On the Complexity of Determinizing Monitors*

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. 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 [23, 10] are computational entities that execute alongside a system so as to *observe* its runtime behavior and possibly determine whether a property is satisfied or violated from the exhibited (system) execution. They are used extensively in runtime verification [17] and are central to software engineering techniques such as monitor-oriented programming [6]. Monitors are often considered to be part of the trusted computing base and, as a result, are expected to behave correctly. A prevailing correctness criterion requires monitors to exhibit *deterministic behavior*. Determinism is also important for lowering the runtime overheads of monitoring a system: in order not to miss possible detections of a non-deterministic monitor, one would need to keep track of all the monitor states that are reachable for the currently observed execution trace.

Non-determinism is inherent to various computational models used to express monitors, such as Büchi automata [26, 8] or process calculi [5, 27, 10]. As a matter of fact, non-deterministic monitor descriptions are often more succinct than deterministic ones, and thus easier to formulate and comprehend. Non-deterministic computation is also intrinsic to concurrent and distributed programming — used increasingly for runtime monitoring [24, 18, 5, 9, 3] —, where the absence of global clocks makes it hard to rule it out, and interleaving under-specification can be used to improve execution efficiency.

In [11], Francalanza *et al.* identified a maximally-expressive monitorable fragment for the branching-time logic μHML [15, 16] and their results relied on a monitor-synthesis procedure for every monitorable μHML-formula. In order to achieve a simple compositional definition, this synthesis procedure may yield non-deterministic monitors. In this paper we tackle the problem of determinizing monitors in the framework of [11], which are described using syntax close to the regular fragment of CCS processes [20]. We demonstrate that every monitor can be transformed into an equivalent

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

Monitor Semantics

$$\textsc{Act}\ \frac{}{\alpha.m \xrightarrow{\alpha} m} \qquad \textsc{Sel}\ \frac{m_i \xrightarrow{\mu} m' \quad i \in I}{\sum_{i\in I} m_i \xrightarrow{\mu} m'} \qquad \textsc{Rec}\ \frac{}{\mathtt{rec}x.m \xrightarrow{\tau} m[\mathtt{rec}x.m/x]} \qquad \textsc{Ver}\ \frac{}{v \xrightarrow{\alpha} v}$$

Instrumentation Semantics

$$\textsc{Mon}\ \frac{p \xrightarrow{\alpha} p' \quad m \xrightarrow{\alpha} m'}{m \lhd p \xrightarrow{\alpha} m' \lhd p'} \qquad \textsc{Ter}\ \frac{p \xrightarrow{\alpha} p' \quad m \overset{\alpha}{\not\rightarrow} \quad m \overset{\tau}{\not\rightarrow}}{m \lhd p \xrightarrow{\alpha} \mathtt{end} \lhd p'} \qquad \textsc{AsP}\ \frac{p \xrightarrow{\tau} p'}{m \lhd p \xrightarrow{\tau} m \lhd p'} \qquad \textsc{AsM}\ \frac{m \xrightarrow{\tau} m'}{m \lhd p \xrightarrow{\tau} m' \lhd p}$$

Table 1. Monitor and Instrumentation Semantics (α ranges over ACT and μ over ACT $\cup \{\tau\}$)

deterministic one, which strengthens the results in [11]. However, we also show that the price of determinization can be a hefty one: there are monitors which require a doubly exponential blow-up in size to determinize. Note that, although our results employ the monitor framework of [11], our methods and findings can be extended to other forms of automata-like monitor descriptions such as those in [2, 5, 27, 10].

Overview: Section 2 provides the preliminaries. In Section 3, we prove that all monitors can be determinized and give methods to transform monitors to automata and back. Section 4 provides lower bounds to complement the constructions of Section 3. Section 5 discusses the main technical results in this paper. Omitted proofs and an extensive treatment of the determinization of monitors can be found in an extended version [1].

2 Background

We overview the main definitions for the monitoring set-up of [11] used in our study.

2.1 Basic Definitions: Monitoring Processes

Systems are denoted as processes whose semantics is given in terms of a labeled transition system (LTS). An LTS is a triple $\langle \textsc{Proc}, (\textsc{Act} \cup \{\tau\}), \rightarrow \rangle$ where PROC is a set of process states ($p \in \textsc{Proc}$), ACT is a finite set of observable actions ($\alpha \in \textsc{Act}$), $\tau \notin \textsc{Act}$ is the distinguished silent action, and $\rightarrow \subseteq (\textsc{Proc} \times (\textsc{Act} \cup \{\tau\}) \times \textsc{Proc})$ is a transition relation. Monitors are described via the specific syntax given below, but their semantics is also given as an LTS.

