Determinizing Monitors for HML with Recursion*

Luca Aceto[1], Antonis Achilleos[1], Adrian Francalanza[2], Anna Ingólfsdóttir[1], and Sævar Örn Kjartansson[1]

[1] School of Computer Science, Reykjavik University, Reykjavik, Iceland
[2] Dept. of Computer Science, ICT, University of Malta, Msida, Malta

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 (as their LTS), 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 entities (referred to hereafter as *systems*, or, in more formal settings, as *processes*) with the aim of accruing system information [23,21], comparing system executions against behavioral specifications [16], or reacting to observed executions via adaptation or enforcement procedures [7,27]. Monitor descriptions can vary substantially, from pseudocode [17,14], to mathematical descriptions [38,11,15], to executable code in a domain-specific or general-purpose language [12,31]. Because they are part of the trusted computing base, monitor descriptions are expected to be *correct*, and a prevalent correctness requirement is that they should exhibit *deterministic behavior*.

Monitors are central for the field of Runtime Verification, where typically we use monitors to observe the trace produced during the 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 [33,13,3,5,4], CTL, CTL* [10], and μHML [24,1,25,16]. For a brief overview of Runtime Verification, see [26].

In [16], Francanalza et al. studied the monitorability of properties expressed in full μHML. They determined a maximal monitorable fragment of the logic, for which they introduced a monitor generating procedure. The monitors constructed from this procedure can be nondeterministic, depending on the μHML formula from which they were constructed. In the light of the importance of

* This research was supported by the project "TheoFoMon: Theoretical Foundations for Monitorability" (grant number: 163406-051) of the Icelandic Research Fund.

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 [16], where monitors are described using syntax close to the regular fragment of CCS processes [32]. 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 [16], 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.

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 monitor 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. Instead, 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[3] 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 [39,28]. We contend that monitors are generally more complex artefacts 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 process-

[3] 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.

ing, 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.

In testing, the impact of nondeterministic (*a.k.a. flaky*) tests on software development processes is substantial enough (for instance, as reported in [28], 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 [30,29,28]. 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 behaviour 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 [19] and problems associated with test dependence are not uncommon [39], 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) [28,30].

Overview: In Section 2, we give the necessary background for our investigation. We introduce μHML formulas 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 [35]. The other directly applies a procedure for transforming systems of equations, inspired from Rabinovich's methods, directly on μHML formulas to turn them into a deterministic form. Then, using the monitor synthesis procedure from [16], 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 for an extensive summary of the technical results in this paper.

2 Background

We provide background on the main definitions and results for monitoring μHML formulae, as defined in [16], and present the conventions we use in this paper.

2.1 Basic Definitions: Monitoring μHML Formulae on Processes

We begin by giving the basic definitions for the main concepts we use. These include 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.

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 $\textsc{Proc}$ is a set of states or processes, $\textsc{Act}$ 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 $\textsc{Proc}$ is defined by the following grammar:

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

where x comes from a countably infinite set of process variables. These processes are a standard variation on the regular fragment of CCS [32]; in [35], these processes are called μ-expressions. We simply call them processes in this paper. As usual, the construct $\mathtt{rec}\ x.p$ binds the free occurrences of x in p. In what follows, unless stated otherwise, we focus on processes without the 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.

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

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

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

Table 1. Dynamics of Processes

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, $p(\xrightarrow{\tau})^*.\ \xrightarrow{\alpha}\ .(\xrightarrow{\tau})^* p'$. 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 $p(\xrightarrow{\tau})^* p'$ if t is the empty trace ϵ.

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:*

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

where X comes from a countably infinite set of logical variables LVar.

Formulae are evaluated in the context of a labeled transition system and an environment, $\rho : \text{LVar} \to 2^{\text{Proc}}$, which gives values to the logical variables in the formula. For an environment ρ, variable X, and set $S \subseteq \text{Proc}$, $\rho[X \mapsto S]$ 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 $X \in \text{LVar}$ is satisfied by the processes in $\rho(X)$. $[\![\cdot]\!]$ is defined as follows:

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

A formula is closed when every occurrence of a variable X is in the scope of 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 formulas, 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, are defined by the grammar:

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

Example 1 (a formula). Formula $\varphi_e = \max\ X.[\alpha]([\alpha]\mathtt{ff} \wedge X) \in$ sHML. Notice that

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

Monitors We now define the notion of a monitor. Just like for processes, we use the definitions given in [16]. 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:*

$$\begin{array}{rcl} m, n \in \textsc{Mon} ::= v & \mid \alpha.m \mid m+n \mid \mathtt{rec}x.m \mid x \\ v \in \textsc{Verd} ::= \mathtt{end} & \mid \mathtt{no} \mid \mathtt{yes} \end{array}$$

where x comes from a countably infinite set of monitor variables.

The behavior of a monitor is defined by the derivation rules of Table 2.

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

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

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

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

Table 2. Monitor Dynamics

For each $m, m' \in \textsc{Mon}$ and $\alpha \in \textsc{Act}$, we use $m \stackrel{\alpha}{\Rightarrow} m'$ to mean that m can derive m' using a single α action and any number of τ actions, $m(\xrightarrow{\tau})^*.\xrightarrow{\alpha}.(\xrightarrow{\tau})^* m'$. For each $m, m' \in \textsc{Mon}$ and trace $t = \alpha_1.\alpha_2.\ldots.\alpha_r \in \textsc{Act}^*$, we use $m \stackrel{t}{\Rightarrow} m'$ to mean $m \stackrel{\alpha_1}{\Longrightarrow} . \stackrel{\alpha_2}{\Longrightarrow} \ldots \stackrel{\alpha_r}{\Longrightarrow} m'$ if t is non-empty and $m(\xrightarrow{\tau})^* m'$ if t is the empty trace. Rules mSelL and mSelR may be referred to collectively as mSel for convenience.

Example 2 (a monitor). Let $m_e = \mathtt{rec}\ x.\alpha.(\alpha.\mathtt{no} + x)$. Notice the similarities between m_e and φ_e. We will be using m_e, φ_e, and related constructs as a running example.

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.*

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

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

Table 3. Monitored Processes

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). $\mathbf{acc}(m,p) \overset{def}{=} \exists t, p'.\ m \triangleleft p \overset{t}{\Rightarrow} \texttt{yes} \triangleleft p'$ *and* $\mathbf{rej}(m,p) \overset{def}{=} \exists t, p'.\ m \triangleleft p \overset{t}{\Rightarrow} \texttt{no} \triangleleft p'$.

Finite Automata We now present a brief overview of Finite Automata Theory, which we use in Section 4. The interested reader can see [37] 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, $\Sigma = \textsc{Act}$ —, $\delta \subseteq Q \times \Sigma \times Q$ 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 ([34]). *If A is an NFA of n states, then there is a DFA of at most 2^n states which recognizes $L(A)$.*

Proof. The way to construct such a DFA is through the classical subset construction. For more details, see [37], 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 lest 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, the reader should read Section 4.

2.2 Previous Results

The main result from [16] is to define a subset of μHML which is monitorable and show that it is maximally expressive. This subset is called MHML and consists of the safe and co-safe syntactic subsets of μHML: $\text{MHML} \stackrel{def}{=} \text{SHML} \cup \text{CHML}$. From now on, we focus on SHML, but the case of CHML is dual. The interested reader can see [16] for more details.

