Consistently-Detecting Monitors*

Adrian Francalanza

1 CS, ICT, University of Malta, Msida, Malta
adrian.francalanza@um.edu.mt

Abstract

We study a contextual definition for deterministic monitoring based on consistent detections. It is defined in terms of the observed behaviour of the monitor when instrumented over arbitrary systems. We give an alternative, coinductive definition based on controllability which does not rely on system quantifications, and show that it is fully-abstract wrt. the former definition. We then develop a symbolic counterpart to the controllability definition to facilitate an automated analysis for controllable monitors involving data.

1998 ACM Subject Classification
F.3.1 Specifying and Verifying and Reasoning about Programs; F.3.2 Semantics of Programming Languages - Process models; D.2.5 Testing and Debugging

Keywords and phrases Runtime Monitoring, Deterministic Behaviour, Controllability, Compositional Reasoning, Symbolic Analysis

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2017.4

1 Introduction

Monitors are computational entities that observe the executions of other entities (referred to hereafter as systems) with the aim of accruing system information [32, 26], comparing system executions against some behavioural specification [23, 7], or reacting to the observed executions via adaptation or enforcement procedures [13, 36]. They are typically considered to be part of the Trusted Computing Base (TCB) and, consequently, their descriptions are expected to be correct. A correctness requirement often presumed of monitors is that they should exhibit deterministic behaviour. Yet, for most monitoring frameworks, such a requirement is seldom specified in unambiguous terms. In fact, there are a number of viable alternatives that one could consider (e.g., [44, 29, 25, 1]) and it is unclear how to choose one over the other in an objective manner. Moreover, these definitions often fail to account for the instrumentation mechanism used to compose a monitor with the system under scrutiny which may, in turn, affect monitoring behaviour. All of this leads to a poor understanding of what should be expected of a monitor, and may give rise to discrepancies between these expectations and what needs to be guaranteed by the monitor implementer in practice.

Non-determinism is intrinsic to a number of computational models used for expressing monitors and monitored systems. In fact, a substantial body of work on monitors is either cast in terms of inherently non-deterministic formalisms such as Büchi automata [45, 17], or formalisms that admit non-determinism such as process calculi and labelled transition systems [12, 46, 30, 22, 10, 23]. Non-deterministic computation arises naturally in concurrent and distributed programming, used increasingly for runtime monitoring [37, 24, 21, 8, 11]. Furthermore, a growing number of monitoring tools employ automata-based specification languages [16, 5, 41, 18] that offer rudimentary support for ensuring deterministic behaviour: their respective implementations are either thread-unsafe [41] or admit arbitrary code for transition-triggered actions [14, 16].

* Partly supported by UoM and RANNIS projects CPSRP05-04 and TheoFoMon:163406-051.
This paper sets out to investigate deterministic behaviour for monitors. The study is limited to execution monitors (sequence recognisers) [43, 36], used extensively for Runtime Verification (RV). Our work is developed in terms of an expository formalism (similar to the aforementioned work on transition-based descriptions) expressing monitored systems that can analyse trace events carrying data and admit degrees of non-determinism. We propose a contextual definition for deterministic monitor behaviour, founded on the observational behaviour that can be discerned when a monitor is instrumented to execute with any arbitrary system under scrutiny. The definition serves two purposes. First, its contextual nature allows us to admit as many correct monitors as possible, as long as these cannot be externally perceived to behave non-deterministically—we contend that the resulting definition is fairly intuitive. Second, it allows us to justify design decisions for an alternative definition describing the deterministic behaviour of monitors, based instead on the notion of controllability [31]. We show a correspondence between these two definitions. In addition, we demonstrate how the alternative definition (which is arguably less intuitive than its contextual counterpart) is more amenable to automated analyses for assessing the deterministic behaviour exhibited by monitors. In particular, we study how this alternative definition can be reformulated in symbolic terms, to facilitate a tractable handling of infinite-state monitor analysis due to data.

Example 1. The monitor description \( m_1 \) accepts traces from an authenticator, that challenges (event \( chl \)) a supplicant for an arbitrary value \( x \) followed by the supplicant’s authentication (\( aut \)) with the (correct) encoding of \( x \), \( y = enc(x) \). The authenticator subsequently acknowledges (\( ack \)) using the same value \( y \). The guard construct \( chl(x) \) quantifies over any value \( x \); guards \( aut(v) \) and \( ack(v) \) require authentications (resp. acknowledgments) for a specific value \( v \). \( \top \) denotes acceptance.

\[
\begin{align*}
m_1 & \triangleq chl(x). \text{let } y = enc(x) \text{ in } (aut(y).ack(y).\top) \\
m_2 & \triangleq chl(x). \text{let } y = enc(x) \text{ in } (aut(y).ack(y).\top + \text{ aut(z).if } z \neq y \text{ then } ack(z').\bot ) \\
m_3 & \triangleq chl(x). \text{let } y = enc(x) \text{ in } (aut(y).ack(y).\top + \text{ aut(z).if } z \neq y \text{ then } ack(z').\bot \text{ else } ack(z).\top)
\end{align*}
\]

It is easy to inadvertently introduce non-determinism. Monitor \( m_2 \) extends \( m_1 \) (using the choice operator, \( + \)) with the intention of rejecting (note the verdict \( \bot \)) acknowledgments following authentications whose value is not the encoding of the challenge value \( x \). When \( v_2 \neq enc(v_1) \), the violating trace \( t = chl(v_1).aut(v_2).ack(v_2) . . . \) is always rejected by \( m_2 \). More subtly, however, when \( v_2 = enc(v_1) \) the trace \( t \) may cause the monitor to non-deterministically choose either branch, whereby the unintended branch does not reach a \( \top \) verdict. But non-determinism is not necessarily conducive to inconsistent verdicts and, in cases where verdicts are considered as the only monitor observable behaviour, such non-determinism may be tolerated. Monitor \( m_3 \) deterministically accepts trace \( t \) when \( v_2 = enc(v_1) \), albeit along different execution paths, and deterministically rejects it whenever \( v_2 \neq enc(v_1) \).

The main contributions of the paper are:
1. A contextual definition for observationally deterministic monitoring behaviour, Def. 6.
2. An alternative definition based on controllability, Def. 11, that coincides with it, Thm. 13.
3. Symbolic variants of the latter definition enabling tractable automations, Thms. 21 and 25.

The paper is structured as follows. Sec. 2 presents the monitor framework used for this study, allowing us motivate our touchstone definition for deterministic monitor behaviour in Sec. 3. Sec. 4 presents a fully-abstract alternative definition that is amenable to compositional reasoning. In Secs. 5 and 6 symbolic variants are developed for automation purposes. Sec. 7 concludes.

