \in M$ that is both sound and complete for P with knowledge R.* ⌟

3 The Strongest Monitorable Consequence

Since not all specifications have a sound and complete monitor, we are interested in computing an *optimal* monitor: a monitor which is sound for the specification, and rejects all violations that can be flagged. In this section we argue that to find the optimal monitor of a specification, we first need to compute its *strongest monitorable consequence*.

Although we focus on rejection monitors, optimal acceptance monitors are dual. An *optimal* monitor for a property P is a sound monitor for P that rejects each trace rejected by some sound monitor for that property.

▶ **Definition 10** (Optimality). *For a fixed monitor set $M \subseteq \textsc{Mon}$, monitor $m \in M$ is* optimal *in M for the property P whenever:*

- *it is sound for P and*
- *for all $n \in M$, if n is sound for P then $n \subseteq m$.* ⌟

Since the definition of a monitor as a set of finite traces does not guarantee computability, it is useful to parameterise this definition with the set of monitors M that determines the computational power of the monitors under scrutiny.

We now aim to characterise optimal monitors in terms of the properties they monitor for. First, for every monitor m, we can easily define a property for which it is both sound and complete:

$$P_m = \{\, p \mid p \text{ does not produce any trace } t \in m \,\}.$$

It is not hard to see that such a property P_m is unique for every monitor m.

▶ **Lemma 11.** *Monitor m is sound and complete for P if and only if $P = P_m$.* ◀

▶ **Proposition 12.** *For all $m, n \in \textsc{Mon}$, $m \subseteq n$ iff $P_n \subseteq P_m$.*

Proof. For the *if case*, assume $P_n \subseteq P_m$ and pick a $t \in m$ and the process p that produces only t. Then, $p \notin P_m$, which implies $p \notin P_n$ from $P_n \subseteq P_m$. By definition of P_n, this implies $t \in n$. We conclude that $m \subseteq n$. For the *only-if case*, assume $m \subseteq n$ and pick a $p \notin P_m$ that produces some $t \in m$. By inclusion $t \in n$ and therefore $p \notin P_n$. ◀

We can now characterise optimal monitors, Def. 10, in terms a notion of a *strongest monitorable consequence*.

▶ **Definition 13** (Strongest Monitorable Consequence). *Let $M \subseteq \textsc{Mon}$. The strongest monitorable consequence of a specification P with respect to M is a property Q that is monitorable in M such that:*

- *it is a consequence of P, i.e., $P \subseteq Q$, and*
- *for any R monitorable in M, $P \subseteq R$ implies $Q \subseteq R$.* ⌟

Note that the existence of a strongest monitorable consequence and of an optimal monitor, depends on M. We establish the correspondence between strongest monitorable consequences and optimal monitors (Thm. 16) using the following two lemmas.

▶ **Lemma 14.** *A* sound *monitor for a consequence of P is sound for P.*

Proof. Pick a consequence Q of P, *i.e.*, $P \subseteq Q$, and a sound monitor m for Q. If $\mathbf{rej}(m, p)$ for some p, then $p \notin Q$ by Def. 5. By $P \subseteq Q$ we obtain $p \notin P$, so m is sound for P. ◀

▶ **Lemma 15.** *If m is complete for P and sound for Q then $Q \subseteq P$.*

Proof. Pick a process $p \notin P$; for P to be a consequence of Q, *i.e.*, $Q \subseteq P$, we need to show that $p \notin Q$. By Def. 5.2 we know $\mathbf{rej}(m, p)$ and by Def. 5.1 we obtain $p \notin Q$. ◀

▶ **Theorem 16.** *A monitor $m \in M$ that is sound for P is optimal for P in M iff it is sound and complete for the strongest monitorable consequence of P with respect to M.*

Proof. For the *if case*, assume that m is sound and complete for Q, the strongest monitorable consequence of P with respect to M. We must show that m is optimal for P in M. Pick any other monitor $n \in M$ that is also sound for P. From Lem. 11, P_n is monitorable in M, and by Lem. 15 we know $P \subseteq P_n$. Since Q is the strongest monitorable consequence of P, we also know $Q \subseteq P_n$, and by Prop. 12 we obtain $n \subseteq m_Q$ as required.

For the *only-if case*, let m be an optimal monitor for P. By Lem. 11 and the soundness of m for P, it follows that P_m is a consequence of P, *i.e.*, $P \subseteq P_m$. Next, we show that P_m is the strongest monitorable consequence of P, from which the claim follows because m is sound and complete for P_m. Let Q be a monitorable consequence of P and let m_Q be a monitor for it. Since m is optimal (Def. 10), we know that $m_Q \subseteq m$. Thus by Prop. 12 we obtain $P_m \subseteq Q$. This implies that P_m is the strongest monitorable consequence of P. ◀

To find the optimal monitor of an arbitrary property, it therefore suffices to compute the sound and complete monitor of its strongest monitorable consequence. We can also extend this result for the cases with prior knowledge about the process to be monitored, Thm. 19.

▶ **Definition 17** (Optimality with Knowledge). *For a fixed monitor set $M \subseteq$ Mon, monitor $m \in M$ is* optimal *in M for property P with knowledge R whenever:*

- *it is sound for P with knowledge R and*
- *for all $n \in M$, if n is sound for P with knowledge R then $n \subseteq m$.* ⌟

Soundness and completeness with prior knowledge can be characterised with respect to soundness and completeness in the setting with no prior knowledge, Prc.

▶ **Proposition 18.** *Monitor m is sound with knowledge R for P iff it is sound for $P \cap R$.*

▶ **Theorem 19.** *For a fixed monitor set $M \subseteq$ Mon, a monitor $m \in M$ is optimal in M for the property P with knowledge R iff m is optimal in M for property $P \cap R$.*

Proof. For the *only-if* case, assume that m is optimal for P with knowledge R. From Def. 17, we know that m is sound for P with knowledge R, and therefore, by Prop. 18, m is sound for $P \cap R$. From Def. 17, we also know that if some n is sound for P with R, then $n \subseteq m$; again, by Prop. 18, if n is sound for $P \cap R$, then $n \subseteq m$. Therefore, m is also optimal for $P \cap R$. The *if* case is symmetric. ◀

4 Monitorability in recHML

Following Thms. 16 and 19, we investigate how to compute the strongest monitorable consequence for properties expressible in the Hennessy–Milner logic with recursion, recHML [37], as a means to obtain optimal monitors for such properties. recHML is a specification logic describing regular properties of processes, and can be seen as a reformulation of the well-studied modal μ-calculus [13, 14]. Since there are standard translations from CTL and

Syntax

$$
\begin{array}{rlll}
\varphi, \psi \in \textsc{recHML} ::= & \mathsf{tt} & \text{(truth)} & \mid \mathsf{ff} \quad \text{(falsehood)} \\
\mid & \varphi \vee \psi & \text{(disjunction)} & \mid \varphi \wedge \psi \quad \text{(conjunction)} \\
\mid & \langle a \rangle \varphi & \text{(existential modality)} & \mid [a]\varphi \quad \text{(universal modality)} \\
\mid & \mathsf{min}\, X.\varphi & \text{(least fixpoint)} & \mid \mathsf{max}\, X.\varphi \quad \text{(greatest fixpoint)} \\
\mid & X & \text{(recursion variable)} &
\end{array}
$$

Branching-Time Semantics

$$
\begin{array}{rcl@{\qquad}rcl}
[\![\mathsf{tt}, \rho]\!] & \stackrel{\text{def}}{=} & \textsc{Prc} & [\![\mathsf{ff}, \rho]\!] & \stackrel{\text{def}}{=} & \emptyset \\
[\![\varphi_1 \vee \varphi_2, \rho]\!] & \stackrel{\text{def}}{=} & [\![\varphi_1, \rho]\!] \cup [\![\varphi_2, \rho]\!] & [\![\varphi_1 \wedge \varphi_2, \rho]\!] & \stackrel{\text{def}}{=} & [\![\varphi_1, \rho]\!] \cap [\![\varphi_2, \rho]\!] \\
[\![\langle a \rangle \varphi, \rho]\!] & \stackrel{\text{def}}{=} & \left\{ p \mid \exists q \cdot p \xrightarrow{a} q \ \text{ and } \ q \in [\![\varphi, \rho]\!] \right\} & [\![X, \rho]\!] & \stackrel{\text{def}}{=} & \rho(X) \\
[\![[a]\varphi, \rho]\!] & \stackrel{\text{def}}{=} & \left\{ p \mid \forall q \cdot p \xrightarrow{a} q \ \text{ implies } \ q \in [\![\varphi, \rho]\!] \right\} & & & \\
[\![\mathsf{min} X.\varphi, \rho]\!] & \stackrel{\text{def}}{=} & \bigcap \{ P \mid [\![\varphi, \rho[X \mapsto P]]\!] \subseteq P \} & [\![\mathsf{max} X.\varphi, \rho]\!] & \stackrel{\text{def}}{=} & \bigcup \{ P \mid P \subseteq [\![\varphi, \rho[X \mapsto P]]\!] \}
\end{array}
$$

Figure 1 recHML Syntax and Branching-Time Semantics.