In order to prove that SHML is monitorable, in [16] Francalanza, Aceto, and Ingólfsdóttir define a monitor synthesis function, $(\!|-|\!)$, which maps formulae to monitors, and show that for each $\varphi \in \text{MHML}$, $(\!|\varphi|\!)$ monitors for φ, in that $\mathbf{rej}((\!|\varphi|\!), p)$ exactly for those processes p for which $p \notin [\![\varphi]\!]$. This function is used in the proofs in Section 3 and so we give the definition here.

Definition 5 (Monitor Synthesis).

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

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

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

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

We say that a monitor m monitors for a formula $\varphi \in \text{SHML}$ when for each process p, $\mathbf{rej}(m, p)$ if and only if $p \notin [\![\varphi]\!]$.

Theorem 2 (Monitorability, [16]). *For each $\varphi \in$ sHML, $(\!|\varphi|\!)$ monitors for φ.*

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, for $m'_e = (\!|\varphi'_e|\!) = \alpha.\alpha.\mathtt{no}$, m_e and m'_e monitor for the same formula, and therefore they are equivalent.

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 know what a deterministic monitor is and what it means that two monitors are equivalent.

Even before having defined what a deterministic monitor is, it is easy to find examples of nondeterministic ones. One can look in [16] for several examples. An easy one is $m_c = \alpha.\mathtt{yes} + \alpha.\mathtt{no}$. 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 amend this situation. One is to make sure that 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 which can reach only one verdict, like $\alpha.\mathtt{yes} + \alpha.p$. We explore both approaches.

Determinism 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 $m_1 + m_2$. Note that a sum $m_1 + m_2 + \cdots + m_k$ can also be written as $\sum_{i=1}^{k} m_i$.

Definition 6. *A monitor m is 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 $A \subseteq$ Act.*

Note that under this definition, from every deterministic monitor m and action $\alpha \in \text{Act}$, if $m \overset{\alpha}{\Rightarrow} m'$, then all derivations $m \overset{\alpha}{\Rightarrow}$ begin with a unique transition. As we will see in the following sections, this set of monitors is in fact maximally expressive.

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 (from [16])). 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

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

We can then define a violation complete monitor for this formula using the monitor synthesis function:

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

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:

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

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. 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. The first proof directly depends on Rabinovich's determinization of processes in [35], while the second one is inspired by Rabinovich's construction.

Multiple Verdicts As Francalanza et al. demonstrated in [16], 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.

Other Conventions and Definitions We will call two monitors, p and q equivalent and write $p \sim q$ if for every trace t and value v, $p \stackrel{t}{\Rightarrow} v$ iff $q \stackrel{t}{\Rightarrow} v$. 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 also [16], Proposition 1). Therefore, we compare two monitors with respect to relation $\sim$. We assume that the set of actions, Act, is finite and, specifically, of constant size.

We call derivations of the form $p \Rightarrow p$ 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 q as follows: q is a sum of q and if r is a sum of q, then so are $r + r'$ and $r' + r$. We say that a sum s of q is acting as q in a derivation $d : s \stackrel{t}{\Rightarrow} r$ if in the same derivation we can replace s by q without consequence (i.e. the first step uses rule mSlet to use a derivation starting from q).

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

We assume that all sums of a verdict v are v. Indeed, Lemma 1 below demonstrates that all instances of $p + v$ can be replaced by $p + \sum_{\alpha \in \text{Act}} \alpha.v$.

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

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

We also assume that there is no (sub)monitor of the form $\texttt{rec}\ x.v$, where v 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 `end`. We say that a monitor p accepts/recognizes a language (set of traces) $L \subseteq \text{Act}^*$ when for every trace $t \in \text{Act}^*$, $t \in L$ iff $p \overset{t}{\Rightarrow} \texttt{yes}$.

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 [35] for the determinization of processes. The second method, presented in subsection 3.2, uses methods similar to Rabinovich's directly on formulas of sHML to transform them to a deterministic form, so that when we apply the monitor synthesis function from [16] (Definition 5 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 7. *For a process p, $Trace(p) = \{t \in \text{Act}^* \mid \exists q.\ p \overset{t}{\Rightarrow} q\}$. We call two processes p, q trace equivalent when $Trace(p) = Trace(q)$.*

We use a result given by Rabinovich in [35], 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.

Definition 8. *Let $D : \text{Proc} \to \text{Proc}$ be a mapping from a process (a process, as defined in Subsection 2.1) to a trace equivalent deterministic process. Such a mapping exists as shown by Rabinovich in [35].*

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 can see 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 define an "inverse" of π, which we call π^{-1}: $\pi^{-1}(s) = v$ when s is a sum of some $v.p$ and in all other cases,

$$\begin{aligned}\pi^{-1}(\alpha.p) &= \alpha.\pi^{-1}(p)\\ \pi^{-1}(p+q) &= \pi^{-1}(p) + \pi^{-1}(q)\\ \pi^{-1}(\mathtt{rec}x.p) &= \mathtt{rec}x.\pi^{-1}(p)\\ \pi^{-1}(x) &= x\end{aligned}$$

Now 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 2. *Let p be a monitor and $q = \pi(p)$. Then, $p \stackrel{t}{\Rightarrow} v$ if and only if there is some $t' \sqsubseteq t$, such that $q \stackrel{t'}{\Rightarrow} v.\mathtt{nil}$.*

Proof. Straightforward induction on the length of the derivation $q \stackrel{t'}{\Rightarrow} v.\mathtt{nil}$ for the "if" direction and on the length of the derivation $p \stackrel{t}{\Rightarrow} v$ for the "only if" direction. □

Lemma 3. *Let $q = \pi^{-1}(p)$. 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 $q \stackrel{t}{\Rightarrow} v$.*

Proof. Like for Lemma 2, by induction on the length of the derivations. □

Lemma 4. *Let m and $n = \pi^{-1}(n')$ be monitors such that $Trace(\pi(m)) = Trace(n')$. Then $m \sim n$.*

Proof. Assume that for a trace t and a verdict v, we have $m \stackrel{t}{\Rightarrow} v$. By Lemma 2, for some $t' \sqsubseteq t$, $\pi(m) \stackrel{t'}{\Rightarrow} v.\mathtt{nil}$ and so $t'.v \in Trace(\pi(m))$. Since $\pi(m)$ and n' are trace equivalent, we have $n' \stackrel{t'v}{\Longrightarrow} r$ for some r; therefore, $n' \stackrel{t'}{\Rightarrow} s$, where s is a sum of some $v.r$, and so, by Lemma 3, $n \stackrel{t}{\Rightarrow} v$.

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

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 4, m and n are equivalent. Let $n = \pi^{-1}(n')$. Monitor n is deterministic. To prove this, let s be a sum in n. We know that $s = \pi^{-1}(s')$, where s' appears in n', which is deterministic. Then, either s' is a sum of a $v.p$, so $s = v$, or $s' = \sum_{\alpha \in A} \alpha.p'_\alpha$ and $s = \sum_{\alpha \in A} \alpha.p_\alpha$. □

Example 6. 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. The reader can also see [35], where Rabinovich uses very similar constructions to determinize processes.

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, [2]).

Given tuples, $a = (a_1, \ldots, a_k)$ 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 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:*

$$\begin{aligned} X_1 &= F_1, \\ X_2 &= F_2, \\ &\vdots \\ X_n &= F_n, \end{aligned}$$

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

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

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 7. 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 5. *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 6. *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,

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

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 $i \in I$.

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 them is a free variable of* φ.