2 Systems, Monitors, Instrumentation and Monitored Systems

We perceive systems as entities that generate events while executing. Observable events, \( \eta \in Evt \), are those visible to a monitor and have the form \( l(\nu) \) where \( l \) is an event label taken from a set \( L, k \in Lab \),
Monitors

\[ \begin{align*}
& w, o \in \text{VERD} := \top \quad \text{(accept)} \quad | \quad \bot \quad \text{(reject)} \\
& m, n \in \text{MON} := w \quad \text{(verdict)} \quad | \quad \text{let } x = e \text{ in } m \quad \text{(evaluate)} \\
& l(e).m \quad \text{(expression guard)} \quad | \quad l(x).m \quad \text{(quantified guard)} \\
& m + n \quad \text{(choice)} \quad | \quad \text{if } b \text{ then } m \text{ else } n \quad \text{(conditional)} \\
& \text{rec } X.m \quad \text{(recursion)} \quad | \quad X \quad \text{(monitor variable)}
\end{align*} \]

\[
\begin{array}{cccc}
\text{Valuations} & \text{Substitutions} & \text{Expressions} \\
\hline
\text{V}_{\text{val}} & \text{I}_{T} & \text{I}_{F} & \text{L}_{\text{ET}} \\
\hline
w \xrightarrow{\eta} w & [b] = \text{true} & [b] = \text{false} & [e] = v \\
\hline
l(e).m \xrightarrow{(\eta)} m & \text{rec } X.m \xrightarrow{m[\text{rec } X.m] X} & m+n \xrightarrow{\alpha} m' & l(x).m \xrightarrow{m[\text{rec } X.m] X}
\end{array}
\]

Instrumentation

\[
\begin{array}{cccc}
\text{GnE} & \text{GuQ} & \text{Rec} & \text{Ctrl} \\
\hline
\hline
s \xrightarrow{\eta} r & m \xrightarrow{\eta} n & \text{AsS} & \text{AsM} \\
\hline
s \xrightarrow{\eta} r \quad m \xrightarrow{\eta} n & s \xrightarrow{\eta} r \quad m \xrightarrow{\eta} n & s \xrightarrow{\eta} r \quad m \xrightarrow{\eta} n & s \xrightarrow{\eta} r \quad m \xrightarrow{\eta} n
\end{array}
\]

Figure 1 A Model for describing Instrumented Systems

and \( v \) is an event payload taken from some unspecified value domain \( v, u \in \text{VAL} \). Events capture a number of computational notions such as inputs/outputs in message-passing programs [22], or method/function calls and returns [13, 38]. To simplify our technical development, we consider monadic (i.e., single-valued) events but the formalism can be extended to accommodate polyadicity.

Systems may be described as Labelled Transition Systems (LTSs) [2, 34], triples \( (\text{Sys}, \text{Act}, \longrightarrow) \) consisting of a set of states, \( s, r \in \text{Sys} \), a set of actions, \( \alpha, \beta \in \text{Act} = \text{Evt} \cup \{\tau\} \) that include all observable events Evt and a distinguished silent action \( \tau \notin \text{Evt} \) for unobservable events, and a transition relation, \( \longrightarrow \subseteq (\text{Sys} \times \text{Act} \times \text{Sys}) \). The suggestive notation \( s \xrightarrow{\alpha} s' \) denotes \( (s, \alpha, s') \in \longrightarrow \), whereas \( s \xrightarrow{\tau} \) denotes \( \neg(\exists s' : s \xrightarrow{\cdot} s') \). As usual, we write \( s \Rightarrow s' \) in lieu of \( s(\xrightarrow{\cdot}) s' \) and \( s \not\Rightarrow s' \) for \( s \not\Rightarrow \cdot : \neg \Rightarrow s' \), referring to \( s' \) as a \( \eta \)-derivative of \( s \); \( s \not\Rightarrow \cdot \) stands for \( \exists s' : s \not\Rightarrow s' \). We let traces, \( t, u \in \text{Evt}^* \), range over (finite) sequences of observable events and write \( s \not\Rightarrow \cdot \not\Rightarrow \cdots \not\Rightarrow s_n \) as \( s \not\Rightarrow s_n \), where \( t = \eta_1, \ldots, \eta_n \). The notation \( u \ldots \) is occasionally used to denote the existence of some trace \( t \) with a prefix \( u \).

We presuppose an expression language \( e, d \in \text{Exp} \) that ranges over the (event) value domain \( \text{Val} \) and a denumerable set of expression variables \( x, y, z \in \text{Vars} \); \( \bar{e} \) and \( \bar{x} \) resp. denote lists of expressions and variables. We also assume a boolean expression language \( b, c \in \text{BExp} \) defined over \( \text{Exp} \) that includes standard constructs for conjunctions, \( b \land c \), and negations, \( \neg b \), but also equality predicates over expressions \( e = d \). The meta-functions \( \text{fv}(e) \) and \( \text{fv}(b) \) return the free variables in the resp. expressions; expressions are closed whenever \( \text{fv}(e) = \emptyset \) and open otherwise, and similarly for boolean expressions. Valuations are total maps from variables to values, \( \rho \in (\text{Vars} \rightarrow \text{Val}) \) whereas substitutions are partial maps from variables to expressions \( \sigma \in (\text{Vars} \rightarrow \text{Exp}) \); substitutions are denoted as \( [\gamma] \), where \( d[\gamma] \) represents the (simultaneous) substitution of all occurrences of \( x_i \in \bar{x} \) in \( d \) by the corresponding
We assume an evaluation function\(\llbracket expr\rrbracket\) that takes an expression and a valuation and returns a value, \(\llbracket expr\rrbracket = v\). Similarly, boolean expressions have a semantic function mapping them to the boolean domain via a valuation, \(\llbracket bp\rrbracket \in \{true, false\}\), where we presume the expected properties, e.g., \(\llbracket (b \land c) \rrbracket \) is true if \(\llbracket bp\rrbracket = true\) and \(\llbracket cq\rrbracket = true\). We also assume a classical interpretation of boolean expressions, i.e., \(\llbracket \neg bp\rrbracket = \neg \llbracket bp\rrbracket\). To alleviate the presentation, we often work up to associativity and commutativity for conjunctions, treating \(\langle BExp, \land, true\rangle\) as an abelian monoid. For closed expressions, we elide the valuation and write \(\llbracket e\rrbracket\) and \(\llbracket b\rrbracket\) for \(\llbracket expr\rrbracket\) and \(\llbracket bp\rrbracket\) resp. The satisfiability judgement for boolean expressions, \(\text{sat}(b) = \exists \rho. \llbracket bp\rrbracket = true\), plays a central role in subsequent development.

Monitors, here defined by the syntax in Fig. 1, constitute the focus of our study. They may reach two kinds of verdicts, Verdf. Conclusive verdicts consist of acceptances, \(\top\), and rejections, \(\bot\). In addition, a monitor may also reach the inconclusive verdict, \(\bot\), a form of premature termination used when the generated system events of the monitor specification itself does not yield sufficient information so as to reach a definite conclusion. The monitor expression guard \(l(e).m\) expects events with label \(l\) and a payload value matching the evaluation of \(e\), whereas the quantified guard \(l(x).m\) allows the monitor to dynamically learn the payload of an event with label \(l\). Monitors may branch (externally) depending on the events observed, \(m + n\), or branch (internally) based on data predicates, if \(b\) then \(m\) else \(n\). They may also perform internal computation themselves by evaluating expressions,\[\text{let } x = e \text{ in } m, \text{ or recurse, rec } X.p, \text{ via term variables } X,Y,Z \in \text{TVARS}\]. The constructs \(l(x).m\) and \(\text{rec } X.m\) act as binders for \(x\) and \(X\) resp. in \(m\), inducing the usual notions of open/closed (monitor) terms. We work up to alpha-conversion of bound expression/term variables and use the shorthand if \(b\) then \(m\) for if \(b\) then \(m\) else \(\bot\) and \(\tau.m\) for let \(x = v\) in \(m\) where \(x \notin \text{fv}(m)\).

