Journal of Logical and Algebraic Methods in Programming 111 (2020) 100515

Contents lists available at ScienceDirect

Journal of Logical and Algebraic Methods in Programming

www.elsevier.com/locate/jlamp

Determinizing monitors for HML with recursion ☆

Luca Aceto [a,b], Antonis Achilleos [b,*], Adrian Francalanza [c], Anna Ingólfsdóttir [b], Sævar Örn Kjartansson [b]

[a] Gran Sasso Science Institute, viale F. Crispi 7, 67100 L'Aquila, Italy
[b] School of Computer Science, Reykjavik University, Menntavegi 1, Reykjavik 101, Iceland
[c] Dept. of Computer Science, ICT, University of Malta, Msida, Malta

ARTICLE INFO

Article history:
Received 29 July 2019
Received in revised form 12 December 2019
Accepted 21 December 2019
Available online 31 December 2019

Keywords:
Monitorability
Runtime verification
Hennessy-Milner logic
Determinization
Complexity bounds

ABSTRACT

We examine the determinization of monitors for HML with recursion. We demonstrate that every monitor is equivalent to a deterministic one, which is at most doubly exponential in size with respect to the original monitor. When monitors are described as CCS-like processes, this doubly exponential bound is optimal. When (deterministic) monitors are described as finite automata (or as their labeled transition systems), then they can be exponentially more succinct than their CCS process form.

1. Introduction

Monitors are computational entities that observe the executions of other computing devices (hereafter referred to as *systems*, or, in more formal settings, as *processes*) with the aim of accruing system information [33,29], comparing system executions against behavioral specifications [23,24], or reacting to observed executions via adaptation or enforcement procedures [4,14,37]. Monitor descriptions can vary substantially, from pseudocode [25,21], to mathematical descriptions [51, 18,22], to executable code in a domain-specific or general-purpose language [19,41]. Since they are part of the trusted computing base, monitor descriptions are expected to be *"correct"*, and a prevalent correctness requirement is that they exhibit *deterministic behavior*.

Monitors are central for the field of Runtime Verification, where typically we use monitors to observe the trace produced during a run of a process. The monitor is expected to be able to reach a verdict after reading a finite part of the execution trace, if it can conclude that the monitored process violates (or, dually, satisfies) a certain specification property. Such specification properties are often expressed in an appropriate logical language, such as LTL [44,20,10,12,11], CTL, CTL* [17], and μHML [34,1,35,23]. We refer the interested readers to [9] for an extensive introduction to Runtime Verification and to [36] for a brief overview of that research field.

☆ This research was partially supported by the projects "TheoFoMon: Theoretical Foundations for Monitorability" (grant number: 163406-051) and "Epistemic Logic for Distributed Runtime Monitoring" (grant number: 184940-051) of the Icelandic Research Fund, and by the MIUR project PRIN 2017FTXR7S "IT-MaTTerS" (Methods and Tools for Trustworthy Smart Systems).

* Corresponding author.
E-mail addresses: luca.aceto@gssi.it, luca@ru.is (L. Aceto), antonios@ru.is (A. Achilleos), adrian.francalanza@um.edu.mt (A. Francalanza), annai@ru.is (A. Ingólfsdóttir).

https://doi.org/10.1016/j.jlamp.2019.100515
2352-2208/

In [23,24], Francalanza et al. studied the monitorability of properties expressed in full μHML interpreted over states in labeled transition systems [32]. They determined a maximal monitorable fragment of the logic in that branching-time setting, called ᴍHML, for which they introduced a compositional monitor-generating procedure as well as a converse, compositional formula-synthesis function from monitor descriptions. Intuitively, ᴍHML consists of the safety properties in μHML and their complements.

Since one of the goals of the work presented in the above-mentioned references was to identify exactly what properties can be monitored within the framework described therein, it was natural to choose a very expressive logic as a specification language. To our mind, μHML [1,35] is an excellent touchstone logic. Indeed, μHML is a variation on the propositional μ-calculus [34], tailored to the description of properties of states in labeled transition systems, which can express all the bisimulation-invariant properties that can be written in monadic second order logic [30] and specification logics such as LTL, CTL and CTL*. This makes the characterization of the properties that can be monitored at runtime given in [23,24] very general.

In the framework developed in [23,24], which is also used in [7] to identify an expressiveness hierarchy of monitorable fragments of μHML in linear-time settings, monitors are described using expressions in a variation on Milner's CCS [43]. The use of a language for specifying monitors naturally supports the definition of correct-by-construction, compositional monitor-synthesis procedures from formulae, as well as the compositional construction of formulae from monitors. The latter, formula-synthesis function plays a key role in showing the maximality of the monitorable fragments of μHML identified in [3,7,23,24]. To the best of our knowledge, these maximality results do not have any counterpart in the literature on runtime monitoring.

The monitors constructed from the synthesis procedure in [23,24] can be nondeterministic, depending on the μHML formula from which they were constructed. In the light of the importance of deterministic monitors in Runtime Verification, we would like to be able to determinize these monitors.

In this paper we tackle the problem of determinizing monitors for μHML in the framework of [23,24], where monitors are described using syntax close to the regular fragment of CCS processes [43], and we provide a detailed analysis of the succinctness of various alternative formalisms for describing such monitors. In particular, we demonstrate that every monitor can be transformed into an equivalent deterministic one, but the price can be a hefty one: there are monitors which require a doubly exponential blowup in size to determinize. Although we focus on the monitors employed in [23,24], our methods and results can be extended to other cases of similar monitors, when these monitors are described using syntax that is close to that of the regular fragment of CCS processes. Furthermore, we demonstrate that finite automata, as a specification language, can be exponentially more succinct than monitors as we define them, exponentially more succinct than the monitorable fragment of μHML, and doubly exponentially more succinct than the deterministic monitorable fragment of μHML.

Table 6 on page 29 summarizes the main complexity bounds we obtain in this article. We trust that those results may serve as a reference for researchers in the choice of a formalism for the description of monitors that, on the one hand, allows for compositional monitor- and formula-synthesis procedures and, on the other, keeps the representation of monitors reasonably small.

As mentioned earlier, the deterministic behavior of monitors is desirable as a correctness requirement and also due to efficiency concerns. A monitor is expected to not overly affect the observed system and thus each transition of the monitor should be performed as efficiently as possible. For each observed action of a system, the required transition of a monitor is given explicitly by a deterministic monitor, but for a nondeterministic monitor, we need to keep track of all its possible configurations, which can introduce a significant overhead to the monitored system.

On the other hand, as we argue below, nondeterminism arises naturally in some scenarios in the behavior of monitors. In those cases, the most natural monitors synthesized from the properties of interest are nondeterministic and might need to be determinized in order to increase the efficiency of the monitoring process. This observation motivates our study of monitor determinization and its complexity.

Empirical evidence for the nondeterministic behavior of monitors By most measures, monitoring is a relatively new software technique and thus not very widespread. Therefore, we substantiate our posit that nondeterministic monitors would occasionally occur (should the technique gain wider acceptance) by considering *testing* as a related pervasive technique. Despite intrinsic differences[1] tests share several core characteristics with many monitors: they rely on the execution of a system to infer correctness attributes, they often fall under the responsibility of quality assurance software teams and, more crucially, they are also expected to behave deterministically [53,38]. We contend that monitors are generally more complex artifacts than tests, which allows us to claim reliably that evidence of nondeterminism found in testing carries over to monitoring. When compared to tests, monitors typically employ more loop and branching constructs in their control structures since monitored runs last longer than test executions. Moreover, monitoring is generally regarded as background processing, and this expected passivity forces monitoring code to cater for various eventualities, aggregating system execution cases that would otherwise be treated by separate (simpler) tests driving the system.

[1] Tests are executed pre-deployment, and employ more mechanisms to direct the execution of the system under scrutiny *e.g.* mocking by inputting specific values.

Table 1
Dynamics of processes.

$$\textsc{Act}\ \frac{}{\alpha.p \xrightarrow{\alpha} p} \qquad \textsc{Rec}\ \frac{}{\texttt{rec}x.\ p \xrightarrow{\tau} p[\texttt{rec}x.\ p/x]}$$

$$\textsc{SelL}\ \frac{p \xrightarrow{\mu} p'}{p+q \xrightarrow{\mu} p'} \qquad \textsc{SelR}\ \frac{q \xrightarrow{\mu} q'}{p+q \xrightarrow{\mu} q'}$$

where $\alpha \in \textsc{Act}$ and $\mu \in \textsc{Act} \cup \{\tau\}$.

In testing, the impact of nondeterministic (*a.k.a. flaky*) tests on software development processes is substantial enough (for instance, as reported in [38], about 4.56% of test failures of the Test Anything Protocol system at Google are caused by flaky tests) to warrant the consideration of various academic studies [40,39,38]. These studies concede that flaky tests are hard to eradicate. Detection is hard, partly because tests are executed in conjunction with the system (where the source of nondeterministic behavior is harder to delineate), or because nondeterminism occurs sporadically, triggered only by specific system runs. Once nondeterminism has been detected, its causes may be even harder to diagnose: test runs are not immune to Heisenbugs [27] and problems associated with test dependence are not uncommon [53], all of which impacts on the capacity to fix the offending tests. In fact, it is common practice to live with nondeterministic tests by annotating them (*e.g.* `@RandomFail` in Jenkins), perhaps requiring reruns for these tests (*e.g.* `@Repeat` in Spring). Curiously, studies have shown that developers are often reluctant to remove detected flaky tests (for instance, by using the `@Ignore` tag in JUnit) because they may still reveal system defects (albeit inconsistently) [38,40].

Overview In Section 2, we give the necessary background for our investigation. We introduce μHML formulae and our framework for processes, monitors, and finite automata. In Section 3, we prove that all monitors for μHML can be determinized. We provide two methods for the determinization of monitors. One reduces the determinization of monitors to the determinization of regular CCS processes, as performed by Rabinovich in [47]. The other applies a procedure for transforming systems of equations, inspired from Rabinovich's methods, directly on MHML formulae to turn them into a deterministic form. Then, using the monitor synthesis procedure from [23], one can construct monitors which are immediately deterministic. In Section 4, we examine the cost of determinizing monitors more closely and we compare the size and behavior of monitors to the size (number of states in this case) and behavior of corresponding finite automata. We examine the simpler case of single-verdict monitors, namely monitors which are allowed to reach verdict `yes` or verdict `no`, but not both (or to halt without reaching an answer). Section 5 explains how to extend the methods and bounds of Section 4 from single-verdict monitors to the general case of monitors with multiple verdicts. The reader is encouraged to see Section 6, that draws further conclusions and presents an extensive summary of the technical results in this paper.

A previous version of this paper, presenting the main results from Section 4, has appeared as [2].

2. Background

We provide background on the main definitions and results for monitoring μHML formulae, as defined in [23], and present the conventions we use in this paper.

2.1. Basic definitions: monitoring μHML formulae on processes

We begin by presenting the calculus used to model a system, the logic used to reason about the systems and finally the monitors used to verify whether a system satisfies some specific formula in the logic.

2.1.1. The model

The processes whose properties we monitor (and on which we interpret the μHML formulae) are states of a labeled transition system (LTS). A labeled transition system is a triple

$$\langle \textsc{Proc}, (\textsc{Act} \cup \{\tau\}), \rightarrow \rangle$$

where Proc is a set of states or processes, Act is a set of observable actions, $\tau \notin \textsc{Act}$ is the distinguished silent action, and $\rightarrow \subseteq (\textsc{Proc} \times (\textsc{Act} \cup \{\tau\}) \times \textsc{Proc})$ is a transition relation. The syntax of the processes in Proc is defined by the following grammar:

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

where $\alpha \in \textsc{Act}$ and x comes from a countably infinite set of process variables. These processes are a standard variation on the regular fragment of *CCS* [43]; in [47], these processes are called μ-expressions. We simply call them processes in this paper. As usual, the construct $\texttt{rec}\ x.\ p$ binds the free occurrences of x in p. In what follows, unless stated otherwise, we focus on processes without occurrences of free variables. The substitution operator $p[q/x]$ is defined in the usual way. The transition relation $\rightarrow$ and thus the behavior of the processes is defined by the derivation rules in Table 1.

For each $p, p' \in \textsc{Proc}$ and $\alpha \in \textsc{Act}$, we use $p \stackrel{\alpha}{\Rightarrow} p'$ to mean that p can derive p' using a single α action and any number of τ actions, $p(\stackrel{\tau}{\rightarrow})^*.\ \stackrel{\alpha}{\rightarrow}.(\stackrel{\tau}{\rightarrow})^* p'$. For each $p, p' \in \textsc{Proc}$ and trace $t = \alpha_1.\alpha_2.\ldots.\alpha_r \in \textsc{Act}^*$, we use $p \stackrel{t}{\Rightarrow} p'$ to mean $p \stackrel{\alpha_1}{\Longrightarrow}.\stackrel{\alpha_2}{\Longrightarrow}\ldots\stackrel{\alpha_r}{\Longrightarrow} p'$ if t is non-empty and $p(\stackrel{\tau}{\rightarrow})^* p'$ if t is the empty trace ϵ.

2.1.2. The logic

We use μHML, the Hennessy-Milner logic with recursion, to describe properties of the processes.

Definition 1. The formulae of μHML are constructed using the following grammar:

$$\begin{array}{lll} \varphi, \psi \in \mu\text{HML} ::= \mathtt{tt} & | \ \mathtt{ff} \\ \quad | \ \varphi \wedge \psi & | \ \varphi \vee \psi \\ \quad | \ \langle\alpha\rangle\varphi & | \ [\alpha]\varphi \\ \quad | \ \min X.\ \varphi & | \ \max X.\ \varphi \\ \quad | \ X \end{array}$$

where X comes from a countably infinite set of logical variables $\textsc{LVar}$. ■

Formulae are evaluated in the context of a labeled transition system and an environment, $\rho : \textsc{LVar} \to 2^{\textsc{Proc}}$, which gives values to the logical variables in the formula. For an environment ρ, variable X, and set $S \subseteq \textsc{Proc}$, $\rho[X \mapsto S]$ is the environment which maps X to S and all $Y \neq X$ to $\rho(Y)$. The semantics for μHML formulae is given through a function $[\![\cdot]\!]$, which, given an environment ρ, maps each formula to a set of processes — namely the processes which satisfy the formula under the assumption that each $X \in \textsc{LVar}$ is satisfied by the processes in $\rho(X)$. $[\![\cdot]\!]$ is defined as follows:

$$\begin{aligned} [\![\mathtt{tt}, \rho]\!] &\stackrel{def}{=} \textsc{Proc} \quad \text{and} \quad [\![\mathtt{ff}, \rho]\!] \stackrel{def}{=} \emptyset \\ [\![\varphi_1 \wedge \varphi_2, \rho]\!] &\stackrel{def}{=} [\![\varphi_1, \rho]\!] \cap [\![\varphi_2, \rho]\!] \\ [\![\varphi_1 \vee \varphi_2, \rho]\!] &\stackrel{def}{=} [\![\varphi_1, \rho]\!] \cup [\![\varphi_2, \rho]\!] \\ [\![[\alpha]\varphi, \rho]\!] &\stackrel{def}{=} \left\{ p \mid \forall q.\ p \stackrel{\alpha}{\Rightarrow} q \text{ implies } q \in [\![\varphi, \rho]\!] \right\} \\ [\![\langle\alpha\rangle\varphi, \rho]\!] &\stackrel{def}{=} \left\{ p \mid \exists q.\ p \stackrel{\alpha}{\Rightarrow} q \text{ and } q \in [\![\varphi, \rho]\!] \right\} \\ [\![\max X.\ \varphi, \rho]\!] &\stackrel{def}{=} \bigcup \{ S \mid S \subseteq [\![\varphi, \rho[X \mapsto S]]\!] \} \\ [\![\min X.\ \varphi, \rho]\!] &\stackrel{def}{=} \bigcap \{ S \mid S \supseteq [\![\varphi, \rho[X \mapsto S]]\!] \} \\ [\![X, \rho]\!] &\stackrel{def}{=} \rho(X). \end{aligned}$$

A formula is closed when every occurrence of a variable X is in the scope of the recursive operator $\mathtt{rec}\ x$. Note that the environment, ρ, has no effect on the semantics of a closed formula. For a closed formula φ, we often drop the environment from the notation for $[\![\cdot]\!]$ and write $[\![\varphi]\!]$ instead of $[\![\varphi, \rho]\!]$. Henceforth we work only with closed formulae, unless stated otherwise. Formulae φ and ψ are (logically) equivalent, written $\varphi \equiv \psi$, if $[\![\varphi, \rho]\!] = [\![\psi, \rho]\!]$ for every environment ρ.

We focus on sHML, the safety fragment of μHML. Both sHML and its dual fragment, cHML (the co-safety fragment), are defined by the grammar:

$$\begin{array}{lllllll} \theta, \vartheta \in \text{sHML} ::= \mathtt{tt} & | \ \mathtt{ff} & | \ [\alpha]\theta & | \ \theta \wedge \vartheta & | \ \max X.\ \theta & | \ X \\ \pi, \varpi \in \text{cHML} ::= \mathtt{tt} & | \ \mathtt{ff} & | \ \langle\alpha\rangle\pi & | \ \pi \vee \varpi & | \ \min X.\ \pi & | \ X. \end{array}$$

In what follows, we write MHML for the collection of all the safety and co-safety formulae in μHML, that is, MHML = sHML ∪ cHML.

Example 1 *(a formula).* The formula $\varphi_e = \max\ X.\ [\alpha]([\alpha]\mathtt{ff} \wedge X) \in \text{sHML}$ will be used in several examples to follow. Notice that the following logical equivalences hold:

$$\begin{aligned} \varphi_e &\equiv [\alpha]([\alpha]\mathtt{ff} \wedge \max\ X.\ [\alpha]([\alpha]\mathtt{ff} \wedge X)) \\ &\equiv [\alpha]([\alpha]\mathtt{ff} \wedge [\alpha]([\alpha]\mathtt{ff} \wedge \max\ X.\ [\alpha]([\alpha]\mathtt{ff} \wedge X))) \\ &\equiv [\alpha][\alpha]\mathtt{ff}. \quad \blacksquare \end{aligned}$$

Table 2
Monitor dynamics.

$$\textsc{mAct}\ \frac{}{\alpha.m \xrightarrow{\alpha} m} \qquad \textsc{mRec}\ \frac{}{\texttt{rec}x.\ m \xrightarrow{\tau} m[\texttt{rec}x.\ m/x]}$$

$$\textsc{mSelL}\ \frac{m \xrightarrow{\mu} m'}{m+n \xrightarrow{\mu} m'} \qquad \textsc{mSelR}\ \frac{n \xrightarrow{\mu} n'}{m+n \xrightarrow{\mu} n'}$$

$$\textsc{mVerd}\ \frac{}{v \xrightarrow{\alpha} v}$$

where $\alpha \in \textsc{Act}$ and $\mu \in \textsc{Act} \cup \{\tau\}$.

Table 3
Monitored processes.

$$\textsc{iMon}\ \frac{p \xrightarrow{\alpha} p' \quad m \xrightarrow{\alpha} m'}{m \triangleleft p \xrightarrow{\alpha} m' \triangleleft p'} \qquad \textsc{iTer}\ \frac{p \xrightarrow{\alpha} p' \quad m \overset{\alpha}{\not\longrightarrow} \quad m \overset{\tau}{\not\longrightarrow}}{m \triangleleft p \xrightarrow{\alpha} \texttt{end} \triangleleft p'}$$

$$\textsc{iAsyP}\ \frac{p \xrightarrow{\tau} p'}{m \triangleleft p \xrightarrow{\tau} m \triangleleft p'} \qquad \textsc{iAsyM}\ \frac{m \xrightarrow{\tau} m'}{m \triangleleft p \xrightarrow{\tau} m' \triangleleft p}$$

2.1.3. Monitors

We now define the notion of a monitor. Just like for processes, we use the definitions given in [23,24] to which we refer our readers for motivation and further examples. Monitors are part of an LTS, much like processes.

Definition 2. The syntax of a monitor is identical to that of a process, with the exception that the `nil` process is replaced by verdicts. A verdict can be one of `yes`, `no` and `end`, which represent acceptance, rejection and termination respectively. A monitor is defined by the following grammar:

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

where x comes from a countably infinite set of monitor variables. The *submonitors* of m are its subterms. ■

The behavior of a monitor is defined by the derivation rules of Table 2, which are discussed at length in [23,24]. Here we limit ourselves to remarking that rule mVerd indicates that monitor verdicts are irrevocable. Rules mSelL and mSelR may be referred to collectively as mSel for convenience.

Remark 1. Note that τ actions in monitor behaviors are generated by rule mRec in Table 2, which uses such actions to unfold recursive definitions of monitors. This choice stems from [24] and we maintain it here since our main goal in this article is to prove complexity bounds that apply directly to the model of monitors studied in that reference. We remark, however, that all the results we prove in the remainder of this study hold also in the setting of τ-free monitors with the classic unfolding rule for recursively-defined monitors, namely:

$$\frac{m[\texttt{rec}x.\ m/x] \xrightarrow{\alpha} m'}{\texttt{rec}x.\ m \xrightarrow{\alpha} m'}.$$

For each $m, m' \in \textsc{Mon}$, $\alpha \in \textsc{Act}$ and trace $t \in \textsc{Act}^*$, the "weak transitions" $m \overset{\alpha}{\Rightarrow} m'$ and $m \overset{t}{\Rightarrow} m'$ are defined as for processes.

Example 2 *(a monitor).* Let $m_e = \texttt{rec}\ x.\ \alpha.(\alpha.\texttt{no} + x)$. Notice the similarities between m_e and φ_e. We will be using m_e and φ_e as a running example in what follows. ■

2.1.4. Monitored system

If a monitor $m \in \textsc{Mon}$ is monitoring a process $p \in \textsc{proc}$, then it must mirror every visible action p performs. If m cannot match an action performed by p and it cannot perform and internal action, then m becomes the inconclusive `end` verdict. We are only looking at the visible actions and so we allow m and p to perform transparent τ actions independently of each other.

Definition 3. A monitored system is a monitor $m \in \textsc{Mon}$ and a process $p \in \textsc{Proc}$ which are run side-by-side, denoted $m \triangleleft p$. The behavior of a monitored system is defined by the derivation rules in Table 3. ■

If a monitored system $m \triangleleft p$ can derive the `yes` verdict, we say that m accepts p, and similarly m rejects p if the monitored system can derive `no`.

Definition 4 *(Acceptance/rejection).* $\mathbf{acc}(m,p) \stackrel{def}{=} \exists t, p'.\ m \triangleleft p \stackrel{t}{\Rightarrow} \mathtt{yes} \triangleleft p'$ and $\mathbf{rej}(m,p) \stackrel{def}{=} \exists t, p'.\ m \triangleleft p \stackrel{t}{\Rightarrow} \mathtt{no} \triangleleft p'$. ■

Remark 2. Note that, according to the above definition, both $\mathbf{acc}(m,p)$ and $\mathbf{rej}(m,p)$ are possible for some monitor m and process p. As shown in [24, Theorem 2, page 109], monitors that may report more than one verdict are necessarily unsound, at least with respect to branching-time properties interpreted over processes. This is the reason why the work in that reference, on which we build, restricts itself to single-verdict monitors, as considered in what follows. (See Section 2.2 to follow for a short summary of the results from [24] on monitorability of branching-time properties of processes.) We remark, however, that, as shown in [7], multi-verdict monitors can be sound (and even 'complete') for some non-trivial properties in a linear-time setting. We refer the interested reader to the above-mentioned references and to [5] for further details on a variety of connections between monitor acceptance/rejection and whether processes or traces satisfy some property.

2.1.5. Finite automata

We now present a brief overview of Finite Automata Theory, which we use in Section 4. The interested reader can see [49] for further details.