CTL* [34] into recHML, our investigation extends to these logics as well. The appeal of recHML comes from its generality, the pre-existence of procedures to compute sound and complete monitors for its monitorable fragment and its good closure properties. Indeed, we show that the strongest monitorable consequence of recHML formulae is itself expressible in recHML. It is unclear whether this is also the case for other branching-time logics, although in the linear time setting, this question is settled positively for LTL in [40].

recHML formulae are generated from the syntax given in Fig. 1, according to the following order of precedence: the existential and universal modal operators ($\langle a \rangle \varphi$ and $[a]\varphi$), conjunctions, disjunctions, and fixpoint operators ($\mathsf{min}\, X.\varphi$ and $\mathsf{max}\, X.\varphi$). The negation of a formula φ can be constructed with the duality rules in the usual way, and we use $\neg\varphi$ as a shorthand for it. In a formula $\mathsf{min}\, X.\varphi$ or $\mathsf{max}\, X.\varphi$, the fixpoint operator binds all free occurrences of X in φ. The subformula φ is then said to be the binding formula of X. We assume that for each variable X, there is exactly one formula $\mathsf{min}\, X.\varphi$ or $\mathsf{max}\, X.\varphi$ that binds X, denoted φ_X. Furthermore, without loss of generality, all formulas are assumed to be guarded [36]: every occurrence of a fixpoint variable within its binding is within the scope of a modal operator. We extend the notion of subformula and say that φ_X is the immediate subformula of X. We write $sf(\varphi)$ for the set of subformulas of φ. We take the size of a formula to be the number of its distinct subformulae, up to α-conversion.

A formula φ from recHML is evaluated on a state of an LTS. In addition to true, false and boolean connectives – which have their usual semantics – recHML has modal and fixpoints operators. The existential modality $\langle a \rangle \varphi$ holds at a state if there is an a-successor in which φ holds, whereas the universal modality $[a]\varphi$ holds if φ holds in all the a-successors of that state. The least fixpoint $\mathsf{min}\, X.\varphi$ and its dual $\mathsf{max}\, X.\varphi$ add recursion to the logic, allowing for the description of temporal properties such as reachability and invariance. Formally, the semantics is defined with respect to an interpretation ρ of the free variables of the formula. We write $[\![\varphi, \rho]\!]$ for the set of process states in an LTS which satisfy φ according to ρ. This set is defined by induction on the structure of the formula φ, following the semantics given

in Fig. 1. Two formulas are equivalent if they agree on all processes. We often consider closed formulas – namely those without free variables. In these cases, we can ignore the environment from the semantics and simply write $[\![\varphi]\!]$ instead of $[\![\varphi, \rho]\!]$.

▶ Remark 20. A system state p trivially satisfies a specification $[a]\varphi$ if it *cannot* transition with action a. Consequently the basic formula $[a]\mathsf{ff}$ describes states that cannot perform a-transitions; the dual basic formula $\langle a\rangle\mathsf{tt}$ denotes states that can perform a-transitions.

▶ **Example 21.** Property Spec 6 from Ex. 7 for $\text{Act} = \{\mathsf{o}, \mathsf{w}, \mathsf{c}\}$ can be expressed as:

$$\varphi_1 = (\max X.([\mathsf{o}]X \wedge [\mathsf{c}]X \wedge [\mathsf{w}]X \wedge [\mathsf{c}][\mathsf{w}]\mathsf{ff})) \ \wedge \ \min Y.([\mathsf{o}]Y \wedge [\mathsf{c}]Y).$$

The first conjunct in φ_1 prohibits the occurrence of cw while the second conjunct requires w to eventually occur on infinite traces (the sub-formula $\langle\mathsf{w}\rangle\mathsf{tt}$ disjuncted with $[\mathsf{o}]Y \wedge [\mathsf{c}]Y$ can be left implicit since $\text{Act} = \{\mathsf{o}, \mathsf{w}, \mathsf{c}\}$). A variation of Spec 1 from Ex. 1 on one component (for $\text{Act} = \{\mathsf{o}, \mathsf{w}, \mathsf{c}\}$) is Spec 7, described below and formalised as the formula φ_2:

"On all infinite executions, w occurs, but w only occurs after o." (Spec 7)

Whereas the outermost fixpoint formula in φ_2 below prohibits w from occurring before o, the innermost fixpoint formula requires w to occur eventually in any infinite execution.

$$\varphi_2 = \min X.([\mathsf{w}]\mathsf{ff} \wedge [\mathsf{c}]X \wedge [\mathsf{o}](\min Y.[\mathsf{c}]Y \wedge [\mathsf{o}]Y)).$$ ⌟

Monitorability for recHML was investigated in [26, 1], where monitors are specified as regular processes and monitorable properties have a syntactic characterisation:

▶ **Theorem 22.** *[26, Theorems 1 and 4] A formula of recHML is* (violation) monitorable *iff it is equivalent to a formula in the fragment sHML defined as follows:*

$$\varphi, \psi \in \text{sHML} ::= \quad \mathsf{tt} \quad | \ \mathsf{ff} \quad | \ [a]\varphi \quad | \ \varphi \wedge \psi \quad | \ \max X.\varphi \quad | \ X$$ ◀

A synthesis function that generates a regular (sound and complete) monitor from a sHML formula is also presented; such monitors are also shown to be finite state [4].

▶ **Example 23.** Since φ_1 and φ_2 are not sHML formulas, we cannot use the synthesis function from [26] to obtain runtime monitors for them. In fact, neither formula is monitorable according to [26]. Although Spec 5 from Ex. 3, with $\text{Act}_i = \{\mathsf{o}_i, \mathsf{w}_i, \mathsf{c}_i\}$ and $\text{Act} = \text{Act}_1 \cup \text{Act}_2$, can be expressed as the sHML formula φ_3, the knowledge (component independence) can be only expressed using formulas like φ_4, which are neither in sHML nor monitorable [26].

$$\varphi_3 = \max X.\Big([\mathsf{c}_1][\mathsf{w}_1]\mathsf{ff} \wedge \bigwedge_{a \in \text{Act}} [a]X\Big)$$

$$\varphi_4 = \max X. \bigwedge_{\substack{a \in \text{Act}_1 \\ b \in \text{Act}_2}} ([a][b]\mathsf{ff} \vee \langle b\rangle\langle a\rangle\mathsf{tt}) \wedge \bigwedge_{\substack{a \in \text{Act}_2 \\ b \in \text{Act}_1}} ([a][b]\mathsf{ff} \vee \langle b\rangle\langle a\rangle\mathsf{tt}) \wedge \bigwedge_{a \in \text{Act}} [a]X$$

The sub-formula $([a][b]\mathsf{ff} \vee \langle b\rangle\langle a\rangle\mathsf{tt})$ in φ_4 encodes the implication $(\langle a\rangle\langle b\rangle\mathsf{tt} \Rightarrow \langle b\rangle\langle a\rangle\mathsf{tt})$. The strongest monitorable consequence of $\varphi_3 \wedge \varphi_4$ is expressed by φ_5:

$$\varphi_5 = \max X.[\mathsf{c}_1](\max Y. \bigwedge_{b \in \text{Act}_2} [b]Y \wedge [\mathsf{w}_1]\mathsf{ff}) \wedge \bigwedge_{a \in \text{Act}} [a]X$$

A sound and complete monitor for this property will reject a process based on executions containing $\mathsf{c}_1\text{Act}_2^*\mathsf{w}_1$, rather than just $\mathsf{c}_1\mathsf{w}_1$. ⌟

In cases such as Ex. 23, we can obtain the optimal monitor of an arbitrary recHML specification φ by: (i) computing the strongest monitorable consequence $\psi \in$ sHML of φ; (ii) synthesising a sound and complete monitor for ψ using the synthesis function from [26].

5 Computing Strongest Monitorable Consequences in recHML

In this section, we describe a method for computing the strongest monitorable consequence of a recHML formula. The full proofs for this section can be found in the appendix. Our constructions rely on a disjunctive representation of formulas, as given in Def. 24.

▶ **Definition 24** (Disjunctive Form [51]). *The set of disjunctive formulas of recHML is given by the following grammar:*

$$\varphi, \psi \in \textit{disHML} ::= \quad \mathsf{tt} \quad | \quad \mathsf{ff} \quad | \quad \varphi \vee \psi \quad | \bigwedge_{a \in A} \Big(\big(\bigwedge_{\varphi \in \mathcal{B}_a} \langle a \rangle \varphi \big) \wedge [a] \bigvee_{\varphi \in \mathcal{B}_a} \varphi \Big)$$
$$| \quad \max X.\varphi \quad | \min X.\varphi \quad | \quad X,$$

where $A \subseteq Act$ and, for each action a in A, the set $\mathcal{B}_a \subseteq$ disHML is a finite set of formulas. ◀

In disjunctive formulas, conjunctions occur to express that for each $a \in A$, every a-successor satisfies a formula in some set $\mathcal{B}_a$ and every formula in $\mathcal{B}_a$ is satisfied by some a-successor.

▶ **Lemma 25** ([51]). *Every recHML formula is equivalent to a disjunctive one.* ◀

In [51], Walukiewicz provides a way to construct an equivalent disjunctive formula from a recHML one, based on a tableau method. He also shows that the satisfiability of disjunctive recHML formulas is decidable in linear time. Thus, we assume that, with the exception of ff, all subformulas of disjunctive formulas are satisfiable. This pre-processing accounts for one exponential in the complexity of our transformation.