Definition 1. *A monitor is described by the following grammar:*

$$m \in \textsc{Mon} ::= \mathtt{yes} \quad | \ \mathtt{no} \quad | \ \mathtt{end} \quad | \ \alpha.m \quad | \ \sum_{i\in I} m_i \quad | \ \mathtt{rec}\, x.m \quad | \ x$$

where x comes from a countably infinite set of variables and $I \neq \emptyset$ is a finite index set. We write $m + n$ in lieu of $\sum_{i\in I} m_i$ when $|I| = 2$. Constants yes*,* no*, and* end *are called verdicts (denoted by v) and represent acceptance, rejection and inconclusive termination respectively. The behavior of a monitor is defined by the rules of Table 1.* ■

A *monitored system* is a monitor m and a system p instrumented to execute side-by-side, denoted as $m \triangleleft p$; its behavior is defined by the instrumentation rules in Table 1. Intuitively, a monitor m mirrors visible actions performed by p (rule MON). Whenever m cannot match an action from p and cannot internally transition to a state that might enable it to do so, $m \stackrel{\tau}{\not\rightarrow}$, then m aborts to the inconclusive `end` verdict (rule TER). Finally instrumentation monitors only for visible actions, and thus we allow m and p to perform internal τ actions independently of each other (rules ASP and ASM). Given an LTS with a set of states P (of processes, monitors, or monitored systems) with $r, r' \in P$ and a set of actions ($\text{ACT} \cup \{\tau\}$), we write $r \stackrel{\alpha}{\Rightarrow} r'$ to mean that r can weakly transition to r' using a single α action and any number of τ actions, $r(\stackrel{\tau}{\rightarrow})^*.\ \stackrel{\alpha}{\rightarrow}\ .(\stackrel{\tau}{\rightarrow})^* r'$. For each $r, r' \in P$ and trace $t = \alpha_1.\alpha_2.\ldots.\alpha_k \in \text{ACT}^*$, we use $r \stackrel{t}{\Rightarrow} r'$ to mean $r \stackrel{\alpha_1}{\Longrightarrow} . \stackrel{\alpha_2}{\Longrightarrow} \ldots \stackrel{\alpha_k}{\Longrightarrow} r'$ if t is non-empty and $r(\stackrel{\tau}{\rightarrow})^* r'$ if t is the empty trace.

In the monitorability results of [11] the verdicts `yes` and `no` (referred to hereafter as *conclusive* verdicts) are linked to satisfaction and violation of μHML formulas, respectively. We say that a monitor m accepts (resp. rejects) process p when there are a trace $t \in \text{ACT}^*$ and process p' such that $m \triangleleft p \stackrel{t}{\Rightarrow} \texttt{yes} \triangleleft p'$ (resp. $m \triangleleft p \stackrel{t}{\Rightarrow} \texttt{no} \triangleleft p'$). In this setting, acceptance is equivalent to saying that p can produce a trace t along which the monitor can derive the `yes` verdict, and similarly for rejection and verdict `no`. Thus, we say that a monitor m accepts (resp. rejects) a trace $t \in \text{ACT}^*$ when $m \stackrel{t}{\Rightarrow} \texttt{yes}$ (resp. when $m \stackrel{t}{\Rightarrow} \texttt{no}$). We say that two monitors, m and n are (verdict) equivalent, denoted as $m \sim n$, if for every trace t and verdict $v \in \{\texttt{yes}, \texttt{no}\}$, $m \stackrel{t}{\Rightarrow} v$ iff $n \stackrel{t}{\Rightarrow} v$. The utility of this monitor equivalence relation stems from the following fact: whenever $m \sim n$, then for *every* process state p, if monitor m accepts (resp. rejects) process p, then monitor n must accept (resp. reject) process p as well.

Multiple Verdicts In [11] the authors show that monitors with a *single conclusive verdict* suffice to adequately monitor for μHML formulae; these monitors can use either `yes` or `no`, but not both. We therefore confine our study to determinizing single-verdict monitors (particularly monitors that use the `yes` verdict), but note that there is a straightforward approach for dealing with multi-verdict monitors, which are used in other settings such as in [10]. For details on determinizing multi-verdict monitors consult [1].