A nondeterministic finite automaton (NFA) is a quintuple

$$A = (Q, \Sigma, \delta, q_0, F),$$

where $Q \neq \emptyset$ is a set of states, Σ is a set of symbols, called the alphabet — in our context, $\Sigma = \text{Act} -$, $\delta \subseteq Q \times \Sigma \times Q$ is a transition relation, $q_0 \in Q$ is the initial state, and $F \subseteq Q$ is the set of final or accepting states. Given a word $t \in \Sigma^*$, a run of A on $t = t_1 \cdots t_k$ is a sequence $q_0 q_1 \cdots q_k$, such that for $1 \leq i \leq k$, $(q_{i-1}, t_i, q_i) \in \delta$; the run is called accepting if $q_k \in F$. We say that A accepts t when A has an accepting run on t. We say that A accepts/recognizes a language $L \subseteq \Sigma^*$ when A accepts exactly the words in L; L is then unique and we call it $L(A)$. If δ is a function $\delta : Q \times \Sigma \to Q$ (depending on the situation, one may just demand that δ is a partial function), then A is a deterministic finite automaton (DFA). It is a classical result that for every NFA with n states, there is an equivalent DFA (i.e. a DFA which recognizes the same language) with at most 2^n states — furthermore, this upper bound is optimal.

Theorem 1 *([46]). If A is an NFA of n states, then there is a DFA of at most 2^n states which recognizes $L(A)$.*

The way to construct such a DFA is through the classical subset construction. For more details, see [49], or another introductory text for automata theory or theoretical computer science.

Are monitors automata? The reader may wonder at this point whether a monitor is simply an NFA in disguise. Certainly, the LTS of monitor resembles an NFA, although there are differences. A monitor — at least in this paper — is identified with its syntactic form, as defined above, although, as Section 4 demonstrates, it pays to be a little more relaxed on this constraint. Monitors can also reach both a `yes` and a `no` verdict — and sometimes both for the same trace. Again, this is often undesirable. Another difference is that once a monitor reaches a verdict, there is no way to change its decision by seeing more of the trace: verdicts are irrevocable for monitors. For more on the relationship between monitors and automata, we refer to Section 4.

2.2. Previous results

The results from [23,24] on which we build in this study are based on the following notion of monitorability for formulae in μHML.

Definition 5 *(Monitorable formula/logic).* Let $\varphi \in \mu\text{HML}$ and let m be a monitor. We say that

- m monitors φ for violations when, for each process p, $\mathbf{rej}(m,p)$ if and only if $p \notin [\![\varphi]\!]$;
- m monitors φ for satisfactions when, for each process p, $\mathbf{acc}(m,p)$ if and only if $p \in [\![\varphi]\!]$;
- m monitors for φ when it monitors φ either for violations or for satisfactions; and
- φ is monitorable when there is some monitor m that monitors for it.

Let $\mathcal{L} \subseteq \mu\text{HML}$. We say that $\mathcal{L}$ is monitorable when each $\varphi \in \mathcal{L}$ is monitorable. $\mathcal{L}$ is a maximally expressive monitorable fragment of μHML if each monitorable formula in μHML is logically equivalent to some formula in $\mathcal{L}$. ■

The main result from [23,24] is to show that MHML, the subset of μHML that consists of the safety and co-safety fragments of that logic, is monitorable and that it is maximally expressive. In particular, SHML is a maximally expressive fragment of μHML whose formulae can be monitored for violations and CHML is a maximally expressive fragment of that logic whose formulae can be monitored for satisfactions.

In the remainder of this section, we focus on surveying the monitorability result for sHML; the case of cHML is dual. The interested reader can see [23,24] for more details, examples and motivation.

In order to prove that sHML is monitorable, in [23,24] Francalanza, Aceto, and Ingólfsdóttir define a monitor synthesis function, $(\!|-|\!)$, which maps formulae to monitors, and show that, for each $\varphi \in$ sHML, $(\!|\varphi|\!)$ monitors φ for violations. This function, which stems from [24, Definition 7], is used in the proofs in Section 3 and so we give the definition here. We remark that the definition of the monitor $(\!|\varphi|\!)$ assumes a bijection between logical variables and monitor variables, which we leave implicit.

Definition 6 *(Monitor synthesis).*

$$(\!|\mathtt{tt}|\!) \stackrel{def}{=} \mathtt{yes} \qquad (\!|\mathtt{ff}|\!) \stackrel{def}{=} \mathtt{no} \qquad (\!|X|\!) \stackrel{def}{=} x$$

$$(\!|[\alpha]\psi|\!) \stackrel{def}{=} \begin{cases} \alpha.(\!|\psi|\!) & \text{if } (\!|\psi|\!) \neq \mathtt{yes} \\ \mathtt{yes} & \text{otherwise} \end{cases}$$

$$(\!|\psi_1 \wedge \psi_2|\!) \stackrel{def}{=} \begin{cases} (\!|\psi_1|\!) & \text{if } (\!|\psi_2|\!) = \mathtt{yes} \\ (\!|\psi_2|\!) & \text{if } (\!|\psi_1|\!) = \mathtt{yes} \\ (\!|\psi_1|\!) + (\!|\psi_2|\!) & \text{otherwise} \end{cases}$$

$$(\!|\max X.\ \psi|\!) \stackrel{def}{=} \begin{cases} \mathtt{rec}x.\ (\!|\psi|\!) & \text{if } (\!|\psi|\!) \neq \mathtt{yes} \\ \mathtt{yes} & \text{otherwise} \end{cases} \blacksquare$$

Remark 3. The monitor-synthesis function over sHML we presented in the previous definition is the restriction to that fragment of the function given in [24, Definition 7] over the whole of mHML. When synthesizing monitors for sHML formulae, it would be possible to replace the clause

$$(\!|\mathtt{tt}|\!) \stackrel{def}{=} \mathtt{yes}$$

with

$$(\!|\mathtt{tt}|\!) \stackrel{def}{=} \mathtt{end}$$

and modify the construction accordingly. We decided not to do so in this paper since we want our results to apply directly to the monitoring setting studied in the above-mentioned references.

Theorem 2 *(Monitorability, [23,24]). For each $\varphi \in$ sHML, $(\!|\varphi|\!)$ monitors φ for violations.*

Example 3. Notice that $(\!|\varphi_e|\!) = m_e$. On the other hand, we also know that for $\varphi'_e = [\alpha][\alpha]\mathtt{ff}$, $\varphi_e \equiv \varphi'_e$. Therefore, m_e and $m'_e = (\!|\varphi'_e|\!) = \alpha.\alpha.\mathtt{no}$ monitor for the same formula. $\blacksquare$

Remark 4. As shown in [24], for each formula $\varphi \in$ cHML, one can synthesize a monitor m that monitors φ for satisfactions.

Remark 5. The notion of monitorable formula was first defined by Pnueli and Zaks in their seminal paper [45]. Other proposals for monitorability may be found in, for instance, the references [12,22,52]. Comparing the notion of monitorability studied by Francalanza et al. in [23,24] with those presented in the literature is a very interesting and non-trivial research endeavor. To begin with, the notion of monitorability considered in [23,24] is given for a branching-time interpretation of mHML, whereas all the other notions are studied in a linear-time setting. Hence, a proper comparison requires a study of the notion of monitorability by Francalanza et al. in a linear-time setting. The recent article [7] is entirely devoted to this topic and also connects the branching- and linear-time notions of monitorability by Francalanza et al. within a unified and principled framework. The work in [7] paves the way to a systematic comparison of various notions of monitorability, which is presented in [5].

2.3. Determinism, verdicts, and the choices that we make

The purpose of this paper is to examine the determinization of monitors, which is the process of constructing a deterministic monitor from an equivalent nondeterministic one. Therefore, we must establish what a deterministic monitor is and what it means that two monitors are equivalent.

The papers [23,24] present several examples of nondeterministic monitors. For example, $m_c = \alpha.\mathtt{yes} + \alpha.\mathtt{no}$ is nondeterministic. This monitor can reach a positive *and* a negative verdict for each trace which starts with α, but has no opinion on any other trace. There are two ways to avoid this situation. One is to make sure that only different actions can transition to different monitors, as in $\alpha.\mathtt{yes} + \beta.\mathtt{no}$, thus removing a nondeterministic choice; the other is to consider single-verdict monitors (see Section 2.3.3) that can reach only one verdict, like $\alpha.\mathtt{yes} + \alpha.m$. We explore both approaches.

2.3.1. Conventions and definitions

We will call two monitors, m and m' equivalent and write $m \sim m'$ if for every non-empty trace t and verdict $v \in \{\texttt{yes}, \texttt{no}\}$, $m \stackrel{t}{\Rightarrow} v$ iff $m' \stackrel{t}{\Rightarrow} v$. Given a monitor, the set of processes it accepts, and thus the set of formulae it monitors, is completely determined by the traces it accepts and rejects (see [23, Proposition 1]). Therefore, we compare two monitors with respect to relation $\sim$. We assume that the set of actions, Act, is finite and, for the complexity results we present, of constant size.

We call derivations of the form $m \Rightarrow m$ trivial. If $t = t_1 t_2 \in \text{Act}^*$, then t_1 is an initial subtrace or prefix of t and we write $t_1 \sqsubseteq t$. We define a sum of m as follows: m is a sum of m and if r is a sum of m, then so are $r + r'$ and $r' + r$. We say that a sum s of m is acting as m in a derivation $d : s \stackrel{t}{\Rightarrow} r$ if in the same derivation we can replace s by m without consequence (i.e. the first step uses rule mSlet to use a derivation starting from m).

The size $|m|$ of a monitor m is the size of its syntactic description, as given in Section 2.1, defined recursively: $|x| = |v| = 1$; $|\alpha.m| = |m| + 1$; $|m + m'| = |m| + |m'| + 1$; and $|\texttt{rec}\ x.\ m| = |m| + 1$. Notice that $|m|$ also coincides with the total number of submonitor occurrences — namely, symbols in m. We also define the height of a monitor, $h(m)$, in the following way: $h(v) = h(x) = 1$; $h(m + m') = \max\{h(m), h(m')\}$; $h(\alpha.m) = h(m) + 1$; $h(\texttt{rec}\ x.\ m) = h(m)$.

In what follows, we consider monitors modulo the equivalence $v + v \sim v$.

The following lemma demonstrates that all instances of $m + v$ can be replaced by $m + \sum_{\alpha \in \text{Act}} \alpha.v$.

Lemma 1. *Monitor $m + v$ is equivalent to $m + \sum_{\alpha \in \text{Act}} \alpha.v$.*

Proof. Let $\mu \in \text{Act} \cup \{\tau\}$ and m' a monitor. Then, $m + v \xrightarrow{\mu} m'$ iff $m \xrightarrow{\mu} m'$ or $v \xrightarrow{\mu} m'$. But $v \xrightarrow{\mu} m'$ exactly when $m' = v$ and $\mu \in \text{Act}$, therefore $v \xrightarrow{\mu} m'$ exactly when $\sum_{\alpha \in \text{Act}} \alpha.v \xrightarrow{\mu} m'$. In conclusion, $m + v \xrightarrow{\mu} m'$ iff $m + \sum_{\alpha \in \text{Act}} \alpha.v \xrightarrow{\mu} m'$. □

We also assume that there is no (sub)monitor of the form $\texttt{rec}\ x.\ v$, where v is a verdict; otherwise, these can be replaced by simply v. Notice then that there is no τ-transition to a verdict. Furthermore, we assume that a verdict appears in all monitors we consider — otherwise the monitor does not accept/reject anything and is equivalent to $\texttt{end}$. We say that a monitor m accepts/recognizes a language (set of traces) $L \subseteq \text{Act}^*$ when for every trace $t \in \text{Act}^*$, $t \in L$ iff $m \stackrel{t}{\Rightarrow} \texttt{yes}$.

2.3.2. Determinism

For the purposes of [23], deterministic monitor behavior need only concern itself with the definite verdicts that can be reached after observing a particular trace t. Stated otherwise, we can say that a monitor m behaves deterministically whenever it transitions to verdict-equivalent monitors for every trace t. For instance, another case of a nondeterministic monitor is $m = \alpha.\alpha.\texttt{yes} + \alpha.\beta.\texttt{yes}$, because when this monitor reads an α, it has to make a choice and transition to either $\alpha.\texttt{yes}$ or $\beta.\texttt{yes}$ which are *not* equivalent, $\alpha.\texttt{yes} \not\sim \beta.\texttt{yes}$. A deterministic monitor that is equivalent to m is $\alpha.(\alpha.\texttt{yes} + \beta.\texttt{yes})$.

For Turing machines, algorithms, and finite automata, determinism means that from every state (in our case, monitor) and input symbol (in our case, action), there is a unique transition to follow. In the case of monitors, we can transition either through an action from Act, but also through a τ-action, which can occur without reading from a trace. In the context of finite automata, these actions would perhaps correspond to ϵ-transitions, which are eliminated from deterministic automata. We cannot eliminate τ-transitions from deterministic monitors, because we need to be able to activate the recursive operators. What we can do is to make sure that there is never a choice between a τ-transition and any other transition.

In other words, in order for a monitor to run deterministically, it cannot contain a nondeterministic choice between two sub-monitors reached by the same action, $m \xrightarrow{\alpha} n_1$ and $m \xrightarrow{\alpha} n_2$, or a nondeterministic choice between performing an action or performing the transparent action, $m \xrightarrow{\alpha} n_1$ and $m \xrightarrow{\tau} n_2$. It must also be impossible to derive these choices using the derivation rules above. As we can see from Table 2, such choices can only be introduced by sums — that is, monitors of the form $m_1 + m_2$; we can therefore attain the required behavior via syntactic constraints. Note that a sum $m_1 + m_2 + \cdots + m_k$ will also be written as $\sum_{i=1}^{k} m_i$.

Definition 7. A monitor m is syntactically deterministic iff every sum of at least two summands which appears in m is of the form $\sum_{\alpha \in A} \alpha.m_\alpha$, where $A \subseteq \text{Act}$. ■

Note that under this definition, from every syntactically deterministic monitor m and action $\alpha \in \text{Act}$, if $m \stackrel{\alpha}{\Rightarrow} m'$, then all derivations $m \stackrel{\alpha}{\Rightarrow}$ begin with a unique transition. As we will see in the following sections, this set of monitors is in fact maximally expressive, meaning that each monitorable formula can be monitored by a syntactically deterministic monitor. Lemma 2 demonstrates that the syntactic determinism of Definition 7 ensures that such monitors will always arrive at the same verdict for a given trace — Lemma 2 appears in Section 4 as Corollary 15, a direct consequence of Lemma 26.

Lemma 2. *If m is syntactically deterministic, $m \stackrel{t}{\Rightarrow} n$, and $m \stackrel{t}{\Rightarrow} n'$, then $n \sim n'$.* □

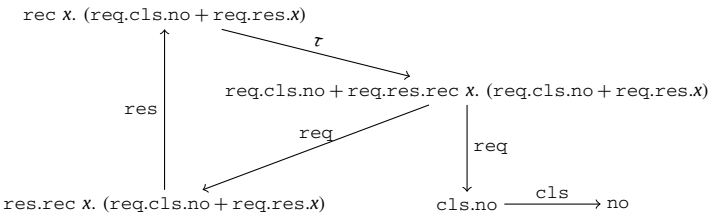

Fig. 1. The LTS of monitor m.

Thus, from now on, we simply refer to syntactically deterministic monitors as deterministic monitors.

Example 4. Notice that monitor m_e is not deterministic, but it has an equivalent deterministic monitor, which is m'_e. ■

Example 5 *(a simple server [23]).* Assume that we have a very simple web server p that alternates between accepting a request from a client, `req`, and sending a response back, `res`, until the server terminates, `cls`. We want to verify that the server cannot terminate while in the middle of serving a client (after executing `req` but before executing `res`). We can encode this property in the following sHML formula

$$\varphi = \max X.\ ([\texttt{req}][\texttt{cls}]\texttt{ff} \wedge [\texttt{req}][\texttt{res}]X).$$

Using the monitor synthesis function, we can then define a monitor that monitors φ for violations thus:

$$m = \texttt{rec}x.\ (\texttt{req.cls.no} + \texttt{req.res}.x).$$

The LTS of monitor m can be seen in Fig. 1. Since m contains a choice between `req.cls.ff` and `req.res.`x, it is nondeterministic. However it is possible to define a deterministic monitor that monitors for φ. For example:

$$m' = \texttt{req.(res.rec}x.\ \texttt{req.(res.}x + \texttt{cls.no)} + \texttt{cls.no)} \quad ■$$

Example 6. Consider the monitor $m = \texttt{rec}\ x.\ (0.x + 1.x + 1.2.\texttt{yes})$. It accepts process states that can produce traces from the language $(0 + 1)^*12(0 + 1 + 2)^*$, that is, traces (words) in which the action 2 appears at least once and the action preceding the first 2 action is a 1. An equivalent deterministic monitor is

$$m' = \texttt{rec}\ y.\ (0.y + 1.\texttt{rec}\ x.\ (0.y + 1.x + 2.\texttt{yes}))$$

Notice that the size of the deterministic monitor m' is greater than that of its original non-deterministic counterpart m. In fact, $|m| = 10$ and $|m'| = 14$. ■

In general, given a monitor produced by the monitor synthesis function, we can define a deterministic monitor that is equivalent to it.

Theorem 3. *For each formula $\varphi \in$ MHML, there exists a deterministic monitor, $m \in$ MON, such that m monitors for φ.*

Section 3 is devoted to providing two proofs for this theorem for formulae in sHML. (The proof for cHML is dual and is therefore omitted.) The first one is presented in Subsection 3.1 and shows that for each valid monitor, there exists a deterministic monitor that is equivalent to it. The second proof is presented in Subsection 3.2 and shows that there is a subset of sHML for which the monitor synthesis function will always produce a deterministic monitor and that this subset is a maximally expressive subset of sHML.

2.3.3. Multiple verdicts

As Francalanza et al. demonstrated in [23], to monitor for formulae of μHML, it is enough (and makes more sense) to consider single-verdict monitors. These are monitors, which can use verdict `yes` or `no`, but not both. As we are interested in determinizing monitors to use them for monitoring MHML properties, confining ourselves to single-verdict monitors is reasonable. The constructions of Section 3 are independent of such a choice and are presented for all kinds of monitors, but in Section 4 it is more convenient to consider only single-verdict monitors. Of course, monitor determinization is an interesting problem in its own right and monitors may be useful in other situations besides just monitoring for MHML. We still confine ourselves to determinizing single-verdict monitors, but we present a straightforward approach for dealing with monitors which use both verdicts in Section 5. In Section 4, whenever we say monitor, we will mean single-verdict monitor.

Specifically, in Section 4, we assume that the verdict is `yes` (and `end`, which is often omitted), because we compare monitors to automata and a monitor reaching a `yes` verdict on a trace intuitively corresponds to an automaton accepting the trace.

3. Rewriting methods for determinization

In this section we present two methods for constructing a deterministic monitor. The first method, presented in subsection 3.1, uses a result by Rabinovich [47] for the determinization of processes. The second method, presented in subsection 3.2, uses methods similar to Rabinovich's directly on formulae of sHML to transform them to a deterministic form, so that when we apply the monitor synthesis function from [23] (Definition 6 of Section 2.1 in this paper), the result is a deterministic monitor for the original formula.

3.1. Monitor rewriting

We show that, for each monitor, there exists an equivalent monitor that is deterministic. Thus, given a formula $\varphi \in$ MHML, we can apply the monitor synthesis function to produce a monitor for φ and subsequently rewrite the monitor so that it runs deterministically.

Definition 8. For a process p, $Trace(p) = \{t \in \text{Act}^* \mid \exists q.\ p \stackrel{t}{\Rightarrow} q\}$. We call two processes p, q trace equivalent when $Trace(p) = Trace(q)$. ■

We use a result given by Rabinovich in [47], which states that each process is equivalent to a deterministic one, with respect to trace equivalence — Rabinovich's definition of determinism for processes coincides with ours when a process has no free variables.

Let $D : \text{Proc} \to \text{Proc}$ be any function mapping each process $p \in \text{Proc}$ (as defined in Subsection 2.1) to a trace equivalent deterministic process. Such a mapping exists, as shown by Rabinovich in [47].

In order to apply the mapping D to the monitors, we must define a way to encode them as processes. We do this by replacing each verdict v with a process $v.\texttt{nil}$, where v is treated as an action.

Definition 9. We define a mapping $\pi : \text{Mon} \to \text{Proc}$ as follows:

$$\begin{aligned}
\pi(\alpha.m) &= \alpha.\pi(m)\\
\pi(m+n) &= \pi(m) + \pi(n)\\
\pi(\texttt{rec}x.\ m) &= \texttt{rec}x.\ \pi(m)\\
\pi(v) &= v.\texttt{nil}\\
\pi(x) &= x
\end{aligned}$$

We define an "inverse" of π, which we call π^{-1}:

$$\begin{aligned}
\pi^{-1}(s) &= v \quad \text{when } s \text{ is a sum of some } v.p\\
\pi^{-1}(\alpha.p) &= \alpha.\pi^{-1}(p)\\
\pi^{-1}(p+q) &= \pi^{-1}(p) + \pi^{-1}(q)\\
\pi^{-1}(\texttt{rec}x.\ p) &= \texttt{rec}x.\ \pi^{-1}(p)\\
\pi^{-1}(x) &= x \quad ■
\end{aligned}$$

Note that π maps monitors to processes where each verdict action v must be followed by the `nil` process and the `nil` process must be prefixed by a verdict action.

We want to use π on a nondeterministic monitor m to construct a process $\pi(m)$; then, use Rabinovich's D on $\pi(m)$ to determinize the process; then, use the inverse π^{-1} on $D(\pi(m))$ to end up with a deterministic monitor which is equivalent to m. So, if two monitor encodings are trace equivalent, we want the monitors they encode to be equivalent. Remember that we have assumed that all sums of verdict v are simply v (see Lemma 1).

Lemma 3. *Let m be a monitor and $p = \pi(m)$. Then, $m \stackrel{t}{\Rightarrow} v$ if and only if there is some $t' \sqsubseteq t$, such that $p \stackrel{t'}{\Rightarrow} v.\texttt{nil}$.*

Proof. Straightforward induction on the length of the derivation $p \stackrel{t'}{\Rightarrow} v.\texttt{nil}$ for the "if" direction and on the length of the derivation $m \stackrel{t}{\Rightarrow} v$ for the "only if" direction. The definition of π is used in both inductions. □

Lemma 4. *Let $m = \pi^{-1}(p)$. Then, for each $t \in \text{Act}^*$, there are some $t' \sqsubseteq t$ and a sum s of some $v.r$, such that $p \stackrel{t'}{\Rightarrow} s$, if and only if $m \stackrel{t}{\Rightarrow} v$.*

Proof. Like for Lemma 3, by induction on the length of the derivations. □

Lemma 5. *Let m and $n = \pi^{-1}(p)$ be monitors such that $Trace(\pi(m)) = Trace(p)$. Then $m \sim n$.*

Proof. Assume that for a trace t and a verdict v, we have $m \stackrel{t}{\Rightarrow} v$. By Lemma 3, for some $t' \sqsubseteq t$, $\pi(m) \stackrel{t'}{\Rightarrow} v.\texttt{nil}$ and so $t'.v \in Trace(\pi(m))$. Since $\pi(m)$ and p are trace equivalent, we have $p \stackrel{t'v}{\Longrightarrow} r$ for some r; therefore, $p \stackrel{t'}{\Rightarrow} s$, where s is a sum of some $v.r$, and so, by Lemma 4, $n \stackrel{t}{\Rightarrow} v$.