The semantics of closed monitors is also defined in terms of an LTS, via the transition rules in Fig. 1: for each \(m \in \text{Mon}\) we have a dedicated LTS \(\langle M, \text{Act}, \rightarrow \rangle\) where \(M \subseteq \text{Mon}\) are the monitors reachable from \(m\) via transitions. The rules model the monitor analysis of observable events. Rule Ver describes how verdicts are irrevocable, meaning that a verdict can analyse any observable event but always transition to itself. In rule GaE, an expression guard \(l(e).m\) only transitions to the continuation \(m\) when observing an event matching the label \(l\) with the payload equal to \(\llbracket e\rrbracket\). By contrast, a quantified guard \(l(x).m\) transitions by analysing any event with label \(l\), binding \(x\) to the event payload \(v\) in the continuation, \(m[v/x]\); see rule GaQ. The remaining rules are as expected where the term \(m[\text{rec } X.m][x]\) denotes the term substitution of \(\text{rec } X.m\) for free occurrences of \(X\) in \(m\).

A system \(s\) instrumented with a monitor \(m\) is referred to as a monitored system and denoted as \(s \cdot m\). The semantics of monitored systems is defined by the instrumentation rules in Fig. 1. We here adopt the composition relation studied in [22, 23], even though other instrumentation relations could have been used. Note that the chosen composition relation is still quite general: it is parametric wrt. the system and monitor abstract LTSs and it is largely independent of their specific language specifications, since it only requires the monitor LTS to contain an inconclusive (persistent) verdict state, \(\bot\). The instrumentation relation of Fig. 1 is asymmetric: a monitored system can transition with an observable event only when the system can produce that event i.e., monitors are passive and cannot instigate transitions. When the system generates an (observable) event that can be analysed by the monitor, the two transition in lockstep according to their respective LTSs (rule Mon). When the monitor cannot analyse the event generated \(^1\) and cannot internally transition to a state that enables it to do so (i.e., it is already stable, \(m \nrightarrow v\)), the instrumentation does not block the monitored system: instead, it allows the system to transition but aborts monitoring to the inconclusive verdict (rule Ter). System-monitor synchronisations are limited to observable events, and the specific entities can transition independently wrt. their respective internal moves (rules AsS and AsM).

\(^1\) This may be due to a number of reasons, such as event knowledge gaps or knowledge disagreements [6].
We focus on identifying payload \( v \in \mathbb{N} \) reports the port number over which the communication operation is performed.

\[
m_2 \triangleq \text{rec } X.((\text{out}(80), \bot) + (\text{in}(x), \text{if } x = 80 \text{ then } \text{out}(81).\top \text{ else } \text{out}(x), X))
\]  

(1)

The monitor rejects system executions starting with an output on port 80 but accepts traces containing an input on port 80 followed by an output on port 81, preceded by an arbitrary number of input-output operations on any matching port other than 80. The execution below shows an accepted monitored computation for a system \( s \) generating the trace \( \text{in}(85) \cdot \text{out}(80) \cdot \text{in}(80) \cdot \text{out}(81) \). In monitor \( m_4 \), the binding on \( \text{in}(x) \) acts as a freeze-variable [19] for the subsequent \( \text{out}(x) \) guard in the else branch.

\[
s : m_4 \xrightarrow{\text{in}(85)} s' \text{ if } 85 = 80 \text{ then } \text{out}(81).\top \text{ else } \text{out}(5), m_4
\]

\[
\xrightarrow{\text{out}(80)} s'' \text{ if } m_4 \xrightarrow{\text{in}(80) \cdot \text{out}(81)} s'' \text{ else } \top
\]

The instrumentation of Fig. 1 delays system transitions to allow the monitor to internally transition to a state that can process the event. E.g., if a system \( r \) can generate event \( \text{out}(80) \), \( r \neq m_4 \) postpones this transition (\( \text{MON} \) and \( \text{TER} \) cannot be applied) until \( m_4 \) unfolds.

\[
r : m_4 \xrightarrow{\text{in}(80)} r + (\text{in}(x), \text{if } x = 80 \text{ then } \ldots) \xrightarrow{\text{out}(80)} r' \xrightarrow{\bot}
\]

A.SM, MON

Rule \( \text{T} \) is crucial both for allowing monitored computations to proceed when the monitor cannot analyse an event, but also to avoid unintended detections. E.g., if system \( r \) can generate the trace \( \text{out}(90) \cdot \text{in}(80) \cdot \text{out}(81) \), this behaviour should still be permitted when instrumented with the monitor \( m_4 \), but the behaviour should not be detected according to the description in Eq. (1). After the initial unfolding of \( m_4 \), \( \text{T} \) allows \( r \) to transition with \( \text{out}(90) \) but transitions \( m_4 \) to the inconclusive state, \( \bot \), since neither guard \( \text{out}(80) \) nor guard \( \text{in}(x) \) can process the event.

\[
r : m_4 \xrightarrow{\text{in}(80)} r + (\text{in}(x), \text{if } x = 80 \text{ then } \ldots) \xrightarrow{\text{out}(90)} r'' \xrightarrow{\bot} \text{in}(80) \cdot \text{out}(81) \xrightarrow{\text{in}(80)} r''' \xrightarrow{\bot} \bot
\]

A.SM, TER

Had rule \( \text{T} \) been designed otherwise (leaving the monitor state unaltered when transiting with \( \text{out}(90) \)) the ensuing events \( \text{in}(80) \cdot \text{out}(81) \) would lead to the unintended acceptance of the trace.