We now establish a fundamental property of sHML formulas: if a process q does not satisfy $\theta \in$ sHML, then no process p that can produce all traces of q satisfies θ.

▶ **Definition 26.** *Process p* covers *process q when all traces of q are traces of p.* ◀

▶ **Lemma 27.** *If process p covers process q, then for closed $\theta \in$ sHML, $q \notin [\![\theta]\!]$ implies $p \notin [\![\theta]\!]$.*

Proof. From [26] there is a sound and complete m for θ. By $q \notin [\![\theta]\!]$ and the completeness of m, there is a trace of q (and of p), rejected by m. By the soundness of m, $p \notin [\![\theta]\!]$. ◀

We present the construction of the strongest monitorable consequence of a formula Ψ in three stages. We first eliminate the existential modalities in a formula. Then we eliminate least fixpoints. Finally, we use a more involved tableau method to remove all disjunctions.

5.1 Eliminating Existential Modalities

▶ **Definition 28.** *The operator $f_1 :$ recHML $\to$ recHML is defined such that $f_1(\langle a \rangle \varphi) = \mathsf{tt}$, while commuting with all other logical connectives.* ◀

That is, $f_1(\Psi)$ results from Ψ by replacing every occurrence of a subformula $\langle a \rangle \varphi$ by tt.

▶ **Lemma 29.** *For every $\Psi \in$ disHML, $f_1(\Psi)$ has the same sHML consequences as Ψ.*

(Proof outline). We show that every sHML formula is a consequence of Ψ iff it is a consequence of $f_1(\Psi)$. For the *if* direction, it suffices to prove $[\![\Psi]\!] \subseteq [\![f_1(\Psi)]\!]$ using the monotonicity of recHML operators resulting from the absence of negation.

$$\frac{\Gamma \cup \{\psi \vee \varphi\}}{\Gamma \cup \{\varphi, \psi\}}\ (\vee) \qquad \frac{\Gamma \cup \{\psi \wedge \varphi\}}{\Gamma \cup \{\varphi\} \quad \Gamma \cup \{\psi\}}\ (\wedge) \qquad \frac{\Gamma \cup \{\max X.\varphi\}}{\Gamma \cup \{\varphi\}}\ (\max) \qquad \frac{\Gamma}{\{\psi \mid [a]\psi \in \Gamma\}}\ ([a])$$

$$\frac{\Gamma \cup \{X\}}{\Gamma \cup \{\varphi_X\}}\ (X) \qquad \frac{\Gamma \cup \{\mathsf{tt}\}}{\{\mathsf{tt}\}}\ (\mathsf{tt}) \qquad \frac{\Gamma \cup \{\mathsf{ff}\}}{\Gamma}\ (\mathsf{ff}) \qquad \frac{\Gamma \cup \{[a]\psi, [b]\varphi\} \quad a \neq b}{\{\mathsf{tt}\}}\ ([a,b])$$

Figure 2 Tableau rules where Γ is a set of formulas.

For the *only-if* direction, the intuition is as follows (see App. A). Let θ be a sHML formula such that $[\![\Psi]\!] \subseteq [\![\theta]\!]$. To show $[\![f_1(\Psi)]\!] \subseteq [\![\theta]\!]$, we proceed by contradiction: starting from a process $p \in [\![f_1(\Psi) \wedge \neg\theta]\!]$ we build a cover q of p such that $q \in [\![\theta]\!]$, which contradicts Lem. 27. To obtain this cover, we use the fact that f_1 turns the conjunctions $\bigwedge_{a\in A}((\bigwedge_{\psi\in\mathcal{B}_a}\langle a\rangle\psi) \wedge [a]\bigvee\mathcal{B}_a)$ of a disjunctive formula into conjunctions of the form $\bigwedge_{a\in A}[a]\bigvee\mathcal{B}_a$. The cover is obtained by finding the states r in which conjunctions of the latter form must be true for $f_1(\Psi)$ to be true in p, and adding an a-successor s_φ to r for each $\varphi \in \mathcal{B}_a$ and $a \in A$. This is possible, because all subformulae of disjunctive formulas are assumed to be satisfiable. The state r with these additional successors then satisfies $\bigwedge_{a\in A}((\bigwedge_{\psi\in\mathcal{B}_a}\langle a\rangle\psi) \wedge [a]\bigvee\mathcal{B}_a)$, which allows us to argue that $q \in [\![\Psi]\!] \subseteq [\![\theta]\!]$. ◀

▶ Remark 30. Disjunctive form is key here: Applying f_1 to formula φ_4 from Ex. 23, which is not disjunctive, yields $\left(\bigwedge_{a\in \text{Act}_1}\bigwedge_{b\in\text{Act}_2}([a][b]\mathsf{ff} \vee \mathsf{tt})\right) \wedge \left(\bigwedge_{a\in\text{Act}_2}\bigwedge_{b\in\text{Act}_1}([a][b]\mathsf{ff} \vee \mathsf{tt})\right)$ which can be simplified to tt and does not provide any useful information for monitoring.

5.2 Eliminating Least Fixpoints

▶ **Definition 31.** *The operator f_2 : recHML $\to$ recHML is defined such that $f_2(\min X.\varphi) = \max X.\varphi$, while commuting with all other logical connectives.* ◀

▶ **Lemma 32.** *For every closed formula $\Psi \in$ recHML without existential modalities, $f_2(\Psi)$ has the same sHML consequences as Ψ.*

(Proof outline). One direction follows from $[\![\min X.\varphi]\!] \subseteq [\![\max X.\varphi]\!]$: since recHML is negation-free, it behaves in a monotone way, and therefore $f_2(\Psi)$ is a consequence of Ψ.

The intuition for the other direction is as follows (see App. A). If a process p violates a consequence $\theta \in$ sHML of Ψ but satisfies $f_2(\Psi)$, then, due to the monitorability of θ, there is a finite trace t of p, where every process producing t must also violate θ. Thus, there is a *finite* process q that violates θ, but also satisfies $f_2(\Psi)$ due to the absence of existential modalities in $f_2(\Psi)$. Since $f_2(\Psi)$ and Ψ only differ with respect to their fixpoint operators, they agree on all finite processes: q satisfies Ψ and its consequence θ, a contradiction. ◀

▶ Remark 33. Lem. 32 does not hold for formulas *with* existential modalities. For instance, the formula $\min X.\langle a\rangle X$ is equivalent to, and thus implies, ff; yet $f_2(\min X.\langle a\rangle X) = \max X.\langle a\rangle X$, which is satisfiable by a system producing the infinite trace a^ω.

▶ **Example 34.** Formula φ_2 from Ex. 21 becomes $\max X.([\mathsf{w}]\mathsf{ff} \wedge [\mathsf{c}]X \wedge [\mathsf{o}](\max Y.[\mathsf{c}]Y \wedge [\mathsf{o}]Y))$ under $f_2(-)$, which simplifies to $\max X.([\mathsf{w}]\mathsf{ff} \wedge [\mathsf{c}]X)$ as $\max Y.[\mathsf{c}]Y \wedge [\mathsf{o}]Y$ simplifies to tt. Since Act$=\{\mathsf{o},\mathsf{c},\mathsf{w}\}$ this formula expresses the property that "w does not occur before o." ◀

5.3 Eliminating Disjunctions

The final and hardest step turns a formula without existential modalities and least fixpoints into its strongest sHML consequence. The intuition is that a violation of a specification of the form $[a]\psi \vee [a]\varphi$ can only be monitored if there is an a-successor in which violations for

both ψ and φ can be detected. Hence, we turn $[a]\psi \vee [a]\varphi$ into $[a](\psi \vee \varphi)$. In contrast, no violation of $[a]\psi \vee [b]\varphi$ can be identified from a single branch, so we rewrite it to tt.

To transform fixpoint-free formulas, it suffices to recursively push disjunctions through the formula. The transformation in the presence of fixpoints is roughly dual to that for disjunctive form presented by Janin and Walukiewicz in [32] and, like theirs, uses a set of tableau rules, but this time to eliminate disjunctions rather than conjunctions. Our rules differ significantly from those in [32] in how they deal with modalities; in particular, our transformation does not preserve the semantics of formulae, but only sHML consequences.

▶ **Definition 35** (Tableau elimination of disjunctions). *Given a closed formula Ψ with neither* min *operators nor existential modalities, 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 subformulae of Ψ, such that:*

- *The root is labelled $\{\Psi\}$,*
- *For each node n and its children, there is a tableau rule (Fig. 2) such that n is labelled with the premise and its children are labelled with its conclusions,*
- *This tableau rule is the rule $[a]$ only if $L_\Psi(n)$ matches the premise of no other tableau-rule.*

The disjunction-free formula equivalent to Ψ is then retrieved from $\mathcal{T}(\Psi)$ by defining the labelling L' as follows and applying it to each node. For each leaf node n:

- *If it has a back-edge to an inner node m, it is labelled X_m;*
- *If it does not have a back-edge, it is labelled with tt, if it contains tt, and ff, otherwise.*

For each inner node n that is not the target of a back-edge:

- *If n has a child m via the rules $\vee, \mathsf{tt}, [a,b], X, \mathsf{max}$, then l has the label $L'(m)$;*
- *If n has children m, m' via rule $\wedge$, then l is label $L'(m) \wedge L'(m')$;*
- *If n has a child m via $[a]$, then l is label $[a]L'(m)$.*

In a second pass, if n is the target of back-edges, then its label is $\mathsf{max}\, X_n.l$, and otherwise it is l, where $l = L'(n)$ as defined above. Let $f_3(\Psi)$ be the L'-label of the root of $\mathcal{T}(\Psi)$.