If $n \stackrel{t}{\Rightarrow} v$ for some verdict v and trace t, then by Lemma 4, $p \stackrel{t'}{\Rightarrow} s$, where s is a sum of $v.q$ and $t' \sqsubseteq t$, so $p \stackrel{t'v}{\Longrightarrow} q$. Then, $\pi(m) \stackrel{t'v}{\Longrightarrow} q'$, so $q' = \texttt{nil}$. Since all sums of v are v in m, it is also the case that all sums of $v.\texttt{nil}$ are $v.\texttt{nil}$ in $\pi(m)$; by Lemma 3 and rule mVerd in Table 2, $m \stackrel{t}{\Rightarrow} v$. □

Theorem 4 *(Monitor rewriting). For each monitor $m \in \text{Mon}$ there exists a deterministic monitor $n \sim m$.*

Proof. We define a new monitor $n = \pi^{-1} \circ D \circ \pi(m)$. By Lemma 5, m and n are equivalent. Let $n = \pi^{-1}(p)$. Monitor n is deterministic. To prove this, let s be a sum in n. We know that $s = \pi^{-1}(p')$, where p' appears in p, which is deterministic. Then, either p' is a sum of a $v.q$, so $s = v$, or $p' = \sum_{\alpha \in A} \alpha.p'_\alpha$ and $s = \sum_{\alpha \in A} \alpha.m_\alpha$. □

Example 7. To determinize $m_e = \texttt{rec}\ x.\ \alpha.(\alpha.\texttt{no} + x)$, we convert it to $p = \pi(m_e) = \texttt{rec}\ x.\ \alpha.(\alpha.[no].\texttt{nil} + x)$. Then, we use Rabinovich's construction to determinize p into (for example) $p' = \alpha.\alpha.[no].\texttt{nil}$. We can now see that $\pi^{-1}(p') = m'_e$. ■

3.2. Formula rewriting

In this second approach, we show that each formula $\varphi \in$ sHML is equivalent to some formula in deterministic form which will yield a deterministic monitor if we apply the monitor synthesis function to it. We focus on formulae in sHML but the proof for cHML is completely analogous. We work through an equivalent representation of formulae as systems of equations, as this makes the steps in our constructions easier to define and follow. The reader can also see [47], where Rabinovich uses very similar constructions to determinize processes. For the sake of readability, most of the proofs of the technical results in this section have been placed in Appendix A.

3.2.1. Systems of equations

We give the necessary definitions and facts about systems of equations for sHML. These definitions and lemmata are simplified versions of more general constructions. We state necessary lemmata without further explanation, but the reader can see an appropriate source on fixed points and the μ-calculus (for example, [8]).

Given tuples of sets, $a = (a_1, \ldots, a_k)$ and $b = (b_1, \ldots, b_l)$, we use the notation $a \cdot b$ to mean $(a_1, \ldots, a_k, b_1, \ldots, b_l)$. We abuse notation and extend the operations $\bigcup$ and $\bigcap$ to tuples of sets, so that $\bigcup(a, b) = a \cup b$ and if $a \cdot B \in S^{k+2}$, $\bigcup(a \cdot B) = a \cup \bigcup B$; similarly for $\bigcap$. Also, for tuples of sets, $a = (a_1, \ldots, a_k)$ and $b = (b_1, \ldots, b_k)$, $a \subseteq b$ iff for all $1 \leq i \leq k$, $a_i \subseteq b_i$. For an environment ρ, a tuple of distinct variables $\mathbb{X} = (X_1, \ldots, X_k)$ and a tuple of sets $\mathbb{S} = (S_1, \ldots, S_k)$, where $k > 1$, we define

$$\rho[X_1 \mapsto S_1, X_2 \mapsto S_2, \ldots X_k \mapsto S_k] = (\rho[X_1 \mapsto S_1])[X_2 \mapsto S_2, \ldots X_k \mapsto S_k]$$

and

$$\rho[\mathbb{X} \mapsto \mathbb{S}] = \rho[X_1 \mapsto S_1, X_2 \mapsto S_2, \ldots X_k \mapsto S_k].$$

Finally, for an environment ρ and formulae $\varphi_1, \ldots, \varphi_k$,

$$\left[\!\!\left[\bigtimes_{i=1}^{n} \varphi_i, \rho \right]\!\!\right] = \bigtimes_{i=1}^{n} [\![\varphi_i, \rho]\!],$$

where $\bigtimes_{i=1}^{n} S_i$ is the n-ary cartesian product $S_1 \times S_2 \times \cdots \times S_n$.

Definition 10. A system of equations is a triple $SYS = (Eq, X, \mathcal{Y})$ where X is called the principal variable in SYS, $\mathcal{Y}$ is a finite set of variables and Eq is an n-tuple of equations:

$$X_1 = F_1,$$
$$X_2 = F_2,$$
$$\vdots$$
$$X_n = F_n,$$

where for $1 \leq i < j \leq n$, X_i is different from X_j, F_i is an expression in sHML which can contain variables in $\mathcal{Y} \cup \{X_1, X_2, \ldots, X_n\}$ and there is some $1 \leq i \leq n$, such that $X = X_i$. $\mathcal{Y}$ is called the set of free variables of *SYS* and is disjoint from $\{X_1, X_2, \ldots, X_n\}$. ■

As we see further below, a system of equations can alternatively be understood as a simultaneous fixed point, but we mostly use the following recursive definition to provide semantics, where $Sol(SYS, \rho) = (S_1(\rho), \ldots, S_n(\rho))$ is an n-tuple of sets of processes from a labeled transition system, giving solutions to every variable X_i of the equation system, given an environment ρ. We use the notations

$$\rho[SYS] \stackrel{def}{=} \overline{\rho}^{SYS} \stackrel{def}{=} \rho\left[\left(\bigtimes_{i=1}^{n} X_i\right) \mapsto Sol(SYS, \rho)\right],$$

depending on which is clearer in each situation. For $n = 1$, let

$$Sol(SYS, \rho) = [\![\max X_1.\ F_1, \rho]\!].$$

Let *SYS*′ be a system *SYS* after adding a new first equation, $X = F$ and removing X from $\mathcal{Y}$; then, for $Sol(SYS, \rho) = (S_1(\rho), \ldots, S_n(\rho))$, let for every environment ρ,

$$S_0(\rho) = \bigcup \left\{ S \mid S \subseteq \left[\!\left[F, \overline{\rho[X \mapsto S]}^{SYS} \right]\!\right] \right\}$$

and

$$Sol(SYS', \rho) = (S_0(\rho), S_1(\rho\,[X \mapsto S_0(\rho)]), \ldots, S_n(\rho\,[X \mapsto S_0(\rho)])\,.$$

If the primary variable of a system of equations, *SYS*, is X_i and

$$Sol(SYS, \rho) = (S_1(\rho), \ldots, S_n(\rho)),$$

then $[\![SYS, \rho]\!] = S_i(\rho)$. We say that a system of equations *SYS* is equivalent to a formula φ of sHML with free variables from $\mathcal{Y}$ when for every environment ρ, $[\![\varphi, \rho]\!] = [\![SYS, \rho]\!]$.

Example 8. A system of equations equivalent to $\varphi_e = \max\ X.\ [\alpha]([\alpha]\mathtt{ff} \wedge X)$ is SYS_e, which has no free variables and includes the equations $X_0 = [\alpha]X_1$ and $X_1 = [\alpha]\mathtt{ff} \wedge X_0$, where X_0 is the principal variable of SYS_e. ■

Given an environment ρ and such a system of equations *SYS*, notice that for any equation $X = F$ of *SYS*, $[\![X, \rho[SYS]]\!] = [\![F, \rho[SYS]]\!]$ and that if X is the principal variable of *SYS*, then $[\![X, \rho[SYS]]\!] = [\![SYS, \rho]\!]$. We note that, as is well known, the order of the calculation of the fixed points does not affect the overall outcome:

Lemma 6. *Let $SYS = (Eq_1, X, \mathcal{Y})$ and $SYS' = (Eq_2, X, \mathcal{Y})$, where Eq_2 is a permutation of Eq_1. Then, for all environments ρ, $[\![SYS, \rho]\!] = [\![SYS', \rho]\!]$.*

Therefore, for every equation $X = F$ of the system *SYS*, if *SYS*′ is the result of removing $X = F$ from the equations and adding X to the free variables,

$$[\![X, \rho[SYS]]\!] = \bigcup \left\{ S \mid S \subseteq \left[\!\left[F, \overline{\rho[X \mapsto S]}^{SYS'} \right]\!\right] \right\}.$$

Furthermore, we can compute parts of the solution of the system (or the whole solution) as simultaneous fixed points:

Lemma 7. *Let $SYS = (Eq_1 \cdot Eq_2, X, \mathcal{Y})$, $Eq_1 = \bigtimes_{1 \leq i \leq k}\{X_i = F_i\}$, $\mathcal{X}_1 = (X_1, \ldots, X_k)$, and $SYS' = (Eq_2, X, \mathcal{Y} \cup \mathcal{X}_1)$. Let for all environments ρ,*

$$\mathbb{S}_0(\rho) = \bigcup \left\{ \mathbb{S} \mid \mathbb{S} \subseteq \left[\!\left[\bigtimes_{i=1}^{k} F_i, \overline{\rho[\bigtimes \mathcal{X}_1 \mapsto \mathbb{S}]}^{SYS'} \right]\!\right] \right\}.$$

Then,

$$Sol(SYS, \rho) = \mathbb{S}_0(\rho) \cdot Sol(SYS', \rho[\bigtimes \mathcal{X}_1 \mapsto \mathbb{S}_0(\rho)]). \quad \blacksquare$$

As a consequence, for a set of variables $\{X_i \mid i \in I\}$ of a system of equations *SYS*,

$$\left[\!\left[\bigtimes_{i\in I} X_i, \rho[SYS] \right]\!\right] = \bigcup \left\{ \mathbb{S} \mid \mathbb{S} \subseteq \left[\!\left[\bigtimes_{i\in I} F_i, \overline{\rho \left[\bigtimes_{i\in I} X_i \mapsto \mathbb{S} \right]}^{SYS'} \right]\!\right] \right\},$$

where SYS' is the result of removing $X_i = F_i$ from the equations and adding X_i to the free variables, for all $i \in I$.

3.2.2. Standard and deterministic forms

We begin by defining a deterministic form for formulae in sHML.

Definition 11. A formula $\varphi \in$ sHML is in deterministic form iff for each pair of formulae $\psi_1 \neq \psi_2$ that occur in the same conjunction in φ, it must be the case that $\psi_1 = [\alpha_1]\psi_1'$ and $\psi_2 = [\alpha_2]\psi_2'$ for some $\alpha_1 \neq \alpha_2$ or that one of ψ_1 and ψ_2 is a free variable of φ. $\blacksquare$

The following lemma justifies calling these formulae deterministic by showing that applying the monitor synthesis function to them will yield a deterministic monitor.

Lemma 8. *Let $\varphi \in$ sHML be a formula in deterministic form with no free variables. Then $m = (\!|\varphi|\!)$ is deterministic.*

Proof. By examining the definition of the monitor synthesis function, we can see that if $(\!|\varphi|\!)$ contains a sum $\sum_{i\in A} m_i$, then φ contains a conjunction $\bigwedge_{i\in A\cup B} \varphi_i$, where for $i \in A$, $(\!|\varphi_i|\!) = m_i \neq \mathtt{yes}$ and $\sum_{i\in A} m_i$ satisfies the constraints in Definition 7. $\square$

Example 9. φ_e is not in deterministic form, but $\psi_e = [\alpha]([\alpha]\mathtt{ff} \wedge X)$ is (because here X is free) and so is $\varphi_e' = [\alpha][\alpha]\mathtt{ff}$. $\blacksquare$

We also define a standard form for formulae in sHML.

Definition 12. A formula $\varphi \in$ sHML is in standard form if all free and unguarded variables in φ are at the top level; that is,

$$\varphi = \varphi' \wedge \bigwedge_{i\in S} X_i$$

where φ' does not contain a free and unguarded variable. $\blacksquare$

Example 10. φ_e is in standard form and so is $\psi_e = [\alpha]([\alpha]\mathtt{ff} \wedge X)$ (although here X is free, it is also guarded). Formula $[\alpha]\mathtt{ff} \wedge X$ is in standard form, because X is at the top level and so is $\varphi_e' = [\alpha][\alpha]\mathtt{ff}$, because it has no variables. Formula $\max\ X.\ ([\alpha]X \wedge Y)$ is not is in standard form, but the construction in the proof of Lemma 9 below transforms it into $(\max\ X.\ [\alpha]X) \wedge Y$ which is. $\blacksquare$

Lemma 9. *Each formula in* sHML *is equivalent to some formula in* sHML *which is in standard form.*

Lemma 10. *Let SYS be a system of equations and $X = F$ an equation of the system and let SYS' result from replacing $X = F$ by $X = F'$. Let SYS_0 be the result of removing the first equation from SYS and adding X_1 to the free variables, $\mathcal{Y}$. If for every environment ρ, $[\![F, \rho[SYS_0]]\!] = [\![F', \rho[SYS_0]]\!]$, then SYS' is equivalent to the original system, SYS.*

Proof. Because of Lemma 6, it suffices to prove the lemma when replacing the equation of the primary variable $X_1 = F_1$ by $X_1 = F_1'$. Then,

$$\begin{aligned}
[\![SYS, \rho]\!] &= \\
&= \bigcup \{S_1 \mid S_1 \subseteq \left[\!\left[F_1, \overline{\rho[X_1 \mapsto S_1]}^{SYS_0} \right]\!\right] \} \\
&= \bigcup \{S_1 \mid S_1 \subseteq \left[\!\left[F_1', \overline{\rho[X_1 \mapsto S_1]}^{SYS_0} \right]\!\right] \} \\
&= [\![SYS', \rho]\!]. \quad \square
\end{aligned}$$

We now extend the notion of standard and deterministic forms to systems of equations.

Definition 13. Let *SYS* be a system of equations that is equivalent to some formula $\varphi \in$ sHML. We say that an equation, $X_i = F_i$ is in standard form if either $F_i = \mathtt{ff}$, or

$$F_i = \bigwedge_{j \in K_i} [\alpha_j] X_j \wedge \bigwedge_{j \in S_i} Y_j$$

for some finite set of indices, K_i and S_i. We say that *SYS* is in standard form if every equation in *SYS* is in standard form.

Lemma 11. *For each formula $\varphi \in$ sHML, there exists a system of equations that is equivalent to φ and is in standard form.*

Example 11. The system of equations that results from the construction of Lemma 11 for formula $\varphi_e = \max\ X.\ [\alpha]([\alpha]\mathtt{ff} \wedge X)$ is SYS'_e, which has no free variables and includes the equations

$$\begin{aligned} X &= [\alpha] X_1, \\ X_1 &= [\alpha] X_2 \wedge [\alpha] X_1, \\ X_2 &= \mathtt{ff}, \end{aligned}$$

where X is the principal variable of SYS'_e. ■

Definition 14. Let $SYS = (Eq, X_1, \mathcal{Y})$ be a system of equations equivalent to a formula in sHML. We say that an equation $X = \bigwedge F$ in Eq is in deterministic form iff for each pair of expressions $F_1, F_2 \in F \setminus \mathcal{Y}$, it must be the case that $F_1 = [\alpha_1] X_i$ and $F_2 = [\alpha_2] X_j$, for some $\alpha_1, \alpha_2 \in$ Act and some i, j such that if $\alpha_1 = \alpha_2$ then $X_i = X_j$. We say that *SYS* is in deterministic form if every equation in Eq is in deterministic form. ■

Lemma 12. *For each* sHML *system of equations in standard form, there exists an equivalent system of equations that is in deterministic form.*

Example 12. The deterministic form of SYS'_e is SYS''_e, which has no free variables and includes the equations

$$\begin{aligned} X &= [\alpha] X_1, \\ X_1 &= [\alpha] X_{12}, \\ X_2 &= \mathtt{ff}, \\ X_{12} &= \mathtt{ff}, \end{aligned}$$

where X is the principal variable of SYS'_e. ■

Lemma 13. *Let $SYS = (Eq, X_1, \mathcal{Y})$ be a* sHML *system of equations in deterministic form. There exists a formula $\varphi \in$ sHML that is in deterministic form and is equivalent to SYS.*

Finally, we are ready to prove the main theorem in this section.

Theorem 5. *For each formula $\varphi \in$ sHML there exists a formula $\psi \in$ sHML that is equivalent to φ and is in deterministic form.*

Proof. Follows from Lemmata 11, 12 and 13. □

Example 13. If we follow through all the constructions, starting from φ_e, using Lemma 11 we construct SYS'_e, which is a system of equations in standard form. From SYS'_e, using Lemma 12, we build system SYS''_e, which is in deterministic form. Finally, from SYS''_e we use Lemma 13 to obtain the deterministic form of φ_e, which happens to be $\varphi'_e = [\alpha][\alpha]\mathtt{ff}$. ■

This section's conclusion is that to monitor an sHML formula, it is enough to consider deterministic monitors. On the other hand, if we are concerned with the computational cost of constructing a monitor (and we are), it is natural to ask what the cost of determinizing a monitor is. The following Section 4 is devoted to answering this and related questions.

Table 4
System N is the result of replacing rule mRec by rules mRecF and mRecB.

$$\textsc{mRecF}\ \frac{}{\texttt{rec}x.\ m_x \xrightarrow{\tau} m_x} \qquad \textsc{mRecB}\ \frac{}{x \xrightarrow{\tau} p_x}$$

Table 5
System M is the result of replacing rule mRec by rules mRecF and mRecP.

$$\textsc{mRecF}\ \frac{}{\texttt{rec}x.\ m_x \xrightarrow{\tau} m_x} \qquad \textsc{mRecP}\ \frac{}{x \xrightarrow{\tau} m_x}$$

4. Bounds for determinizing monitors

The purpose of this section is to establish bounds on the sizes of monitors, to compare the succinctness of (nondeterministic) finite automata and monitors, and to determine the cost of converting nondeterministic monitors into equivalent deterministic ones. It is hard to extract bounds from the constructions of Section 3. Therefore, we examine a different approach in this section. We remind the reader that, in this section, we consider monitors which use only the `yes` verdict — although treating monitors which use only the `no` verdict instead is completely analogous and we can also treat monitors which use both verdicts as described in Section 5.

4.1. Semantic transformations

For convenience, we slightly alter the behavior of monitors to simplify this section's arguments. Specifically, we provide three different sets of rules to define the behavior of the monitors, but we prove these are equivalent with respect to the traces which can reach a `yes` value, which is what we need for this section. Fix a reference monitor, which intuitively corresponds to the start state of a monitor LTS we consider. All other monitors are submonitors of this reference monitor. We assume without loss of generality that each variable x appears in the scope of a unique monitor $p_x = \texttt{rec}\ x.\ m_x$. The monitors may behave according to a system of rules. System O is the old system of rules, as given in Table 2 of Section 2.1; system N is given by replacing rule mRec by the rules given by Table 4.

Derivations $\longrightarrow$ and $\Rightarrow$ are defined as before, but the resulting relations are called $\longrightarrow_O$ and $\Rightarrow_O$, and $\longrightarrow_N$ and $\Rightarrow_N$, respectively for systems O and N. This subsection's main result, as given by Corollary 1, is that systems O and N are equivalent. That is, for any monitor m, trace t, and value v,

$$m \overset{t}{\Rightarrow}_O v \text{ if and only if } m \overset{t}{\Rightarrow}_N v.$$

To prove the equivalence of the two systems, we introduce an intermediate system, which we call system M. This is the result of replacing rule mRec of system O by the rules given by Table 5.

For system M as well, derivations $\longrightarrow$ and $\Rightarrow$ are defined as before, but the resulting relations are called $\longrightarrow_M$ and $\Rightarrow_M$ for system M, to distinguish them from $\longrightarrow_O$, $\Rightarrow_O$, $\longrightarrow_N$, and $\Rightarrow_N$. Notice that the syntax used in system O and systems M and N is necessarily different. While systems M and N require unique p_x and m_x and no substitutions occur in a derivation, the substitutions which occur in rule mRec produce new monitors and can result for each variable x in several monitors of the form $\texttt{rec}\ x.\ m$. For example, consider monitor

$$m = \texttt{rec}\ x.\ (\alpha.\texttt{yes} + \beta.\texttt{rec}\ y.\ (\alpha.x + \beta.y)).$$

In this case, $p_x = m$ and $p_y = \texttt{rec}\ y.\ (\alpha.x + \beta.y)$, but by rule mRec of system O,

$$\begin{aligned} m &\xrightarrow{\tau} \alpha.\texttt{yes} + \beta.\texttt{rec}\ y.(\alpha.\texttt{rec}\ x.\ (\alpha.\texttt{yes} + \beta.\texttt{rec}\ y.\ (\alpha.x + \beta.y)) + \beta.y) \\ &\xrightarrow{\beta} \texttt{rec}\ y.\ (\alpha.\texttt{rec}\ x.\ (\alpha.\texttt{yes} + \beta.\texttt{rec}\ y.\ (\alpha.x + \beta.y)) + \beta.y) = p'_y, \end{aligned}$$

but $p_y \neq p'_y$ and they are both of the form $\texttt{rec}\ y.\ m$, which means that they cannot both appear in a derivation of system N or M. Thus, when comparing the two systems, we assume one specific initial monitor p_0, from which all derivations in both systems are assumed to originate. Monitor p_0 satisfies the syntactic conditions for systems M and N and this is enough, since all transitions that can occur in these systems only generate submonitors of p_0.

The reason for changing the rules in the operational semantics of monitors is that for our succinctness results we need to track when recursion is used to reach a previous state of the monitor (i.e. a previous monitor in the derivation) by tracking when rule mRecB is used and then claim that this move takes us back a few steps in the derivation. Thus, we will be using system N for the remainder of this section. However, before that we need to demonstrate that it is equivalent to system O, in that for every monitor m, trace t, and value v, $m \overset{t}{\Rightarrow}_O v$ iff $m \overset{t}{\Rightarrow}_N v$. We prove this claim in this subsection by demonstrating equivalence of both systems to system M.

Lemma 14. *Given monitors m, n and action $\alpha \in \text{Act}$, $m \xrightarrow{\alpha}_N n$ iff $m \xrightarrow{\alpha}_M n$ iff $m \xrightarrow{\alpha}_O n$.*

Proof. Notice that in all three systems, $m \xrightarrow{\alpha} n$ can only be produced by a combination of rules mAct, mSelL, and mSelR (all variants of mRec can only produce τ-transitions, which are preserved through mSelL and mSelR). Since all three rules are present in all three systems, when this transition is provable in one of the systems, it can be replicated in each of them. □

Lemma 15. *If $m \xrightarrow{\tau}_M n$, then there is some variable x, such that m is a sum of x or of p_x and $n = m_x$.*

Proof. Notice that τ-transitions can only be introduced by rules mRecF and mRecP. Moreover, τ-transitions of monitors m_1 and m_2 are propagated to $m_1 + m_2$ via rules mSelL and mSelR. Then, it is not hard to verify that if $m \xrightarrow{\tau}_M n$ was produced by mRecF or mRecP, then m, n satisfy the property asserted by the lemma. Moreover, rules mSelL and mSelR preserve this property. □

Lemma 16. *If $m \xrightarrow{\tau}_N n$, then there is some variable x, such that either m is a sum of x and $n = p_x$, or m is a sum of p_x and $n = m_x$.*

Proof. Very similar to the proof of Lemma 15. □

Lemma 17. *If $m \xrightarrow{\tau}_O n$, then there is a monitor $r = \texttt{rec}\, x.\, r'$, such that m is a sum of r and $n = r'[r/x]$.*

Proof. Again, very similar to the proof of Lemma 15. □

We first prove the equivalence of the systems M and N.