\section{Deterministic Monitoring Behaviour}

In a monitored system, non-deterministic behaviour can be caused by either the system or the monitor. We focus on identifying non-determinism attributed to monitors, teasing it apart from non-determinism caused by system behaviour. This is motivated by the fact that, generally, one has limited control over the behaviour of a system under scrutiny. We target a definition that admits monitor non-determinism that is not externally observable. Concretely, we consider detections (i.e., conclusive verdicts) as the only externally visible aspect of a monitor and base our definition on the notion of deterministic detections—in applications such as RV, detections are associated with property satisfactions and violations [23, 7]. This immediately rules out a number of candidate definitions for deterministic monitor behaviour. For instance, a definition that considers a monitor \( m \) to be deterministic whenever, for all systems \( s \) and traces \( t \), \( s \neq m \Rightarrow s' \neq m' \) and \( s \neq m \Rightarrow s' \neq m'' \) implies \( m' = m'' \) is too stringent: it precludes the monitor description below (a slight modification on \( m_1 \) of E.g. 1).

\[
m_5 \triangleq \text{chl}(x).((\text{let } y = \text{enc}(x) \text{ in } \text{aut}(y).\text{ack}(y).\top) + (\text{aut}(\text{enc}(x)).\text{ack}(\text{enc}(x)).\top))
\]

(2)

Even though \( m_5 \) deterministically accepts traces of the form \( \text{chl}(v_1) \cdot \text{aut}(v_2) \cdot \text{ack}(v_2) \) where \( v_2 = \text{enc}(v_1) \). In fact, after an event \( \text{chl}(v) \) (for some value \( v \)), monitor \( m_5 \) can reach two possible internal states,
4:6 Consistently-Detecting Monitors

namely (let y = enc(v) in aut(y).ack(y).τ) + (aut(enc(v)).ack(enc(v)).τ) or aut(v').ack(v').τ where v' = enc(v). Other candidates (e.g., confluence defined over transitions [25, 40]) are either inadequate or not immediately applicable because they do not account for executions that do not lead to detections. E.g., m6 (below) would not be confluent (consider event in(81)), even though it consistently rejects any trace with the prefix u = in(80) (and consistently does not detect all the other traces).

\[ m_6 \triangleq (\text{in}(80), \bot) + ((\text{in}(81), \text{out}(81), 0) + (\text{in}(81), \text{out}(81), \text{in}(82), 0)) \]  

Definition 3 (Detected Computations). The transition sequence

\[ s \prec m \xrightarrow{\tau} s_0 \prec m_0 \xrightarrow{\tau} s_1 \prec m_1 \xrightarrow{\tau} s_2 \prec m_2 \xrightarrow{\tau} \ldots \]

is called a t-computation if it is maximal i.e., either it is infinite or it is finite and cannot be extended further using τ-transitions. The t-computation above is called accepted whenever \( \exists i \in \mathbb{N} \cdot m_i = \top \) and rejected when \( \exists i \in \mathbb{N} \cdot m_i = \bot \). A detected t-computation is either an accepted or a rejected one.

Detected computations are indexed by their trace to allow us to partition computations according to the system behaviour exhibited at runtime, thus accounting for system non-determinism. Def. 3 also permits monitors to stabilise and reach verdicts in the trailing τ-sequence following a t-trace.

Definition 4 (Deterministic Detection and Withholding). Monitor m deterministically accepts (resp. deterministically rejects) for system s along trace t ∈ Evτ*, denoted as \( \text{da}(m, s, t) \) and \( \text{dr}(m, s, t) \) resp., if all t-computation from s * m are accepting (resp. rejecting). Monitor m deterministically detects for s along t, \( \text{dd}(m, s, t) \), whenever \( \text{da}(m, s, t) \) or \( \text{dr}(m, s, t) \). Monitor m deterministically withholds for s along trace t, \( \text{dw}(m, s, t) \), iff no t-computation from s * m is accepting or rejecting.

Example 5. For arbitrary system s, monitors m1 of Eq. 1 and m5 of Eq. 2 deterministically accept traces with the prefix \( t = \text{ch}(v_1).\text{aut}(v_2).\text{ack}(v_2) \) where \( v_2 = \text{enc}(v_1) \) and deterministically withhold on all the other traces. Monitor m2 deterministically rejects traces with the prefix t above when \( v_2 \neq \text{enc}(v_1) \) but does not deterministically detect traces with prefix t when \( v_2 = \text{enc}(v_1) \). For arbitrary s, monitor m3 deterministically detects any trace with the above prefix t (accepting or rejecting the trace depending on whether \( v_2 = \text{enc}(v_1) \) or not) and deterministically withholds otherwise. For any system s, monitor m4 of Eq. 2 satisfies \( \text{dr}(m_4, s, t) \) when the trace t is of the form \( t = \text{out}(80) \ldots \text{da}(m_4, s, t) \) when \( t = (\text{in}(v_i) \cdot \text{out}(v_i)) \cdot (\text{in}(80) \cdot \text{out}(81)) \ldots \) for some \( i \in \mathbb{N} \), and \( \text{dw}(m_4, s, t) \) otherwise. Similarly, for all systems s, m6 satisfies \( \text{dr}(m, s, t) \) when \( t = \text{in}(80) \ldots \) and \( \text{dw}(m, s, t) \) otherwise.

For the rest of our study, monitors with deterministic behaviour are defined as consistently-detecting.

Definition 6 (Consistent Detection). Monitor m consistently detects for system s, denoted as \( \text{cd}(m, s) \) iff for all traces t we have \( \text{dd}(m, s, t) \) or \( \text{dw}(m, s, t) \). A monitor m is consistently-detecting, denoted as \( \text{cd}(m) \), whenever \( \text{cd}(m, s) \) holds for any system s.

Example 7. Monitors m1, m3, m4, m5 and m6 are consistently-detecting, but m2 is not. Def. 6 does not require monitors to perform any detections. The monitor \( m_7 \triangleq \text{rec} X.(\text{in}(81), X) + (\text{in}(81), \text{out}(81), X) \) can consistently analyse an infinite number of traces for any s, \( \text{cd}(m_7) \), even though it never flags.

A few comments are in order. First, Def. 6 abstracts away from the particular instances of the systems considered, the specifics of the monitor language and instrumentation mechanism used; this makes is applicable to arbitrary monitoring setups. Second, \( \text{cd}(m, s) \) may be seen as requiring an ambiguity of d=1 from automata theory [29, 4], for the observable behaviour specified in Def. 4. Our setting is however more general, allowing for infinite states and alphabets (actions). Moreover, \( \text{cd}(m) \) quantifies over all possible system compositions. Third, since Def. 6 is defined over monitored system behaviour, it allows us to assess the actual monitor behaviour at runtime. Particularly, the system quantification in \( \text{cd}(m) \) accounts for any (indirect) effects of a system on the execution of a monitor.
Example 8. Whereas monitor $m_8 \triangleq \text{in}(81) \cdot \bot$ is (trivially) consistently-detecting in the framework of Fig. 1, the monitor $m_9 \triangleq \text{in}(x)\text{.If } x = 81 \text{ then } \bot \text{ else } \top$ is, perhaps surprisingly, not. Consider a (diverging) system $s$ with behaviour $s \xrightarrow{\text{in}(81)} s' \xrightarrow{\text{ff}} s''$. Although $s \not\xrightarrow{m_9}$ can reject the $t$-computation for $t=\text{in}(81)$, another possible $t$-computation of $s \xrightarrow{m_9}$ is

$$s \xrightarrow{m_9} (s' \not\xrightarrow{81 = 81 \text{ then } \bot \text{ else } \top} \xrightarrow{\text{ff}} s'' \not\xrightarrow{81 = 81 \text{ then } \bot \text{ else } \top} \xrightarrow{\text{ff}} \ldots$$

which never reaches a verdict. Therefore, we have $\neg \text{cd}(m_9)$ according to Def. 6.