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

Lemma 7. *Let* $\varphi \in$ sHML *be a formula 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* $\sum_{i \in A} p_i$, *then* φ *contains a conjunction* $\bigwedge_{i \in A \cup B} \varphi_i$, *where for* $i \in A$, $(\!|\varphi_i|\!) = p_i \neq \texttt{yes}$. □

Example 8. φ_e is not in deterministic form, but $\psi_e = [\alpha]([\alpha]\texttt{ff} \wedge X)$ is (because here X is free) and so is $\varphi_e' = [\alpha][\alpha]\texttt{ff}$.

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.

Example 9. 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 of Lemma 8 transforms it into $(\max\ X.[\alpha]X) \wedge Y$ which is.

Lemma 8. *Each formula in* sHML *is equivalent to some formula in* sHML *which is in standard form.*

Proof. 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

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

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:

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

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

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

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:

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

□

Lemma 9. *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 5, it suffices to prove the lemma when replacing the equation of the primary variable $X_1 = F_1$ by $X_1 = F_1'$. Then,

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

□

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

Proof. 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 8, 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 $j = 3 - i$,

$$\left[\!\!\left[SYS_i, \overline{\rho[SYS_i]}^{SYS_j}\right]\!\!\right] = \left[\!\!\left[SYS_i, \overline{\rho[SYS_j]}^{SYS_i}\right]\!\!\right] = [\![SYS_i, \rho[SYS_i]]\!]. \qquad (4)$$

Finally,

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

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:

$$\begin{aligned} X_0 &= F_1 \wedge G_1 \\ &= \bigwedge_{j \in K_1} [\alpha_j] X_j \wedge \bigwedge_{j \in S_1} Y_j \wedge \bigwedge_{j \in K_1'} [\alpha_j] Z_j \wedge \bigwedge_{j \in S_1'} Y_j \\ &\equiv \left(\bigwedge_{j \in K_1'} [\alpha_j] X_j \wedge \bigwedge_{j \in K_1'} [\alpha_j] Z_j \right) \wedge \bigwedge_{j \in S_1 \cup S_1'} Y_j \end{aligned}$$

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

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

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 9):

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

(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. □

Example 10. The system of equations which which results from the construction of Lemma 10 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 if $X_i \neq X_j$, then $\alpha_1 \neq \alpha_2$. We say that SYS is in deterministic form if every equation in Eq is in deterministic form.*

Lemma 11. *For each* sHML *system of equations in standard form, there exists an equivalent system of equations that is in deterministic form.*

Proof. 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:

$$\begin{aligned} S(i) &= \{\alpha_j \mid [\alpha_j] X_j \text{ is a sub formula in } F_i\} \\ D(i, \alpha) &= \{r \mid [\alpha] X_r \text{ is a sub formula in } F_i\} \\ E(i) &= \{r \mid Y_r \text{ is unguarded in } F_i\} \end{aligned}$$

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 $F_i \neq \texttt{ff}$, 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 $\bigwedge_{j \in Q} X_j$; we identify $X_{\{i\}}$ with X_i.

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

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

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. 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).

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

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

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 5). 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:

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

For the first direction,

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

On the other hand,

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

because if $Q \neq D(Q, \alpha)$, then $[\![X_{D(Q,\alpha)}, \rho[SYS_0]]\!] = \left[\!\!\left[\bigwedge_{i \in D(Q,\alpha)} X_i, \rho[SYS_0]\right]\!\!\right]$, 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, $[\![X_i, \rho[SYS']]\!] = [\![F_i, \rho[SYS']]\!]$. 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, $\mathcal{Y}$.

$$
\begin{aligned}
[\![F_Q, \rho[SYS']]\!] &= \left[\!\!\left[\bigwedge_{i \in Q} F_i, \rho[SYS'] \right]\!\!\right] \quad \text{(by definition)} \\
&= \left[\!\!\left[\bigwedge_{i \in Q} X_i, \rho[SYS'] \right]\!\!\right] \quad = \bigcap_{i \in Q} [\![X_i, \rho[SYS']]\!] \\
&= \bigcap \bigtimes_{i \in Q} [\![X_i, \rho[SYS']]\!] = \bigcap \left[\!\!\left[\bigtimes_{i \in Q} X_i, \rho[SYS'] \right]\!\!\right] \\
&= \bigcap \bigcup \left\{ \mathbb{T} \mid \mathbb{T} \subseteq \left[\!\!\left[\bigtimes_{i \in Q} F_i, \rho \overline{\left[\bigtimes_{i \in Q} X_i \mapsto \mathbb{T} \right]}^{SYS'_Q} \right]\!\!\right] \right\},
\end{aligned}
$$

because of Lemma 6. Similarly as to how we analyzed F_Q,

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

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

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

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

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

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

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

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

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

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 (5), $p_i \in [\![[\alpha] X_Q, \rho[SYS']]\!]$, so $q_i \in [\![X_Q, \rho[SYS']]\!] = B$. Therefore,

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

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

Example 11. The deterministic form of SYS'_e is 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 12. *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.*

Proof. We use proof by 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,

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

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

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

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

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 10, 11 and 12. □

Example 12. If we follow through all the constructions, starting from φ_e, using Lemma 10 we construct SYS'_e, which is a system of equations in standard form; from SYS'_e we construct using Lemma 11 we construct system SYS''_e, which is in deterministic form; finally, from SYS''_e we can construct, using Lemma 12, 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.

4 Bounds for Determinizing Monitors

The purpose of this section is to establish bounds on the sizes of monitors, both with respect to finite automata and when comparing deterministic to nondeterministic monitors. The constructions of Section 3 are not easy to extract bounds

from. 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

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. Consider a single monitor, which appears at the beginning of the derivation under consideration — that is, all other monitors are submonitors of this. We assume without loss of generality that each variable x appears in the scope of a unique monitor of the form $\texttt{rec}\ x.m$, which we call p_x; m_x is the monitor such that $p_x = \texttt{rec}\ x.m_x$. The monitors may behave according 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.

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

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

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 p, trace t, and value v,

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

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.

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

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

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

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

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

$$p \xrightarrow{\tau} \alpha.\texttt{yes} + \beta.\texttt{rec}\ y.(\alpha.\texttt{rec}\ x.(\alpha.\texttt{yes} + \beta.\texttt{rec}\ y.(\alpha.x + \beta.y)) + \beta.y)$$

$$\xrightarrow{\beta} \texttt{rec}\ y.(\alpha.\texttt{rec}\ x.(\alpha.\texttt{yes} + \beta.\texttt{rec}\ y.(\alpha.x + \beta.y)) + \beta.y) = p'_y,$$

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 p, trace t, and value v, $p \overset{t}{\Rightarrow}_O v$ iff $p \overset{t}{\Rightarrow}_N v$. We prove this claim in this subsection by demonstrating equivalence of both systems to system M.