▶ **Example 36.** Consider the bespoke formula $\mathsf{max}\, X.[a]([a]X \wedge [b]\mathsf{ff}) \vee [a]([a]\mathsf{ff} \wedge [b]X)$. The tableau for this formula labelled with subsets of subformulas using Def. 35 is given below.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\mathsf{max}\, X.[a]([a]X \wedge [b]\mathsf{ff}) \vee [a]([a]\mathsf{ff} \wedge [b]X)}{[a]([a]X \wedge [b]\mathsf{ff}) \vee [a]([a]\mathsf{ff} \wedge [b]X)}\,(\mathsf{max})}{[a]([a]X \wedge [b]\mathsf{ff}), [a]([a]\mathsf{ff} \wedge [b]X)}\,(\vee)}{[a]X \wedge [b]\mathsf{ff}, [a]\mathsf{ff} \wedge [b]X}\,([a])}{[a]X, [a]\mathsf{ff} \wedge [b]X}\,(\wedge) \qquad \dfrac{[b]\mathsf{ff}, [a]\mathsf{ff} \wedge [b]X}{[b]\mathsf{ff}, [a]\mathsf{ff}}\,(\wedge)$$

$$\dfrac{\dfrac{[a]X, [a]\mathsf{ff}}{X, \mathsf{ff}}\,([a])}{X}\,(\mathsf{ff}) \qquad \dfrac{[a]X, [b]X}{\mathsf{tt}}\,([a,b]) \qquad \dfrac{[b]\mathsf{ff}, [a]\mathsf{ff}}{\mathsf{tt}}\,([a,b]) \qquad \dfrac{[b]\mathsf{ff}, [b]X}{\mathsf{ff}, X}\,(\mathsf{ff})}{X}\,([b])$$

The corresponding tableau relabelled as L' yielding the strongest sHML consequence is:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\mathsf{max}\, X_1.[a]([a]X_1 \wedge \mathsf{tt} \wedge \mathsf{tt} \wedge [b]X_1)}{[a]([a]X_1 \wedge \mathsf{tt} \wedge \mathsf{tt} \wedge [b]X_1)}\,(\mathsf{max})}{[a]([a]X_1 \wedge \mathsf{tt} \wedge \mathsf{tt} \wedge [b]X_1)}\,(\vee)}{[a]X \wedge \mathsf{tt} \wedge \mathsf{tt} \wedge [b]X_1}\,([a])}{[a]X_1 \wedge \mathsf{tt}}\,(\wedge) \qquad \dfrac{\mathsf{tt} \wedge [b]X_1}{\mathsf{tt}}\,(\wedge)$$

$$\dfrac{\dfrac{[a]X_1}{X_1}\,([a])}{X_1}\,(\mathsf{ff}) \qquad \dfrac{\mathsf{tt}}{\mathsf{tt}}\,([a,b]) \qquad \dfrac{\mathsf{tt}}{\mathsf{tt}}\,([a,b]) \qquad \dfrac{[b]X_1}{X_1}\,([b])}{X_1}\,(\mathsf{ff})$$

⌟

▶ **Lemma 37.** *Given a closed formula Ψ of* recHML without min *operators or existential modalities, $f_3(\Psi)$ has the same* sHML *consequences as Ψ.*

Proof sketch. The proof of this lemma rests on the observation that all violations of $f_3(\Psi)$ and Ψ correspond to a single path in $\mathcal{T}(\Psi)$. We can then use the two labellings of $\mathcal{T}(\Psi)$ to move between the witnesses that we use for the violation of $f_3(\Psi)$ and Ψ. ◀

5.4 The strongest sHML consequence

▶ **Theorem 38.** *$f_3 \circ f_2 \circ f_1(\Psi)$ is the strongest* sHML *consequence of any closed $\Psi \in$ recHML.*

Proof. Follows from Lems. 29 and 37 and Def. 31. By construction $f_3 \circ f_2 \circ f_1(\Psi) \in$ sHML. Moreover, $f_3 \circ f_2 \circ f_1(\Psi)$ has the same sHML consequences as Ψ, making it the strongest sHML consequence of Ψ. ◀

We can symmetrically compute the weakest satisfaction-monitorable antecedent of Ψ, in order to synthesize an optimal acceptance-monitor, or construct the weakest satisfaction-monitorable antecedent by negating $f_3 \circ f_2 \circ f_1(\neg\Psi)$ where $\neg\Psi$ is the negation of Ψ in disjunctive form. In principle, one could also consider constructing optimal monitors from *both* violations and satisfactions of a property Ψ, by deducing the strongest violation-monitorable consequence φ_V of Ψ and the weakest satisfaction-monitorable antecedent φ_S of Ψ; the monitors could be used in tandem to detect all possible satisfactions or violations for Ψ. However, in a branching-time setting either φ_V or φ_S must be trivial:

▶ **Proposition 39.** *For any branching-time property P, its strongest monitorable consequence P_V and its weakest monitorable antecedent P_S, we either have $P_V = P_{RC}$ or $P_S = \emptyset$.* ◀

Proof. If there is a process $p \notin P_V$ and a process $q \in P_S$, then by merging the initial states of p and q we obtain a process that covers p and therefore violates P_V and therefore also P, and that covers q and therefore satisfies P_S and therefore also P, a contradiction. ◀

5.5 Complexity

Eliminating existential modalities and fixpoints does not increase the size of a formula. However, the two tableau constructions used – the first one required to turn the initial formula into disjunctive form, and the second one used to eliminate disjunctions – each can cause an exponential blow-up.

Morally, this is just the cost of determinising alternating automata (already double exponential for finite automata [16]): the automaton corresponding to our final formula, obtained via standard formula-automata correspondences [20], is deterministic (even though automata over trees are not in general determinisable). Indeed, the synthesis from [26], when applied to the formulas we obtain, yields deterministic monitors, in the sense of [6], because our formulas contain no disjunctions, and only conjunctions over disjoint modalities (of the form $\bigwedge_{a \in A}[a]\psi_a$). Whether a more compact non-deterministic monitor can be synthesised instead, or whether the last step, of constructing $f_3(-)$, can be implemented on-the-fly (in the spirit of [35]) is left for future work.

This double-exponential complexity is already present, and necessary, in the corresponding linear-time problem computing a deterministic automaton that recognises the *bad prefixes* of a linear-time property [35]. As Kupferman and Vardi write, this procedure has the flavour of determinisation, hence its double-exponential complexity. Our procedure, despite the added complications associated with branching-time, follows the same principle without a significant

additional cost. Interestingly, obtaining the strongest monitorable consequence of an LTL formula in LTL form is much harder. While the (counter-free) non-deterministic automaton that recognises executions without bad prefixes, *i.e.,* the strongest monitorable consequence of an LTL formula, requires exponential blow-up, the best procedure known to date to go from a (counter-free) non-deterministic automaton to an LTL formula uses star-free regular expressions and does not have an elementary complexity upper bound [41, 45].

On a more pragmatic note, both f_1 and f_2 only simplify formulas while f_3 eliminates subformulae containing mixed modalities $[a]\psi \wedge [b]\varphi$, so blow-ups can only occur in f_3 if disjunctions and modalities over the same action interact in a pathological way.

6 Related Work

Linear- vs. Branching-time. Runtime monitoring can be used to verify whether an execution satisfies a linear-time property, for example before the output of a third party component is used as input for a critical component. It can also be used to verify whether a system satisfies a branching-time property, for example as a best-effort light-weight verification strategy. The branching-time properties that one verifies at runtime often consist of properties of the form "on all paths, φ_L holds", where φ_L is a linear-time property. For these kinds of properties, the distinction between the branching-time and linear-time cases can be subtle. In particular, the branching-time case is then implicitly reduced to the linear-time case, *i.e.,* just checking for violations of φ_L. However, in this situation it *only* makes sense to check for violations of φ_L, as satisfactions do not give enough information to deduce anything about the system itself. In contrast, if we are interested in truly linear-time properties, then a monitor can simultaneously check both for violations and satisfactions, as it is done in [27].

Here we are in the branching-time setting: the prior knowledge can be an arbitrary branching-time property, and the property to be monitored can either be a linear-time property quantified universally over all branches, or any other branching-time property. Note that given an LTL formula φ_L, there are standard translations to build a recHML formula φ_B such that φ_B holds in a system if and only if φ_L holds in all of its executions [15]. These can be used to combine a linear-time property, to verify at runtime, with a branching-time property representing the prior knowledge.

As discussed in Sec. 5.5, finding optimal monitors for properties over infinite traces corresponds to computing the good/bad prefixes of the property. Kupferman and Vardi [35] describe how to do this for safety properties described as LTL formulas or Büchi automata. Havelund and Peled [28] describe the same procedure for arbitrary trace properties.