Example 9. Although $m_8$ (from E.g. 8) and monitor $m_{10} = \text{in}(81) \cdot \top$ are both consistently-detecting according to Def. 6, their composition, i.e., $m_8 + m_{10}$, is clearly not.

4 Controllability

In spite of its generality and intuitive nature, Def. 6 it is hard to automate directly as a correctness analysis. One major obstacle is the inherent universal quantification over systems and traces defining $\text{cd}(m)$. In this section, we set out to give an alternative definition for describing consistently-detecting monitors that does not suffer from these shortcomings. It is based on the notion of controllability [20, 31] which, in discrete event settings, roughly refers to the ability to steer a (passive) entity to designated terminal states via a series of admissible controls. In our case, the monitors will constitute the passive entities to be steered, whereas the monitored systems assume the controller’s role: the admissible controls are effectively the observable events in a monitoring setup that cause the monitor to transition, whereas the terminal states of interest are the conclusive verdicts. The proposed definition thus inverts the focus from how a system is monitored to how a monitor can be driven.

Before giving the actual definition, we first need to lift the technical machinery of Fig. 1 to sets of monitors, $M, N \subseteq \text{Mon}$: this allows us to express the status whereby a monitor that can be in a number of potential states after being driven by a sequence of steering controls, which facilitates the analysis of non-compositional properties such as ours (see E.g. 9).

Definition 10. A monitor-set $M$ potentially reaches a verdict $w$, $\text{pr}(M, w)$, when $\exists m \in M \cdot m \Rightarrow w$, and potentially analyses an event $\eta$, $\text{pa}(M, \eta)$, when $\exists m \in M \cdot m \xrightarrow{\eta}$. Function $\text{aft}(M, \eta)$ is defined as:

$$\text{aft}(M, \eta) \triangleq \bigcup_{m \in M} \text{aft}(m, \eta) \quad \text{aft}(m, \eta) \triangleq \{ n \mid m \Rightarrow \eta \Rightarrow n \cup \{0 \mid \exists n \cdot m \Rightarrow n \xrightarrow{\eta} \text{ and } n \not\xrightarrow{\eta} \} \}

Intuitively, $\text{aft}(M, \eta)$ computes the set of reachable states from every $m \in M$ when it is asked by the instrumentation of Fig. 1 to analyse an event $\eta$. The two conditions defining $\text{aft}(m, \eta)$ correspond to the monitored system transitions dictated by the respective rules $\text{Mon}$ and $\text{Inf}$ in Fig. 1.

Definition 11 (Controllability). A relation $R \subseteq \mathcal{P}(\text{Mon})$ is controllable iff for all $M \in R$:

1. $\text{pr}(M, w)$ and $w \in \{\top, \bot\}$ implies $M = \{w\}$;
2. $\text{pa}(M, \eta)$ implies $\text{aft}(M, \eta) \subseteq R$.

Controllability, denoted as the relation $C$, is the largest controllable relation. A monitor $m$ (resp. monitor-set $M$) is said to be controllable iff $\{m\} \in C$ (resp. $M \in C$).

Controllability is coinductive: to show that a monitor $m$ is controllable, i.e., $\{m\} \in C$, it suffices to provide a witness controllable relation $R$ such that $\{m\} \in R$. Condition (i) in Def. 11 requires that if some $m \in M$ can reach a conclusive verdict, then every $m' \in M$ must be able to do so immediately, without requiring any preceding $\tau$-moves (hence $M = \{w\}$); this rules out the possibility of inconsistent detections and, at the same time, prohibits diverging systems from interfering with the reaching of such verdicts (see E.g. 8). Condition (ii) in Def. 11 intuitively requires that this condition is satisfied for any event $\eta$ observed, by all the states that any $m \in M$ may transition to when analysing $\eta$.  

CONCUR 2017
Consistently-Detecting Monitors

- **Example 12.** We can show that \( m_6 \) (Eq. (3)) is controllable via the controllable relation \( R_1 \) below:

\[
R_1 = \left\{ \{m_6\}, \{\top\}, \{0, \text{in}(82), 0\}, \{0\}, \{\text{out}(81), 0, \text{in}(82), 0\} \right\}
\]

\[
R_2 = \left\{ \{m_7\}, \{m_7, \text{out}(81), m_7\}, \{0, m_7, 0\}, \{0, m_7, \text{out}(81), m_7\}, \{0\} \right\}
\]

Note that \( \{m_6\} \in R_1 \). We can also finitely determine that the recursive monitor \( m_7 \) (E.g. 7) is controllable via the relation \( R_2 \). The reader may want to check that \( R_2 \) is controllable. For instance, \( \text{aft}([m_7, \text{out}(81), m_7], \text{in}(81)) = \{0, m_7, \text{out}(81), m_7\}, \text{aft}([m_7, \text{out}(81), m_7], \text{in}(81)) = \{0, m_7\} \) and, importantly, \( \text{aft}([0, m_7, \text{out}(81), m_7], \text{in}(81)) = \{0, m_7, \text{out}(81), m_7\} \) itself.

Controllability coincides with Def. 6: we can use Def. 11 as a sound and complete proof technique to determine whether a monitor \( m \) satisfies \( \text{cd}(m) \), side-stepping universal quantifications over systems.

- **Theorem 13** (Consistent Detection Full Abstraction). \( \text{cd}(m) \iff \{m\} \in C \)

- **Example 14.** As a result of Thm. 13, we can show that \( m_6 \) and \( m_7 \) are consistently-detecting via the controllable relations \( R_1 \) and \( R_2 \) of E.g. 12. We can also indirectly show that \( \neg \text{cd}(m_8 + m_{10}) \) from E.g. 9, by arguing that there cannot be a controllable relation \( R \) with \( \{m_8 + m_{10}\} \in R \). For suppose that such an \( R \) exists. By Def. 11(ii) the monitor-set \( \text{aft}([m_8 + m_{10}], \text{in}(81)) = \{\top, \bot\} \) must also be in \( R \); this, in turn, would necessarily mean that \( R \) is not controllable since \( \{\top, \bot\} \) violates Def. 11(i).

5. **Symbolic Controllability**

Controllability, Def. 11, is still not adequate for a fully automated analysis of consistently-detecting monitors. Particularly, whenever the resp. value domain is infinite, quantified guards induce an infinite number of transitions, e.g., \( l(x) m \) generates a transition with the label \( l(y) \) for every \( v \in \text{Vars} \) (see rule GsQ). As a result, condition Def. 11(ii) may require the monitor analysis to consider a potentially infinite number of monitor-set states whenever monitor descriptions use quantified guards.

To this end, we define a symbolic semantics over both open\(^2\) and closed monitor terms using symbolic events, \( \theta \in \text{SEvt} \). These are similar to the events of Sec. 2 except that they carry variables instead of values as payloads, \( l(x) \). Symbolic transitions, \( m \xrightarrow{\mu} n \) are defined by the rules in Fig. 2, where \( \mu \in \text{SEvt} \cup \{\tau\} \) ranges over both symbolic and \( \tau \) events, and the boolean expression \( b \) records the condition under which the LTS action may take place. For instance, the term \( \text{sRec} X.m \) may unfold in all circumstances (i.e., \( b = \text{true} \) in rule sRec) whereas the term if \( b \text{ then } m \text{ else } n \) can either \( \tau \)-transition to \( m \) when \( b \) holds, or to \( n \) when the converse, \( \neg b \), holds (rules slfT and slfF). The other key rules in Fig. 2 are sGxE and sGxQ: the former transitions with a symbolic event \( l(x) \) under the condition \( x = e \), whereas the latter transitions with a similar symbolic event under any circumstance.