Lemma 13. *Given monitors p, q and action $\alpha \in$ Act, $p \xrightarrow{\alpha}_N q$ iff $p \xrightarrow{\alpha}_M p'$ iff $p \xrightarrow{\alpha}_O p'$.*

Proof. Notice that in all three systems, $p \xrightarrow{\alpha} p'$ 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 14. *If $p \xrightarrow{\tau}_M q$, then there is some variable x, such that p is a sum of x or of p_x and $q = m_x$.*

Proof. Notice that τ-transitions can only be introduced by rules mRecF and mRecP and then propagated by rules mSelL and mSelR. then, it is not hard to verify that if $p \xrightarrow{\tau}_M q$ was produced by mRecF or mRecP, then p, q satisfy the property asserted by the lemma; and that rules mSelL and mSelR preserve this property. □

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

Proof. Very similar to the proof of Lemma 14. □

Lemma 16. *If $p \xrightarrow{\tau}_O q$, then there is a monitor $r = \texttt{rec}\ x.r'$, such that p is a sum of r and $q = r'[r/x]$.*

Proof. Again, very similar to the proof of Lemma 14. □

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

Lemma 17. *For a monitor p, trace t, and value v, $p \overset{t}{\Rightarrow}_N v$ iff $p \overset{t}{\Rightarrow}_M v$.*

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

If $p = v$**,** then we are done, as the trivial derivation exists for both systems.

If $p \xrightarrow{\alpha}_M p' \overset{t'}{\Rightarrow}_M v$**,** then $p \xrightarrow{\alpha}_N p'$ by Lemma 13. By the inductive hypothesis, $p' \overset{t'}{\Rightarrow}_N v$ and we are done.

If $p \xrightarrow{\tau}_M p' \overset{t}{\Rightarrow}_M v$**,** then by Lemma 14, there is some variable x, such that p is a sum of x or of p_x and $p' = m_x$. Then, if p a sum of p_x, $p \xrightarrow{\tau}_N m_x$ and if p a sum of x, then $p \xrightarrow{\tau}_N p_x \xrightarrow{\tau}_N m_x = p'$. By the inductive hypothesis, $p' \overset{t}{\Rightarrow}_N v$ and we are done.

We now prove that if $p \overset{t}{\Rightarrow}_N v$, then $p \overset{t}{\Rightarrow}_M v$, again by induction on the length of the derivation $p \overset{t}{\Rightarrow}_N v$. The only different case from above is for when $p \xrightarrow{\tau}_N p' \overset{t}{\Rightarrow}_N v$. In this case, by Lemma 15, there is some variable x, such that p is a sum of x and $p' = p_x$, or p is a sum of p_x and $p' = m_x$. Then, if p a sum of p_x, $p \xrightarrow{\tau}_M m_x = p'$ and by the inductive hypothesis, $p' \overset{t}{\Rightarrow}_M v$ and we are done. If p a sum of x, then $p' = p_x \neq v$, so the derivation is of the form $p \xrightarrow{\tau}_N p' = p_x \xrightarrow{\tau}_N m_x \overset{t}{\Rightarrow}_N v$ (as p_x can only transition to m_x), thus $p \xrightarrow{\tau}_M m_x$. 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 18. *For a monitor p, trace t, and value v, $p \overset{t}{\Rightarrow}_M v$ iff $p \overset{t}{\Rightarrow}_O v$.*

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 $\texttt{rec}\ x.m$ and 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 p and $q \equiv x$, $p \equiv p[q/x]$. We claim the following:

For every $p = \texttt{rec}\ x.q$ which appears in a derivation in O from p_0, $p \equiv p_x$: since only substitution can introduce new monitors of the form $\texttt{rec}\ x.q$ 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 $p_0 \overset{w}{\Rightarrow}_O \texttt{rec}\ y.r \xrightarrow{\tau}_O r[\texttt{rec}\ y.r/y]$, 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 $p_0 \overset{w}{\Rightarrow}_O \texttt{rec}\ y.r$. Let $p = \texttt{rec}\ x.q$ 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 $p' = \texttt{rec}\ x.q'$ submonitor of $\texttt{rec}\ y.r$ (thus, $p' \equiv p_x$) and $p = p'[\texttt{rec}\ y.r/y]$, but by the definition of $\equiv$, $p \equiv p' \equiv p_x$.

For a value v and monitor p, if $v \equiv p$, then $p = v$: notice that $\equiv_M$ such that for value v, $v \equiv_M p$ iff $v = p$ and for p, p' which are not values, always $p \equiv_M p'$, satisfies the above conditions, therefore $\equiv \subseteq \equiv_M$.

If $p + q \equiv r$, then there are $p' + q' = r$, where $p \equiv p'$ and $q \equiv q'$: notice that $\equiv$ can be constructed as the union $\bigcup_{i \in \mathbb{N}} \equiv_i$, where $\equiv_0$ includes all pairs (p, p), (x, p_x), (p_x, x), while $\equiv_{i+1}$ has all pairs in $\equiv_i$ and all $(p, p[q/x])$, $(p[q/x], p)$ and (p, r), (r, p), where $q \equiv_i x$ and $p \equiv_i p' \equiv_i r$; also, notice that neither x not p_x is of the form $p+q$ and all steps of the construction preserve the claim.

If $\alpha.p \equiv r$, then there is some $\alpha.q = r$, where $p \equiv q$: same as above.

If $\texttt{rec}\ x.p \equiv r$ or $x \equiv r$, then $x = r$ or there is some $\texttt{rec}\ x.q = r$, where $p \equiv q$: as above.

If $p \equiv r$ and p is a sum of q, then r is a sum of some $q' \equiv q$: by induction on the construction of a sum and the claim for $+$.

By the second claim, it is enough to prove that if $p \equiv p'$ and $p \xrightarrow{\mu}_O q$, then there is some $q' \equiv q$, such that $p' \xrightarrow{\mu}_M q'$ and vice-versa, if $p \equiv p'$ and $p' \xrightarrow{\mu}_M q'$, then there is some $q \equiv q'$, such that $p \xrightarrow{\mu}_O q$ (notice that this makes $\equiv$ a bisimulation).

For the first direction, let $p \equiv p'$ and $p \xrightarrow{\mu}_O q$.

If $\mu = \tau$, then by Lemma 16, p is a sum of $\texttt{rec}\ x.r$ and $q = r[p/x]$ — from the last claim above, we assume for simplicity that $p = \texttt{rec}\ x.r$ and it does not affect the proof. By the claims above, either $p' = x$ or $p' = \texttt{rec}\ x.q'$, where $r \equiv q'$. Since $r \equiv r[p/x] = q$, if $p' = \texttt{rec}\ x.q'$, then $p' = \texttt{rec}\ x.q' \xrightarrow{\tau}_M q' \equiv r \equiv q$ and we are done. Otherwise, $p' = x$ and by the first claim, $p \equiv p_x$, so $m_x \equiv r \equiv q$, therefore $p' = x \xrightarrow{\tau}_M m_x \equiv q$ and we are done.

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

For the opposite direction, let $p \equiv p'$ and $p' \xrightarrow{\mu}_M q'$.

If $\mu \in \text{Act}$, then the argument is as above. If $\mu = \tau$, then by Lemma 14, p' is a sum of x or p_x and $q' = m_x$. Therefore, by the claims above, either $p = x$, which cannot occur in system O, or $p = \texttt{rec}\ x.q$, for some $q \equiv m_x$. Therefore, $p \xrightarrow{\tau}_O q[p/x] \equiv q \equiv q'$. □

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

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 remaining section, we use system N and we call $\rightarrow_N$ and $\Rightarrow_N$ simply $\rightarrow$ and $\Rightarrow$, respectively.