Hierarchies of monitorability. There are many definitions of monitorability (surveyed in [5]) and property classifications (for instance [28, 44]) that help us understand the guarantees we can expect from RV tools for different properties. However monitorable a property is, its optimal monitor is by definition the gold standard to which any RV tool can aspire. Optimal monitors might help determine the degree of monitorability of a property.

Monitoring with prior knowledge. Recently, Henzinger and Saraç [30] studied how assumptions (prior knowledge) can make non-monitorable linear-time properties monitorable. Interestingly, in their setting a property P is not monitorable under assumption K if and only if $P \wedge K$ is monitorable without assumptions, as is the case here. This is because they study a different definition of monitorability, which is not as well behaved under assumption as the notion we use. (Our choice of notion of monitorability is utilitarian: it enables us to

compute optimal monitors.) Independently, Cimatti *et al.* and Leucker have also considered a form of monitoring of linear-time properties with (linear-time) prior knowledge in [17, 38]. Leucker proposes an LTL semantics parameterised by this prior knowledge while Cimatti *et al.* incorporate the assumption directly into the monitoring algorithm, thereby treating violations of the assumptions and violations of the property to be monitored differently. Stucki *et al.* [49] parameterise monitorability for hyperproperties with the system under consideration. Their notion of *perfect monitor* corresponds to our *optimal monitor*. Although the authors in [22] study the decidability of monitorability for hyperproperties, neither work describes methods for computing the optimal monitors of hyperproperties.

Multi-valued logics. Logics with three-valued semantics (yes, no, indecisive) can be used to describe monitors [12, 21, 19]. However, whether monitor semantics are given by a many-valued logic or other means, questions of soundness, completeness and optimality with respect to the (two valued) specification formula remain the same.

Monitoring for under-specified components. In orthogonal work that has similarities with ours, Sistla and co-authors [42, 48, 47] address the following problem: given an under-specification φ, and a goal specification ψ, compute a safety property θ such that $\varphi \wedge \theta \implies \psi$. The intuition for this is that if φ is assumed, and violations of θ can be monitored at runtime, then ψ can be assumed whenever the monitor does not detect a violation of θ. This problem then reduces to computing a *safety antecedent* of a specification, namely $\neg\varphi \wedge \psi$. Unlike the strongest monitorable consequence, there is no weakest safety antecedent: properties can be approximated from below with arbitrary precision using a safety formula.

7 Conclusion

We have shown how to compute optimal monitors for arbitrary regular branching-time properties, following a procedure which is sound for arbitrary (not just regular) properties. Our core insight is that the theory of runtime monitors can be extended to the (partial) verification of specifications previously dismissed as unmonitorable, such as most branching-time properties. In particular, this enables us to integrate any prior contextual knowledge of the system into our monitors. We show that this is indeed the best a monitor can do.

References

1 Luca Aceto, Antonis Achilleos, Adrian Francalanza, and Anna Ingólfsdóttir. Monitoring for silent actions. In Satya Lokam and R. Ramanujam, editors, *FSTTCS*, volume 93 of *LIPIcs*, pages 7:1–7:14, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

2 Luca Aceto, Antonis Achilleos, Adrian Francalanza, and Anna Ingólfsdóttir. A framework for parameterized monitorability. In Christel Baier and Ugo Dal Lago, editors, *Foundations of Software Science and Computation Structures - 21st International Conference, FOSSACS 2018*, volume 10803 of *Lecture Notes in Computer Science*, pages 203–220. Springer, 2018. doi:10.1007/978-3-319-89366-2_11.

3 Luca Aceto, Antonis Achilleos, Adrian Francalanza, Anna Ingólfsdóttir, and Sævar Örn Kjartansson. On the complexity of determinizing monitors. In Arnaud Carayol and Cyril Nicaud, editors, *Implementation and Application of Automata - 22nd International Conference, CIAA 2017*, volume 10329 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 2017. doi:10.1007/978-3-319-60134-2_1.

4 Luca Aceto, Antonis Achilleos, Adrian Francalanza, Anna Ingólfsdóttir, and Karoliina Lehtinen. Adventures in monitorability: From branching to linear time and back again. *Proceedings*

of the ACM on Programming Languages, 3(POPL):52:1–52:29, 2019. URL: https://dl.acm.org/citation.cfm?id=3290365.

5 Luca Aceto, Antonis Achilleos, Adrian Francalanza, Anna Ingólfsdóttir, and Karoliina Lehtinen. An operational guide to monitorability. In *Software Engineering and Formal Methods - 17th International Conference, SEFM 2019, Oslo, Norway, September 18-20, 2019, Proceedings*, volume 11724 of *LNCS*, pages 433–453. Springer, 2019. doi:10.1007/978-3-030-30446-1_23.

6 Luca Aceto, Antonis Achilleos, Adrian Francalanza, Anna Ingólfsdóttir, and Sævar Örn Kjartansson. Determinizing monitors for HML with recursion. *Journal of Logical and Algebraic Methods in Programming*, 111:100515, February 2020. doi:10.1016/j.jlamp.2019.100515.

7 Bowen Alpern and Fred B Schneider. Defining liveness. *Information processing letters*, 21(4):181–185, 1985.

8 Duncan Paul Attard, Ian Cassar, Adrian Francalanza, Luca Aceto, and Anna Ingolfsdottir. *Behavioural Types: from Theory to Tools*, chapter A Runtime Monitoring Tool for Actor-Based Systems, pages 49–74. River Publishers, 2017.

9 Duncan Paul Attard and Adrian Francalanza. A monitoring tool for a branching-time logic. In Yliès Falcone and César Sánchez, editors, *Runtime Verification - 16th International Conference, RV 2016*, volume 10012 of *Lecture Notes in Computer Science*, pages 473–481. Springer, 2016. doi:10.1007/978-3-319-46982-9_31.

10 Christel Baier, Joost-Pieter Katoen, and Kim Guldstrand Larsen. *Principles of model checking*. MIT press, 2008.

11 Ezio Bartocci, Yliès Falcone, Adrian Francalanza, and Giles Reger. *Introduction to Runtime Verification*, pages 1–33. Springer International Publishing, Cham, 2018. doi:10.1007/978-3-319-75632-5_1.

12 Andreas Bauer, Martin Leucker, and Christian Schallhart. The good, the bad, and the ugly, but how ugly is ugly? In Oleg Sokolsky and Serdar Taşıran, editors, *Runtime Verification*, pages 126–138, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

13 Julian Bradfield and Colin Stirling. Chapter 4 - Modal logics and mu-calculi: An introduction. In J.A. Bergstra, A. Ponse, and S.A. Smolka, editors, *Handbook of Process Algebra*, pages 293–330. Elsevier Science, Amsterdam, 2001. doi:10.1016/B978-044482830-9/50022-9.

14 Julian Bradfield and Colin Stirling. Modal μ-calculi. *Studies in Logic and Practical Reasoning*, 3:721–756, 2007.

15 Julian Bradfield and Igor Walukiewicz. *The mu-calculus and Model Checking*, pages 871–919. Springer, May 2018. doi:10.1007/978-3-319-10575-8_26.

16 Ashok K. Chandra, Dexter C. Kozen, and Larry J. Stockmeyer. Alternation. *Journal of the ACM*, 28(1):114–133, January 1981. doi:10.1145/322234.322243.

17 Alessandro Cimatti, Chun Tian, and Stefano Tonetta. Assumption-based runtime verification with partial observability and resets. In *International Conference on Runtime Verification*, pages 165–184. Springer, 2019.

18 Edmund M Clarke, Orna Grumberg, and Doron Peled. *Model Checking*. MIT press, 1999.

19 Volker Diekert and Martin Leucker. Topology, monitorable properties and runtime verification. *Theoretical Computer Science*, 537:29–41, 2014. Theoretical Aspects of Computing (ICTAC 2011). doi:10.1016/j.tcs.2014.02.052.

20 E Allen Emerson and Charanjit S Jutla. Tree automata, mu-calculus and determinacy. In *FoCS*, volume 91, pages 368–377. Citeseer, 1991.

21 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, 2012. doi:10.1007/s10009-011-0196-8.

22 Bernd Finkbeiner, Christopher Hahn, Marvin Stenger, and Leander Tentrup. Monitoring hyperproperties. *Formal Methods in System Design*, 54(3):336–363, 2019.

23 Adrian Francalanza. A Theory of Monitors (Extended Abstract). In *FoSSaCS*, volume 9634 of *LNCS*, pages 145–161, 2016.

24 Adrian Francalanza. Consistently-detecting monitors. In Roland Meyer and Uwe Nestmann, editors, *28th International Conference on Concurrency Theory (CONCUR 2017)*, volume 85 of *LIPIcs*, pages 8:1–8:19, Dagstuhl, Germany, 2017. Schloss Dagstuhl. doi:10.4230/LIPIcs.CONCUR.2017.8.

25 Adrian Francalanza, Luca Aceto, Antonis Achilleos, Duncan Paul Attard, Ian Cassar, Dario Della Monica, and Anna Ingólfsdóttir. A foundation for runtime monitoring. In Shuvendu K. Lahiri and Giles Reger, editors, *Runtime Verification - 17th International Conference, RV 2017*, volume 10548 of *Lecture Notes in Computer Science*, pages 8–29. Springer, 2017. doi:10.1007/978-3-319-67531-2_2.