Finite Automata We overview briefly Finite Automata Theory, used in Section 3; the interested reader should consult [25] for further details. A nondeterministic finite automaton (NFA) is a quintuple $A = (Q, \Sigma, \delta, q_0, F)$, where Q is a finite set of states, Σ is a finite set of symbols, called the alphabet (in our context, $\Sigma = \text{ACT}$), $\delta \subseteq Q \times \Sigma \times Q$ is a transition relation, $q_0 \in Q$ is the initial state, and $F \subseteq Q$ is the set of final or accepting states. Given a word $t \in \Sigma^*$, a run r of A on $t = t_1 \cdots t_k$ ($t_i \in \Sigma$, $1 \leq i \leq k$) is a sequence $q_0 q_1 \cdots q_k$, such that $(q_{i-1}, t_i, q_i) \in \delta$ for $1 \leq i \leq k$; r is an accepting run if $q_k \in F$. We say that A accepts t when A has an accepting run on t, and A accepts/recognizes a language $L \subseteq \Sigma^*$ whenever A accepts exactly all $t \in L$. In such cases L is unique and we call it $L(A)$. If δ is a function $\delta : Q \times \Sigma \to Q$ then A is a deterministic finite automaton (DFA). A classical result is that for every NFA A with n states, there is an equivalent DFA (i.e. a DFA that recognizes the language $L(A)$) with at most 2^n states [21]; this upper bound is optimal [19].

2.2 Determinism and Other Choices

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. We must therefore establish what we understand by a deterministic monitor. For the purposes of [11], monitor deterministic behavior need only concern itself with the definite verdicts that can be reached after observing a particular trace t. Stated otherwise, we can say that a monitor m behaves deterministically whenever it transitions to verdict-equivalent monitors for every trace t. The work in [11] contains several examples of monitors that break this behavioral condition: an easy one is $m_c = \alpha.\alpha.\mathtt{yes}+\alpha.\beta.\mathtt{yes}$, because when this monitor reads an α, it has to make a choice and transition to either $\alpha.\mathtt{yes}$ or $\beta.\mathtt{yes}$ which are *not* equivalent, $\alpha.\mathtt{yes} \not\sim \beta.\mathtt{yes}$. A deterministic monitor that is equivalent to m_c is $\alpha.(\alpha.\mathtt{yes} + \beta.\mathtt{yes})$.

For Turing machines, algorithms, and finite automata, determinism is typically more restrictive, requiring 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 by means of an observable action, α, but also via a τ-action, which can occur without reading from a trace. In finite automata, these τ actions could perhaps correspond to ϵ-transitions, which are eliminated from deterministic automata. However, we cannot readily eliminate τ-transitions from deterministic monitors. For instance, we need to be able to activate the recursive operators. Instead we require that monitor transitions denote functions that take us to a unique next state, and moreover that whenever a monitor can transition with an observable action α, it cannot perform silent actions. A closer inspection of the derivation rules of Table 1 immediately reveals that such choices can only be introduced by sums — that is, monitors of the form $\sum_{i\in I} m_i$ with $|I| \geq 2$; we can therefore attain the required behavior via syntactic constraints.

Definition 2. *A monitor m is* syntactically deterministic *iff every sum of at least two summands that appears in m is of the form $\sum_{\alpha\in A} \alpha.m_\alpha$, where $A \subseteq$ ACT.* ■

As we will see below, this set of monitors is in fact maximally expressive. Lemma 1 demonstrates that the syntactic determinism of Definition 2 ensures that such monitors will always arrive at the same verdict for a given trace. Following Lemma 1, we will simply refer to syntactically deterministic monitors as deterministic monitors.

Lemma 1. *If m is syntactically deterministic, $m \stackrel{t}{\Rightarrow} n$, and $m \stackrel{t}{\Rightarrow} n'$, then $n \sim n'$.* □

The first main result of the paper is that given a nondeterministic monitor, we can always find an equivalent deterministic monitor.