When using these new systems, we need to alter the definition of determinism. Notice that in System O, it is possible to have a nondeterministic monitor, which has a deterministic submonitor. Specifically, all variables are deterministic. This is fine in System O, because variables do not derive anything on their own. In systems M and N, though, a variable x can derive p_x, so it is not a good idea to judge that any variable is deterministic — and thus judge the determinism of a monitor only from its structure. For example, if $p_x = \texttt{rec}\ x.(\alpha.x + \alpha.\texttt{yes})$, $x \overset{\tau}{\rightarrow}_N p_x$ and x is deterministic (by default), while p_x is not. Of course, so far all sums encountered in the course of an O-derivation can be already observed in the initial monitor, so the definition of determinism so far was enough (in our example above, simply $x \not\rightarrow$). The issue now are free variables in a monitor p, which can derive supermonitors of p. Therefore, we demand of the initial monitor p_0, that p_0 is deterministic (using the same definition as before). Since it is the determinism of p_0 itself we are interested in, we will not see much difference from using the usual definition.

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 $2^{2^{\Omega(n)}}$.

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 rela-

tion. Here we make this observation explicit.[4] For a monitor p, we define the automaton $A(p)$ to be $(Q, \textsc{Act}, \delta, q_0, F)$, where

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

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

Proof. We actually prove that for every $q \in Q$, $A_q(p) = (Q, \textsc{Act}, \delta, q, F)$ accepts t iff $q \stackrel{t}{\Rightarrow} \texttt{yes}$ and we do this by induction on t. If $t = \epsilon$, then $A_q(p)$ accepts iff $q \in F$ iff $q = \texttt{yes}$ iff $q \Rightarrow \texttt{yes}$. If $t = \alpha t'$, then
$A_q(p)$ accepts t
iff there is some $q' \in \delta(q, \alpha)$ such that $A_{q'}(p)$ 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 $A(p)$ has at most $|p|$ states (because Q only includes submonitors of p), but probably fewer, since two occurrences of the same monitor as submonitors of p 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 p is deterministic, then $A(p)$ is deterministic.

Corollary 2. *For every (deterministic) monitor p, there is an (resp. deterministic) automaton which accepts the same language and has at most $|p|$ states.*

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

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 was such a monitor for the language, since ϵ is in the language, the monitor can only be `yes`, which accepts everything (so, this conclusion is also true for any regular language of the form $\epsilon + L \neq \textsc{Act}^*$).

One of the properties which differentiates monitors from automata is the fact that verdicts are irrevocable for monitors. Therefore, if for a monitor p and finite trace t, $p \stackrel{t}{\Rightarrow} \texttt{yes}$, then for every trace t', it is also the case that $p \stackrel{tt'}{\Longrightarrow} \texttt{yes}$ (because of rule mVerd, which yields that for every t', $\texttt{yes} \stackrel{t'}{\Rightarrow} \texttt{yes}$). So, if L is

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

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.

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 $p(P)$ by recursion on $n - k$.
If $q_P = Y$, then $p(P) = \texttt{yes}$. Otherwise,

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

This was a recursive definition, because if $n = k$, all transitions from q_P lead back to a state in P. Let $p = p(q_0)$. Notice that p satisfies our assumptions that all variables x (i.e. x_P) are bound by a unique p_x (i.e. $p(P)$). 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 $p(P)$ — this includes all the generated sums of submonitors of the forms $\alpha.p(P\alpha q)$ and $\alpha.x_{P_q}$ and $p(P)$ itself. Specifically, if the

number of transitions from each state is at most Δ (Δ is at most $n|\textsc{Act}|$), then for each path P originating at q_0 we have generated at most 2Δ new submonitors of $p(P)$. If the number of paths originating at q_0 is Π, then

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

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

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

and thus,

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

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

Claim: for every such path P, $p(P) \stackrel{t}{\Rightarrow} \texttt{yes}$ if and only if A_{q_P} accepts t.
We prove the claim by induction on t.
If $t = \epsilon$, then $p(P) \stackrel{t}{\Rightarrow} \texttt{yes}$ iff $p(P) = \texttt{yes}$ 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 $p(P) = \texttt{yes} \stackrel{t}{\Rightarrow} \texttt{yes}$;
- if $q \in P$, then $p(P) \stackrel{\alpha}{\rightarrow} x_{P_q} \stackrel{\tau}{\rightarrow} p(P_q)$ and by the inductive hypothesis, $p(P_q) \stackrel{t'}{\Rightarrow} \texttt{yes}$;
- otherwise, $p(P) \stackrel{\tau}{\rightarrow}\stackrel{\alpha}{\rightarrow} p(P\alpha q)$ and by the inductive hypothesis, $p(P\alpha q) \stackrel{t'}{\Rightarrow} \texttt{yes}$.

If $t = \alpha t'$ and $p(P) \stackrel{t}{\Rightarrow} \texttt{yes}$ and $q_P \neq Y$, then there is some $q \in \delta(q_P, \alpha)$, such that either $p(P) \stackrel{\tau}{\rightarrow}\stackrel{\alpha}{\rightarrow} p(P\alpha q) \stackrel{t'}{\Rightarrow} \texttt{yes}$ (when $q \notin P$), or $p(P) \stackrel{\tau}{\rightarrow}\stackrel{\alpha}{\rightarrow} x_{P|_q} \stackrel{\tau}{\rightarrow} p(P|_q) \stackrel{t'}{\Rightarrow} \texttt{yes}$ (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 $2n \cdot |\textsc{Act}|^n = 2^{O(n)}$ which accepts the same traces as the automaton.*[5]

Proof. Notice that the proof of Theorem 6 works for DFAs as well. We use the same construction. Unless $p = \texttt{yes}$, every recursive operator (except the first

[5] Note that if $|\textsc{Act}| = 1$, 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.

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, 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 paths in A is thus at most

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

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

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

and therefore,

$$|p| \leq 2n|\textsc{Act}|^n.$$

□

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

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

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

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

Lower Bound for (nondeterministic) Monitor Size 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,

$$M_n = \{\alpha e \beta \in \{0,1,e\}^* \mid \alpha \in L_n\}.^6$$

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

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 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 p_α by induction on $n-k$: if $k = n-1$, $p_\alpha = e.\texttt{yes}$; otherwise, $p_\alpha = 0.p_{\alpha 0} + 1.p_{\alpha 1}$. Let $p = \texttt{rec}\ x.(0.x + 1.x + 1.p_\epsilon)$. Then, p mimics the behavior of the NFA for M_n and $|p| = 8 + |p_\epsilon| = O(2^n)$.

Definition 15. *We call a derivation $p \stackrel{t}{\Rightarrow} q$ 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 p if there is a simple derivation $p \stackrel{t}{\Rightarrow} q$. We say that a set G of simple traces for p is simple for p.*

Lemma 19. *Every subderivation of a simple derivation is simple.*

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

Lemma 20. *Let p be a monitor and G a (finite) simple set of traces for p. Then, $|p| \geq |G|$.*

Proof. By structural induction on p.

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

If $p = \alpha.q$, then all non-trivial derivations that start from p, begin with $p \xrightarrow{\alpha} q$. 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, $|q| \geq |G_\alpha|$, so $|p| = |q| + 1 \geq |G_\alpha| + 1 \geq |G|$.