Lemma 18. *For a monitor m, trace t, and value v, $m \overset{t}{\Rightarrow}_N v$ iff $m \overset{t}{\Rightarrow}_M v$.*

Proof. We first prove that if $m \overset{t}{\Rightarrow}_M v$, then $m \overset{t}{\Rightarrow}_N v$ by induction on the length of the derivation $m \overset{t}{\Rightarrow}_M v$.

If $m = v$, then we are done, as the trivial derivation exists for both systems.
If $m \xrightarrow{\alpha}_M m' \overset{t'}{\Rightarrow}_M v$, then $m \xrightarrow{\alpha}_N m'$ by Lemma 14. By the inductive hypothesis, $m' \overset{t'}{\Rightarrow}_N v$ and we are done.
If $m \xrightarrow{\tau}_M m' \overset{t}{\Rightarrow}_M v$, then by Lemma 15, there is some variable x, such that m is a sum of x or of p_x and $m' = m_x$. Then, if m a sum of p_x, $m \xrightarrow{\tau}_N m_x$ and if m a sum of x, then $m \xrightarrow{\tau}_N p_x \xrightarrow{\tau}_N m_x = m'$. By the inductive hypothesis, $m' \overset{t}{\Rightarrow}_N v$ and we are done.

We now prove that if $m \overset{t}{\Rightarrow}_N v$, then $m \overset{t}{\Rightarrow}_M v$, again by induction on the length of the derivation $m \overset{t}{\Rightarrow}_N v$. The only different case from above is for when $m \xrightarrow{\tau}_N m' \overset{t}{\Rightarrow}_N v$. In this case, by Lemma 16, there is some variable x, such that m is a sum of x and $m' = p_x$, or m is a sum of p_x and $m' = m_x$. Then, if m a sum of p_x, $m \xrightarrow{\tau}_M m_x = m'$ and by the inductive hypothesis, $m' \overset{t}{\Rightarrow}_M v$ and we are done. If m a sum of x, then $m' = p_x \neq v$, so the derivation is of the form $m \xrightarrow{\tau}_N m' = p_x \xrightarrow{\tau}_N m_x \overset{t}{\Rightarrow}_N v$ (as p_x can only transition to m_x), thus $m \xrightarrow{\tau}_M m_x$. By the inductive hypothesis, $m_x \overset{t}{\Rightarrow}_M v$ and we are done. □

Now we prove the equivalence of systems O and M.

Lemma 19. *For a monitor m, trace t, and value v, $m \overset{t}{\Rightarrow}_M v$ iff $m \overset{t}{\Rightarrow}_O v$.*

Proof. As we mentioned before, the syntax used in systems O and M is necessarily different. While rule mRecP requires a unique m_x (and thus a unique p_x), the substitutions which occur in rule mRec often result in several monitors of the form $\texttt{rec}\ x.\ m$. The central idea of this proof is that if the derivation we consider starts with a monitor whose submonitors satisfy the uniqueness conditions we have demanded in systems M and N for $\texttt{rec}\ x.\ m$, then all produced monitors of the form $\texttt{rec}\ x.\ m$ are somehow equivalent.

Thus, we assume an initial monitor p_0, such that all derivations considered are subderivations of a derivation initialized at p_0 and such that every x is bound in a unique submonitor $\texttt{rec}\ x.\ m$, which is p_x as required in system M. We define $\equiv$ to be the smallest equivalence relation such that for all variables x, $x \equiv p_x$ and that for all m and $d \equiv x$, $m \equiv m[d/x]$. We now proceed to establish a number of auxiliary claims.

1. *For every $m = \texttt{rec}\ x.\ d$ which appears in a derivation in O from p_0, $m \equiv p_x$.* We prove this claim as follows. Since only substitution can introduce new monitors of the form $\texttt{rec}\ x.\ d$ and initially, for each variable x the only such monitor is p_x, we can use induction on the overall number of substitutions in the derivation from p_0 to prove the claim. If no

substitutions happened, the claim is trivial. Say the derivation so far is of the form $p_0 \stackrel{w}{\Rightarrow}_O \texttt{rec}\ y.\ r \stackrel{\tau}{\rightarrow}_O r[\texttt{rec}\ y.\ r/y]$, with $r[\texttt{rec}\ y.\ r/y]$ being the latest substitution in the derivation, such that the claim holds for all submonitors of monitors appearing in $p_0 \stackrel{w}{\Rightarrow}_O \texttt{rec}\ y.\ r$. Let $m = \texttt{rec}\ x.\ d$ be a submonitor of $r[\texttt{rec}\ y.\ r/y]$. We know by the inductive hypothesis that $\texttt{rec}\ y.\ r \equiv p_y \equiv y$; then, there is some $m' = \texttt{rec}\ x.\ d'$ submonitor of $\texttt{rec}\ y.\ r$ (thus, $m' \equiv p_x$) and $m = m'[\texttt{rec}\ y.\ r/y]$, but by the definition of $\equiv$, $m \equiv m' \equiv p_x$.

2. *For a value v and monitor m, if $v \equiv m$, then $m = v$.* To see this, notice that $\equiv_M$ such that for value v, $v \equiv_M m$ iff $v = m$ and for m, m' which are not values, always $m \equiv_M m'$, satisfies the above conditions, therefore $\equiv \subseteq \equiv_M$.
3. *If $m + d \equiv r$, then there are $m' + d' = r$, where $m \equiv m'$ and $d \equiv d'$.* Indeed, notice that $\equiv$ can be constructed as the union $\bigcup_{i\in\mathbb{N}} \equiv_i$, where $\equiv_0$ includes all pairs (m, m), (x, p_x), (p_x, x), while $\equiv_{i+1}$ has all pairs in $\equiv_i$ and all $(m, m[d/x])$, $(m[d/x], m)$ and (m, r), (r, m), where $d \equiv_i x$ and $m \equiv_i m' \equiv_i r$; also, notice that neither x not p_x is of the form $m + d$ and all steps of the construction preserve the claim.
4. *If $\alpha.m \equiv r$, then there is some $\alpha.d = r$, where $m \equiv d$.* The proof of this claim is the same as the one for the one above.
5. *If $\texttt{rec}\ x.\ m \equiv r$ or $x \equiv r$, then $x = r$ or there is some $\texttt{rec}\ x.\ d = r$, where $m \equiv d$.* This is proved as above.
6. *If $m \equiv r$ and m is a sum of d, then r is a sum of some $d' \equiv d$.* By induction on the construction of a sum and the claim for $+$.

By the second claim, it is enough to prove that if $m \equiv m'$ and $m \stackrel{\mu}{\rightarrow}_O d$, then there is some $d' \equiv d$, such that $m' \stackrel{\mu}{\rightarrow}_M d'$ and conversely, if $m \equiv m'$ and $m' \stackrel{\mu}{\rightarrow}_M d'$, then there is some $d \equiv d'$, such that $m \stackrel{\mu}{\rightarrow}_O d$ (notice that this makes $\equiv$ a bisimulation). We now show these two statements separately.

Assume, first of all that $m \equiv m'$ and $m \stackrel{\mu}{\rightarrow}_O d$.

If $\mu = \tau$, then by Lemma 17, m is a sum of $\texttt{rec}\ x.\ r$ and $d = r[m/x]$ — from the last claim above, we assume for simplicity that $m = \texttt{rec}\ x.\ r$ and it does not affect the proof. By the claims above, either $m' = x$ or $m' = \texttt{rec}\ x.\ d'$, where $r \equiv d'$. Since $r \equiv r[m/x] = d$, if $m' = \texttt{rec}\ x.\ d'$, then $m' = \texttt{rec}\ x.\ d' \stackrel{\tau}{\rightarrow}_M d' \equiv r \equiv d$ and we are done. Otherwise, $m' = x$ and by the first claim, $m \equiv p_x$, so $m_x \equiv r \equiv d$, therefore $m' = x \stackrel{\tau}{\rightarrow}_M m_x \equiv d$ and we are done.

If $\mu = \alpha \in \text{Act}$, then m is a sum of some $\alpha.d$, so by the claims above, m' is the sum of some $\alpha.d'$, where $d' \equiv d$; thus, $m' \stackrel{\mu}{\rightarrow}_M d'$.

For the converse, let $m \equiv m'$ and $m' \stackrel{\mu}{\rightarrow}_M d'$.

If $\mu \in \text{Act}$, then the argument is as above. If $\mu = \tau$, then by Lemma 15, m' is a sum of x or p_x and $d' = m_x$. Therefore, by the claims above, either $m = x$, which cannot occur in system O, or $m = \texttt{rec}\ x.\ d$, for some $d \equiv m_x$. Therefore, $m \stackrel{\tau}{\rightarrow}_O d[m/x] \equiv d \equiv d'$. □

Corollary 1. *For a monitor m, trace t, and value v, $m \stackrel{t}{\Rightarrow}_N v$ iff $m \stackrel{t}{\Rightarrow}_M v$ iff $m \stackrel{t}{\Rightarrow}_O v$.*

By Corollary 1, systems M, N, and O are equivalent, so we will be using whichever is more convenient in the proofs that follow. For the remainder of this section, we use system N and we call $\rightarrow_N$ and $\Rightarrow_N$ simply $\rightarrow$ and $\Rightarrow$, respectively.

There are three reasons for changing the operational semantics rules of monitors. One is that, for the bounds we prove, we need to track when recursion is used in a derivation. Another is that in System *N* (unlike in System *O*) it is clear which monitors may appear in a derivation starting from monitor m (namely, at most all submonitors of m), which in turn makes it easier to construct an LTS — and also to transform a monitor into an automaton. For instance, consider $m = \texttt{rec}\ x.\ (\alpha.x + \beta.\texttt{yes})$. In System O, $m \stackrel{\tau}{\rightarrow} \alpha.(\texttt{rec}\ x.\ (\alpha.x + \beta.\texttt{yes})) + \beta.\texttt{yes}$, which is *not* a subterm of m. On the other hand, in System N, $m \stackrel{\tau}{\rightarrow} \alpha.x + \beta.\texttt{yes}$, which is a subterm of m. Finally, and partly due to the previous observation, we can see that a monitor, viewed as an LTS, has a specific form: it is a rooted tree with labeled edges provided by $\stackrel{\mu}{\rightarrow}$, with some back edges, which result from recursion (namely, from the rule RecB in Table 4).

When using System N, we need to be more careful with the definition of determinism. Notice that it is possible to have a nondeterministic monitor, which has a deterministic submonitor. For instance, $p_x = \texttt{rec}\ x.\ (\alpha.x + \alpha.\texttt{yes})$ is nondeterministic, while according to our definition of determinism, $\alpha.x$ is deterministic (specifically, all variables are deterministic). The issue here is that although $\alpha.x$ is deterministic in form, it can transition to (x and then to) p_x, which is not. This is not a situation we encountered in System O, because there variables do not derive anything on their own and all monitors we consider are closed. In System N, though, a variable x can appear in a derivation and it can derive p_x, so it would not be reasonable to consider a variable deterministic — and thus judge the determinism of a monitor only from its structure. In other words, our definition of a deterministic monitor additionally demands that said monitor is closed; alternatively, for a monitor which appears in a derivation to be deterministic, we demand that the initial monitor p_0 be deterministic (by Definition 7).

4.2. Size bounds for monitors

We present upper and lower bounds on the size of monitors. We first compare monitors to finite automata and then we examine the efficiency of monitor determinization. Note that monitors can be considered a special case of nondeterministic finite automata (NFA) and this observation is made explicit.

This section's results are the following. We first provide methods to transform monitors to automata and back. One of the consequences of these transformations is that we can use the classic subset construction for the determinization of NFAs and thus determinize monitors. One advantage of this method over the one given in the previous sections is that it makes it easier to extract upper bounds on the size of the constructed monitors. Another is that it can be applied to an equivalent NFA, which can be smaller than the given monitor, thus resulting in a smaller deterministic monitor. Then, we demonstrate that there is an infinite family of languages $(L_n)_{n\in\mathbb{N}}$, such that for each n, L_n is recognizable by an NFA of $n+1$ states, a DFA of 2^n states, a monitor of size $O(2^n)$, and a deterministic monitor of size $2^{2^{O(n)}}$. Furthermore, we cannot do better, as we demonstrate that any monitor which accepts L_n must be of size $\Omega(2^n)$ and every deterministic monitor which accepts L_n must be of size $2^{2^{\Omega(n)}}$.

4.2.1. From monitors to finite automata

A monitor can be considered to be a finite automaton with its submonitors as states and $\Rightarrow$ as its transition relation. Here we make this observation explicit.[2] For a monitor m, we define the automaton $A(m)$ to be $(Q, \text{Act}, \delta, q_0, F)$, where

- Q, the set of states, is the set of submonitors of m;
- Act, the set of actions, is also the alphabet of the automaton;
- $d' \in \delta(d,\alpha)$ iff $d \Rightarrow \xrightarrow{\alpha} d'$;
- q_0, the initial state is m;
- $F = \{\texttt{yes}\}$, that is, `yes` is the only accepting state.

Proposition 1. *Let m be a monitor and $t \in \text{Act}^*$ a trace. Then, $A(m)$ accepts t iff $t \stackrel{t}{\Rightarrow} \texttt{yes}$.*

Proof. We actually prove that for every $q \in Q$, $A_q(m) = (Q, \text{Act}, \delta, d, F)$ accepts t iff $q \stackrel{t}{\Rightarrow} \texttt{yes}$ and we do this by induction on t. If $t = \epsilon$, then $A_q(m)$ accepts iff $q \in F$ iff $q = \texttt{yes}$ iff $q \Rightarrow \texttt{yes}$. If $t = \alpha t'$, then
$A_q(m)$ accepts t
iff there is some $q' \in \delta(q,\alpha)$ such that $A_{q'}(m)$ accepts t'
iff there is some q' s.t. $q \Rightarrow \xrightarrow{\alpha} q'$ and $q' \stackrel{t'}{\Rightarrow} \texttt{yes}$
iff $q \stackrel{t}{\Rightarrow} \texttt{yes}$. □

Notice that $A(m)$ has at most $|m|$ states (because Q only includes submonitors of m), but probably fewer, since two occurrences of the same monitor as submonitors of m give the same state — and we can cut that down a bit by removing submonitors which can only be reached through τ-transitions. Furthermore, if m is deterministic, then $A(m)$ is deterministic.

Corollary 2. *For every monitor m, there is an automaton A which accepts the same language and has at most $|m|$ states. Furthermore, if m is deterministic, then A is a DFA.*

Corollary 3. *All languages recognized by monitors are regular.*

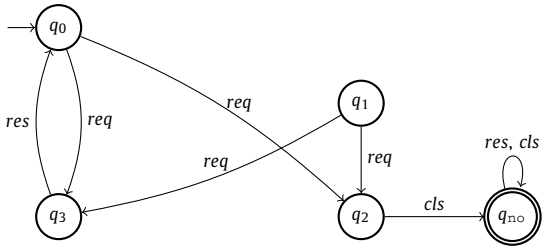

Fig. 2. The automaton corresponding to the monitor of Fig. 1 (in this case, the verdict used is `no` instead of `yes`).

See Fig. 2.

[2] The definition to follow is possible because system N only transitions to submonitors of an initial monitor; otherwise we would need to consider all monitors reachable through transitions and it would not be as clear which ones these are.

4.2.2. From automata to monitors

We would also like to be able to transform a finite automaton to a monitor and thus recognize regular languages by monitors. This is not always possible, though, since there are simple regular languages not recognized by any monitor. Consider, for example, $(11)^*$, which includes all strings of ones of even length. If there were such a monitor for the language, since ϵ is in the language, the monitor could only be $\texttt{yes}$, which accepts everything (so, this conclusion is also true for any regular language of the form $\epsilon + L \neq \text{Act}^*$).

One of the properties which differentiates monitors from automata is the fact that verdicts are irrevocable for monitors. Therefore, if for a monitor m and finite trace t, $m \stackrel{t}{\Rightarrow} \texttt{yes}$, then for every trace t', it is also the case that $m \stackrel{tt'}{\Longrightarrow} \texttt{yes}$ (because of rule mVerd, which yields that for every t', $\texttt{yes} \stackrel{t'}{\Rightarrow} \texttt{yes}$). So, if L is a regular language on Act recognized by a monitor, then L has the property that for every $t, t' \in \text{Act}^*$, if $t \in L$, then $tt' \in L$. We call such languages irrevocable (we could also call them suffix-closed).

Now, consider an automaton which recognizes an irrevocable language. Then, if q is any (reachable) accepting state of the automaton, notice that if we can reach q through a word t, then t is in the language and so is every $t\alpha$; therefore, we can safely add an α-transition from q to an accepting state (for example, itself) if no such transition exists. We call an automaton which can always transition from an accepting state to an accepting state irrevocable. Note that in an irrevocable DFA, all transitions from accepting states go to accepting states.

Corollary 4. *A language is regular and irrevocable if and only if it is recognized by an irrevocable NFA.*

Corollary 5. *A language is regular and irrevocable if and only if it is recognized by an irrevocable DFA.*

Proof. Simply notice that the usual subset construction on an irrevocable NFA gives an irrevocable DFA. □

Let $A = (Q, \text{Act}, \delta, q_0, F)$ be an automaton and $n = |Q|$. For $1 \leq k \leq n$, $q_1, q_2, \ldots, q_k \in Q$ and $\alpha_1, \alpha_2, \ldots, \alpha_{k-1} \in \text{Act}$, $P = q_1\alpha_1 q_2 \alpha_2 \cdots \alpha_{k-1} q_k$ is a path of length k on the automaton if all $q_1, q_2, \ldots, q_k$ are distinct and for $0 < i < k$, $q_{i+1} \in \delta(q_i, \alpha_i)$. Given such path, $P|_{q_i} = q_1, q_2, \ldots, q_i$ and $q_P = q_k$. By $q \in P$ we (abuse notation and) mean that q appears in P. When $q_1 = q_0$, we say that the path originates at q_0.

Given an irrevocable NFA, we can construct an equivalent (in that it accepts the same traces) monitor through a procedure which can be described informally in the following way. We first unravel the NFA into a tree: we make a copy of each state q for all paths from the initial state that end with q. Then, we map each node of this tree to a monitor, such that in the end, the root is mapped to the resulting equivalent monitor. The leaves that correspond to an accepting state are mapped to $\texttt{yes}$; we use action applications to describe forward tree edges and recursion for back edges — there is no need for cross edges. (See Fig. 3 for an example of the transformation.) If the automaton is deterministic, so is the resulting monitor.

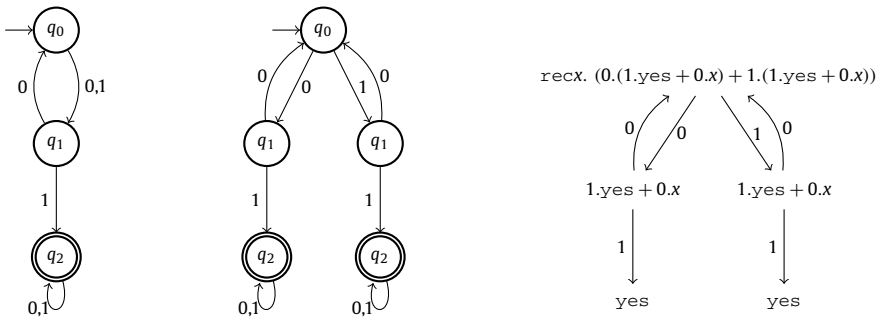

Fig. 3. Transformation of an automaton into a monitor: DFA to tree unraveling to deterministic monitor.

Theorem 6. *Given an irrevocable NFA of n states, there is a monitor of size $2^{O(n \log n)}$ which accepts the same traces as the automaton.*

Proof. Let $A = (Q, \text{Act}, \delta, q_0, F)$ be an irrevocable finite automaton and $n = |Q|$. We can assume that F has a single accepting state (since all states in F behave the same way), which we call Y. For some $q \in Q$, $A_q = (Q, \text{Act}, \delta, q, F)$, which is the same automaton, but the run starts from q. Given a set S of monitors, $\sum S$ is some sum of all elements of S.

For every path $P = q_1\alpha_1 \cdots \alpha_{k-1} q_k$ of length $k \leq n$ on Q, we construct a monitor $m(P)$ by recursion on $n - k$.
If $q_P = Y$, then $m(P) = \texttt{yes}$. Otherwise,

$$m(P) = \mathtt{rec}\, x_P. \left(\begin{array}{c} \sum\{\alpha.m(P\alpha q) \mid \alpha \in \text{Act and } q \in \delta(q_P, a) \text{ and } q \notin P\} \\ + \\ \sum\{\alpha.x_{P|_q} \mid \alpha \in \text{Act and } q \in \delta(q_P, a) \text{ and } q \in P\} \end{array} \right).$$

This was a recursive definition, because if $n = k$, all transitions from q_P lead back to a state in P. Let $m = m(q_0)$. Notice that m satisfies our assumptions that all variables x (i.e. x_P) are bound by a unique p_x (i.e. $m(P)$). Furthermore, following the definition above, for each path P originating at q_0 we have generated at most $2|\text{Act}| \cdot n$ new submonitors of $m(P)$ — this includes all the generated sums of submonitors of the forms $\alpha.m(P\alpha q)$ and $\alpha.x_{P_q}$ and $m(P)$ itself. Specifically, if the number of transitions from each state is at most Δ (Δ is at most $n|\text{Act}|$), then for each path P originating at q_0 we have generated at most 2Δ new submonitors of $m(P)$. If the number of paths originating at q_0 is Π, then

$$|m| \leq 2\Delta \cdot \Pi.$$

Let $P = q_1\alpha_1 q_2 \alpha_2 \cdots \alpha_{k-1} q_k$ be a path on Q. Notice that $q_1 q_2 \cdots q_k$ is a permutation of length k of elements of Q and $\alpha_1\alpha_2\cdots\alpha_{k-1}$ is a $(k-1)$-tuple of elements form Act. Therefore, the number of paths originating at q_0 is at most

$$\Pi \;\leq\; \sum_{k=1}^{n-1} |\text{Act}|^k \cdot (k!) \leq (n-1)! \cdot |\text{Act}|^{n-1} \cdot n \;=\; |\text{Act}|^{n-1} \cdot (n!)$$

and thus,

$$|m| \;\leq\; 2n|\text{Act}| \cdot |\text{Act}|^{n-1} \cdot (n!) \;=\; 2n|\text{Act}|^n \cdot (n!) \;=\; 2^{O(n \log n)}.$$

To prove that m accepts the same traces as A, we prove the following claim.

Claim: for every such path P, $m(P) \stackrel{t}{\Rightarrow} \mathtt{yes}$ if and only if A_{q_P} accepts t.

We prove the claim by induction on t.

If $t = \epsilon$, then $m(P) \stackrel{t}{\Rightarrow} \mathtt{yes}$ iff $m(P) = \mathtt{yes}$ iff $q_P = Y$.

If $t = \alpha t'$ and A_{q_P} accepts t, then there is some $q \in \delta(q_P, \alpha)$, such that A_q accepts t'. We consider the following cases:

- if $q_P = Y$, then $m(P) = \mathtt{yes} \stackrel{t}{\Rightarrow} \mathtt{yes}$;
- if $q \in P$, then $m(P) \xrightarrow{\alpha} x_{P_q} \xrightarrow{\tau} m(P_q)$ and by the inductive hypothesis, $m(P_q) \stackrel{t'}{\Rightarrow} \mathtt{yes}$;
- otherwise, $m(P) \xrightarrow{\tau} \xrightarrow{\alpha} m(P\alpha q)$ and therefore, by the inductive hypothesis, $m(P\alpha q) \stackrel{t'}{\Rightarrow} \mathtt{yes}$.