26 Adrian Francalanza, Luca Aceto, and Anna Ingólfsdóttir. Monitorability for the Hennessy-Milner logic with recursion. *Formal Methods in System Design*, 51(1):87–116, 2017. doi:10.1007/s10703-017-0273-z.

27 M.C.W. Geilen. On the construction of monitors for temporal logic properties. *Electronic Notes in Theoretical Computer Science*, 55(2):181–199, 2001. RV'2001, Runtime Verification (in connection with CAV '01). doi:10.1016/S1571-0661(04)00252-X.

28 Klaus Havelund and Doron Peled. Runtime Verification: From Propositional to First-Order Temporal Logic. In *Runtime Verification - 18th International Conference, RV 2018, Limassol, Cyprus, November 10-13, 2018, Proceedings*, volume 11237 of *LNCS*, pages 90–112. Springer, 2018. doi:10.1007/978-3-030-03769-7_7.

29 Klaus Havelund and Grigore Rosu. Synthesizing monitors for safety properties. In *TACAS*, volume 2, pages 342–356. Springer, 2002.

30 Thomas A Henzinger and N Ege Saraç. Monitorability under assumptions. In *International Conference on Runtime Verification*, pages 3–18. Springer, 2020.

31 David Janin and Igor Walukiewicz. On the expressive completeness of the propositional mu-calculus with respect to monadic second order logic. In *International Conference on Concurrency Theory*, pages 263–277. Springer, 1996.

32 David Janin and Igor Walukiewicz. On the expressive completeness of the propositional mu-calculus with respect to monadic second order logic. In *CONCUR '96: Concurrency Theory*, pages 263–277. Springer Berlin Heidelberg, 1996. doi:10.1007/3-540-61604-7_60.

33 Robert M. Keller. Formal verification of parallel programs. *Commun. ACM*, 19(7):371–384, 1976. doi:10.1145/360248.360251.

34 Dexter C. Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27:333–354, 1983.

35 Orna Kupferman and Moshe Y. Vardi. Model checking of safety properties. *Formal Methods in System Design*, 19(3):291–314, 2001.

36 Orna Kupferman, Moshe Y Vardi, and Pierre Wolper. An automata-theoretic approach to branching-time model checking. *Journal of the ACM*, 47(2):312–360, 2000.

37 Kim G. Larsen. Proof Systems for Satisfiability in Hennessy-Milner Logic with recursion. *Theoretical Computer Science*, 72(2):265–288, 1990. doi:10.1016/0304-3975(90)90038-J.

38 Martin Leucker. Sliding between model checking and runtime verification. In *International Conference on Runtime Verification*, pages 82–87. Springer, 2012.

39 Martin Leucker, César Sánchez, Torben Scheffel, Malte Schmitz, and Daniel Thoma. Runtime verification for timed event streams with partial information. In Bernd Finkbeiner and Leonardo Mariani, editors, *Runtime Verification - 19th International Conference, RV 2019, Porto, Portugal, October 8-11, 2019, Proceedings*, volume 11757 of *LNCS*, pages 273–291. Springer, 2019. doi:10.1007/978-3-030-32079-9_16.

40 Grgur Petric Maretić, Mohammad Torabi Dashti, and David Basin. Ltl is closed under topological closure. *Information Processing Letters*, 114(8):408–413, 2014.

41 Grgur Petric Maretić, Mohammad Torabi Dashti, and David Basin. Ltl is closed under topological closure. *Information Processing Letters*, 114(8):408–413, 2014.

42 Tiziana Margaria, A. Prasad Sistla, Bernhard Steffen, and Lenore D. Zuck. Taming interface specifications. In Martín Abadi and Luca de Alfaro, editors, *CONCUR 2005 – Concurrency Theory*, pages 548–561, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

43 R Milner. A calculus of communicating systems. *Lecture Notes in Comput. Sci. 92*, 1980.

44 Doron Peled and Klaus Havelund. Refining the safety–liveness classification of temporal properties according to monitorability. In *Models, Mindsets, Meta: The What, the How, and the Why Not?*, pages 218–234. Springer, 2019.

45 A Peuli and Lenore Zuck. In and out of temporal logic. In *[1993] Proceedings Eighth Annual IEEE Symposium on Logic in Computer Science*, pages 124–135. IEEE, 1993.

46 Fred B. Schneider. Enforceable security policies. *ACM Transactions on Information and System Security*, 3(1):30–50, 2000.

47 A Prasad Sistla and Abhigna R Srinivas. Monitoring temporal properties of stochastic systems. In *International Workshop on Verification, Model Checking, and Abstract Interpretation*, pages 294–308. Springer, 2008.

48 A Prasad Sistla, Min Zhou, and Lenore D Zuck. Monitoring off-the-shelf components. In *International Workshop on Verification, Model Checking, and Abstract Interpretation*, pages 222–236. Springer, 2006.

49 Sandro Stucki, César Sánchez, Gerardo Schneider, and Borzoo Bonakdarpour. Gray-box monitoring of hyperproperties. In *Formal Methods–The Next 30 Years: Third World Congress, FM 2019, Porto, Portugal, October 7–11, 2019, Proceedings*, volume 11800, page 406. Springer Nature, 2019.

50 Mahesh Viswanathan and Moonzoo Kim. Foundations for the run-time monitoring of reactive systems - fundamentals of the MaC language. In Zhiming Liu and Keijiro Araki, editors, *Theoretical Aspects of Computing - ICTAC 2004, First International Colloquium*, volume 3407 of *Lecture Notes in Computer Science*, pages 543–556. Springer, 2004. doi:10.1007/978-3-540-31862-0_38.

51 Igor Walukiewicz. Completeness of Kozen's axiomatisation of the propositional μ-calculus. *Information and Computation*, 157(1-2):142–182, February 2000. doi:10.1006/inco.1999.2836.

A Technical Proofs

In our proofs, instead of working with the classical semantics, we use *consistent annotations* and *counter-annotations* which respectively witness that a property holds or does not hold for a process. The intuition is that an evaluation of $\psi \vee \varphi$ to true must also evaluate either ψ or φ to true, and an annotation indicates which one. Similarly, for $\langle a \rangle \psi$ to be true at a state, one of the state's a-successors must be annotated with ψ.

▶ **Example 40.** The witness of the reachability specification $\mathsf{min}\, X.(\langle a \rangle X \vee \langle b \rangle \mathsf{tt})$ (there is a sequence of a-transitions that leads to a b-transition) for a process p would consist of the following annotation: p is annotated with

$$\{\mathsf{min}\, X.\langle a \rangle X \vee \langle b \rangle \mathsf{tt}, \langle a \rangle X \vee \langle b \rangle \mathsf{tt}, \langle a \rangle X\},$$

a finite sequence of a-successors will be annotated with

$$\{X, \mathsf{min}\, X.\langle a \rangle X \vee \langle b \rangle \mathsf{tt}, \langle a \rangle X \vee \langle b \rangle \mathsf{tt}, \langle a \rangle X\}$$

and finally an a-successor will be annotated with

$$\{X, \mathsf{min}\, X.\langle a \rangle X \vee \langle b \rangle \mathsf{tt}, \langle a \rangle X \vee \langle b \rangle \mathsf{tt}, \langle b \rangle \mathsf{tt}\}$$

and its b-successor will be annotated with tt. ◀

In the following, for each formula φ with free variables we consider the closure $c(\varphi)$ of φ, which results by replacing in φ all free variables X by φ_X. We use $[\![\varphi]\!]$ for an open formula φ, to mean $[\![c(\varphi)]\!]$. We say that a formula φ is satisfiable when $[\![\varphi]\!] \neq \emptyset$.

▶ **Definition 41** (Locally consistent annotation). *An* annotation $A : P \to \mathcal{P}(sf(\Psi))$, *where* $P \subseteq \textsc{Prc}$ *is a labelling of* P *(a partial labelling of* $\textsc{Prc}$*) with sets of subformulae of a closed formula* Ψ *of* recHML. *An annotation is* locally consistent *if for all states* $s \in P$*:*

- $\mathsf{ff} \notin A(s)$;
- *If* $\min X.\varphi \in A(s)$ *or* $\max X.\varphi \in A(s)$ *then* $\varphi \in A(s)$;
- *If* $X \in A(s)$ *then* $\min X.\varphi \in A(s)$ *if* X *is a least fixpoint variable and* $\max X.\varphi \in A(s)$ *otherwise;*
- *If* $\varphi \wedge \psi \in A(s)$ *then* $\varphi \in A(s)$ *and* $\psi \in A(s)$;
- *If* $\varphi \vee \psi \in A(s)$ *then* $\varphi \in A(s)$ *or* $\psi \in A(s)$;
- *If* $\langle a \rangle \varphi \in A(s)$ *then* $\varphi \in A(s')$ *for some* $s' \in P$, *such that* $s \xrightarrow{a} s'$;
- *If* $[a]\varphi \in A(s)$ *then* $\varphi \in A(s')$ *for all* $s' \in \textsc{Prc}$, *such that* $s \xrightarrow{a} s'$. ◀