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

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

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

Proof. Very similar to the proof of Lemma 20. □

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

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

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

Proof. A direct consequence of Corollary 10. □

Lemma 21. *In a derivation $p \stackrel{t}{\Rightarrow} x$, such that x is bound in p, 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 $p = x$ and x is not bound. Otherwise, notice that naturally, x is bound in p, but x is not bound in x. Let $p \stackrel{t'}{\Rightarrow} q \xrightarrow{\mu} q'$ be the longest initial subderivation such that x remains bound in q. Thus, it must be the case that x is not bound in q'. The only rule which can have this effect is mRecF, possibly combined with mSel, so q 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 22. *Let $p \stackrel{t}{\Rightarrow} q$, such that t is not simple for p. Then, there are $t = xuz$, such that $|u| > 0$ and for every $i \in \mathbb{N}$, $p \stackrel{xu^iz}{\Longrightarrow} q$.*

Proof. Let $p \stackrel{t}{\Rightarrow} q$ 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 $p \stackrel{s}{\Rightarrow} r$ is simple. Then, there is a $s\alpha \sqsubseteq t$, such that there is a subderivation of d, $p \stackrel{s}{\Rightarrow} r \xrightarrow{\alpha} r'$ and in $r \xrightarrow{\alpha} r'$, either mVerd or mRecB is used and neither is used in $p \stackrel{s}{\Rightarrow} r$. If mVerd is used, then $p \stackrel{s}{\Rightarrow} v$ as part of d, so $q = v$ and $p \stackrel{su^i}{\Longrightarrow} v$, for $su = t$ and $i \in \mathbb{N}$. If mRecB is used, then $p \stackrel{s}{\Rightarrow} x \xrightarrow{\tau=\alpha} p_x$; by Lemma 21, a sum of p_x acting as p_x, say s_x must appear in $p \stackrel{s}{\Rightarrow} x$, so $s_x \stackrel{u}{\Rightarrow} p_x$ 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 $i \in \mathbb{N}$, $p \stackrel{xu^iz}{\Longrightarrow} q$. □

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

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

Proof. Because of Lemma 20, it suffices to find a set G of simple traces for p, 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 22, since $p \stackrel{1se}{\Longrightarrow} \texttt{yes}$, there is a (strictly) shorter trace t, such that $p \stackrel{t}{\Rightarrow} \texttt{yes}$, which contradicts our assumption that p 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.

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 [8] for unary languages (languages on only one symbol).

Lemma 23. *Let p be a deterministic monitor. If $p \stackrel{t}{\Rightarrow} q$, then q is deterministic.*

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

Lemma 24. *If $q + q'$ is deterministic, then $q + q' \stackrel{\tau}{\not\rightarrow}$.*

Proof. The monitor $q + q'$ is a submonitor of the initial monitor, which is deterministic, so $q + q'$ must be of the form $\sum_{\alpha \in A} \alpha.p_\alpha$. We continue by induction on $q + q'$: if $q = \alpha.r$ and $q' = \beta.r'$, then by the production rules, only $q + q' \stackrel{\alpha}{\rightarrow} r$ and $q + q' \stackrel{\beta}{\rightarrow} r'$ are allowed; if one of q, q' is also a sum, then by the derivation rules, if $q + q' \stackrel{\tau}{\rightarrow}$, then also $q \stackrel{\tau}{\rightarrow}$ or $q' \stackrel{\tau}{\rightarrow}$, 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 25. *Let $p \sim p'$ be deterministic monitors. If $p \stackrel{t}{\Rightarrow} q$ and $p' \stackrel{t}{\Rightarrow} q'$, then $q \sim q'$.*

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

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

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

Corollary 15. *If p is deterministic, $p \overset{t}{\Rightarrow} q$, and $p \overset{t}{\Rightarrow} q'$, then $q \sim q'$.*

Corollary 16. *If p is deterministic, $p \overset{t}{\Rightarrow} q$, and $p \overset{t}{\Rightarrow} v$, where v is a value, then $q = v$.*

Lemma 26. *If p is a deterministic monitor and q is a submonitor of p, such that q is a sum of some p_x, then $q = p_x$.*

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

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

Proof. Combine Lemmata 21 and 26. □

Lemma 27. *Let $p \overset{t}{\Rightarrow} q$, such that t is not simple for p and p is deterministic. Then, there are $t = xuz$ and monitor r, such that $|u| > 0$ and $p \overset{x}{\Rightarrow} r \overset{u}{\Rightarrow} r \overset{z}{\Rightarrow} q$.*

Proof. Similar to the proof of Lemma 22, but use Corollary 17, instead of Lemma 21. □

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

Proof. We construct a set of simple traces of size $\Omega((2^{n/3-1}-1)!)$ and by Lemma 20 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 p. 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 $g_i = 1 b_i 10^k 1 b_i$.

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 $a \in A$, $0^k 1a$ 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. $l(x) = 0^{n_1} 1 a_1 1 a_2 10^{n_2}$ for some $n_1 + n_2 = k$ and $a_1, a_2 \in A$ (in this case, $10^k 1 a_1 1 a_2 10^{n_2} = 10^{n_2} l(x)$ are the last $n + n_2$ positions of x), or
2. $l(x) = d_1 1 a_1 10^k 1 d_2$ for some $d_2 \sqsubseteq a_1 \in A$, or
3. $l(x) = d_1 10^k 1 a_1 d_2$ 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 $l(y) = 0^{n_1} 1 a_1 1 a_2 10^{n_2}$, then $x = 1 a_1 1 a_2 10^{n_2}$, which is a contradiction, because x is not in an appropriate form (right after $1a_1 1$, there should be $0^k \neq a_2$);
if $l(x) = l(y) = d_1 1 a_1 10^k 1 d_2$, then $10^k 1$ 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 $l(x) = d_1 10^k 1 a_1 d_2$, then we already have a contradiction, because there is some $|a'| > k + 1$, such that x starts with $a' 10^k 1$, 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 $l(x) = l(y) = 0^{n_1} 1 a_1 1 a_2 10^{n_2}$, then $10^k 1 a_1 1 a_2 10^{n_2}$ 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 $10^k 1 a_1$ appears, a contradiction; the cases of the other forms are similar.

Now, if g is not simple, then by Lemma 27, there are $x \sqsubset y \sqsubseteq g$, such that $p \stackrel{x}{\Rightarrow} q$ and $p \stackrel{y}{\Rightarrow} q$. Furthermore, by Corollary 15, if $p \stackrel{x}{\Rightarrow} q_1$ and $p \stackrel{y}{\Rightarrow} q_2$, $q_1 \sim q \sim q_2$. If $xz \in M_n$, then $p \stackrel{x}{\Rightarrow} r \stackrel{z}{\Rightarrow} \texttt{yes}$, so $p \stackrel{y}{\Rightarrow} r'$, 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 $2^{2^{o(\sqrt{n \log n})}}$.*

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 [8] (errata at [9]), 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 $e^{o(\sqrt{n \log n})}$ 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 $1 \leq j \leq k$, p_i^j is the monitor defined recursively in the following way. Let

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

for $0 \leq l < m_j$, let

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

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

$$p_i^j = \texttt{rec}\ x_0.(\bar{i}.x_0 + \underbrace{i.\texttt{rec}\ x_1.(\bar{i}.x_1 + \cdots i.\texttt{rec}\ x_{m_j}.(\bar{i}.x_{m_j}}_{m_j} + i.x_1 + e.\texttt{yes})\cdots))).^7$$

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 `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 p 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 20, gives the same lower bound for $|p|$. Specifically, we prove that if t is not simple for p, 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 $p \overset{t_1}{\Rightarrow} r \overset{t_2}{\Rightarrow} s \overset{i}{\Rightarrow} r$ (by Lemma 27). Without loss of generality, we assume $i = 1$. Let A be the DFA $(Q, \{1\}, \delta, p, F)$, where Q is the set of all monitors appearing in the derivation $p \overset{t_1}{\Rightarrow} r \overset{t_2}{\Rightarrow} s \overset{1}{\Rightarrow} r$, $\delta(q_1, 1) = q_2$ 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