Fig. 2 also defines rules for weak symbolic transitions, \( m \xrightarrow{\theta} n \), and reductions, \( m \Rightarrow n \), where both relations aggregate boolean constraints via conjunctions. Note that weak symbolic transitions describe transition sequences where \( \tau \)-transitions must precede the (final) symbolic event transition.

The predicate \( m \xrightarrow{\mu} \) denotes \( \exists b, n : m \xrightarrow{\mu} n \) whereas \( m \Rightarrow n \) stands for \( \exists \theta : m \xrightarrow{\theta} n \).

A constrained monitor-set \( \langle b, M \rangle \) is a tuple where every \( m \in M \) may be open, and \( b \) is a condition constraining free variables in \( M \). Every \( \langle b, M \rangle \) abstractly represents a (potentially infinite) set of closed monitor-sets for every valuation \( \rho \) satisfying \( b \).

\[
\{ mp \mid \text{[b]} = \text{true} \text{ and } m \in M \}
\]  

\(^2\) Open wrt. expression variables \( x, y, \ldots \in \text{Vars} \) not term variables \( X, Y, \ldots \in \text{TVars} \).
In this sense, the monitor-sets in Sec. 4 are special cases of constrained monitor-sets where $b = \text{true}$ and $M$ is closed. Note that whenever $\neg \text{sat}(b)$, the constrained monitor-set $(b, M)$ denotes the empty set of monitor-sets, $\emptyset$, which is trivially controllable by Def. 11. We lift functions such as that for free variables $fv(-)$ to constrained monitor-sets in the obvious manner, e.g., $fv((b, M)) \equiv fv(b) \cup fv(M)$.

**Example 15.** The constrained monitor-set $(x \geq 3, [if \ x = 2 \ then \ T \ else \ \bot])$ abstractly describes all monitor-sets $\{if \ x = 2 \ then \ \top \ \bot\} \rho$ where $\rho(x) \geq 3$. For any such $\rho$, no monitor of the form $(if \ x = 2 \ then \ \top \ \bot\) \rho$ can transition to a $\top$ verdict according to the concrete semantics of Fig. 1. Symbolically, this may be expressed as $\neg \text{sat}(x \geq 3 \land x = 2)$.

We abstractly model controllability, Def. 11, in terms of constrained monitor-sets, the symbolic semantics of Fig. 2 and the satisfiability judgement $\text{sat}(b)$ defined earlier in Sec. 2.

**Definition 16.** A constrained monitor-set $(b, M)$ potentially reaches a verdict $w$, denoted as $\text{spr}(b, M, w)$, whenever $\exists m \in M, c \cdot m \Rightarrow w$ and $\text{sat}(b\land c)$. Moreover, $(b, M)$ potentially analyses a symbolic event $\theta$ along $c$, denoted as $\text{spal}(b, M, \theta, c)$, whenever $\exists m \in M \cdot m \overset{\theta}{\Rightarrow} c$ and $\text{sat}(b\land c)$.

Defining the symbolic counterpart to $\text{aff}(M, \eta)$ of Def. 10 is less straightforward. Intuitively, from all the valuations $\rho$ satisfying (and represented by) $b$ in $(b, M)$, only a subset of them may satisfy the condition $c$ in a potentially-analyses judgement $\text{spal}(b, M, \theta, c)$ from Def. 16. A correct modelling of Def. 11 therefore requires us to take this fact into account.

**Example 17.** Consider the constrained monitor-set $(\text{true}, M)$ where $M = \{m_{10}, m_{11}\}$ and

$$m_{10} \equiv if \ x = 2 \ then \ k(1).\top \ else \ k(1).\bot \quad m_{11} \equiv if \ x < 1 \lor \ x \geq 3 \ then \ k(1).\bot \ else \ k(1).\top$$

It turns out that for any $\rho$ satisfying $\text{true}$, $M\rho$ is controllable. Therefore, for the judgement $\text{spal}(\text{true}, M, k(y), (x = 2 \land y = 1))$ of Def. 16, which holds since $m_{10} \overset{k(y)}{\Rightarrow} m_{11} \overset{x = 2 \land y = 1}{\Rightarrow} \text{sat}(\text{true} \land x = 2 \land y = 1)$, the reachable states to be considered by a corresponding symbolic analysis (modelling Def. 11(ii)) should not include the residual state $\bot$, even though it may be reached after the event $k(y)$ with $y = 1$ (see rule $\text{GrE}$). The reason for this is that the conditions required to symbolically reach this state, i.e., $\neg ((x = 2) \land y = 1)$ or $((x \leq 1 \lor x \geq 3) \land y = 1)$, cannot be satisfied by any $\rho$ that also satisfies the $\text{spa}(\neg)$ condition $(x = 2 \land y = 1)$. Symbolically, this may be expressed as $\neg \text{sat}(x = 2 \land y = 1) \land ((x = 2 \land y = 1))$ and $\neg \text{sat}(x = 2 \land y = 1) \land ((x \leq 1 \lor x \geq 3) \land y = 1)$.

The complications elicited in Example 17 are even more intricate. For instance, for a particular judgement $\text{spal}(b, M, \theta, c)$, one could have some $m_1, m_2 \in M$ whereby $m_i \overset{c_i}{\Rightarrow} m_i'$ and $\text{sat}(b\land c_i\land c_i')$ for $i \in [1, 2]$, but at the same time having $c_1$ and $c_2$ being incompatible with one another, i.e., $\neg \text{sat}(c_1 \land c_2)$. In such cases, the respective residual states $m_i'$ and $m_i''$ should be analysed separately.