▶ **Definition 42.** *For annotation* A, *an* annotated sequence *is a (finite or infinite) sequence* $\pi = (\varphi_0, s_0)(\varphi_1, s_1) \cdots$, *such that*

- *for each* i, $\varphi_i \in A(s_0)$;
- *for all* $i, i+1$ *that appear as indexes in* π, φ_i *is of the form* $\varphi_{i+1} \wedge \psi$, $\psi \wedge \varphi_{i+1}$, $\varphi_{i+1} \vee \psi$, $\psi \vee \varphi_{i+1}$, $[a]\varphi_{i+1}$, $\langle a \rangle \varphi_{i+1}$, $\min X.\varphi_{i+1}$, $\min X.\varphi_{i+1}$, *or* X, *where* $\varphi_{i+1} = \min X.\psi$ *or* $\min X.\psi$;
- *if* $\varphi_i = [a]\varphi_{i+1}$ *or* $\langle a \rangle \varphi_{i+1}$, *then* $s_i \xrightarrow{a} s_{i+1}$, *and otherwise* $s_i = s_{i+1}$; ◀

It is not hard to see that if two fixpoint formulas φ_1, φ_2 appear in an annotated sequence, then in the subsequence between (but including) the respective appearances of φ_1 and φ_2, there appears a fixpoint formula φ_3, such that φ_1 and φ_2 are subformulae of φ_3. Therefore, in every infinite annotated sequence there appears infinitely often a fixpoint formula ψ, such that all other fixpoint formulas that appear infinitely often are subformulae of ψ. Then, ψ is called the outermost fixpoint formula that appears infinitely often in the sequence.

▶ **Definition 43** (Consistent Annotation). *An annotation is* consistent *if it is both locally consistent and for every infinite annotated sequence, the outermost fixpoint formula that appears infinitely often in the sequence is a* max*-formula.* ◀

It is a standard result (see for example [14] for a more thorough discussion) that for a process p and a subformula φ of Ψ, we have that $p \in [\![\varphi]\!]$ if and only if there is a consistent annotation such that $\varphi \in A(p)$. We call this a consistent φ-annotation of p.

We observe that, because formulas are assumed to be guarded, every annotation on processes with no infinite traces is consistent if and only if it is locally consistent. The same is true if no min-fixpoints appear in the annotation.

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

▶ **Definition 44** (Consistent counter-annotation). *A counter-annotation* $C : P \to P(sf(\Psi))$ *is a labelling of* $P \subseteq \textsc{Prc}$ *with sets of subformulae of a formula* Ψ *of* recHML. *A counter-annotation is locally consistent if for all states* $s \in P$*:*

- $\mathsf{tt} \notin C(s)$;
- *If* $\min X.\varphi \in C(s)$ *or* $\max X.\varphi \in C(s)$ *then* $\varphi \in C(s)$;
- *If* $X \in C(s)$ *then* $\min X.\varphi \in C(s)$;
- *If* $\varphi \wedge \psi \in C(s)$ *then* $\varphi \in C(s)$ *or* $\psi \in C(s)$;

- *If $\varphi \vee \psi \in C(s)$ then $\varphi \in C(s)$ and $\psi \in C(s)$;*
- *If $\langle a \rangle \varphi \in C(s)$ then $\varphi \in C(s')$ for all $s' \in P_{RC}$, such that $s \xrightarrow{a} s'$;*
- *If $[a]\varphi \in C(s)$ then $\varphi \in C(s')$ for some $s' \in P$, such that $s \xrightarrow{a} s'$.*

Counter-annotated sequences are defined similarly to annotated sequences. A counter-annotation is consistent if it is both locally consistent and for every infinite annotated sequence of subformulae, the outermost fixpoint formula that appears infinitely often in the sequence is a μ-formula. ◀

Then, a process p violates a subformula φ of Ψ if and only if there is a consistent counter-annotation C, such that $\varphi \in C(p)$.

Eliminating existentials

Lemma 29. *For every closed disjunctive recHML formula Ψ, the formula $f_1(\Psi)$ has the same sHML consequences as Ψ.*

Proof. Observe that we can construct a consistent annotation for $f_1(\Psi)$ from a consistent annotation for Ψ, by simply replacing each ψ in the annotation by $f_1(\psi)$. Then, all conditions for a consistent annotation are satisfied, and therefore Ψ implies $f_1(\Psi)$.

Let $\theta \in$ sHML be a consequence of Ψ. We show that $f_1(\Psi)$ also implies θ

Assume otherwise: let p be a process such that $p \in [\![f_1(\Psi) \wedge \neg\theta]\!]$. Let A_1 be an annotation that witnesses $p \in [\![f_1(\Psi)]\!]$.

We know, by Thm. 22, that θ is monitorable, so there is a finite trace $t = a_1 a_2 \dots a_k$ of p, such that for every p', if t is a trace of p', then $p' \in [\![\neg\theta]\!]$. Let $p = p_0 \xrightarrow{a_1} p_1 \xrightarrow{a_2} \cdots \xrightarrow{a_k} p_k$ be states reachable from p while producing t, and let $q_0, q_1, \dots, q_k$ be processes with only the following transitions: $q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \cdots \xrightarrow{a_k} q_k$. From the above, we see that $q_0 \notin [\![\theta]\!]$. Furthermore, we can define an annotation A_2 on $\{q_0, q_1, \dots, q_k\}$, such that for all $i = 1, \dots, k$, $A_2(q_i) = A_1(p_i)$. It is not hard to see, exploiting the absence of existential modalities in $f_1(\Psi)$, that A_2 is a consistent annotation, witnessing that $q_0 \in [\![f_1(\Psi)]\!]$.

Let A_3 be a *minimal* consistent annotation witnessing that $q_0 \in [\![f_1(\Psi)]\!]$. Let us observe that by the definition of Ψ and f_1, all formulas in A_3 can be of the form tt, X, $\mathsf{min}\, X.\psi$, $\mathsf{max}\, X.\psi$, $\psi_1 \vee \psi_2$, or $\bigwedge_{a \in B} [a] \bigvee \mathcal{B}_a$. The last kind of formula we call a conjunction. We say that $\bigwedge_{a \in B} [a] \bigvee \mathcal{B}_a \in A_3(q_i)$ is maximal in $A_3(q_i)$ if it is not a conjunct in any other conjunction in $A_3(q_i)$. We now show that due to the disjunctive form of our formulas, a minimal annotation only has one maximal conjunction per state.

We also define recursively for two formulas $\psi_1, \psi_2 \in A_3(q_i)$ what is a path from ψ_1 to ψ_2: $\{\psi_1\}$ is a path from ψ_1 to ψ_1; and if F is a path from ψ_1 to ψ_2, then:

- if $\psi_1 \vee \psi_1' \in A_3(q_i)$, then $F \cup \{\psi_1 \vee \psi_1'\}$ is a path from $\psi_1 \vee \psi_1'$ to ψ_2;
- if $\psi_1' \vee \psi_1 \in A_3(q_i)$, then $F \cup \{\psi_1' \vee \psi_1\}$ is a path from $\psi_1' \vee \psi_1$ to ψ_2;
- if $\mathsf{max}\, X.\psi_1 \in A_3(q_i)$, then $F \cup \{\mathsf{max}\, X.\psi_1\}$ is a path from $\mathsf{max}\, X.\psi_1$ to ψ_2; and
- if $\mathsf{min}\, X.\psi_1 \in A_3(q_i)$, then $F \cup \{\mathsf{min}\, X.\psi_1\}$ is a path from $\mathsf{min}\, X.\psi_1$ to ψ_2.

Finally, for each $\bigwedge_{a \in B} [a] \bigvee \mathcal{B}_a \in A_3(q_i)$, we define the set of conjunctions that it subsumes, in the following sense:

$$ss\left(\bigwedge_{a \in B} [a] \bigvee \mathcal{B}_a \in A_3(q_i)\right) = \left\{ \bigwedge_{a \in C} [a] \bigvee \mathcal{B}_a \in A_3(q_i) \mid \emptyset \neq C \subseteq B \right\}.$$

We are now ready to prove the following claim on the minimal annotation A_3, which will allow us to focus on a single maximal conjunction per state:

Claim: for every i, if $\bigwedge_{a\in B}[a]\bigvee \mathcal{B}_a \in A_3(q_i)$ and $\bigwedge_{a\in B'}[a]\bigvee \mathcal{B}'_a \in A_3(q_i)$, are maximal in $A_3(q_i)$, then $B = B'$ and for all $a \in B$, $\mathcal{B}_a = \mathcal{B}'_a$ – in other words, there is at most one maximal conjunction in $A_3(q_i)$

We prove the claim by induction on i. *For the case where* $i = 0$, we first observe that $f_1(\Psi) \in A_3(q_0)$. Since our formulas are guarded and $f_1(\Psi)$ is closed, there is no path from $f_1(\Psi)$ to a variable X. Therefore, according to the conditions for local consistency, there must be a path F from $f_1(\Psi)$ to either tt or to a conjunction ψ_c. In the first case, we observe that there can be no conjunction in F (by the definition of a path), and substituting $A_3(q_0)$ by F results in a locally consistent annotation, and therefore, $A_3(q_0) = F$ as A_3 is minimal. In the second case, we observe that there can be no other conjunction in F (by the definition of a path), and that substituting $A_3(q_0)$ by $F \cup ss(\psi_c)$ (which is a subset of $A_3(q_0)$) results in a locally consistent annotation, and therefore, by minimality, $A_3(q_0) = F \cup ss(\psi_c)$. Therefore, in both cases, there is at most one maximal conjunction in $A_3(q_0)$.