If $t = \alpha t'$ and $m(P) \stackrel{t}{\Rightarrow} \mathtt{yes}$ and $q_P \neq Y$, then there is some $q \in \delta(q_P, \alpha)$, such that either $m(P) \xrightarrow{\tau} \xrightarrow{\alpha} m(P\alpha q) \stackrel{t'}{\Rightarrow} \mathtt{yes}$ (when $q \notin P$), or $m(P) \xrightarrow{\tau} \xrightarrow{\alpha} x_{P|_q} \xrightarrow{\tau} m(P|_q) \stackrel{t'}{\Rightarrow} \mathtt{yes}$ (when $q \in P$); in both cases, by the inductive hypothesis, A_q accepts t', so A accepts t. □

Corollary 6. *Given an irrevocable DFA of n states, there is a deterministic monitor of size at most* $2n \cdot |\text{Act}|^n = 2^{O(n)}$ *which accepts the same traces as the automaton.*[3]

Proof. Notice that the proof of Theorem 6 works for DFAs as well. We use the same construction. Unless $m = \mathtt{yes}$, every recursive operator (except the first one), variable, and value is prefixed by an action; furthermore, if A is deterministic and $\alpha.p_1$, $\alpha.p_1$ appear as part of the same sum, then $p_1 = p_2$. So, we have constructed a deterministic monitor.

Since A is deterministic, given a state q_1, every path in A, $q_1\alpha_1\cdots\alpha_{k-1}q_k$ is fully defined by the sequence of actions which appear in the path, $\alpha_1\alpha_2\cdots\alpha_{k-1}$. Since $k \leq n$, the number of such paths in A is thus at most

$$\sum_{k=1}^{n} |\text{Act}|^{k-1} < n \cdot |\text{Act}|^{n-1}.$$

As we mentioned in the proof of Theorem 6, if the number of paths originating at q_0 is Π and the number of transitions from each state is at most Δ, then

$$|m| \leq 2\Delta \cdot \Pi$$

and therefore,

$$|m| \leq 2n|\text{Act}|^n. \quad \square$$

[3] Note that if $|\text{Act}| = 1$, then this Corollary claims a linear bound on the size of the deterministic monitor with respect to the number of states of the DFA. See Corollary 18 at the end of this section and the discussion right above it for more details.

Corollary 7. *A language is regular and irrevocable if and only if it is recognized by a (deterministic) monitor.*

Corollary 8. *Let m be a monitor for φ. Then, there is a deterministic monitor for φ of size $2^{O(2^{|m|})}$.*

Proof. Transform m into an equivalent NFA of at most $|m|$ states, then to a DFA of $2^{|m|}$ states, and then, into an equivalent deterministic monitor of size $2^{O(2^{|m|})}$. □

Now, this is a horrible upper bound. Unfortunately, as the remainder of this section demonstrates, we cannot do much better.

4.2.3. Lower bound for (nondeterministic) monitor size

It is easier to understand the intuition behind the lower bounds for constructing monitors after realizing that the LTS of a monitor is a rooted tree with additional back edges. The tree is the monitor's syntactic tree; a transition generated by rules mAct and mRecF (and then, possibly mSel) is a transition from a parent to a child and a transition generated by rule mRecB (and then, possibly mSel) is a transition to an ancestor (rule mVerd gives self-loops for the leaves). Furthermore, we note that from every node of this tree, distinct actions transition to distinct nodes. This is the form generated from the construction of Theorem 6.

The family of languages we consider is (initially) the following. For $n \geq 1$, let

$$L_n = \{\alpha 1 \beta \in \{0,1\}^* \mid |\beta| = n-1\}.$$

This is a well-known example of a regular language recognizable by an NFA of $n+1$ states, by a DFA of 2^n states, but by no DFA of fewer than 2^n states. As we have mentioned, monitors do not behave exactly the way automata do and can only recognize irrevocable languages. Therefore, we modify L_n to mark the ending of a word with a special character, e, and make it irrevocable. Thus,[4]

$$M_n = \{\alpha e \beta \in \{0,1,e\}^* \mid \alpha \in L_n\}.$$

Note that an automaton (deterministic or not) accepting L_n can easily be transformed into one (of the same kind) accepting M_n by introducing two new states, Y and N, where Y is accepting and N is not, so that all transitions from Y go to Y and from N go to N (N is a junk state, thus unnecessary for NFAs); then we add an e-transition from all accepting states to Y and from all other states to N. The reverse transformation is also possible: From an automaton accepting M_n, we can have a new one accepting L_n by shedding all e-transitions and turning all states that can e-transition to an accepting state of the old automaton to accepting states. The details are left to the reader.

Thus, there is an NFA for M_n with $n+2$ states and a DFA for M_n with 2^n+2 states, but no less. We construct a (nondeterministic) monitor for M_n of size $O(2^n)$. For every $\alpha \in \{0,1\}^*$, where $|\alpha| = k \leq n-1$, we construct a monitor m_α by induction on $n-k$: if $k = n-1$, $m_\alpha = e.\mathtt{yes}$; otherwise, $m_\alpha = 0.m_{\alpha 0} + 1.m_{\alpha 1}$. Let $m = \mathtt{rec}\ x.\ (0.x + 1.x + 1.m_\epsilon)$. Then, m mimics the behavior of the NFA for M_n and $|m| = 8 + |m_\epsilon| = O(2^n)$.

The idea behind showing that there is no monitor for M_n of size less than 2^n is that for every $w \in \{0,1\}^{n-1}$, $1we$ is an accepted trace. Furthermore, after reading the first letter, the monitor tree is not allowed to use a back edge (i.e. recursion), or it could accept a shorter trace. By the observation above about the form of a monitor as a tree, the monitor is (at least) a complete binary tree of height $n-1$. In the following, we make this argument more explicit.

Definition 15. We call a derivation $m \stackrel{t}{\Rightarrow} m'$ simple, if rules mRecB and mVerd are not used in the proof of any transition of the derivation. We say that a trace $t \in \text{Act}^*$ is simple for monitor m if there is a simple derivation $m \stackrel{t}{\Rightarrow} m'$. We say that a set G of simple traces for m is simple for m. ■

Lemma 20. *Every subderivation of a simple derivation is simple.*

Corollary 9. *If $t' \sqsubseteq t$ and t is a simple trace for monitor m, then t' is also a simple trace for m.*

Lemma 21. *Let m be a monitor and G a (finite) simple set of traces for m. Then, $|m| \geq |G|$.*

Proof. By structural induction on m.

If $m = v$ or x, then $|G|$ is either empty or $\{\epsilon\}$ and $|m| \geq 1$.

[4] Note that we can also allow for infinite traces without consequence.

If $m = \alpha.d$, then all non-trivial derivations that start from m, begin with $m \xrightarrow{\alpha} d$. Therefore, all traces in G, except perhaps for ϵ, are of the form αt. Let $G_\alpha = \{t \in \text{Act}^* \mid \alpha t \in G\}$. Then, $|G| \leq |G_\alpha| + 1$, as there is a 1-1 and onto mapping from $G \setminus \{\epsilon\}$ to G_α, namely $\alpha t \mapsto t$. By the inductive hypothesis, $|d| \geq |G_\alpha|$, so $|m| = |d| + 1 \geq |G_\alpha| + 1 \geq |G|$.

If $m = d + r$, then notice that all derivations that start from m (including the trivial one, if you consider $m \Rightarrow m$ and $d \Rightarrow d$ to be the same) can also start from either d or r. Therefore, $G = G_d \cup G_r$, where G_d is simple for d and G_r is simple for r. Then, $|m| = |d| + |r| \geq |G_d| + |G_r| \geq |G|$.

If $m = \mathtt{rec}\ x.\ d$, then all non-trivial derivations that start from m, must begin with $m \xrightarrow{\tau} d$. Therefore, it is not hard to see that G is simple for d as well, so $|m| = |d| + 1 > |G|$. □

Corollary 10. *Let t be a simple trace for m. Then, $h(m) \geq |t|$.*

Proof. Very similar to the proof of Lemma 21. □

Corollary 11. *Let m be a monitor and G a simple set of traces for m. Then, G is finite.*

Proof. Notice that $|m| \in \mathbb{N}$. □

Corollary 12. *Let m be a monitor and t a simple trace for m. Then, t is finite.*

Proof. A direct consequence of Corollary 10. □

Lemma 22. *In a derivation $m \overset{t}{\Rightarrow} x$, such that x is bound in m, a sum of p_x must appear acting as p_x.*

Proof. By induction on the length of the derivation. If it is 0, then $m = x$ and x is not bound. Otherwise, notice that naturally, x is bound in m, but x is not bound in x. Let $m \overset{t'}{\Rightarrow} d \xrightarrow{\mu} d'$ be the longest initial subderivation such that x remains bound in d. Thus, it must be the case that x is not bound in d'. The only rule which can have this effect is mRecF, possibly combined with mSel, so d is a sum of p_x acting as p_x. □

The following lemma demonstrates that derivations which are not simple enjoy a property which resembles the classic Pumping Lemma for regular languages.

Lemma 23. *Let $m \overset{t}{\Rightarrow} d$, such that t is not simple for m. Then, there are $t = xuz$, such that $|u| > 0$ and for every $i \in \mathbb{N}$, $m \overset{xu^iz}{\Longrightarrow} d$.*

Proof. Let $m \overset{t}{\Rightarrow} d$ be a non-simple derivation D. We assume that there are no $s(\xrightarrow{\tau})^+ p_x$ parts in it, where s is a sum of p_x acting as p_x; otherwise remove them and the resulting derivation is non-simple, because t is non-simple. Let $s \sqsubseteq t$ be the longest prefix of t, such that subderivation $m \overset{s}{\Rightarrow} r$ is simple. Then, there is a $s\alpha \sqsubseteq t$, such that there is a subderivation of D, $m \overset{s}{\Rightarrow} r \xrightarrow{\alpha} r'$ and in $r \xrightarrow{\alpha} r'$, either mVerd or mRecB is used and neither is used in $m \overset{s}{\Rightarrow} r$. If mVerd is used, then $m \overset{s}{\Rightarrow} v$ as part of D, so $d = v$ and $m \overset{su^i}{\Longrightarrow} v$, for $su = t$ and $i \in \mathbb{N}$. If mRecB is used, then $m \overset{s}{\Rightarrow} x \xrightarrow{\tau=\alpha} p_x$; by Lemma 22, a sum of p_x acting as p_x, say s_x must appear in $m \overset{s}{\Rightarrow} x$, so $s_x \overset{u}{\Rightarrow} p_x$ is thus part of the derivation and $|u| > 0$ by our assumption at the beginning of the proof. Then, for some x, z, $t = xuz$ and for every $i \in \mathbb{N}$, $m \overset{xu^iz}{\Longrightarrow} d$. □

Corollary 13. *Let m be a monitor and t a trace, such that $|t| > h(m)$. Then, there are $t = xuz$ and a monitor d, such that $|u| > 0$ and for every $i \in \mathbb{N}$, $m \overset{xu^iz}{\Longrightarrow} d$.*

Proposition 2. *Let m be a monitor for M_n. Then, $|m| \geq 3 \cdot 2^{n-1}$.*

Proof. Because of Lemma 21, it suffices to find a set G of simple traces for m, such that $|G| \geq 3 \cdot 2^{n-1}$. We define

$$G = \{t \in \{0, 1, e\}^* \mid t \sqsubseteq 1se, \text{ where } s \in \{0, 1\}^{n-1}\}.$$

Then, $|G| \geq 3 \cdot 2^{n-1}$, so it suffices to demonstrate that all traces in G are simple. In turn, it suffices to demonstrate that for $s \in \{0, 1\}^{n-1}$, $1se$ is simple. If it is not, then by Lemma 23, since $m \overset{1se}{\Longrightarrow} \mathtt{yes}$, there is a (strictly) shorter trace t, such that $m \overset{t}{\Rightarrow} \mathtt{yes}$, which contradicts our assumption that m is a monitor for M_n. □

We have thus demonstrated that for every $n \geq 1$, there is a monitor for M_n of size $O(2^n)$ and furthermore, that there is no monitor for M_n of size less than $3 \cdot 2^{n-1}$. So, to recognize languages M_n, monitors of size exponential with respect

to n are required and thus we have a lower bound on the construction of a monitor from an NFA, which is close to the respective upper bound provided by Theorem 6.

4.2.4. Lower bound for deterministic monitor size

We now consider deterministic monitors. We demonstrate (see Theorem 7) that to recognize languages M_n a deterministic monitors needs to be of size $2^{2^{\Omega(n)}}$. Therefore, a construction of a deterministic monitor from an equivalent NFA can result in a double-exponential blowup in the size of the monitor; constructing a deterministic monitor from an equivalent nondeterministic one can result in an exponential blowup in the size of the monitor. As Theorem 8 demonstrates, the situation is actually worse for the determinization of monitors, as there is a family U_n of irrevocable regular languages, such that for $n \geq 1$, U_n is recognized by a nondeterministic monitor of size $O(n)$, but by no deterministic one of size $2^{2^{o(\sqrt{n \log n})}}$.

The proof of Theorem 8 relies on a result by Chrobak [15] for unary languages (languages on only one symbol), who showed that for every n, there is a unary language Ch_n which is recognized by an NFA with n states, but by no DFA with $e^{o\left(\sqrt{n \log n}\right)}$ states. U_n is then the set of word $w \in \{0, 1\}$, such that the 0's or the 1's in w are a word from Ch_n. Then, from a deterministic monitor for U_n we can extract a unary DFA for Ch_n by following the 0^*1- or 1^*0-transitions of the monitor, until the first time recursion was used (i.e. a back edge was followed). Therefore, the first time the deterministic monitor has a back edge is at distance at least $e^{o\left(\sqrt{n \log n}\right)}$ from the root; so, it contains at least a complete binary tree of height $e^{o\left(\sqrt{n \log n}\right)}$.

Lemma 24. *Let m be a deterministic monitor. If $m \stackrel{t}{\Rightarrow} m'$, then m' is deterministic.*

Proof. Both m and m' are submonitors of the original monitor p_0. If m is deterministic, then so is p_0, and so is m'. □

Lemma 25. *If $m + m'$ is deterministic, then $m + m' \stackrel{\tau}{\not\rightarrow}$.*

Proof. The monitor $m + m'$ is a submonitor of the initial monitor, which is deterministic, so $m + m'$ must be of the form $\sum_{\alpha \in A} \alpha.m_\alpha$. We continue by induction on $m + m'$: if $m = \alpha.r$ and $m' = \beta.r'$, then by the production rules, only $m + m' \stackrel{\alpha}{\rightarrow} r$ and $m + m' \stackrel{\beta}{\rightarrow} r'$ are allowed; if one of m, m' is also a sum, then by the derivation rules, if $m + m' \stackrel{\tau}{\rightarrow}$, then also $m \stackrel{\tau}{\rightarrow}$ or $m' \stackrel{\tau}{\rightarrow}$, but by the inductive hypothesis, this is a contradiction. □

Corollary 14. *Only τ-transitions of the form $x \stackrel{\tau}{\rightarrow} p_x$ and $\texttt{rec}\, x.\, m \stackrel{\tau}{\rightarrow} m$ are allowed for deterministic monitors.*

Lemma 26. *Let $m \sim m'$ be deterministic monitors. If $m \stackrel{t}{\Rightarrow} d$ and $m' \stackrel{t}{\Rightarrow} d'$, then $d \sim d'$.*

Proof. First, notice that $x \sim p_x$, since all derivations from x are either trivial or must start with $x \stackrel{\tau}{\rightarrow} p_x$. For the same reason, $\texttt{rec}\ x.\ m \sim m$. Therefore, by Corollary 14, if $r \stackrel{\tau}{\rightarrow} r'$, then $r \sim r'$. Now we can prove the lemma by induction on $|t|$.

If $t = \epsilon$, then $m \Rightarrow d$ and $m' \Rightarrow d'$, so (by the observation above) $d \sim m \sim m' \sim d'$.

If $t = \alpha t'$, then we demonstrate that if $m \stackrel{\alpha}{\Rightarrow} r$ and $m' \stackrel{\alpha}{\Rightarrow} r'$, then $r \sim r'$ and, by induction, we are done. In fact, by our observations, it is enough to prove that if $m \stackrel{\alpha}{\rightarrow} r$ and $m' \stackrel{\alpha}{\rightarrow} r'$, then $r \sim r'$. Thus, let $m \stackrel{\alpha}{\rightarrow} r$ and $m' \stackrel{\alpha}{\rightarrow} r'$; then, m is a sum of $\alpha.r$ and of no other $\alpha.f$ (because m is deterministic) and m' is a sum of $\alpha.r'$ and of no other $\alpha.f$. If $r \nsim r'$, then there is a trace s and value v, such that $r \stackrel{s}{\Rightarrow} v$ and $r' \stackrel{s}{\not\Rightarrow} v$ (or $r' \stackrel{s}{\Rightarrow} v$ and $r \stackrel{s}{\not\Rightarrow} v$, but this case is symmetric). Therefore, $m \stackrel{\alpha s}{\Rightarrow} v$, but since all derivations from m' on trace αs must start from $m' \stackrel{\alpha}{\rightarrow} r'$, if $m' \stackrel{\alpha s}{\Rightarrow} v$, also $r' \stackrel{s}{\Rightarrow} v$, so $m \nsim m'$, a contradiction. Therefore, $r \sim r'$ and the proof is complete. □

Corollary 15. *If m is deterministic, $m \stackrel{t}{\Rightarrow} d$, and $m \stackrel{t}{\Rightarrow} d'$, then $d \sim d'$.*

Corollary 16. *If m is deterministic, $m \stackrel{t}{\Rightarrow} d$, and $m \stackrel{t}{\Rightarrow} v$, where v is a value, then $d = v$.*

Lemma 27. *If m is a deterministic monitor and d is a submonitor of m, such that d is a sum of some p_x, then $d = p_x$.*

Proof. By the definition of deterministic monitors, $r + p_x$ is not allowed. □

Corollary 17. *In a derivation $m \stackrel{t}{\Rightarrow} x$, such that x is bound in m and m is deterministic, p_x must appear.*

Proof. Combine Lemmata 22 and 27. □

Lemma 28. *Let $m \stackrel{t}{\Rightarrow} d$, such that t is not simple for m and m is deterministic. Then, there are $t = xuz$ and monitor r, such that $|u| > 0$ and $m \stackrel{x}{\Rightarrow} r \stackrel{u}{\Rightarrow} r \stackrel{z}{\Rightarrow} d$.*

Proof. Similar to the proof of Lemma 23, but use Corollary 17, instead of Lemma 22. □

Theorem 7. *Let m be a deterministic monitor for M_n. Then, $|m| = 2^{2^{\Omega(n)}}$.*

Proof. We construct a set of simple traces of size $\Omega((2^{n/3-1}-1)!)$ and by Lemma 21 this is enough to prove our claim. Let $k = \lfloor n/3 \rfloor - 1$ (actually, for simplicity we assume $n = 3k+3$, as this does not affect our arguments); let

$$A = \{a \in \{0,1\}^k \mid \text{ at least one 1 appears in } a\},$$

let

$$B = \{1a10^k1a \in \{0,1\}^n \mid a \in A\},$$

and let

$$G = \{x \in \{0,1\}^* \mid x \sqsubseteq g, \text{ where } g \text{ is a permutation of } B\};$$

then, $|A| = |B| = 2^k - 1$ and $|G| > |B|! = 2^{\Omega(|B|\log|B|)} = 2^{\Omega(k\cdot 2^k)} = 2^{2^{\Omega(n)}}$. It remains to demonstrate that all permutations g of B, are simple traces for m. Let g be such a permutation. Let $h = (2^k-1)!$ and $g = g_1 g_2 \cdots g_h$, where $g_1, g_2, \ldots, g_h \in B$ and for all such g_i, let $g_i = 1b_i10^k1b_i$.

Notice that g is designed so that in every subsequence of length n of g, 0^k can appear at most once, specifically as the area of 0^k in the middle of a $g_i \in B$ as defined. If 0^k appears in an area of k contiguous positions, then that area cannot include one of the separating 1's, so it must be an $a \in A$, which cannot happen by the definition of A, or the area of 0^k in the middle of a $b \in B$ as defined. If there is another area of 0^k closer than $2k+3$ positions away, again, it must be some $a \in A$, which cannot be the case. Furthermore, and because of this observation, for every $a \in A$, 0^k1a and $a10^k$ appear exactly once in g.

For traces x, y, we say that $x \sim y$ if for every trace z,

$$xz \in M_n \text{ iff } yz \in M_n.$$

For trace x, if $|x| \geq n$, we define $l(x)$ to be such that $|l(x)| = n$ and there is $x'l(x) = x$; if $|x| < n$, we define $l(x) = 0^{n-|x|}x$. Notice that for traces x, y, $x \sim y$ iff $l(x) = l(y)$. Also, that if $|x| \geq n$, $l(x)$ must be in one of the three forms below:

1. $l(x) = 0^{n_1}1a_11a_210^{n_2}$ for some $n_1 + n_2 = k$ and $a_1, a_2 \in A$ (in this case, $10^k1a_11a_210^{n_2} = 10^{n_2}l(x)$ are the last $n+n_2$ positions of x), or
2. $l(x) = d_11a_110^k1d_2$ for some $d_2 \sqsubseteq a_1 \in A$, or
3. $l(x) = d_110^k1a_1d_2$ for some $a'd_1 = a_1 \in A$.

Claim: For $x, y \sqsubseteq g$, if $x \sim y$, then $x = y$. Otherwise, there are $x, y \sqsubseteq g$, such that $l(x) = l(y)$, but $x \neq y$. We have the following cases:

$|x|, |y| \leq n$: in this case, there are $n_1, n_2 \leq n$, such that $0^{n_1}x = 0^{n_2}y$; because x and y start with 1, then $n_1 = n_2$ and $x = y$.

$|x| < n < |y|$: y must be in one of the forms described above, so if $l(y) = 0^{n_1}1a_11a_210^{n_2}$, then $x = 1a_11a_210^{n_2}$, which is a contradiction, because x is not in an appropriate form (right after $1a_11$, there should be $0^k \neq a_2$);
if $l(x) = l(y) = d_11a_110^k1d_2$, then 10^k1 must appear exactly once in x, so $d_1 = 0^{n_1}$ and $a_1 = b_1$, meaning that y is an initial fragment of g, thus $d_1 = \epsilon$ a contradiction, because $l(x)$ starts with 0;
if $l(x) = d_110^k1a_1d_2$, then we already have a contradiction, because there is some $|a'| > k+1$, such that x starts with $a'10^k1$, so $d_1 = a'$, but $|d_1| \leq k$.

$|x|, |y| \geq n$: $l(x) = l(y)$, so they must be of the same form; if $l(x) = l(y) = 0^{n_1}1a_11a_210^{n_2}$, then $10^k1a_11a_210^{n_2}$ are the last $n+n_2$ positions of x and of y, so if $x \neq y$, then we found two different places in g where 10^k1a_1 appears, a contradiction; the cases of the other forms are similar.