Theorem 1. *For each monitor $m \in$ MON there exists a deterministic monitor, $m' \in$ MON, such that $m \sim m'$.* □

Besides the constructions we present in this paper, in [1] we present two more methods to determinize monitors. The first is by reducing monitor determinization to the determinization of CCS processes modulo trace equivalence, which has been accomplished by Rabinovich in [22]. The second method is specific to the synthesis procedure of [11] via the determinization of μHML formulas. In either case, it is not easy to extract complexity bounds from these methods. See [1] for more details.

$$\textsc{RecF}\ \frac{}{\texttt{rec}x.m_x \xrightarrow{\tau} m_x} \qquad\qquad \textsc{RecB}\ \frac{}{x \xrightarrow{\tau} p_x}$$

Table 2. System N is the result of replacing rule REC by rules RECF and RECB.

2.3 Size Conventions

When we extract complexity bounds for our constructions, we assume that the set of actions, ACT, is of constant size. The size $|m|$ of a monitor m is the size of its syntactic description as given in Section 2.1, defined recursively thus: $|x| = |\texttt{yes}| = 1$; $|a.m| = |m| + 1$; $|\sum_{i\in I} m_i| = \sum_{i\in I} |m_i| + |I| - 1$; and $|\texttt{rec}\ x.m| = |m| + 1$. Notice that $|m|$ coincides with the total number of submonitor occurrences — namely, symbols in m.

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

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

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

2.4 Semantic Transformations

For convenience, we slightly alter the behavior of monitors from [11] to simplify our constructions and arguments. Specifically, we provide another set of transition rules and show that the new and old rules are equivalent with respect to the traces that can reach a `yes` verdict (the same applies for `no` verdicts). Consider a single monitor, m_0, which appears at the beginning of the derivation under consideration — that is, all other monitors are submonitors of m_0. 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; namely, m_x is the monitor such that $p_x = \texttt{rec}\ x.m_x$. The monitors may behave according to one of two systems of rules. System O is the old system of rules, as given in Table 1. System N is given by replacing rule REC by the rules given in Table 2. The transition relations $\xrightarrow{\mu}$ and $\stackrel{t}{\Rightarrow}$ are defined as before, but they are called $\xrightarrow{\mu}_O$ and $\stackrel{t}{\Rightarrow}_O$ when they result from System O and $\xrightarrow{\mu}_N$ and $\stackrel{t}{\Rightarrow}_N$ when they result from System N. We can show that the two LTSs are equivalent with respect to verdicts.

Lemma 2. *For a monitor m and trace t, $m \stackrel{t}{\Rightarrow}_N \texttt{yes}$ iff $m \stackrel{t}{\Rightarrow}_O \texttt{yes}$.* □

There are three reasons for changing the operational semantics rules of monitors. One is that, for the bounds we prove, we need to track when recursion is used in a derivation. Another is that in System N (unlike in System O) it is clear which monitors may appear in a derivation starting from monitor m (namely, at most all submonitors of m), which

in turn makes it easier to construct an LTS — and also to transform a monitor into an automaton. For instance, consider $m = \texttt{rec}\ x.(\alpha.x + \beta.\texttt{yes})$. In System O, $m \xrightarrow{\tau} \alpha.(\texttt{rec}\ x.(\alpha.x + \beta.\texttt{yes})) + \beta.\texttt{yes}$, which is *not* a subterm of m. On the other hand, in System N, $m \xrightarrow{\tau} \alpha.x + \beta.\texttt{yes}$, which is a subterm of m. Finally, and partly due to the previous observation, we can see that a monitor, viewed as an LTS, has a specific form: it is a rooted tree with labeled edges provided by $\xrightarrow{\mu}$, with some back edges, which result from recursion (namely, from the rule RECB in Table 2). For the remainder, we use system N and drop subscripts from $\xrightarrow{\mu}_N$ and $\overset{t}{\Rightarrow}_N$.

When using this new system, we need to be more careful with the definition of determinism. Notice that it is possible to have a nondeterministic monitor, which has a deterministic submonitor. For instance, $p_x = \texttt{rec}\ x.(\alpha.x + \alpha.\texttt{yes})$ is nondeterministic, while according to our definition of determinism, $\alpha.x$ is deterministic (specifically, all variables are deterministic). The issue here is that although $\alpha.x$ is deterministic in form, it can transition to (x and then to) p_x, which is not. This is not a situation we encountered in System O, because there variables do not derive anything on their own and all monitors we consider are closed. In System N, though, a variable x can appear in a derivation and it can derive p_x, so it is not a good idea to judge that any variable is deterministic — and thus judge the determinism of a monitor only from its structure. In other words, our definition of a deterministic monitor additionally demands that said monitor is closed; alternatively, for a monitor which appears in a derivation to be deterministic, we demand that the initial monitor p_0 be deterministic (by Definition 2).