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

Notice that A is, indeed, deterministic, since transitions move along the derivation and all monitors in Q transition exactly once in $p \overset{t_1}{\Rightarrow} r \overset{t_2}{\Rightarrow} s \overset{1}{\Rightarrow} r$ — so the

[7] Notice that variable x appears in several scopes of `rec` x and thus violates our agreement that each variable is bound by a unique recursion operator (and the same can be said for x_b). However, this can easily be dealt with by renaming the variables and it helps ease notation.

first $\xrightarrow{1}$ 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) q, then there is some $w^t \in \{0,1\}^*$, such that $w^t|_1 = w$ and $p \overset{w^t}{\Longrightarrow} q$. Then,

A accepts w iff $q \in F$ iff for all $c \geq 0$, $q \overset{0^c e}{\Longrightarrow} \texttt{yes}$ iff for all $c \geq 0$, $p \overset{w^t 0^c e}{\Longrightarrow} \texttt{yes}$ (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 [16], to monitor for mHML-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:

$$\begin{aligned}
&\nu(\mathtt{no}) = [no].\mathtt{yes},\\
&\nu(x) = x,\\
&\nu(\alpha.m) = \alpha.\nu(m),\\
&\nu(m+n) = \nu(m) + \nu(n), \text{ and}\\
&\nu(\mathtt{rec}x.m) = \mathtt{rec}x.\nu(m).
\end{aligned}$$

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

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

Lemma 28. *Let $q = \nu(p)$. Then, $p \stackrel{t}{\Rightarrow} \mathtt{no}$ if and only if there is some $t' \sqsubseteq t$, such that $q \stackrel{t'}{\Rightarrow} \mathtt{no}.r$.*

Proof. Straightforward induction on the derivations. □

Lemma 29. *Let $q = \nu^{-1}(p)$, where verdict $\mathtt{no}$ does not appear in p. Then, there is some $t' \sqsubseteq t$ and a sum s of $[no].\mathtt{yes}$, such that $p \stackrel{t'}{\Rightarrow} s$, if and only if $q \stackrel{t}{\Rightarrow} \mathtt{no}$.*

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,
$m \stackrel{t}{\Rightarrow} \mathtt{no}$
iff $m \stackrel{t}{\Rightarrow} \mathtt{no}$ and $m \stackrel{t}{\not\Rightarrow} \mathtt{yes}$ (m is not conflicting)

iff there is a $t' \sqsubseteq t$ and $\alpha \in \text{Act}$, such that $m' \overset{t'}{\Rightarrow} [no].\texttt{yes}$ and $m' \overset{t'\alpha}{\not\Longrightarrow} \texttt{yes}$ (by Lemma 28)

iff there is a $t' \sqsubseteq t$ and $\alpha \in \text{Act}$, such that $m' \overset{t'[no]}{\Longrightarrow} \texttt{yes}$ and $m' \overset{t'\alpha}{\not\Longrightarrow} \texttt{yes}$ (there are no sums of `no` in m)

iff there is a $t' \sqsubseteq t$ and $\alpha \in \text{Act}$, such that $n' \overset{t'[no]}{\Longrightarrow} \texttt{yes}$ and $n' \overset{t'\alpha}{\not\Longrightarrow} \texttt{yes}$ (m' and n' are equivalent)

iff there is a $t' \sqsubseteq t$, an $\alpha \in \text{Act}$, and some $s \neq \texttt{yes}$, such that $n' \overset{t'}{\Rightarrow} s \overset{[no]}{\Longrightarrow} \texttt{yes}$ and $n' \overset{t'\alpha}{\not\Longrightarrow} \texttt{yes}$

iff there is a $t' \sqsubseteq t$, an $\alpha \in \text{Act}$, and a sum s of $[no].\texttt{yes}$, such that $n' \overset{t'}{\Rightarrow} s$ and $n' \overset{t'\alpha}{\not\Longrightarrow} \texttt{yes}$

iff there is a $t' \sqsubseteq t$ and $\alpha \in \text{Act}$, such that $n \overset{t'}{\Rightarrow} \texttt{no}$ and $n \overset{t'\alpha}{\not\Longrightarrow} \texttt{yes}$ (by Lemma 29)

iff $n \overset{t}{\Rightarrow} \texttt{no}$.

The case of the `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

$$V = \{(p, q) \mid p, q \text{ submonitors of } m\} \quad \text{and}$$

$$E = \{(p, q, p', q') \in V^2 \mid \exists \alpha \in \text{Act}.\ p \overset{\alpha}{\Rightarrow} p' \text{ and } q \overset{\alpha}{\Rightarrow} q'\}.$$

Then, m is conflicting iff there is a path from (m, m) to $(\texttt{yes}, \texttt{no})$ 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 [35]; 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 [16], 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 [16]. 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 amounts to keeping track of all possible derivations along the trace. This can take exponential time with regards to the size of the monitor, as n submonitors give rise to an exponential number of combinations of submonitors we could reach along a trace, if the trace is long enough. More importantly, 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 certain 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; furthermore, this bound is tight.
- 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 4, 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 [35].
– Similarly, using the constructions of [16], 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 formulas as in Subsection 3.2. Therefore, a CHML formula which 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 $2^{2^{\Omega(n)}}$. Therefore, NFAs as a specification language, 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.
– 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 [36] in deterministic space $O(\log^2 n)$).

The bounds we were able to prove can be found in Table 6.

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

from/to	DFA	monitor	det. monitor
NFA	tight: $O(2^n)$	upper: $2^{O(n \log n)}$ lower: $2^{\Omega(n)}$	tight: $2^{O(2^n)}$
DFA	X	upper: $2^{O(n)}$	tight: $2^{O(n)}$
nondet. monitor	upper: $O(2^n)$ lower: $2^{\Omega(\sqrt{n \log n})}$	X	upper: $2^{O(2^n)}$ lower: $2^{2^{\Omega(\sqrt{n \log n})}}$

Table 6. Bounds on the cost of construction (X signifies that the conversion is trivial)

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 $t \in \textsc{Act}^*$. We show that for every deterministic monitor p 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 p. By Lemma 27, there are $t'u = t$, such that $|u| > 0$ and $p \stackrel{t'}{\Rightarrow} r \stackrel{u}{\Rightarrow} r$. 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, $p \stackrel{ts}{\Rightarrow} \texttt{yes}$ iff $p \stackrel{t's}{\Rightarrow} \texttt{yes}$. □

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, NFA minimization is a hard problem — specifically PSPACE-complete [22] — and it remains NP-hard even for classes of NFAs which are very close to DFAs [6]. NFA minimization is even hard to approximate or parameterize [18,20]. Still, it would be better to use an efficient approximation algorithm from [20] 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.