Now, if g is not simple, then by Lemma 28, there are $x \sqsubset y \sqsubseteq g$, such that $m \stackrel{x}{\Rightarrow} d$ and $m \stackrel{y}{\Rightarrow} d$. Furthermore, by Corollary 15, if $m \stackrel{x}{\Rightarrow} q_1$ and $m \stackrel{y}{\Rightarrow} q_2$, $q_1 \sim d \sim q_2$. If $xz \in M_n$, then $m \stackrel{x}{\Rightarrow} r \stackrel{z}{\Rightarrow} \texttt{yes}$, so $m \stackrel{y}{\Rightarrow} r'$, where $r' \sim r$, so $r' \stackrel{z}{\Rightarrow} \texttt{yes}$, meaning that $yz \in M_n$. Similarly, if $yz \in M_n$, then $xz \in M_n$, therefore $x \sim y$. Finally, since $x \neq y$ and $x \sim y$, we have a contradiction by the claim we proved above, so g is simple and the proof complete. □

Language M_n above demonstrates an exponential gap between the state size of NFAs, (DFAs,) nondeterministic monitors, and deterministic monitors. Therefore, the upper bounds provided by Theorem 6 and Corollary 6 cannot be improved significantly. On the other hand, it is not clear what the gap between a nondeterministic monitor and an equivalent deterministic one has to be. Corollary 8 informs us that for every monitor of size n, there is an equivalent deterministic one of size $2^{O(2^n)}$; on the other hand, Theorem 7 presents a language (namely M_n) which is recognized by a (nondeterministic) monitor of size $k\ (= O(2^n))$, but by no deterministic monitor of size $2^{o(k)}$. These bounds are significantly different, as there is an exponential gap between them, and they raise the question whether there is a more sophisticated (and efficient) procedure than turning a monitor into an NFA, then using the usual subset construction, and then turning the resulting DFA back into a monitor, as was done in Corollary 8. Theorem 8 demonstrates that the upper bound of Corollary 8 cannot be improved much.

Theorem 8. *For every $n \in \mathbb{N}$, there is an irrevocable regular language on two symbols which is recognized by a nondeterministic monitor of size $O(n)$ and by no deterministic monitor of size $2^{2^{o\left(\sqrt{n \log n}\right)}}$.*

Proof. For a word $t \in \{0,1\}^*$, we define the projections of t, $t|_0$ and $t|_1$ to be the result of removing all 1's, respectively all 0's, from t: for $i \in \{0,1\}$, $\epsilon|_i = \epsilon$, $it'|_i = i(t'|_i)$, and $(1-i)t'|_i = t'|_i$. For $n \in \mathbb{N}$, let

$$F(n) = \max_{m_1+\cdots+m_k=n} lcm(m_1, \ldots, m_k),$$

where $lcm(m_1, \ldots, m_k)$ is the least common multiple of $m_1, \ldots, m_k$, and $X(n) = \{m_1, \ldots, m_k\}$, where $m_1 + \cdots + m_k = n$ and $lcm(m_1, \ldots, m_k) = F(n)$.

The proof of Theorem 8 is based on a result by Chrobak [15] (errata at [16]), who demonstrated that for every natural number n, there is a unary language (a language over exactly one symbol) which is recognized by an NFA with n states, but by no DFA with $e^{o\left(\sqrt{n \log n}\right)}$ states. The unique symbol used can be (in our case) either 0 or 1. We can use 1, unless we make explicit otherwise. The unary language Chrobak introduced was

$$Ch_n = \{1^{cm} \mid m \in X(n),\ 0 < c \in \mathbb{N}\}.$$

For some $n \in \mathbb{N}$, let Ch_n^0 be Chrobak's language, where the symbol used is 0 and Ch_n^1 be the same language, but with 1 being the symbol used — so for $i \in \{0,1\}$, $Ch_n^i = \{i^{cm} \mid m \in X(n),\ 0 < c \in \mathbb{N}\}$. Now, let

$$U_n = \{xey \in \{0,1,e\}^* \mid x \in \{0,1\}^* \text{ and for some } i \in \{0,1\},\ x|_i \in Ch_n^i\}.$$

Fix some $n \in \mathbb{N}$ and let $X(n) = \{m_1, m_2, \ldots, m_k\}$. U_n can be recognized by the monitor $p_0 + p_1$ of size $O(n)$, where for $\{i, \bar{i}\} = \{0,1\}$, $p_i = p_i^1 + p_i^2 + \cdots + p_i^k$, where for $1 \le j \le k$, p_i^j is the monitor defined recursively in the following way. Let

$$p_i^j[m_j] = \mathtt{rec}\, x_{m_j}.\ (\bar{i}.x_{m_l} + i.x_1 + e.\mathtt{yes});$$

for $0 \le l < m_j$, let

$$p_i^j[l] = \mathtt{rec}\, x_l.\ (\bar{i}.x_l + i.p_i^j[l+1]);$$

finally, let $p_i^j = p_i^j[0]$. That is, after putting everything together and simplifying the variable indexes,

$$p_i^j = \mathtt{rec}\, x_0.\ (\bar{i}.x_0 + \underbrace{i.\mathtt{rec}\, x_1.\ (\bar{i}.x_1 + \cdots i.\mathtt{rec}\, x_{m_j}.\ (\bar{i}.x_{m_j}}_{m_j} + i.x_1 + e.\mathtt{yes}) \cdots))).$$

Monitor p_i^j essentially ignores all appearances of $\bar{i}$ and counts how many times i has appeared. If i has appeared a multiple of m_j times, then p_i^j is given a chance to reach verdict $\mathtt{yes}$ if e then appears; otherwise it continues counting from the beginning. That the size of $p_0 + p_1$ is $O(n)$ is evident from the fact that for every i, j, $|p_i^j| = O(m_j)$, which is not hard to calculate since $\left|p_i^j[m_j]\right| = 9$ and for $l < m_j$, $\left|p_i^j[l]\right| = \left|p_i^j[l+1]\right| + 5$.

Let m be a deterministic monitor for U_n and $t \in \{0,1\}^*$. To complete the proof of the theorem, it suffices to prove for some constant $c > 0$ that if $|t| < e^{c \cdot \sqrt{n \log n}}$, then t must be simple. Indeed, proving the above would mean that we have constructed a simple set of traces of cardinality more than $2^{e^{c \cdot \sqrt{n \log n}}}$ — that is, the set of traces on $\{0,1\}$ of length less than $e^{c \cdot \sqrt{n \log n}}$ — which, by Lemma 21, gives the same lower bound for $|m|$. Specifically, we prove that if t is not simple for m, then there is a DFA for Chrobak's unary language, Ch_n, of at most $|t|$ states, so by Chrobak's results it cannot be that t is not simple and $|t| < e^{c \cdot \sqrt{n \log n}}$.

So, let t be a shortest non-simple trace on $\{0,1\}$. Since t is not simple and is minimal, $t = t_1 t_2 i$, where $i = 0$ or 1 and $m \xRightarrow{t_1} r \xRightarrow{t_2} s \xRightarrow{i} r$ (by Lemma 28). Without loss of generality, we assume $i = 1$. Let A be the DFA $(Q, \{1\}, \delta, m, F)$, where Q

is the set of all monitors appearing in the derivation $m \overset{t_1}{\Rightarrow} r \overset{t_2}{\Rightarrow} s \overset{1}{\Rightarrow} r$, $\delta(q_1, 1) = q_2$ iff $q_1 \left(\overset{0}{\Rightarrow}\right)^* \left(\overset{\tau}{\rightarrow}\right)^* \overset{1}{\rightarrow} q_2$ is part of the derivation above, and

$$F = \{d \in Q \mid \text{for all } c \geq 0,\ d \overset{0^c e}{\Longrightarrow} \texttt{yes}\}.$$

Notice that A is, indeed, deterministic, since transitions move along the derivation and all monitors in Q transition exactly once in $m \overset{t_1}{\Rightarrow} r \overset{t_2}{\Rightarrow} s \overset{1}{\Rightarrow} r$ – so the first $\overset{1}{\rightarrow}$ transition that appears after a state/monitor in the derivation exists and is unique.

We claim that A accepts Ch_n. By the definition of δ, if the run of A on w reaches state (submonitor) d, then there is some $w^t \in \{0,1\}^*$, such that $w^t|_1 = w$ and $m \overset{w^t}{\Longrightarrow} d$. Then,

$$A \text{ accepts } w \quad \text{iff} \quad d \in F \quad \text{iff} \quad \text{for all } c \geq 0,\ d \overset{0^c e}{\Longrightarrow} \texttt{yes} \quad \text{iff}$$

for all $c \geq 0$, $m \overset{w^t 0^c e}{\Longrightarrow} \texttt{yes}$ (by Corollary 16) iff for every $c \geq 0$, $w^t 0^c e \in U_n$ iff $w \in Ch_n$.

Finally, although we promised that we would only use two symbols, we have used three. However, a language of three symbols can easily be encoded as one of two symbols, using the mapping: $0 \mapsto 00$, $1 \mapsto 01$, and $e \mapsto 11$. We can encode U_n like this and simply continue with the remaining of the proof. □

Notice that the lower bound given by Theorem 8 depends on the assumption that we can use two symbols in our regular language; this can be observed both from the statement of the theorem and from its proof, which makes non-trivial use of the two symbols. So, a natural question to ask is whether the same bounds hold for NFAs and monitors on one symbol.

Consider an irrevocable regular language on one symbol. If k is the length of the shortest word in the language, then we can immediately observe two facts. The first is that the smallest NFA which recognizes the language must have at least $k+1$ states (and indeed, $k+1$ states are enough). The second fact we can observe is that there is a deterministic monitor of size exactly $k+1$ which recognizes the language: $1^k.\texttt{yes}$. Therefore, things are significantly easier when working with unary languages.

Corollary 18. *If there is an irrevocable NFA of n states which recognizes unary language L, then, there is a deterministic monitor of size at most n which recognizes L.*

5. Determinizing with two verdicts

In Section 4 we have dealt with monitors which can only reach a positive verdict or a negative one but not both. This was mainly done for convenience, since a single-verdict monitor is a lot easier to associate with a finite automaton and it helped simplify several cases. It is also worth mentioning that, as demonstrated in [23], to monitor for μHML properties, we are interested in single-verdict monitors. In this section, we demonstrate how the constructions and bounds of Section 4 transfer to the general case of monitors.

First, notice that there is no deterministic monitor equivalent to the monitor $m_c = \alpha.\texttt{yes} + \alpha.\texttt{no}$, also defined in Subsection 2.3, since m_c can reach both verdicts with the same trace. Thus, there are monitors, which are not, in fact, equivalent to deterministic ones. These are the ones for which there is a trace through which they can transition to both verdicts. We call these monitors *conflicting*.

To treat non-conflicting monitors, we reduce the problem to the determinization of single-verdict monitors. For this, we define two transformations, very similar to what we did in Section 3 to reduce the determinization of monitors to the determinization of processes. Let $[no]$ be a new action, not in Act. We define ν in the following way:

$$\nu(v) = v, \text{ if } v \in \{\texttt{yes}, \texttt{end}\},$$
$$\nu(\texttt{no}) = [no].\texttt{yes},$$
$$\nu(x) = x,$$
$$\nu(\alpha.m) = \alpha.\nu(m),$$
$$\nu(m+n) = \nu(m) + \nu(n), \text{ and}$$
$$\nu(\texttt{rec}x.\, m) = \texttt{rec}x.\, \nu(m).$$

We also define ν^{-1}: if s is a sum of $[no].\texttt{yes}$, then $\nu^{-1}(s) = \texttt{no}$ and otherwise,

$$\nu^{-1}(v) = v, \text{ if } v \in \{\texttt{yes}, \texttt{end}\},$$
$$\nu^{-1}(x) = x,$$
$$\nu^{-1}(\alpha.m) = \alpha.\nu^{-1}(m),$$

$$\nu^{-1}(m+n) = \nu^{-1}(m) + \nu^{-1}(n), \text{ and}$$
$$\nu^{-1}(\mathtt{rec}x.\, m) = \mathtt{rec}x.\, \nu^{-1}(m).$$

Lemma 29. *Let $m' = \nu(m)$. Then, $m \overset{t}{\Rightarrow} \mathtt{no}$ if and only if there is some $t' \sqsubseteq t$, such that $m' \overset{t'}{\Rightarrow} \mathtt{no}.r$.*

Proof. Straightforward induction on the derivations. □

Lemma 30. *Let $m' = \nu^{-1}(m)$, where verdict $\mathtt{no}$ does not appear in m. Then, there is some $t' \sqsubseteq t$ and a sum s of $[no].\mathtt{yes}$, such that $m \overset{t'}{\Rightarrow} s$, if and only if $m' \overset{t}{\Rightarrow} \mathtt{no}$.*

Proof. Straightforward induction on the derivations. □

Theorem 9. *Let m be a monitor which is not conflicting. Then, there is an equivalent deterministic one of size $2^{2^{O(|m|)}}$.*

Proof. Let m be a monitor which is not conflicting, but uses both verdicts $\mathtt{yes}$ and $\mathtt{no}$. By Lemma 1, we can assume that there are no sums of $\mathtt{no}$ in m. Let m' be the result of replacing $\mathtt{no}$ in m by $[no].\mathtt{yes}$, where $[no]$ is a new action not appearing in m. Then, for some constant $c > 0$, m' is a single-verdict monitor of size $c \cdot |m|$, so as we have shown in the preceding sections (Corollary 8), m' is equivalent to a deterministic monitor n' of size $2^{O(2^{c\cdot|m|})}$. Let n be the result of replacing all maximal sums of $[no].\mathtt{yes}$ by $\mathtt{no}$. Then, n is deterministic, because there are no sums of $\mathtt{no}$ (they would have been replaced) and all other sums remain in the form required by deterministic monitors. It remains to demonstrate that $m \sim n$, which we now do. Let $t \in \text{Act}^*$. Then,

$m \overset{t}{\Rightarrow} \mathtt{no}$
iff $m \overset{t}{\Rightarrow} \mathtt{no}$ and $m \overset{t}{\not\Rightarrow} \mathtt{yes}$ (m is not conflicting)
iff there is a $t' \sqsubseteq t$ and $\alpha \in \text{Act}$, such that $m' \overset{t'}{\Rightarrow} [no].\mathtt{yes}$ and $m' \overset{t'\alpha}{\not\Longrightarrow} \mathtt{yes}$ (by Lemma 29)
iff there is a $t' \sqsubseteq t$ and $\alpha \in \text{Act}$, such that $m' \overset{t'[no]}{\Longrightarrow} \mathtt{yes}$ and $m' \overset{t'\alpha}{\not\Longrightarrow} \mathtt{yes}$ (there are no sums of $\mathtt{no}$ in m)
iff there is a $t' \sqsubseteq t$ and $\alpha \in \text{Act}$, such that $n' \overset{t'[no]}{\Longrightarrow} \mathtt{yes}$ and $n' \overset{t'\alpha}{\not\Longrightarrow} \mathtt{yes}$ (m' and n' are equivalent)
iff there is a $t' \sqsubseteq t$, an $\alpha \in \text{Act}$, and some $s \neq \mathtt{yes}$, such that $n' \overset{t'}{\Rightarrow} s \overset{[no]}{\Longrightarrow} \mathtt{yes}$ and $n' \overset{t'\alpha}{\not\Longrightarrow} \mathtt{yes}$
iff there is a $t' \sqsubseteq t$, an $\alpha \in \text{Act}$, and a sum s of $[no].\mathtt{yes}$, such that $n' \overset{t'}{\Rightarrow} s$ and $n' \overset{t'\alpha}{\not\Longrightarrow} \mathtt{yes}$
iff there is a $t' \sqsubseteq t$ and $\alpha \in \text{Act}$, such that $n \overset{t'}{\Rightarrow} \mathtt{no}$ and $n \overset{t'\alpha}{\not\Longrightarrow} \mathtt{yes}$ (by Lemma 30)
iff $n \overset{t}{\Rightarrow} \mathtt{no}$.
The case of the $\mathtt{yes}$ verdict is straightforward. □

Naturally, the same lower bounds as for single-verdict monitors hold for the general case of monitors as well.

Conflicting monitors

We demonstrated in this section how we can determinize any non-conflicting monitor. A deterministic and conflicting monitor is a contradiction, as it would have to deterministically reach two different verdicts on the same trace. It would be good, therefore, to be able to detect conflicting monitors. Here we sketch how this can be done using nondeterministic logarithmic space.

For any monitor m, let $G_m = (V, E)$ be a graph, such that

$$V = \{(m, d) \mid m, d \text{ submonitors of } m\} \text{ and}$$
$$E = \{(m, d, m', d') \in V^2 \mid \exists \alpha \in \text{Act}.\ m \overset{\alpha}{\Rightarrow} m' \text{ and } d \overset{\alpha}{\Rightarrow} d'\}.$$

Then, m is conflicting iff there is a path from (m, m) to $(\mathtt{yes}, \mathtt{no})$ in G_m. This is a subproblem of the st-connectivity problem, known to be in NL and solvable in time $O(|V| + |E|) = O(|m|^4)$ (by running a search algorithm in G_m).

6. Conclusions

We have provided three methods for determinizing monitors. One of them is by reducing the problem to the determinization of processes, which has been handled by Rabinovich in [47]; another is by using Rabinovich's methods directly on the formulae of μHML, bringing them to a deterministic form, and then employing Francalanza et al.'s monitor synthesis method from [23,24], ending up with a deterministic monitor; the last one transforms a monitor into an NFA, uses the classical subset construction from Finite Automata Theory to determinize the NFA, and then, from the resulting DFA constructs a deterministic monitor.

The first method is probably the simplest and directly gives results, at the same time describing how the behavior of monitors is linked to processes. The second method is more explicit, in that it directly applies Rabinovich's methods; furthermore, it is used directly on a μHML formula and helps us relate the (non-)deterministic behavior of monitors to the form of the formula they were constructed to monitor. The third method links monitors to finite automata and allows us to extract more precise bounds on the size of the constructed deterministic monitors.

Monitors are central for the runtime verification of processes. We have focused on monitors for μHML, as constructed in [23,24]. We showed that we can add a runtime monitor to a system without having a significant effect on the execution time of the system. Indeed, in general, evaluating a nondeterministic monitor of size n for some specific trace may amount to keeping track of all possible monitor states reachable along that trace. Computing each transition can take time up to n^2, because for up to n submonitors we may need to transition to up to n submonitors. By using a deterministic monitor, each transition is provided explicitly by the monitor description, so we can transition immediately at every step along a trace — with a cost depending on the implementation. This speed-up can come at a severe cost, though, since we may have to use up to doubly-exponential more space to store the monitor, and even stored in a more efficient form as its LTS, the deterministic monitor may require exponential extra space.

Summary of bounds We were able to prove a number of upper and lower bounds for several constructions. Here we give a summary of the bounds we have proven, the bounds which are known, and the ones we can further infer from these results.

- Corollary 8 (actually, Theorem 9 for the general case) informs us that from a nondeterministic monitor of size n, we can construct a deterministic one of size $2^{O(2^n)}$.
- Theorem 8 explains that we cannot do much better, because there is an infinite family of monitors, such that for any monitor of size n in the family, there is no equivalent deterministic monitor of size $2^{2^{o(\sqrt{n \log n})}}$.
- As for when we start with an NFA, it is a classical result that an NFA of n states is equivalent to a DFA of 2^n states [46]; furthermore, this bound is tight [42].
- Theorem 6 informs us that an irrevocable NFA of n states can be converted to an equivalent monitor of size $2^{O(n \log n)}$.
- Proposition 2 reveals that there is an infinite family of NFAs, for which every NFA of the family of n states is not equivalent to any monitor of size $2^{o(n)}$.
- Corollary 8 yields that an irrevocable NFA of n states can be converted to an equivalent deterministic monitor of size $2^{O(2^n)}$; Theorem 7 makes this bound tight.
- Corollary 6 allows us to convert a DFA of n states to a deterministic monitor of $2^{O(n)}$ states; Theorem 7 makes this bound tight.
- This $2^{O(n)}$ is also the best upper bound we have for converting a DFA to a (general) monitor; it is unclear at this point what lower bounds we can establish for this transformation.
- We can convert a (single-verdict) monitor of size n to an equivalent DFA of $O(2^n)$ states, by first converting the monitor to an NFA of n states (Proposition 1) and then using the classical subset construction.
- If we could convert any monitor of size n to a DFA of $2^{o(\sqrt{n \log n})}$ states, then we could use the construction of Corollary 6 to construct a deterministic monitor of $2^{2^{o(\sqrt{n \log n})}}$ states, which contradicts the lower bound of Theorem 8; therefore, $2^{\Omega(\sqrt{n \log n})}$ is a lower bound for converting monitors to equivalent DFAs.
- Using Lemma 5, we can reduce the problem of determinizing monitors to the determinization of processes (up to trace-equivalence); by Theorem 8, this gives a lower bound of $2^{2^{\Omega(\sqrt{n \log n})}}$ for Rabinovich's construction in [47].
- Similarly, using the constructions of [23,24], one can convert a MHML formula into a monitor for it and a monitor into a MHML formula of the same size (or smaller). Therefore, we can conclude that the lower bounds for determinizing monitors also hold for determinizing MHML formulae as in Subsection 3.2. Hence, a CHML formula that holds precisely for the traces in language M_n from Section 4 must be of size $2^{\Omega(n)}$ and an equivalent deterministic CHML formula must be of size $2^{2^{\Omega(n)}}$. We conclude that, as a specification language, NFAs can be exponentially more succinct than the monitorable fragment of μHML and doubly exponentially more succinct than the deterministic monitorable fragment of μHML; DFAs can be exponentially more succinct than the deterministic monitorable fragment of μHML. We refer the interested reader to [6, Section 5] for more complexity bounds on translations between formulae in a variety of monitorable fragments of μHML in a linear-time setting.
- Corollary 18 informs us that it is significantly easier to convert an irrevocable NFA or monitor into a (deterministic) monitor when the alphabet we use (the set of actions) is limited to one element: when there is only one action/symbol, an irrevocable NFA of n states or nondeterministic monitor of size n can be converted into an equivalent deterministic monitor of size at most n.
- In Section 5, we have argued that detecting whether a monitor is conflicting can be done in time $O(n^4)$ or in nondeterministic space $O(\log n)$ (and thus, by Savitch's Theorem [48] in deterministic space $O(\log^2 n)$).

The bounds we were able to prove can be found in Table 6. We remark that the doubly exponential blow-up in converting NFAs into deterministic monitors is due to the syntactic representation of monitor expressions. Just like processes and formulae, one could describe (the DFA associated with) a deterministic monitor using systems of equations. The use of

Table 6
Bounds on the cost of construction and where they can be found (X signifies that the conversion is trivial).

From/to	DFA	Monitor	Det. monitor
NFA	`tight:` $O(2^n)$ from [46,42]	`upper:` $2^{O(n \log n)}$ by Theorem 6 `lower:` $2^{\Omega(n)}$ by Proposition 2	`tight:` $2^{O(2^n)}$ by Corollary 8, Theorem 7
DFA	X	`upper:` $2^{O(n)}$ by Corollary 6	`tight:` $2^{O(n)}$ by Corollary 6, Theorem 7
nondet. monitor	`upper:` $O(2^n)$ by Corollary 2, Theorem 1, [46] `lower:` $2^{\Omega(\sqrt{n \log n})}$ by Corollary 6 Theorem 8	X	`upper:` $2^{O(2^n)}$ by Theorems 1,6, Corollary 6; for dual-verdict 9 `lower:` $2^{2^{\Omega(\sqrt{n \log n})}}$ by Theorem 8

systems of equations leads to an exponentially more succinct representation because, unlike syntactic monitor descriptions, it allows one to express sharing. To our mind, one of the main contributions of our article is in spelling out exactly, and in proving, the succinctness price that has to be paid when representing monitors using classic process-algebraic syntax, such as the one we adopt. The use of syntax has its benefits, as it leads naturally to compositional definitions of monitor-synthesis functions and to proofs of properties of monitors by structural and rule induction. We trust that the results presented in this study can help researchers in choosing a suitable formalism for describing monitors that offers a good trade-off between usability and complexity.