3 Monitor Determinization

We provide methods to transform monitors to automata and back which, in turn, allows us to use the classic subset construction for the determinization of NFAs and thus determinize monitors. An advantage of this method is that it is not hard to extract upper bounds on the size of the constructed monitors. Another benefit is that, when transforming a monitor into an equivalent automaton, the constructed NFA may be smaller than the original monitor, thus resulting in a smaller deterministic monitor.

3.1 From Monitors to Finite Automata

A monitor can be seen as a finite automaton with its submonitors as states and the composition $\overset{\epsilon}{\Rightarrow} \cdot \xrightarrow{\alpha}$ as its transition relation. Here we make this observation explicit.[3] For a monitor m, we define the automaton $A(m)$ to be $(Q, \text{ACT}, \delta, q_0, F)$, where

- Q, the set of states, is the set of submonitors of m;
- ACT, the set of actions, is also the alphabet of the automaton;
- $q' \in \delta(q, \alpha)$ iff $q \overset{\epsilon}{\Rightarrow} \cdot \xrightarrow{\alpha} q'$;
- q_0, the initial state, is m;
- $F = \{\texttt{yes}\} \cap Q$, that is, $\texttt{yes}$ is the only accepting state (if it exists).

[3] 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 (see the previous explanation in Section 2.4).

Proposition 1. *For every monitor m and $t \in \text{ACT}^*$, $A(m)$ accepts t iff $m \stackrel{t}{\Rightarrow} \texttt{yes}$.* □

Thus, all languages recognized by monitors are regular. Notice that $A(m)$ has at most $|m|$ states (because Q only includes submonitors of m), but possibly fewer, since two occurrences of the same monitor as submonitors of m give the same state; we can cut the state size down further by removing submonitors which can only be reached through τ-transitions. Furthermore, if m is deterministic, then $A(m)$ is deterministic.

Corollary 1. *For every monitor m, there is an automaton that accepts the same language and has at most $|m|$ states. The automaton is deterministic if so is m.* □

3.2 From Automata to Monitors

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

One of the properties differentiating monitors from automata is that verdicts are irrevocable for monitors. Therefore, if for a monitor m and finite trace t, $m \stackrel{t}{\Rightarrow} \texttt{yes}$, then for every trace t', it is also the case that $m \stackrel{tt'}{\Longrightarrow} \texttt{yes}$ (this is due to rule VER which ensures that $\texttt{yes} \stackrel{t'}{\Rightarrow} \texttt{yes}$, for every t'). Stated otherwise, if L is a regular language on ACT that is recognized by a monitor, then L must be *suffix-closed*. Since this property stems from the fact that monitor verdicts are irrevocable, in the rest of this paper we instead refer to such languages as *irrevocable*.

Now, consider an automaton that recognizes an irrevocable language. Then, if q is any (reachable) accepting state of the automaton, and q can be reached through a word t, then t is clearly in the language but so is every word $t\alpha$. Thus, 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 that can *always* transition from an accepting state to an accepting state for each $\alpha \in \text{ACT}$ irrevocable. Note that, in the case of an irrevocable DFA, all transitions from accepting states *must* go to accepting states.

Corollary 2. *A language is regular and irrevocable if and only if it is recognized by an irrevocable NFA (or DFA).* □

Given an irrevocable NFA, we can construct an equivalent monitor through a procedure that can be described informally as follows (see Figure 1 for an example). We first unravel the NFA into a tree: for every transition sequence that starts from the initial state and that does not repeat any states, we keep a copy of its ending state. For example, for the automaton of Figure 1, we can reach q_2 through $q_0 \stackrel{0}{\rightarrow} q_1 \stackrel{1}{\rightarrow} q_2$ and $q_0 \stackrel{1}{\rightarrow} q_1 \stackrel{1}{\rightarrow} q_2$, which gives us two copies of q_2. Then, we map each node of this tree to a monitor, so that, at the end, the root is mapped to the resulting equivalent monitor. The leaves that correspond to an accepting state are mapped to $\texttt{yes}$. We use action transitions to