References

1. Luca Aceto, Anna Ingólfsdóttir, Kim Guldstrand Larsen, and Jiri Srba. *Reactive Systems: Modelling, Specification and Verification*. Cambridge Univ. Press, New York, NY, USA, 2007.
2. A. Arnold and D. Niwinski. *Rudiments of μ-Calculus*. Studies in Logic and the Foundations of Mathematics. Elsevier Science, 2001.
3. Andreas Bauer, Martin Leucker, and Christian Schallhart. The good, the bad, and the ugly, but how ugly is ugly? In *RV*, volume 4839 of *LNCS*, pages 126–138. Springer, 2007.
4. Andreas Bauer, Martin Leucker, and Christian Schallhart. Comparing LTL semantics for runtime verification. *Logic and Comput.*, 20(3):651–674, 2010.
5. Andreas Bauer, Martin Leucker, and Christian Schallhart. Runtime verification for LTL and TLTL. *TOSEM*, 20(4):14, 2011.
6. Henrik Björklund and Wim Martens. The tractability frontier for NFA minimization. *Journal of Computer and System Sciences*, 78(1):198–210, 2012.
7. Ian Cassar and Adrian Francalanza. On implementing a monitor-oriented programming framework for actor systems. In Erika Ábrahám and Marieke Huisman, editors, *Integrated Formal Methods - 12th International Conference, iFM 2016*, volume 9681 of *Lecture Notes in Computer Science*, pages 176–192. Springer, 2016.
8. Marek Chrobak. Finite automata and unary languages. *Theoretical Computer Science*, 47:149–158, 1986.
9. Marek Chrobak. Errata to: "Finite Automata and Unary Languages". *Theoretical Computer Science*, 302(1):497–498, 2003.
10. Edmund M Clarke and E Allen Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Workshop on Logic of Programs*, pages 52–71. Springer, 1981.
11. Marcelo d'Amorim and Grigore Rosu. Efficient monitoring of ω-languages. In Kousha Etessami and Sriram K. Rajamani, editors, *Computer Aided Verification, 17th International Conference, CAV 2005*, volume 3576 of *Lecture Notes in Computer Science*, pages 364–378. Springer, 2005.
12. Søren Debois, Thomas T. Hildebrandt, and Tijs Slaats. Safety, liveness and runtime refinement for modular process-aware information systems with dynamic sub processes. In Nikolaj Bjørner and Frank S. de Boer, editors, *FM 2015: Formal Methods - 20th International Symposium*, volume 9109 of *Lecture Notes in Computer Science*, pages 143–160. Springer, 2015.
13. Cindy Eisner, Dana Fisman, John Havlicek, Yoad Lustig, Anthony McIsaac, and David Van Campenhout. Reasoning with temporal logic on truncated paths. In *CAV*, volume 2725 of *LNCS*, pages 27–39. Springer, 2003.
14. Ulfar Erlingsson. *The Inlined Reference Monitor approach to Security Policy Enforcement*. PhD thesis, Cornell University, 2004.
15. Yliès Falcone, Jean-Claude Fernandez, and Laurent Mounier. What can you verify and enforce at runtime? *STTT*, 14(3):349–382, 2012.
16. Adrian Francalanza, Luca Aceto, and Anna Ingolfsdottir. On verifying Hennessy-Milner logic with recursion at runtime. In Ezio Bartocci and Rupak Majumdar, editors, *Runtime Verification*, volume 9333 of *Lecture Notes in Computer Science*, pages 71–86. Springer International Publishing, 2015.
17. Marc Geilen. On the Construction of Monitors for Temporal Logic Properties. In *RV*, volume 55 of *ENTCS*, pages 181–199, 2001.

18. Gregor Gramlich and Georg Schnitger. Minimizing NFA's and regular expressions. *Journal of Computer and System Sciences*, 73(6):908–923, 2007.
19. Jim 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*, pages 3–12. IEEE Computer Society, 1986.
20. Hermann Gruber and Markus Holzer. Inapproximability of nondeterministic state and transition complexity assuming P≠NP. In Tero Harju, Juhani Karhumäki, and Arto Lepistö, editors, *Developments in Language Theory, 11th International Conference, DLT 2007*, volume 4588 of *Lecture Notes in Computer Science*, pages 205–216. Springer, 2007.
21. Yuqin He, Xiangping Chen, and Ge Lin. Composition of monitoring components for on-demand construction of runtime model based on model synthesis. In Hong Mei, Jian Lv, and Xiaoguang Mao, editors, *Proceedings of the 5th Asia-Pacific Symposium on Internetware, Internetware 2013*, pages 20:1–20:4. ACM, 2013.
22. Tao Jiang and B. Ravikumar. Minimal NFA problems are hard. *SIAM J. Comput.*, 22(6):1117–1141, 1993.
23. John Klein and Ian Gorton. Runtime performance challenges in big data systems. In C. Murray Woodside, editor, *Proceedings of the 2015 Workshop on Challenges in Performance Methods for Software Development, WOSP-C'15*, pages 17–22. ACM, 2015.
24. Dexter Kozen. Results on the propositional Îij-calculus. *Theoretical Computer Science*, 27(3):333–354, 1983.
25. Kim Guldstrand Larsen. Proof systems for satisfiability in Hennessy-Milner logic with recursion. *Theoretical Computer Science*, 72(2&3):265–288, 1990.
26. Martin Leucker and Christian Schallhart. A brief account of Runtime Verification. *JLAP*, 78(5):293 – 303, 2009.
27. Jay Ligatti, Lujo Bauer, and David Walker. Edit automata: enforcement mechanisms for run-time security policies. *Int. J. Inf. Secur.*, 4(1-2):2–16, 2005.
28. Qingzhou Luo, Farah Hariri, Lamyaa Eloussi, and Darko Marinov. An empirical analysis of flaky tests. In Shing-Chi Cheung, Alessandro Orso, and Margaret-Anne D. Storey, editors, *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, (FSE-22)*, pages 643–653. ACM, 2014.
29. Paul Dan Marinescu, Petr Hosek, and Cristian Cadar. Covrig: A framework for the analysis of code, test, and coverage evolution in real software. In Corina S. Pasareanu and Darko Marinov, editors, *International Symposium on Software Testing and Analysis, ISSTA '14*, pages 93–104. ACM, 2014.
30. Atif M. Memon and Myra B. Cohen. Automated testing of GUI applications: models, tools, and controlling flakiness. In David Notkin, Betty H. C. Cheng, and Klaus Pohl, editors, *35th International Conference on Software Engineering, ICSE '13*, pages 1479–1480. IEEE Computer Society, 2013.
31. Patrick O'Neil Meredith, Dongyun Jin, Dennis Griffith, Feng Chen, and Grigore Roşu. An overview of the MOP runtime verification framework. *STTT*, 14(3):249–289, 2012.
32. R. Milner. *A Calculus of Communicating Systems*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1982.
33. Amir Pnueli. The temporal logic of programs. In *Foundations of Computer Science, 1977., 18th Annual Symposium on*, pages 46–57. IEEE, 1977.
34. Michael O Rabin and Dana Scott. Finite automata and their decision problems. *IBM journal of research and development*, 3(2):114–125, 1959.

35. Alexander Rabinovich. A complete axiomatisation for trace congruence of finite state behaviors. In *International Conference on Mathematical Foundations of Programming Semantics*, pages 530–543. Springer, 1993.
36. Walter J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4(2):177–192, 1970.
37. M. Sipser. *Introduction to the Theory of Computation*. Computer Science Series. PWS Publishing Company, 1997.
38. M.Y. Vardi and P. Wolper. Reasoning about infinite computations. *Inf.& Comp.*, 115(1):1–37, 1994.
39. Sai Zhang, Darioush Jalali, Jochen Wuttke, Kıvanç Muşlu, Wing Lam, Michael D. Ernst, and David Notkin. Empirically revisiting the test independence assumption. In *ISSTA*, 2014.