**Definition 18.** The relevant conditions for a monitor-set $M$ wrt. a symbolic event $\theta$ are:

$$\text{rc}(M, \theta) \equiv \{ c \mid \exists m \in M \cdot (m \overset{\theta}{\Rightarrow} c \text{ or } \exists n \cdot (m \Rightarrow n \land n \overset{\theta}{\Rightarrow} c)) \}$$

The satisfiability combinations for a condition-set $\{c_1, \ldots, c_n\}$ wrt. a condition $b$ are:

$$\text{sc}(b, \{c_1, \ldots, c_n\}) \equiv \{ (b, c_i', \ldots, c_n') \mid \forall i \in [1..n] \cdot (c_i' = c_i \text{ or } c_i' = \neg c_i) \}$$

The reachable constrained monitor-sets from $(b, M)$ after $\theta$ with condition $c$ are:

$$\text{saft}(b, M, \theta, c) \equiv \{ \langle \bot, \text{sat}(M, B, \theta) \rangle \mid B \in \text{sc}(b\land c, \text{rc}(M, \theta)) \text{ and } \text{sat}(\bot B) \}$$

$$\text{saft}(M, B, \theta) \equiv \{ \langle m \cdot \text{sat}((\bot B) \land c) \rangle \mid \exists n \in M \cdot (m \overset{\theta}{\Rightarrow} n \lor (\exists n' \cdot m \overset{\theta}{\Rightarrow} n' \land n' \overset{\theta}{\Rightarrow} c)) \}$$
Symbolic Transitions

<table>
<thead>
<tr>
<th>sVra</th>
<th>slrT</th>
<th>srfT</th>
<th>sRfc</th>
</tr>
</thead>
<tbody>
<tr>
<td>( w \xrightarrow{\eta} w )</td>
<td>if ( b ) then ( n \xrightarrow{\tau} m ) else ( \neg b )</td>
<td>if ( b ) then ( n \xrightarrow{\tau} m ) else ( \neg b )</td>
<td>( \text{rec } X.m \xrightarrow{\tau} m[\text{rec } X.m[X]] )</td>
</tr>
</tbody>
</table>

Definition 19

- Symbolic Controllability

\( \text{spa}(b,M), l(x), c \) where \( \text{frsh}(l((b,M))) = x \) implies \( \text{saft}((b,M), l(x), c) \subseteq \mathcal{S} \).

Symbolic Controllability, denoted as \( C_{\text{sym}} \), is the largest symbolically-controllable relation. A (closed) monitor \( m \) is symbolically-controllable iff \( \langle \text{true}, [m] \rangle \in C_{\text{sym}} \).

Example 20. Recall \( m_{10} \) and \( m_{11} \) from E.g. 17. The monitor \( m_{12} \) can be shown to be symbolically-controllable via the relation \( S_{1} \) defined below, where \( b_{1} = (x = 2 \land y = 1) \), etc.
b_2 = (\neg(x = 2) \land y = 1), \ b_3 = ((x \leq 1 \lor x \geq 3) \land y = 1) \text{ and } b_4 = (\neg(x \leq 1 \lor x \geq 3) \land y = 1); \text{ these are obtained from the relevant conditions } \text{rec}(m_{[10, m_{11}}), k(y)) = \{b_1, b_2, b_3, b_4\}.

\begin{align*}
S_1 = & \left\{ \langle \text{true}, \{m_{12}\} \rangle, \langle \text{true}, \{m_{10}, m_{11}\} \rangle, \langle \text{true} \land b_1 \land b_2 \land \neg b_3 \land b_4, \{\top\} \rangle, \langle \text{true} \land b_4 \land b_1 \land \neg b_2 \land \neg b_3 \land b_4, \{\top\} \rangle, \\
& \langle \text{true} \land b_3 \land \neg b_1 \land b_2 \land \neg b_3 \land b_4, \{\bot\} \rangle, \langle \text{true} \land b_3 \land \neg b_1 \land b_2 \land \neg b_3 \land b_4, \{\bot\} \rangle \right\}
\end{align*}

For illustrative purposes, we do not simplify the constraints in the constrained monitor-sets of \(S\) to show how these are derived. E.g., \(\langle \text{true} \land b_1 \land \neg b_2 \land \neg b_3 \land b_4, \{\top\} \rangle\) is obtained as a result of \(\text{saft}((\text{true}, \{m_{10}, m_{11}\}), l(x), \text{true} \land b_1)\). In fact, the combination \(((\text{true} \land b_1), b_1, \neg b_2, \neg b_3, b_4)\) is the only satisfiable condition-set and all the others are filtered out by \(\text{saft}((\text{true}, \{m_{10}, m_{11}\}), l(x), \text{true} \land b_1)\).

Symbolic Controllability, Def. 19, is sound and complete wrt. Controllability, Def. 11.

\subsection*{Example 22.} \(\text{Recall } m_3 \triangleq \text{chi}(x).m'_3\) from E.g. 1, recast in terms of \(m'_3\) defined below as:

\[m'_3 \triangleq \forall y = \text{enc}(x) \in (\text{aut}(y) \land \text{ack}(y) \lor \text{true} \land \text{if } z \neq y \text{ then } \text{ack}(z) \land \bot \text{ else } \text{ack}(z) \land \text{true})\]

E.g. 7 stated that \(m_3\) is consistently-detecting. This fact is hard to determine using Def. 6, whereas analyses using Def. 11 are complicated by quantifications over the values of events. By Thms. 13 and 21, we can show that \(m_3\) is consistently-detecting via the symbolic controllability relation:

\[S_2 = \left\{ \langle \text{true}, \{m_3\} \rangle, \langle \text{true}, m'_3 \rangle, \langle z = \text{enc}(x), \text{ack}((\text{enc}(x)) \land \text{true} \land \text{if } z \neq \text{enc}(x) \text{ then } \text{ack}(z) \land \bot \text{ else } \text{ack}(z) \land \text{true}) \rangle, \langle \langle z = \text{enc}(x) \land (w = z) \land (w = \text{enc}(x)), \{\top\} \rangle, \langle \langle \neg(z = \text{enc}(x)), \{\top\} \rangle \rangle \right\}
\]

In \(S_2\) and the ensuing discussion, we alleviate our presentation by simplifying the boolean conditions used, e.g., we simply write \((z = \text{enc}(x))\) in lieu of \(((\text{true} \land z = \text{enc}(x)))\). We highlight a few points.

First, consider the second constrained monitor-set in \(S_2\), namely \((\text{true}, m'_3)\). Since the semantics of Fig. 2 allows expression guards and quantified guards to transition with the same symbolic event (albeit with different conditions) we are able to consider the resp. continuations in unison for the event \(\text{aut}(z)\). Concretely, according to Def. 19(ii), for \(\text{spa}((\text{true}, m'_3), \text{aut}(z), (z = \text{enc}(x))\rangle\) generated by the expression guard weak transition of \(m'_3\), we need to ensure that the resulting monitor-set \(((z = \text{enc}(x)), \text{saft}(m'_3), (z = \text{enc}(x)), \text{true}, \text{aut}(z)))\) (which evaluates to the third constrained monitor-set in \(S_2\)) is also in the symbolic relation. At the same time, for \(\text{spa}((\text{true}, m'_3), \text{aut}(z), \text{true})\langle z = \text{enc}(x))\rangle\) generated by the quantified guard weak transition of \(m'_3\), we need to ensure that two monitor-sets are in \(S_2\), namely \((z = \text{enc}(x)), \text{saft}(m'_3), (z = \text{enc}(x)), \text{true}, \text{aut}(z))\) (as before) but also the constrained monitor-set \(((z = \text{enc}(x)), \text{saft}(m'_3), (z = \text{enc}(x)), \text{true}, \text{aut}(z)))\) (which evaluates to the fifth constrained monitor-set in \(S_2\)).

The second point we highlight about \(S_2\) concerns its third constrained monitor-set. In particular, the left branch of the conditional term in this set, namely \(\text{ack}(z) \land \bot \text{ in the term if } z \neq \text{enc}(x)\text{ then } \text{ack}(z) \land \bot \text{ else } \text{ack}(z) \land \text{true})\), is incompatible with the constraining condition of the monitor-set, i.e., \(\neg\text{saft}(z = \text{enc}(x) \land z \neq \text{enc}(x))\).