Optimizations Monitors to be used for the runtime verification of processes are expected to not affect the systems they monitor as much as possible. Therefore, the efficiency of monitoring must be taken into account to restrict overhead. To use a deterministic monitor for a μHML property, we would naturally want to keep its size as small as possible. It would help to preserve space (and time for each transition) to store the monitor in its LTS form — as a DFA. We should also aim to use the smallest possible monitor we can. There are efficient methods for minimizing a DFA, so one can use these to find a minimal DFA and then turn it into monitor form using the construction from Theorem 6, if such a form is required. The resulting monitor will be (asymptotically) minimal:

Proposition 3. *Let A be a minimal DFA for an irrevocable language L, such that A has n states and there are at least Π paths in A originating at its initial state. Then, there is no deterministic monitor of size less than Π, which recognizes L.*

Proof. Since A is deterministic, all paths in A are completely described by a trace $t \in \text{Act}^*$. We show that for every deterministic monitor m which recognizes L, such a t is simple. Let t be the shortest trace which gives a path in A, but is not simple for m. By Lemma 28, there are $t'u = t$, such that $|u| > 0$ and $m \stackrel{t'}{\Rightarrow} r \stackrel{u}{\Rightarrow} r$. Since t represents a path in A and A is deterministic, A can reach state q with trace t and $q' \neq q$ with trace t'. Since A is a minimal DFA, there must be a trace s, such that (without loss of generality) A reaches an accepting state from q through s and a non-accepting state from q' through s. Therefore $ts \in L$ and $t's \notin L$, which is a contradiction, because by Corollary 16 and the observation above, $m \stackrel{ts}{\Rightarrow} \texttt{yes}$ iff $m \stackrel{t's}{\Rightarrow} \texttt{yes}$. □

As we see, DFA minimization also solves the problem of deterministic monitor minimization. On the other hand, it would be good to keep things small from an earlier point of the construction, before the exponential explosion of states of the subset construction takes place. In other words, it would be good to minimize the NFA we construct from the monitor, which can already be smaller than the original monitor. Unfortunately, under standard complexity-theoretic assumptions, NFA minimization is a hard problem — specifically PSPACE-complete [31] — and it remains NP-hard even for classes of NFAs which are very close to DFAs [13]. NFA minimization is even hard to approximate or parameterize [26,28]. Still, it would be better to use an efficient approximation algorithm from [28] to process the NFA and save on the number of states before we determinize. This raises the question of whether (nondeterministic) monitors are easier to minimize than NFAs, although a positive answer seems unlikely in light of the hardness results for NFA minimization.

The complexity bounds we present in this paper pertain to the worst-case scenarios that may arise in converting monitor descriptions between pairs of formalisms for their representation that we consider. An interesting avenue for future research is to conduct an experimental study of the efficiency of the various conversion procedures on benchmark or synthetic examples and to study the overhead on system performance of various monitor representations. A paradigmatic example of a study of this kind is given in [50]. In that paper, Tabakov, Rozier and Vardi argue for the algorithmic generation of "correct" monitors from properties and report on an experimental study of the generation of monitors that offer the best performance in terms of runtime overhead at simulation time.

Declaration of competing interest

The authors confirm that there are no known conflicts of interest associated with this publication and there has been no significant financial support for this work that could have influenced its outcome.

Acknowledgements

We thank the anonymous reviewers for their careful reading of our paper and the constructive suggestions that have led to several improvements.

Appendix A. Proofs from Section 3.2

Proof of Lemma 9. We define a function f as follows:

$$f(\varphi) = \{i \mid X_i \text{ occurs free and unguarded in } \varphi\}$$

where $X_1, X_2, \ldots$ are all the variables that can occur in the formulae.

Then formally our claim is that for each $\varphi \in$ sHML, there exists a formula, $\psi \in$ sHML such that

$$\varphi \equiv \psi \wedge \bigwedge_{i \in f(\varphi)} X_i$$

where $f(\psi) = \emptyset$.

We use induction on the size of φ to prove this claim and go through each case below.

$\varphi \in \{\mathtt{tt}, \mathtt{ff}\}$: This case holds trivially since $f(\varphi) = \emptyset$ and

$$\varphi \equiv \varphi \wedge \bigwedge_{i \in \emptyset} X_i.$$

$\varphi = X_j$: This case holds trivially since $f(\varphi) = \{j\}$ and

$$\varphi \equiv \mathtt{tt} \wedge \bigwedge_{i \in \{j\}} X_i.$$

$\varphi = [\alpha]\varphi'$: Since φ is prefixed with the $[\alpha]$ operator, all variables are guarded in φ and so $f(\varphi) = \emptyset$ and

$$\varphi \equiv \varphi \wedge \bigwedge_{i \in \emptyset} X_i.$$

$\varphi = \varphi_1 \wedge \varphi_2$: By the induction hypothesis, there exist formulae $\psi_1, \psi_2 \in$ sHML such that

$$f(\psi_1) = f(\psi_2) = \emptyset,$$
$$\varphi_1 \equiv \psi_1 \wedge \bigwedge_{i \in f(\varphi_1)} X_i,$$
$$\text{and } \varphi_2 \equiv \psi_2 \wedge \bigwedge_{i \in f(\varphi_2)} X_i.$$

Using the fact that $\varphi \wedge \varphi \equiv \varphi$ for each formula $\varphi \in \mu$HML, we have

$$\begin{aligned}
\varphi &\equiv \varphi_1 \wedge \varphi_2 \\
&\equiv \left(\psi_1 \wedge \bigwedge_{i \in f(\varphi_1)} X_i\right) \wedge \left(\psi_2 \wedge \bigwedge_{i \in f(\varphi_2)} X_i\right) \\
&\equiv (\psi_1 \wedge \psi_2) \wedge \bigwedge_{i \in f(\varphi_1)} X_i \wedge \bigwedge_{i \in f(\varphi_2)} X_i \\
&\equiv (\psi_1 \wedge \psi_2) \wedge \bigwedge_{i \in f(\varphi_1) \cup f(\varphi_2)} X_i
\end{aligned}$$

and since $f(\psi_1) = f(\psi_2) = \emptyset$, we have $f(\psi_1 \wedge \psi_2) = \emptyset$.

Each free and unguarded variable in φ must either be free and unguarded in φ_1 or φ_2 and each such variable in φ_1 or φ_2 must also be free and unguarded in φ. This gives us $f(\varphi_1) \cup f(\varphi_2) = f(\varphi)$ and so we have

$$\varphi \equiv (\psi_1 \wedge \psi_2) \wedge \bigwedge_{i \in f(\varphi)} X_i.$$

$\varphi = \max X_j.\, \varphi'$**:** By the induction hypothesis, there exists a formula $\psi \in$ sHML such that

$$\varphi' \equiv \psi \wedge \bigwedge_{i \in f(\varphi')} X_i$$

where $f(\psi) = \emptyset$.

We use the following equivalences:

$$\max X.\, \vartheta \equiv \vartheta[\max X.\, \theta / X] \tag{A.1}$$

$$\max X.\, (\vartheta \wedge X) \equiv \max X.\, \vartheta \tag{A.2}$$

$$Y[\vartheta / X] \equiv Y \quad \text{where } X \neq Y \tag{A.3}$$

From this we get:

$$\begin{aligned}
\varphi &\equiv \max X_j.\, \varphi' \\
&\equiv \max X_j.\, \left(\psi \wedge \bigwedge_{i \in f(\varphi') \setminus \{j\}} X_i \right) \\
&\equiv \left(\psi \wedge \bigwedge_{i \in f(\varphi') \setminus \{j\}} X_i \right) \left[\max X_j.\, \left(\psi \wedge \textstyle\bigwedge_{i \in f(\varphi') \setminus \{j\}} X_i \right) \Big/ X_j \right] \\
&\equiv \psi \left[\max X_j.\, \left(\psi \wedge \textstyle\bigwedge_{i \in f(\varphi') \setminus \{j\}} X_i \right) \Big/ X_j \right] \wedge \bigwedge_{i \in f(\varphi') \setminus \{j\}} X_i.
\end{aligned}$$

Since each variable in ψ is guarded, substituting a variable for a formula will not introduce unguarded variables and so

$$f\left(\psi \left[\max X_j.\, \left(\psi \wedge \textstyle\bigwedge_{i \in f(\varphi') \setminus \{j\}} X_i \right) \Big/ X_j \right] \right) = \emptyset.$$

The variables in φ that are free and unguarded are exactly the ones that are free and unguarded in φ', excluding X_j and so we have

$$f(\varphi) = f(\varphi') \setminus \{j\}.$$

This gives us:

$$\varphi \equiv \psi \left[\max X_j.\, \left(\psi \wedge \textstyle\bigwedge_{i \in f(\varphi') \setminus \{j\}} X_i \right) \Big/ X_j \right] \wedge \bigwedge_{i \in f(\varphi)} X_i. \quad \square$$

Proof of Lemma 11. We use structural induction to show how we can construct a system of equations from a formula φ that is in standard form. As shown in Lemma 9, given a formula $\vartheta \in$ sHML we can define an equivalent formula ϑ' where each free and unguarded variable is at the top level. We can therefore assume that for each fixed point $\max X.\, \psi$ that occurs as a subformula in φ, each free and unguarded variable in ψ is at the top level of ψ and using the equivalence $\max X.\, (\vartheta \wedge X) \equiv \max X.\, \vartheta$, we can also assume that X does not appear at the top level of ψ. Furthermore, we assume that if $\varphi_1 \wedge \psi_2$ appears as a subformula of φ, then there is no variable which is free in φ_1 and bound in φ_2 (or vice-versa).

We now go through the base cases and each of the top level operators that can occur in φ.

$\varphi = \mathtt{tt}$**:** We define a system of equations $SYS = (\{X = \mathtt{tt} \wedge \mathtt{tt}\}, X, \emptyset)$. Since $\bigwedge_{j \in \emptyset} \vartheta_j \equiv \mathtt{tt}$, SYS is in standard form and is equivalent to φ.

$\varphi = \mathtt{ff}$**:** We define a system of equations $SYS = (\{X = \mathtt{ff}\}, X, \emptyset)$. SYS is in standard form and is equivalent to φ.

$\varphi = Y$**:** We define a system of equations $SYS = (\{X = Y \wedge \mathtt{tt}\}, X, \{Y\})$. SYS is in standard form and is equivalent to φ.

$\varphi = [\alpha]\psi$**:** By the induction hypothesis, there exists a system of equations, $SYS = (Eq, X_1, \mathcal{Y})$ that is equivalent to ψ and is in standard form.

We define a new system of equations

$$SYS' = (Eq \cup \{X_0 = [\alpha]X_1 \wedge \mathtt{tt}\}, X_0, \mathcal{Y})$$

Each equation from SYS' is in standard form and so SYS' is in standard form. Since SYS is equivalent to ψ and X_0 does not appear in SYS (so the first equation does not affect the fixed-point calculations),

$$\begin{aligned}
[\![SYS', \rho]\!] &= \\
&= [\![[\alpha]X_1, \rho[X_1 \mapsto [\![SYS, \rho]\!]]]\!] \\
&= \{p \mid p \stackrel{\alpha}{\Rightarrow} q \text{ implies } q \in [\![SYS, \rho]\!]\} \\
&= \{p \mid p \stackrel{\alpha}{\Rightarrow} q \text{ implies } q \in [\![\psi, \rho]\!]\} \\
&= [\![[\alpha]\psi, \rho]\!],
\end{aligned}$$

so SYS' is equivalent to $[\alpha]\psi$, which is φ.

$\varphi = \psi_1 \wedge \psi_2$**:** By the induction hypothesis, we know that there exist systems of equations $SYS_1 = (Eq_1, X_1, \mathcal{Y}_1)$ and $SYS_2 = (Eq_2, Z_1, \mathcal{Y}_2)$ that are equivalent to ψ_1 and ψ_2 respectively and are in standard form. Let $X_1 = F_1$ be the principal equation from SYS_1 and let $Z_1 = G_1$ be the principal equation from SYS_2.

We define a new system of equations

$$SYS = (Eq, X_0, \mathcal{Y}_1 \cup \mathcal{Y}_2),$$

where

$$Eq = Eq_1 \cup Eq_2 \cup \{X_0 = F_1 \wedge G_1\}.$$

Again, X_0 does not appear in SYS_1 or SYS_2, so it does not play a part in the fixed-point calculation; furthermore, for $X_i = F_i$ an equation in Eq_1, $X_i \notin \mathcal{Y}_2$, i.e. X_i cannot be a free variable (or appear at all) in SYS_2 and vice-versa. Therefore, for $i = 1, 2$ and $X_j = F_j$ an equation in $Eq_1 \cup Eq_2$, $[\![SYS_i, \rho[X_j \mapsto S_j]]\!] = [\![SYS_i, \rho]\!]$ (that is, $\rho(X_j)$ does not affect the computation of $[\![SYS_i, \rho]\!]$) and therefore $\rho[SYS_1][SYS_2] = \rho[SYS_2][SYS_1]$ and for $i = 1, 2$ and $j = 3 - i$,

$$\left[\!\!\left[SYS_i, \overline{\rho[SYS_i]}^{SYS_j}\right]\!\!\right] = \left[\!\!\left[SYS_i, \overline{\rho[SYS_j]}^{SYS_i}\right]\!\!\right] = [\![SYS_i, \rho[SYS_i]]\!]. \tag{A.4}$$

Finally,

$$\begin{aligned}
[\![SYS, \rho]\!] &= \\
&= \bigcup \left\{ S_0 \mid S_0 \subseteq \left[\!\!\left[F_1 \wedge G_1, \overline{\overline{\rho[X_0 \mapsto S_0]}^{SYS_1}}^{SYS_2} \right]\!\!\right] \right\} \\
&= \left[\!\!\left[F_1 \wedge G_1, \overline{\rho[SYS_1]}^{SYS_2} \right]\!\!\right] \qquad (X_0 \text{ does not appear anywhere}) \\
&= \left[\!\!\left[F_1, \overline{\rho[SYS_1]}^{SYS_2} \right]\!\!\right] \cap \left[\!\!\left[G_1, \overline{\rho[SYS_1]}^{SYS_2} \right]\!\!\right] \\
&= [\![SYS_1, \rho[SYS_2]]\!] \cap [\![SYS_2, \rho[SYS_1]]\!] \qquad (\text{by (A.4)}) \\
&= [\![SYS_1, \rho]\!] \cap [\![SYS_2, \rho]\!].
\end{aligned}$$

Since SYS_1 is equivalent to ψ_1 and SYS_2 is equivalent to ψ_2, SYS is equivalent to $\varphi = \psi_1 \wedge \psi_2$.

Both $X_1 = F_1$ and $Z_1 = G_1$ are in standard form and so we can write them as

$$\begin{aligned}
F_1 &= \bigwedge_{j \in K_1} [\alpha_j]X_j \wedge \bigwedge_{j \in S_1} Y_j \\
G_1 &= \bigwedge_{j \in K'_1} [\alpha_j]Z_j \wedge \bigwedge_{j \in S'_1} Y_j
\end{aligned}$$

This allows us to rewrite the equation for X_0 as follows:

$$\begin{aligned}
X_0 &= F_1 \wedge G_1 \\
&= \bigwedge_{j \in K_1} [\alpha_j]X_j \wedge \bigwedge_{j \in S_1} Y_j \wedge \bigwedge_{j \in K'_1} [\alpha_j]Z_j \wedge \bigwedge_{j \in S'_1} Y_j
\end{aligned}$$

$$\equiv \left(\bigwedge_{j \in K_1'} [\alpha_j] X_j \wedge \bigwedge_{j \in K_1'} [\alpha_j] Z_j \right) \wedge \bigwedge_{j \in S_1 \cup S_1'} Y_j$$

Now SYS is in standard form and is equivalent to φ.

$\varphi = \max Y.\ \psi$**:** By the induction hypothesis, there exists a system of equations $SYS = (Eq, X_1, \mathcal{Y})$ that is equivalent to ψ and is in standard form.

If ψ does not contain Y, then $\varphi \equiv \psi$ which means φ is equivalent to SYS and we are done.

If ψ does contain Y then we define a new system of equation:

$$SYS' = (Eq \cup \{Y = F_1\}, Y, \mathcal{Y} \setminus \{Y\})$$

where $X_1 = F_1$ appears in SYS.

Let ρ be an environment. Then,

$$\begin{aligned}
[\![\varphi, \rho]\!] &= \\
&= \bigcup \{S_0 \mid S_0 \subseteq [\![\psi, \rho[X_0 \mapsto S_0]]\!]\} \\
&= \bigcup \{S_0 \mid S_0 \subseteq [\![SYS, \rho[X_0 \mapsto S_0]]\!]\} \\
&= \bigcup \{S_0 \mid S_0 \subseteq \left[\!\!\left[F_1, \overline{\rho[X_0 \mapsto S_0]}^{SYS} \right]\!\!\right]\} \\
&= [\![SYS', \rho]\!].
\end{aligned}$$

By our assumption that for each maximum fixed point $\max X.\ \psi$ in φ, X does not appear unguarded in ψ, we know that Y does not appear unguarded in F_1.

However in general we cannot guarantee that Y does not appear unguarded in the equations from SYS, since $Y \in \mathcal{Y}$. To overcome this, we replace each unguarded occurrence of Y with its corresponding formula F_1. Let $X_i = F_i$ be a formula that contains an unguarded occurrence of Y. Since $X_i = F_i$ is in standard form in SYS, we have

$$\begin{aligned}
X_i &= \bigwedge_{j \in K_i} [\alpha_j] X_j \wedge \bigwedge_{j \in S_i} Y_j \\
&\equiv \bigwedge_{j \in K_i} [\alpha_j] X_j \wedge \bigwedge_{j \in S_i \setminus \{t\}} Y_j \ \wedge Y_t
\end{aligned}$$

where $Y = Y_t$. We now change the equation for X_i by replacing the unguarded occurrence of Y_t with F_1 (by Lemma 10):

$$\begin{aligned}
X_i &= \bigwedge_{j \in K_i} [\alpha_j] X_j \wedge \bigwedge_{j \in S_i \setminus \{t\}} Y_j \ \wedge F_1 \\
&= \bigwedge_{j \in K_i} [\alpha_j] X_j \wedge \bigwedge_{j \in S_i \setminus \{t\}} Y_j \wedge \bigwedge_{j \in K_1} [\alpha_j] X_j \wedge \bigwedge_{j \in S_1} Y_j \\
&\equiv \bigwedge_{j \in K_i \cup K_1} [\alpha_j] X_j \wedge \bigwedge_{j \in (S_i \setminus \{j\}) \cup S_1} Y_j
\end{aligned}$$

(for simplicity, assume K_1, K_i are disjoint) and the i'th equation is in standard form.

Since $X_1 = F_1$ is in standard form in SYS, $Y = F_1$ is in standard form in SYS'. For all other equations from SYS, we can define equivalent equations that are in standard form in SYS' by replacing every unguarded occurrence of Y with F_1. All equations in SYS' are now in standard form and since SYS' is equivalent to φ, this case holds. □

Proof of Lemma 12. Let $SYS = (Eq, X_1, \mathcal{Y})$ be a system of n equations in standard form that is equivalent to a formula $\varphi \in$ sHML and

$$Eq = (X_1 = F_1,\ X_2 = F_2, \ldots,\ X_n = F_n).$$

We define some useful functions:

$$S(i) = \{\alpha_j \mid [\alpha_j] X_j \text{ is a sub formula in } F_i\}$$

$$D(i, \alpha) = \{r \mid [\alpha] X_r \text{ is a sub formula in } F_i\}$$

$$E(i) = \{r \mid Y_r \text{ is unguarded in } F_i\}$$

We also define these functions for subsets of indices $Q \subseteq \{1, 2, \ldots, n\}$:

$$S(Q) = \bigcup_{i \in Q} S(i),$$
$$D(Q, \alpha) = \bigcup_{i \in Q} D(i, \alpha), \text{ and}$$
$$E(Q) = \bigcup_{i \in Q} E(i).$$

For each equation $X_i = F_i$ where $F_i \neq \mathtt{ff}$, using these functions we can rewrite the equation as follows:

$$X_i = \bigwedge_{\alpha \in S(i)} \left([\alpha] \bigwedge_{j \in D(i,\alpha)} X_j \right) \wedge \bigwedge_{j \in E(i)} Y_i.$$

This equation may not be in deterministic form since it contains the conjunction of variables $\bigwedge_{j \in D(i,\alpha)} X_j$ and $D(i, \alpha)$ may not be a singleton. To fix this, we define a new variable X_Q for each subset $Q \subseteq \{1, 2, \ldots, n\}$, such that X_Q behaves like $\bigwedge_{j \in Q} X_j$ (in the sense that X_Q is defined in such a way that the formula associated with it in the resulting equation system is the conjunction of the formulae associated with the X_j's); we identify $X_{\{i\}}$ with X_i.

If for any $j \in Q$ we have $F_i = \mathtt{ff}$ then $\bigwedge_{j \in Q} F_j \equiv \mathtt{ff}$ and the equation for X_Q is $X_Q = \mathtt{ff}$. Otherwise, the equation for X_Q is:

$$X_Q = \bigwedge_{\alpha \in S(Q)} \left([\alpha] \bigwedge_{i \in D(Q,\alpha)} X_i \right) \wedge \bigwedge_{j \in E(Q)} Y_j \ (= F_Q).$$

We apply the following steps, each of which preserves the two conditions:

1. the resulting system is equivalent to the original system and
2. for every equation $X_A = F'_A$, where $A \subseteq \{1, 2, \ldots, n\}$, in the current system SYS and environment ρ, $[\![X_A, \rho[SYS]]\!] = [\![F_A, \rho[SYS]]\!]$. Note that it may be the case that $F'_A \neq F_A$, as we may have replaced F_A in the original equation for X_A by F'_A. This condition assures us that it doesn't matter, because F_A and F'_A are semantically equivalent.

Notice that condition 2 implies that $[\![X_A, \rho[SYS]]\!] = [\![\bigwedge_{i \in A} X_i, \rho[SYS]]\!]$.

Consider an equation

$$X_Q = \bigwedge_{\alpha \in S(Q)} \left([\alpha] \bigwedge_{i \in D(Q,\alpha)} X_i \right) \wedge \bigwedge_{i \in E(Q)} Y_i,$$

which is already in the system and is not in deterministic form. As a first step, if there is no equation for $X_{D(Q,\alpha)}$ in the system, then we introduce the equation for $X_{D(Q,\alpha)}$ as defined above (this gives an equivalent system, because $X_{D(Q,\alpha)}$ does not appear in any other equation if its own equation is not already in the system and condition 2 is preserved trivially).