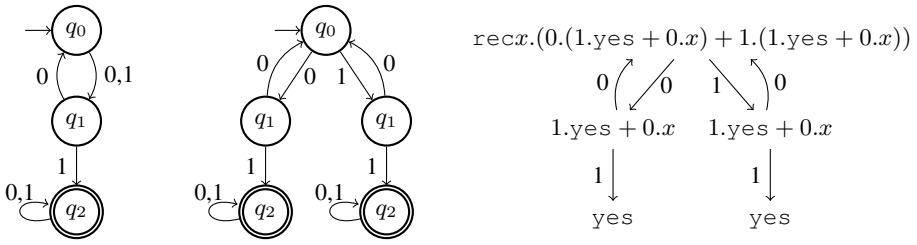

Fig. 1. Transforming an automaton into a monitor: DFA to tree unraveling to monitor.

describe forward tree edges and recursion for back edges — there is no need for cross edges. If the automaton is deterministic, so is the resulting monitor.

Theorem 2. *Given an irrevocable NFA (resp. DFA) A of n states, there is a monitor (resp. deterministic monitor) of size $2^{O(n \log n)}$ (resp. $2^{O(n)}$) that accepts $L(A)$.* □

By applying the transformations from a monitor into an NFA, into a DFA, and into a deterministic monitor, we obtain the following space complexity upper bound.

Corollary 3. *For every monitor m, there exists an equivalent deterministic monitor of size $2^{O(2^{|m|})}$.* □

4 Lower Bounds

As this section demonstrates, we cannot significantly improve the bounds of Section 3.

4.1 Lower Bound for (Nondeterministic) Monitor Size

It is easier to understand the intuition behind the lower bounds for constructing monitors after realizing that the LTS of a monitor is a rooted tree with additional back edges (when we consider each submonitor occurrence to be distinct). The tree is the monitor's syntactic tree; a transition generated by rules ACT and RECF (and then, possibly, SEL) is a transition from a parent to a child and a transition generated by rule RECB (and then, possibly, SEL) is a transition to an ancestor (rule VER gives self-loops for the leaves). Furthermore, from every node, distinct actions transition to distinct nodes. This is the form generated from the construction of Theorem 2.

We initially consider the family of regular languages $(L_n)_n$, where L_n, for $n \geq 1$, is described by $(0+1)^*1(0+1)^{n-1}$. 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 previously remarked, monitors do not behave exactly the same 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. Let $M_n = \{\alpha e \beta \in \{0,1,e\}^* \mid \alpha \in L_n\}$.

Note that an automaton (deterministic or not) accepting L_n can easily be transformed into one (of the same kind) accepting M_n by introducing two new states, Y and

N, where Y is accepting and N is not, so that all transitions from Y go to Y and from N go to N (N is a junk state, thus unnecessary for NFAs); then we add an e-transition from all accepting states to Y and from all other states to N. The reverse transformation is also possible: From an automaton accepting M_n, we can have a new one accepting L_n by removing 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.

So, 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 fewer. Let $A = \{0,1\}^{n-1}$ and $m = \texttt{rec}\, x.(0.x{+}1.x{+}1.\sum_{t\in A} t.e.\texttt{yes})$. Then, m mimics the behavior of the NFA for M_n and $|m| = O(2^n)$.

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

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

The above result means that monitors of size exponential with respect to n are required to recognize languages M_n, and thus we have a lower bound on the construction of a monitor from an NFA, which is close to the respective upper bound of Theorem 2.

4.2 Lower Bounds for the Size of Deterministic Monitors

Theorem 3. *Let m be a deterministic monitor for M_n. Then, $|m| = 2^{2^{\Omega(n)}}$.* □

Therefore, a construction of a deterministic monitor from an equivalent NFA can result in a doubly-exponential blow-up in the size of the monitor, and building a deterministic monitor from an equivalent nondeterministic one can result in an exponential blow-up in the size of the monitor. Hence, the upper bounds provided by Theorem 2 cannot be improved significantly. As Theorem 4 demonstrates, the situation is actually even worse for the determinization of monitors.