We now tackle the case for $i > 0$. By the inductive hypothesis, there is at most one maximal conjunction $\bigwedge_{a\in B''}[a]\bigvee \mathcal{B}''_a \in A_3(q_{i-1})$. If there is none, or if $a_i \notin B''$, then $A_3(q_i) = \emptyset$ and we are done. Otherwise, let $\psi_1 = \bigvee \mathcal{B}''_{a_i}$. By the requirements of local consistency, $\psi_1 \in A_3(q_i)$, and there is a path F_1 from ψ_1 to tt, to a conjunction ψ_2, or to a variable X_1. We can handle the first two cases similarly for the case of $i = 0$. For the last case, from the requirements of local consistency, for some $k > 0$, we can construct k paths, F_j, $1 \le j \le k$, such that F_1 is as defined above, and for $1 < j < k$, F_j is a path from $\mathsf{max}\, X_{j-1}.\psi_j$ to X_j (with $X_j \neq X_{j'}$ for $j \neq j'$), and, due to the guardedness of our formulas and the finiteness of $A_3(q_i)$, F_k is a path from $\mathsf{max}\, X_{k-1}.\psi_k$ to ψ_k, where $\psi_k = \mathsf{tt}$ or ψ_k is a conjunction. Let $F = \bigcup_{j=1}^{k} F_i$. With the possible exception of ψ_k, there is no conjunction in F. In the case $\psi_k = \mathsf{tt}$, substituting $A_3(q_0)$ by F results in a locally consistent annotation, and therefore, $A_3(q_0) = F$. In the case ψ_k a conjunction, substituting $A_3(q_0)$ by $F \cup ss(\psi_2)$ results in a locally consistent annotation, and therefore, $A_3(q_0) = F \cup ss(\psi_2)$. Therefore, in both cases, there is at most one maximal conjunction in $A_3(q_0)$.

We can now use these maximal conjunctions (which all come from the elimination of existential modalities from the conjunctions of Ψ) to turn q_0 into a process that also satisfies Ψ, by adding successors that satisfy the consistency requirements of the eliminated existential modalities.

We have assumed that all subformulae of disjunctive formulas (except for ff) are satisfiable. Therefore, for every subformula $\langle a\rangle\psi$ of Ψ, we can fix a process $s_\psi \in [\![\psi]\!]$, and assume a consistent annotation A_4 that witnesses these facts. We now construct a process $r \in [\![\Psi]\!]$, such that t is a trace of r. For each q_i, $i = 0, \ldots, k$, we construct a process q'_i with exactly the following transitions: $q'_i \xrightarrow{a_{i+1}} q'_{i+1}$, if $i < k$, and $q'_i \xrightarrow{a} s_\psi$ for every $\psi \in \mathcal{B}_a$, for every $f_1((\bigwedge_{\psi\in\mathcal{B}_a}\langle a\rangle\psi) \wedge [a]\bigvee \mathcal{B}_a) \in A_3(q_i)$. We can now construct a consistent annotation A_5 to witness that $q'_i \in [\![\psi]\!]$, for every $f(\psi) \in A_3(q_i)$. For each subformula $(\bigwedge_{\psi\in\mathcal{B}_a}\langle a\rangle\psi) \wedge [a]\bigvee \mathcal{B}_a$ of Ψ and every $\psi \in \mathcal{B}_a$, $A_5(s_\psi) = A_4(s_\psi) \cup \{\bigvee \mathcal{B}_a\}$; for $i = 0, \ldots, k$, $A_5(q'_i) = \{\psi \in sf(\Psi) \mid f_1(\psi) \in A_3(q'_i)\}$, and for every other state s, $A_5(s) = A_4(s)$ if A_4 is defined on s. It is then not hard to see that all conditions for a consistent annotation are satisfied by A_5. Therefore, A_5 witnesses that $r \stackrel{\text{def}}{=} q'_0 \in [\![\Psi]\!]$. Furthermore, it is immediately evident that t is a trace of r, and therefore $r \notin [\![\theta]\!]$, and therefore θ cannot be a consequence of Ψ, contradicting our assumptions. This completes the proof of the lemma. ◀

▶ **Example 45.** The necessity of disjunctive form can be seen from the following example: $\psi = (\langle a\rangle[b]\mathsf{ff}) \wedge ([a]\langle b\rangle\mathsf{tt} \vee [a][c]\mathsf{ff})$. For $F = \{[b]\mathsf{ff} \wedge [c]\mathsf{ff}, [c]\mathsf{ff}\}$, the equivalent disjunctive formula is:

$$\bigwedge_{\varphi\in F} \langle a\rangle\varphi \wedge [a]\bigvee F.$$

In ψ, replacing existentials with tt would yield a formula itself equivalent to tt. However, from its disjunctive form we can extract its strongest sHML consequence $[a][c]\mathsf{ff}$ (rather than tt). ◀

Eliminating least fixpoints

Lemma 32. *For every closed formula Ψ of recHML without existentials, $f_2(\Psi)$ has the same sHML consequences as Ψ.*

Proof. First, observe that Ψ implies $f_2(\Psi)$: an annotation for Ψ is locally consistent, so by replacing all occurrences of min by max, we are certain to have no sequences with infinite occurrences of min-formulas, so we have a consistent annotation for $f_2(\Psi)$.

Now, let $\theta \in$ sHML, such that θ is not a consequence of $f_2(\Psi)$. We prove that θ is also not a consequence of Ψ, which completes the proof of the lemma. Since θ is not a consequence of $f_2(\Psi)$, there is a $p \in [\![f_2(\Psi) \wedge \neg\theta]\!]$. Similarly to the proof of Lem. 29, we know, by Thm. 22, that θ is monitorable, so we can construct a process q that has no infinite traces and satisfies $f_2(\Psi) \wedge \neg\theta$. Let A be a consistent annotation that witnesses the fact. From A, we can then construct an annotation A': $A'(s) = \{\psi \in sf(\Psi) \mid f_2(\psi) \in A(s)\}$, when $A(s)$ is defined. It is straightforward to see that A' is locally consistent, using the fact that A is locally consistent. It is also consistent, because q has no infinite traces. Therefore, A' witnesses that $q \in [\![\Psi]\!]$, which completes the proof. ◀

A.1 Eliminating disjunctions

Lemma 37. *Given a closed formula Ψ of recHML with neither* min *operators nor existentials, $f_3(\Psi)$ has the same sHML consequences as Ψ.*

Proof. We fix a tableau $\mathcal{T}(\Psi)$ and the corresponding labellings L and L' of its nodes, as defined in Def. 35.

We first show that $f_3(\Psi)$ is a consequence of Ψ, *i.e.*, $\neg f_3(\Psi)$ implies $\neg\Psi$. Let p be a process such that $p \notin [\![f_3(\Psi)]\!]$. Since $f_3(\Psi)$ is a sHML formula, and similarly to the proofs of Lems. 29 and 32, we can assume that process p has a single maximal trace t. Let $C : P \to \mathcal{P}(sf(f_3(\Psi)))$ be a counter-annotation that witnesses the fact that $p \notin [\![f_3(\Psi)]\!]$. Since p only has finite traces and $f_3(\Psi)$ is guarded, C has no infinite counter-annotated sequences. Therefore, ff appears somewhere in C. We now define C', a counter annotation for Ψ that is defined on the set $P \subseteq$ Prc of processes that are reachable from p by a (possibly empty) sequence of transitions:

$$C'(q) = \{\psi \in L(n) \mid n \text{ is a tableau node s.t. } L'(n) \in C(q)\},$$

for every $q \in P$. It is then, not hard to verify that C' is locally consistent, and therefore it is also consistent, thus witnessing that $p \notin [\![\Psi]\!]$.

We now show that if Ψ implies a formula $\theta \in$ sHML, then $f_3(\Psi)$ also implies θ. Assume that Ψ implies $\theta \in$ sHML. Let p be a process such that $p \notin [\![\theta]\!]$ – therefore, $p \notin [\![\Psi]\!]$. Since θ

is a sHML formula, as above, we can assume that process p has a single maximal trace t. Let C be a consistent counter-annotation that witnesses that $p \notin [\![\Psi]\!]$, defined over $P \subseteq \text{PRC}$. We now define C', a counter annotation for $f_3(\Psi)$ that is defined on $P' = \{q \in P \mid C(q) \neq \emptyset\}$:

$$C'(q) = \{L'(n) \mid n \text{ is a tableau node s.t. } L(n) \subseteq C(q)\},$$

for every $q \in P'$. Again, it is not hard to verify that C' is locally consistent – and since p has no infinite traces and our formulas are guarded, C' is also consistent. Therefore, $p \notin [\![f_3(\Psi)]\!]$, so we have showed that $f_3(\Psi)$ implies all sHML consequences of Ψ, which completes the proof. ◀