We now proceed with the second step of the construction. Let $S_1(Q) \cup S_2(Q) = S(Q)$ be such that for every $\alpha \in S_1(Q)$, $Q = D(Q, \alpha)$ and for every $\alpha \in S_2(Q)$, $Q \neq D(Q, \alpha)$. Then, let SYS_0 be the result of removing the equation for X_Q from the system (and adding X_Q to the free variables) and SYS' the result of replacing it by

$$X_Q = \bigwedge_{\alpha \in S(Q)} [\alpha] X_{D(Q,\alpha)} \wedge \bigwedge_{j \in E(Q)} Y_i \ (= F'_Q).$$

We claim that SYS and SYS' are equivalent and after proving this claim we are done, because we can repeat these steps until all equations are in deterministic form and we are left with an equivalent deterministic system. To prove the claim, it is enough to prove that for every environment ρ, $[\![X_Q, \rho[SYS]]\!] = [\![X_Q, \rho[SYS']]\!]$ (by Lemma 6). Equivalently, we show that $A = B$, where

$$A = \bigcup \left\{ S \mid S \subseteq \left[\!\!\left[F_Q, \overline{\rho[X_Q \mapsto S]}^{SYS_0} \right]\!\!\right] \right\} = [\![X_Q, \rho[SYS]]\!]$$

and

$$B = \bigcup \left\{ S \mid S \subseteq \left[\!\!\left[F'_Q, \overline{\rho[X_Q \mapsto S]}^{SYS_0} \right]\!\!\right] \right\} = [\![X_Q, \rho[SYS']]\!].$$

Thus, it suffices to prove that $A \subseteq B$ and $B \subseteq A$. Notice that

$$A = \left[\!\left[F_Q , \overline{\rho[X_Q \mapsto A]}^{SYS_0} \right]\!\right],$$
$$B = \left[\!\left[F'_Q , \overline{\rho[X_Q \mapsto B]}^{SYS_0} \right]\!\right],$$

and that $\overline{\rho[X_Q \mapsto A]}^{SYS_0} = \rho[SYS]$ and $\overline{\rho[X_Q \mapsto B]}^{SYS_0} = \rho[SYS']$. Therefore, it suffices to prove:

$$A \subseteq \left[\!\left[F'_Q , \rho[SYS] \right]\!\right]$$
$$\text{and } B \subseteq \left[\!\left[F_Q , \rho[SYS'] \right]\!\right].$$

For the first direction,

$$\begin{aligned}
\left[\!\left[F'_Q , \rho[SYS] \right]\!\right] &= \\
&= \left[\!\left[\bigwedge_{\alpha \in S(Q)} \left([\alpha] X_{D(Q,\alpha)} \right) \wedge \bigwedge_{j \in E(Q)} Y_i , \rho[SYS] \right]\!\right] \\
&= \left[\!\left[\bigwedge_{\alpha \in S(Q)} \left([\alpha] \bigwedge_{i \in D(Q,\alpha)} X_i \right) \wedge \bigwedge_{j \in E(Q)} Y_j , \rho[SYS] \right]\!\right] \\
&\quad \text{(because of the preservation of condition 2)} \\
&= \left[\!\left[F_Q , \rho[SYS] \right]\!\right] = A.
\end{aligned}$$

On the other hand,

$$\begin{aligned}
&\left[\!\left[F_Q , \rho[SYS'] \right]\!\right] = \\
&= \left[\!\left[\bigwedge_{\alpha \in S(Q)} \left([\alpha] \bigwedge_{i \in D(Q,\alpha)} X_i \right) \wedge \bigwedge_{j \in E(Q)} Y_j , \rho[SYS'] \right]\!\right] \\
&= \left[\!\left[\bigwedge_{\alpha \in S_1(Q)} \left([\alpha] \bigwedge_{i \in Q} X_i \right) \wedge \bigwedge_{\alpha \in S_2(Q)} \left([\alpha] \bigwedge_{i \in D(Q,\alpha)} X_i \right) \wedge \bigwedge_{j \in E(Q)} Y_j , \rho[SYS'] \right]\!\right] \\
&= \left[\!\left[\bigwedge_{\alpha \in S_1(Q)} \left([\alpha] \bigwedge_{i \in Q} X_i \right) \wedge \bigwedge_{\alpha \in S_2(Q)} [\alpha] X_{D(Q,\alpha)} \wedge \bigwedge_{j \in E(Q)} Y_j , \rho[SYS'] \right]\!\right],
\end{aligned}$$

because if $Q \neq D(Q,\alpha)$, then $\left[\!\left[X_{D(Q,\alpha)} , \rho[SYS_0] \right]\!\right] = \left[\!\left[\bigwedge_{i \in D(Q,\alpha)} X_i , \rho[SYS_0] \right]\!\right]$, by the preserved condition 2. If Q is a singleton, then we are done, because then $\bigwedge_{i \in Q} X_i = X_Q$ and the last expression is just B. Therefore, we assume Q is not a singleton; so, for all $i \in Q$, $Q \neq \{i\}$ and thus, $\left[\!\left[X_i , \rho[SYS'] \right]\!\right] = \left[\!\left[F_i , \rho[SYS'] \right]\!\right]$. For convenience, let

$$\begin{aligned}
C &= \left[\!\left[\bigwedge_{\alpha \in S_2(Q)} \left([\alpha] \bigwedge_{i \in D(Q,\alpha)} X_i \right) \wedge \bigwedge_{j \in E(Q)} Y_j , \rho[SYS'] \right]\!\right] \\
&= \left[\!\left[\bigwedge_{\alpha \in S_2(Q)} [\alpha] X_{D(Q,\alpha)} \wedge \bigwedge_{j \in E(Q)} Y_j , \rho[SYS'] \right]\!\right].
\end{aligned}$$

Then, let SYS'_Q be SYS' after removing the equations for all X_i, $i \in Q$ and inserting all X_i, where $i \in Q$ in the set of free variables, $\mathcal{Y}$.

$$\begin{aligned}
\left[\!\left[F_Q , \rho[SYS'] \right]\!\right] &= \left[\!\left[\bigwedge_{i \in Q} F_i , \rho[SYS'] \right]\!\right] \quad \text{(by definition)} \\
&= \left[\!\left[\bigwedge_{i \in Q} X_i , \rho[SYS'] \right]\!\right] = \bigcap_{i \in Q} \left[\!\left[X_i , \rho[SYS'] \right]\!\right]
\end{aligned}$$

$$= \bigcap \bigtimes_{i\in Q} [\![X_i, \rho[SYS']]\!] = \bigcap \left[\!\!\left[\bigtimes_{i\in Q} X_i, \rho[SYS'] \right]\!\!\right]$$

$$= \bigcap \bigcup \left\{ \mathbb{T} \mid \mathbb{T} \subseteq \left[\!\!\left[\bigtimes_{i\in Q} F_i, \rho \overline{\left[\bigtimes_{i\in Q} X_i \mapsto \mathbb{T} \right]}^{SYS'_Q} \right]\!\!\right] \right\},$$

because of Lemma 7. Similarly as to how we analyzed F_Q,

$$B = [\![F'_Q, \rho[SYS']]\!] = \left[\!\!\left[\bigwedge_{\alpha\in S_1(Q)} [\alpha] X_Q, \rho[SYS'] \right]\!\!\right] \cap C. \tag{A.5}$$

So, it suffices to prove that for $k = |Q|$,

$$B^k \subseteq \left[\!\!\left[\bigtimes_{i\in Q} F_i, \rho \overline{\left[\bigtimes_{i\in Q} X_i \mapsto B^k \right]}^{SYS'_Q} \right]\!\!\right].$$

Let $p = (p_1, \ldots, p_k) \in [\![F'_Q, \rho[SYS']]\!]^k = B^k$. By (A.5), $p \in C^k$. Therefore, to prove that

$$p \in \left[\!\!\left[\bigtimes_{i\in Q} F_i, \rho \overline{\left[\bigtimes_{i\in Q} X_i \mapsto B^k \right]}^{SYS'_Q} \right]\!\!\right],$$

it suffices to prove that for all $1 \le i \le k$,

$$p_i \in \left[\!\!\left[\bigwedge_{\alpha\in S(i)\cap S_1(Q)} [\alpha] \bigwedge_{j\in D(i,\alpha)} X_j, \rho \overline{\left[\bigtimes_{i\in Q} X_i \mapsto B^k \right]}^{SYS'_Q} \right]\!\!\right],$$

or equivalently that for every $\alpha \in S(i) \cap S_1(Q)$ and $j \in D(i,\alpha) \subseteq D(Q,\alpha)$,

$$p_i \in \left[\!\!\left[[\alpha] X_j, \rho \overline{\left[\bigtimes_{i\in Q} X_i \mapsto B^k \right]}^{SYS'_Q} \right]\!\!\right]. \tag{A.6}$$

For any $\alpha \in S(i) \cap S_1(Q)$, $j \in D(i,\alpha) \subseteq D(Q,\alpha) = B$, and $p_i \stackrel{\alpha}{\Rightarrow} q_i$, because $p_i \in B$ and because of (A.5), $p_i \in [\![[\alpha]X_Q, \rho[SYS']]\!]$, so $q_i \in [\![X_Q, \rho[SYS']]\!] = B$. Therefore,

$$q_i \in \left[\!\!\left[X_j, \rho \overline{\left[\bigtimes_{i\in Q} X_i \mapsto B^k \right]}^{SYS'_Q} \right]\!\!\right] = B,$$

which gives us (A.6) and the proof is complete. □

Proof of Lemma 13. We use induction on the number of equations in *SYS*.

For the base case, we assume that *SYS* contains a single equation, $X_1 = F_1$. Since $X_1 = F_1$ is the principal equation in *SYS*, *SYS* is equivalent to the formula $\max X_1.\ F_1$, which is in deterministic form.

Now assume that *SYS* contains $n > 1$ equations. Let *SYS*′ be the result of removing the first equation from *SYS* and adding X_1 to $\mathcal{Y}$ if X_1 appears in the remaining equations of *SYS*′. Then,

$$[\![\text{SYS}, \rho]\!] = \bigcup \left\{ S_1 \mid S_1 \subseteq \left[\!\!\left[F_1, \overline{\rho[X_1 \mapsto S_1]}^{SYS'} \right]\!\!\right] \right\};$$

by the inductive hypothesis, there are formulae $\varphi_2, \ldots, \varphi_n$ in deterministic form, such that

$$\begin{aligned} [\![\text{SYS}, \rho]\!] &= \\ &= \bigcup \{ S_1 \mid S_1 \subseteq [\![F_1[\varphi_2/X_2, \ldots, \varphi_n/X_n], \rho[X_1 \mapsto S_1]]\!] \\ &= [\![\max X_1.\ F_1[\varphi_2/X_2, \ldots, \varphi_n/X_n], \rho]\!] \} \end{aligned}$$

and $\max X_1.\ F_1[\varphi_2/X_2, \ldots, \varphi_n/X_n]$ is in deterministic form. □

References

[1] L. Aceto, A. Ingólfsdóttir, K.G. Larsen, J. Srba, Reactive Systems: Modelling, Specification and Verification, Cambridge Univ. Press, New York, NY, USA, ISBN 0521875463, 2007.

[2] L. Aceto, A. Achilleos, A. Francalanza, A. Ingólfsdóttir, S.Ö. Kjartansson, On the complexity of determinizing monitors, in: A. Carayol, C. Nicaud (Eds.), Implementation and Application of Automata – 22nd International Conference, CIAA 2017, in: Lecture Notes in Computer Science, vol. 10329, Springer, ISBN 978-3-319-60133-5, 2017, pp. 1–13.

[3] L. Aceto, A. Achilleos, A. Francalanza, A. Ingólfsdóttir, A framework for parameterized monitorability, in: C. Baier, U.D. Lago (Eds.), Foundations of Software Science and Computation Structures - 21st International Conference, FOSSACS 2018, in: Lecture Notes in Computer Science, vol. 10803, Springer, ISBN 978-3-319-89365-5, 2018, pp. 203–220.

[4] L. Aceto, I. Cassar, A. Francalanza, A. Ingólfsdóttir, On runtime enforcement via suppressions, in: S. Schewe, L. Zhang (Eds.), 29th International Conference on Concurrency Theory, CONCUR 2018, in: LIPIcs, vol. 118, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, ISBN 978-3-95977-087-3, 2018, pp. 34:1–34:17.

[5] L. Aceto, A. Achilleos, A. Francalanza, A. Ingólfsdóttir, K. Lehtinen, An operational guide to monitorability, in: P.C. Ölveczky, G. Salaün (Eds.), Software Engineering and Formal Methods – 17th International Conference, SEFM 2019, in: Lecture Notes in Computer Science, vol. 11724, Springer, ISBN 978-3-030-30445-4, 2019, pp. 433–453.

[6] L. Aceto, A. Achilleos, A. Francalanza, A. Ingólfsdóttir, K. Lehtinen, The cost of monitoring alone, in: E. Bartocci, R. Cleaveland, R. Grosu, O. Sokolsky (Eds.), From Reactive Systems to Cyber-Physical Systems - Essays Dedicated to Scott A. Smolka on the Occasion of His 65th Birthday, in: Lecture Notes in Computer Science, vol. 11500, Springer, ISBN 978-3-030-31513-9, 2019, pp. 259–275.

[7] L. Aceto, A. Achilleos, A. Francalanza, A. Ingólfsdóttir, K. Lehtinen, Adventures in monitorability: from branching to linear time and back again, Proc. ACM Program. Lang. 3 (POPL 2019) (2019) 52:1–52:29, https://doi.org/10.1145/3290365.

[8] A. Arnold, D. Niwinski, Rudiments of μ-Calculus, Studies in Logic and the Foundations of Mathematics, Elsevier Science, ISBN 9780080516455, 2001, https://books.google.is/books?id=MkWZaiECJvQC.

[9] E. Bartocci, Y. Falcone (Eds.), Lectures on Runtime Verification — Introductory and Advanced Topics, Lecture Notes in Computer Science, vol. 10457, Springer, ISBN 978-3-319-75631-8, 2018.

[10] A. Bauer, M. Leucker, C. Schallhart, The good, the bad, and the ugly, but how ugly is ugly?, in: RV, in: LNCS, vol. 4839, Springer, ISBN 978-3-540-77394-8, 2007, pp. 126–138.

[11] A. Bauer, M. Leucker, C. Schallhart, Comparing LTL semantics for runtime verification, J. Log. Comput. 20 (3) (2010) 651–674.

[12] A. Bauer, M. Leucker, C. Schallhart, Runtime verification for LTL and TLTL, ACM Trans. Softw. Eng. Methodol. 20 (4) (2011) 14.

[13] H. Björklund, W. Martens, The tractability frontier for NFA minimization, J. Comput. Syst. Sci. 78 (1) (2012) 198–210.

[14] I. Cassar, A. Francalanza, On implementing a monitor-oriented programming framework for actor systems, in: E. Ábrahám, M. Huisman (Eds.), Integrated Formal Methods - 12th International Conference, IFM 2016, in: Lecture Notes in Computer Science, vol. 9681, Springer, ISBN 978-3-319-33692-3, 2016, pp. 176–192.

[15] M. Chrobak, Finite automata and unary languages, Theor. Comput. Sci. (ISSN 0304-3975) 47 (1986) 149–158, https://doi.org/10.1016/0304-3975(86)90142-8, http://www.sciencedirect.com/science/article/pii/0304397586901428.

[16] M. Chrobak, Errata to: "Finite Automata and Unary Languages", Theor. Comput. Sci. (ISSN 0304-3975) 302 (1) (2003) 497–498, https://doi.org/10.1016/S0304-3975(03)00136-1, http://www.sciencedirect.com/science/article/pii/S0304397503001361.

[17] E.M. Clarke, E.A. Emerson, Design and synthesis of synchronization skeletons using branching time temporal logic, in: Workshop on Logic of Programs, Springer, 1981, pp. 52–71.

[18] M. d'Amorim, G. Rosu, Efficient monitoring of ω-languages, in: K. Etessami, S.K. Rajamani (Eds.), Computer Aided Verification, 17th International Conference, CAV 2005, in: Lecture Notes in Computer Science, vol. 3576, Springer, ISBN 3-540-27231-3, 2005, pp. 364–378.

[19] S. Debois, T.T. Hildebrandt, T. Slaats, Safety, liveness and run-time refinement for modular process-aware information systems with dynamic sub processes, in: N. Bjørner, F.S. de Boer (Eds.), FM 2015: Formal Methods – 20th International Symposium, in: Lecture Notes in Computer Science, vol. 9109, Springer, ISBN 978-3-319-19248-2, 2015, pp. 143–160.

[20] C. Eisner, D. Fisman, J. Havlicek, Y. Lustig, A. McIsaac, D.V. Campenhout, Reasoning with temporal logic on truncated paths, in: CAV, in: LNCS, vol. 2725, Springer, ISBN 3-540-40524-0, 2003, pp. 27–39.

[21] U. Erlingsson, The Inlined Reference Monitor Approach to Security Policy Enforcement, PhD thesis, Cornell University, 2004.

[22] Y. Falcone, J.-C. Fernandez, L. Mounier, What can you verify and enforce at runtime?, Int. J. Softw. Tools Technol. Transf. 14 (3) (2012) 349–382.

[23] A. Francalanza, L. Aceto, A. Ingólfsdóttir, On verifying Hennessy-Milner logic with recursion at runtime, in: E. Bartocci, R. Majumdar (Eds.), Runtime Verification, in: Lecture Notes in Computer Science, vol. 9333, Springer International Publishing, ISBN 978-3-319-23819-7, 2015, pp. 71–86.

[24] A. Francalanza, L. Aceto, A. Ingólfsdóttir, Monitorability for the Hennessy-Milner logic with recursion, Form. Methods Syst. Des. 51 (1) (2017) 87–116, https://doi.org/10.1007/s10703-017-0273-z.

[25] M. Geilen, On the construction of monitors for temporal logic properties, in: RV, in: ENTCS, vol. 55, 2001, pp. 181–199.

[26] G. Gramlich, G. Schnitger, Minimizing NFA's and regular expressions, J. Comput. Syst. Sci. (ISSN 0022-0000) 73 (6) (2007) 908–923, https://doi.org/10.1016/j.jcss.2006.11.002, http://www.sciencedirect.com/science/article/pii/S0022000006001735.

[27] J. Gray, Why do computers stop and what can be done about it?, in: Fifth Symposium on Reliability in Distributed Software and Database Systems, SRDS 1986, IEEE Computer Society, ISBN 0-8186-0690-8, 1986, pp. 3–12.

[28] H. Gruber, M. Holzer, Inapproximability of nondeterministic state and transition complexity assuming P$\neq$NP, in: T. Harju, J. Karhumäki, A. Lepistö (Eds.), Developments in Language Theory, 11th International Conference, DLT 2007, in: Lecture Notes in Computer Science, vol. 4588, Springer, ISBN 978-3-540-73207-5, 2007, pp. 205–216.

[29] Y. He, X. Chen, G. Lin, Composition of monitoring components for on-demand construction of runtime model based on model synthesis, in: H. Mei, J. Lv, X. Mao (Eds.), Proceedings of the 5th Asia-Pacific Symposium on Internetware, Internetware 2013, ACM, ISBN 978-1-4503-2369-7, 2013, pp. 20:1–20:4.

[30] D. Janin, I. Walukiewicz, On the expressive completeness of the propositional mu-calculus with respect to monadic second order logic, in: U. Montanari, V. Sassone (Eds.), CONCUR '96, Concurrency Theory, 7th International Conference, in: Lecture Notes in Computer Science, vol. 1119, Springer, ISBN 3-540-61604-7, 1996, pp. 263–277.

[31] T. Jiang, B. Ravikumar, Minimal NFA problems are hard, SIAM J. Comput. (ISSN 0097-5397) 22 (6) (1993) 1117–1141, https://doi.org/10.1137/0222067.

[32] R.M. Keller, Formal verification of parallel programs, Commun. ACM 19 (7) (1976) 371–384, https://doi.org/10.1145/360248.360251.

[33] J. Klein, I. Gorton, Runtime performance challenges in big data systems, in: C.M. Woodside (Ed.), Proceedings of the 2015 Workshop on Challenges in Performance Methods for Software Development, WOSP-C'15, ACM, ISBN 978-1-4503-3340-5, 2015, pp. 17–22.

[34] D. Kozen, Results on the propositional μ-calculus, Theor. Comput. Sci. (ISSN 0304-3975) 27 (3) (1983) 333–354, https://doi.org/10.1016/0304-3975(82)90125-6, http://www.sciencedirect.com/science/article/pii/0304397582901256.

[35] K.G. Larsen, Proof systems for satisfiability in Hennessy-Milner logic with recursion, Theor. Comput. Sci. 72 (2&3) (1990) 265–288, https://doi.org/10.1016/0304-3975(90)90038-J.

[36] M. Leucker, C. Schallhart, A brief account of runtime verification, J. Log. Algebraic Program. (ISSN 1567-8326) 78 (5) (2009) 293–303, https://doi.org/10.1016/j.jlap.2008.08.004, http://www.sciencedirect.com/science/article/pii/S1567832608000775.

[37] J. Ligatti, L. Bauer, D. Walker, Edit automata: enforcement mechanisms for run-time security policies, Int. J. Inf. Secur. 4 (1–2) (2005) 2–16, https://doi.org/10.1007/s10207-004-0046-8.
[38] Q. Luo, F. Hariri, L. Eloussi, D. Marinov, An empirical analysis of flaky tests, in: S. Cheung, A. Orso, M.D. Storey (Eds.), Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE-22, ACM, ISBN 978-1-4503-3056-5, 2014, pp. 643–653.
[39] P.D. Marinescu, P. Hosek, C. Cadar Covrig, A framework for the analysis of code, test, and coverage evolution in real software, in: C.S. Pasareanu, D. Marinov (Eds.), International Symposium on Software Testing and Analysis, ISSTA '14, ACM, ISBN 978-1-4503-2645-2, 2014, pp. 93–104.
[40] A.M. Memon, M.B. Cohen, Automated testing of GUI applications: models, tools, and controlling flakiness, in: D. Notkin, B.H.C. Cheng, K. Pohl (Eds.), 35th International Conference on Software Engineering, ICSE '13, IEEE Computer Society, ISBN 978-1-4673-3076-3, 2013, pp. 1479–1480.
[41] P.O. Meredith, D. Jin, D. Griffith, F. Chen, G. Roşu, An overview of the MOP runtime verification framework, Int. J. Softw. Tools Technol. Transf. 14 (3) (2012) 249–289.
[42] A.R. Meyer, M.J. Fischer, Economy of description by automata, grammars, and formal systems, in: 12th Annual Symposium on Switching and Automata Theory, IEEE Computer Society, 1971, pp. 188–191.
[43] R. Milner, A Calculus of Communicating Systems, Springer-Verlag New York, Inc., Secaucus, NJ, USA, ISBN 0387102353, 1982.
[44] A. Pnueli, The temporal logic of programs, in: 18th Annual Symposium on Foundations of Computer Science, 1977, IEEE, 1977, pp. 46–57.
[45] A. Pnueli, A. Zaks, PSL model checking and run-time verification via testers, in: J. Misra, T. Nipkow, E. Sekerinski (Eds.), FM 2006: Formal Methods, 14th International Symposium on Formal Methods, in: Lecture Notes in Computer Science, vol. 4085, Springer, ISBN 3-540-37215-6, 2006, pp. 573–586.
[46] M.O. Rabin, D.S. Scott, Finite automata and their decision problems, IBM J. Res. Dev. 3 (2) (1959) 114–125, https://doi.org/10.1147/rd.32.0114.
[47] A. Rabinovich, A complete axiomatisation for trace congruence of finite state behaviors, in: International Conference on Mathematical Foundations of Programming Semantics, Springer, 1993, pp. 530–543.
[48] W.J. Savitch, Relationships between nondeterministic and deterministic tape complexities, J. Comput. Syst. Sci. (ISSN 0022-0000) 4 (2) (1970) 177–192, https://doi.org/10.1016/S0022-0000(70)80006-X.
[49] M. Sipser, Introduction to the Theory of Computation. Computer Science Series, PWS Publishing Company, ISBN 9780534947286, 1997, https://books.google.is/books?id=cXcpAQAAMAAJ.
[50] D. Tabakov, K.Y. Rozier, M.Y. Vardi, Optimized temporal monitors for SystemC, Form. Methods Syst. Des. 41 (3) (2012) 236–268, https://doi.org/10.1007/s10703-011-0139-8.
[51] M. Vardi, P. Wolper, Reasoning about infinite computations, Inf. Comput. (ISSN 0890-5401) 115 (1) (1994) 1–37, https://doi.org/10.1006/inco.1994.1092, http://www.sciencedirect.com/science/article/pii/S0890540184710923.
[52] M. Viswanathan, M. Kim, Foundations for the run-time monitoring of reactive systems–fundamentals of the mac language, in: International Colloquium on Theoretical Aspects of Computing, Springer, 2004, pp. 543–556.
[53] S. Zhang, D. Jalali, J. Wuttke, K. Muşlu, W. Lam, M.D. Ernst, D. Notkin, Empirically revisiting the test independence assumption, in: Proceedings of the 2014 International Symposium on Software Testing and Analysis, ACM, New York, NY, USA, ISBN 978-1-4503-2645-2, 2014, pp. 385–396.