Theorem 4. *For every $n \in \mathbb{N}$, there is an irrevocable regular language on two symbols*[4] *that is recognized by a nondeterministic monitor of size $O(n)$, but which cannot be recognized by any deterministic monitor of size $2^{2^{o(\sqrt{n \log n})}}$.* □

The proof of Theorem 4 relies on a result by Chrobak [7] for unary languages (languages on only one symbol), who showed that, for every n, there is a unary language Ch_n that is recognized by an NFA with n states, but by no DFA with $e^{o(\sqrt{n \log n})}$ states. U_n is then the set of words $w \in \{0,1\}$, such that the 0's or the 1's in w are a word from Ch_n. Then, from a deterministic monitor for U_n we can extract a unary DFA for Ch_n by following the 0^*1- or 1^*0-transitions of the monitor, until the first time recursion was used (i.e. a back edge was followed). Therefore, the first time the deterministic monitor has a back edge is at distance at least $e^{\Omega(\sqrt{n \log n})}$ from the root; so, the deterministic monitor contains at least a complete binary tree of height $e^{\Omega(\sqrt{n \log n})}$.

[4] For unary languages, determinizing monitors is significantly easier; see [1].

5 Conclusions

We provided a method for determinizing monitors. We have focused on monitors for the co-safety fragment of μHML, as constructed in [11]. We showed that we can add a runtime monitor to a system without having a significant effect on the execution time of the system. Specifically, evaluating a nondeterministic monitor for a runtime trace may amount to keeping track of all possible monitor states reachable along that trace. By using a deterministic monitor, each trace event leads to a unique monitor state from the current state, which is easier to compute. However, this speed-up can come at a severe cost, since we may have to use up to doubly-exponential more space to store the monitor; even if this is stored in a more efficient form such as its LTS, the deterministic monitor may require an exponential additional space.

From the established bounds, NFAs can be exponentially more succinct than monitors as a specification language, and doubly exponentially more succinct than deterministic monitors; DFAs can be exponentially more succinct than deterministic monitors. Therefore, it is much more efficient to use monitors not in their syntactic forms, but as automata — or to use a monitor's syntax DAG instead of its syntax tree.

Summary of Bounds: We proved upper and lower bounds for several constructions related to monitor determinization. Table 3 summarizes the bounds we have proven, those which were known, and the ones we can further infer from these results. We discuss these below:

- Corollary 3 informs us that from a nondeterministic monitor of size n, we can construct a deterministic one of size $2^{O(2^n)}$.
- Theorem 4 explains that we cannot do much better, because there is an infinite family of monitors such that, for each monitor of size n in the family, there is no equivalent deterministic monitor of size $2^{2^{o(\sqrt{n \log n})}}$.
- Theorem 2 tells 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 n-state NFA of the family is not equivalent to any monitor of size $2^{o(n)}$.
- Corollary 3 yields that an irrevocable NFA of n states can be converted to an equivalent deterministic monitor of size $2^{O(2^n)}$; Theorem 3 makes this bound tight.
- Theorem 2 also allows us to convert a DFA of n states to a deterministic monitor of $2^{O(n)}$ states; Theorem 3 makes this bound tight.
- 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 in the proof of Theorem 2 to construct a deterministic monitor of $2^{2^{o(\sqrt{n \log n})}}$ states, which contradicts the lower bound of Theorem 4; therefore, $2^{\Omega(\sqrt{n \log n})}$ is a lower bound for converting monitors to equivalent DFAs.

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

Optimizations: Monitors to be used in runtime verification are expected not to affect the systems they monitor as much as possible. Therefore, the efficiency of monitoring must be taken into account to reduce overhead. To use a deterministic monitor, we would naturally want to keep its size as small as possible. It would help to preserve space (and time for each transition) to store the monitor in its LTS form — as a DFA. We should also aim to use the smallest possible monitor we can. There are efficient methods for minimizing a DFA, so one can use these to find a minimal DFA and then turn it into monitor form using the construction from Theorem 2, if such a form is required. The resulting monitor will be (asymptotically) minimal.

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 [14] — and it remains NP-hard even for classes of NFAs that are very close to DFAs [4]. NFA minimization is even hard to approximate or parameterize [12, 13]. Still, it would be better to use an efficient approximation algorithm from [13] 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 the light of the hardness results for NFA minimization.

References

1. Luca Aceto, Antonis Achilleos, Adrian Francalanza, Anna Ingólfsdóttir, and Sævar Örn Kjartansson. Determinizing monitors for HML with recursion. *arXiv preprint arXiv:1611.10212*, 2016.
2. Howard Barringer, Yliès Falcone, Klaus Havelund, Giles Reger, and David E. Rydeheard. Quantified Event Automata: Towards Expressive and Efficient Monitors. In *International Symposium on Formal Methods*, pages 68–84, 2012.
3. Shay Berkovich, Borzoo Bonakdarpour, and Sebastian Fischmeister. Runtime verification with minimal intrusion through parallelism. *Formal Methods in System Design*, 46(3):317–348, 2015.
4. Henrik Björklund and Wim Martens. The tractability frontier for NFA minimization. *Journal of Computer and System Sciences*, 78(1):198–210, 2012.
5. Laura Bocchi, Tzu-Chun Chen, Romain Demangeon, Kohei Honda, and Nobuko Yoshida. Monitoring networks through multiparty session types. In *Formal Techniques for Distributed Systems*, pages 50–65, 2013.

6. Ian Cassar and Adrian Francalanza. On Implementing a Monitor-Oriented Programming Framework for Actor Systems. In *integrated Formal Methods (iFM)*, pages 176–192, 2016.
7. Marek Chrobak. Finite automata and unary languages. *Theoretical Computer Science*, 47:149–158, 1986.
8. Marcelo d'Amorim and Grigore Roşu. Efficient monitoring of ω-languages. In *International Conference on Computer Aided Verification*, pages 364–378. Springer, 2005.
9. Pierre Fraigniaud, Sergio Rajsbaum, and Corentin Travers. On the number of opinions needed for fault-tolerant run-time monitoring in distributed systems. In *Runtime Verification*, pages 92–107, 2014.
10. Adrian Francalanza. A theory of monitors. In *International Conference on Foundations of Software Science and Computation Structures*, pages 145–161. Springer, 2016.
11. Adrian Francalanza, Luca Aceto, and Anna Ingolfsdottir. On verifying Hennessy-Milner Logic with Recursion at Runtime. In *Runtime Verification*, pages 71–86, 2015.
12. Gregor Gramlich and Georg Schnitger. Minimizing NFA's and regular expressions. *Journal of Computer and System Sciences*, 73(6):908–923, 2007.
13. Hermann Gruber and Markus Holzer. Inapproximability of nondeterministic state and transition complexity assuming $\mathsf{P} \neq \mathsf{NP}$. In *International Conference on Developments in Language Theory*, pages 205–216, 2007.
14. Tao Jiang and B. Ravikumar. Minimal NFA problems are hard. *SIAM Journal on Computing*, 22(6):1117–1141, 1993.
15. Dexter Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27(3):333–354, 1983.
16. Kim Guldstrand Larsen. Proof systems for satisfiability in Hennessy-Milner logic with recursion. *Theoretical Computer Science*, 72(2&3):265–288, 1990.
17. Martin Leucker and Christian Schallhart. A brief account of Runtime Verification. *The Journal of Logic and Algebraic Programming*, 78(5):293 – 303, 2009.
18. Qingzhou Luo and Grigore Roşu. EnforceMOP: A Runtime Property Enforcement System for Multithreaded Programs. In *International Symposium on Software Testing and Analysis*, pages 156–166, 2013.
19. Albert R Meyer and Michael J Fischer. Economy of Description by Automata, Grammars, and Formal Systems. In *12th Annual Symposium on Switching and Automata Theory*, 1971.
20. R. Milner. *A Calculus of Communicating Systems*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1982.
21. Michael O Rabin and Dana Scott. Finite automata and their decision problems. *IBM journal of research and development*, 3(2):114–125, 1959.
22. Alexander Rabinovich. A complete axiomatisation for trace congruence of finite state behaviors. In *International Conference on Mathematical Foundations of Programming Semantics*, pages 530–543, 1993.
23. Fred B. Schneider. Enforceable security policies. *ACM Transactions on Information and System Security (TISSEC)*, 3(1):30–50, February 2000.
24. Koushik Sen, Abhay Vardhan, Gul Agha, and Grigore Roşu. Efficient decentralized monitoring of safety in distributed systems. *International Conference on Software Engineering*, pages 418–427, 2004.
25. M. Sipser. *Introduction to the Theory of Computation*. Computer Science Series. PWS Publishing Company, 1997.
26. M.Y. Vardi and P. Wolper. Reasoning about infinite computations. *Information and Computation*, 115(1):1–37, 1994.
27. Yoriyuki Yamagata, Cyrille Artho, Masami Hagiya, Jun Inoue, Lei Ma, Yoshinori Tanabe, and Mitsuharu Yamamoto. Runtime monitoring for concurrent systems. In *Runtime Verification*, pages 386–403, 2016.