Third, we also note how the condition aggregation mechanism for the consecutive symbolic events \(\text{aut}(z)\) and \(\text{ack}(w)\) and \(\text{ack}(w)\) — transferring us from the second constrained monitor-set, \((\text{true}, m'_3)\), to the fourth, \(((z = \text{enc}(x)) \land (w = z) \land (w = \text{enc}(x)), \{\top\})\), via the third constrained monitor-set in \(S_2\) — enables us to symbolically relate the expression guards in \(m_3\), which impose a condition such as \((z = \text{enc}(x))\) upon transition, with the quantified guard that imposes the same condition after the transition (by means of a conditional branch in its continuation). We leave it up to the interested reader to check that the remaining monitor-sets in \(S_2\) satisfy the conditions required by Def. 19.
6 On Automating Symbolic Controllability

Despite its merits, a direct implementation of the symbolic controllability from Def. 19 still would not perform well for certain recursive monitor descriptions, as shown in the following example.

► Example 23. Recall monitor $m_4$ from E.g. 2. To show that it is controllable, we need to exhibit a symbolic relation that includes $(\text{true}, [m_4])$. For some fresh variable $x$ where $\text{frsh}(\text{fv}((\text{true}, [m_4])) = x$, since the judgement $\text{spa}(\text{true}, [m_4], \text{in}(x), \text{true})$ holds, this relation needs to include the ensuing monitor-set $(\text{true}, [m_4])$ as well, where $m_4' \doteq \begin{cases} 1 & \text{if } x = 80 \text{ then out}(81). \top \text{ else out}(x). m_4. \end{cases}$ In turn, since $\text{spa}((\text{true}, [m_4']), \text{out}(y), (\neg(x = 80) \land y = x))$ (where $\text{frsh}(\text{fv}((b, [m_4'])) = y)$, the symbolic relation must also contain $(\neg(x = 80) \land y = x, [m_4])$. We thus reach the original monitor set $[m_4]$ but with a stronger condition, namely $(\neg(x = 80) \land y = x)$. By extension of this reasoning, it is not hard to see that the symbolic relation required by Def. 19 needs to be infinitely large.

The problem exhibited by E.g. 23 is that the condition aggregating mechanism of Def. 19 does not specify any means for consolidating the boolean condition $b$ constraining a monitor set $M$ in $(b, M)$, i.e., a form of garbage collection of redundant conditions. For instance, in the constrained monitor-set $(\neg(x = 80) \land y = x, [m_4])$ of E.g. 23, the condition $(\neg(x = 80) \land y = x)$ plays no effective role in constraining the free variables in $[m_4]$, of which there are none. We therefore optimise Def. 19 in a sound (and complete) manner by taking into consideration boolean sub-conditions that can be isolated and discarded. This leads to an improved automated analysis for consistently-detecting monitors.

► Definition 24 (Optimised Symbolic Controllability). The consolidation of a boolean expression $b$ wrt. a variable set $V$, denoted as $\text{cns}(b, V)$, is defined as:

$$\text{cns}(b, V) \doteq b_1 \text{ whenever } \text{prt}(b, V) = \langle b_1, b_2 \rangle \text{ for some } b_2$$

where the boolean expression partitioning operation $\text{prt}(b, V)$ is defined as:

$$\text{prt}(b, V) \doteq \begin{cases} \langle b_1, b_2 \rangle & \text{if } \text{sat}(b) \text{ and } b = b_1 \land b_2 \text{ and } (\text{fv}(b_1) \subseteq V) \text{ and } (V \cap \text{fv}(b_2) = \emptyset) \\ \langle b, \text{true} \rangle & \text{otherwise} \end{cases}$$

Let the optimised symbolic reachability from $\langle b, M \rangle$ for $\theta$ and $c$, $\text{osaf}t(\langle b, M \rangle, \theta, c)$, be defined as:

$$\text{osaf}t((b, M), \theta, c) \doteq \left\{ \langle \text{cns}(\land V, \land B, \land M), \text{sat}(M, B, \theta) \rangle \mid B \in \text{sc}(b \land c, \text{rec}(M, \theta)) \text{ and } \text{sat}(\land V) \text{ and } V \in \text{fv}(\text{sat}(M, B, \theta)) \right\}$$

A relation $S \subseteq (\mathbb{BExp} \times \mathbb{P}^{\text{Mon}})$ is called optimised symbolically-controllable iff for all $\langle b, M \rangle \in S$:

1. $\text{spr}(b, M, w)$ and $w \in \{\top, \bot\}$ implies $M = \{w\}$;
2. $\text{spa}(\langle b, M \rangle, l(x), c)$ where $\text{frsh}(\text{fv}((b, M))) = x$ implies $\text{osaf}t((b, M), l(x), c) \subseteq S$.

The largest optimised symbolically-controllable relation is denoted by $C_{\text{sym}}^{\text{opt}}$. A (closed) monitor $m$ is said to be optimised symbolically-controllable iff $\langle \text{true}, [m] \rangle \in C_{\text{sym}}^{\text{opt}}$.

We highlight the salient points from Def. 24. First, boolean consolidation in a constrained monitor-set, $(\text{cns}(b, M))$, should not change the set of concrete monitor sets represented by $(b, M)$ and, for this reason, we cannot consolidate unsatisfiable boolean conditions. For instance, even if $x \notin \text{fv}(M)$, it is still unsound to optimise $\langle \text{true} \land (x \neq x), M \rangle$ to $\langle \text{true}, M \rangle$ based on the fact that $\neg\text{sat}(x \neq x)$. Concretely, from Eq. (4) of Sec. 5 we know that $\langle \text{true} \land x \neq x, M \rangle$ denotes the empty set of monitor-sets, $\emptyset$, whereas $(\text{true}, M)$ represents the set $\{m_4 \mid \exists \{m_4 \} \text{ true and } m_4 \in M\}$. Second, consolidation should ideally filter out as much redundant constraints as possible, e.g., in $(b_1 \land b_2, M)$ we should remove $b_2$ whenever $\text{fv}(b_2) \cap \text{fv}(M) = \emptyset$. In Def. 24 we require the strongest possible condition for the residual condition $b_1$ in $(b_1 \land b_2, M)$, i.e., $\text{fv}(b_1) \subseteq \text{fv}(M)$, which indirectly implies
that the resp. condition variables are partitioned $\text{fv}(b_1) \cap \text{fv}(h_1) = \emptyset$. This partitioning is crucial for a sound consolidation, e.g., in \((\neg(x = 80) \land y = x), M\), it is unsound to just remove the subcondition \(\neg(x = 80)\) when \(x \not\in \text{fv}(M)\) and \(y \in \text{fv}(M)\). Although $\text{prt}(b, V)$ can be refined further (while still observing core requirements for soundness such as variable condition partitioning), in Def. 24 we opted for a less elaborate condition that suffices our exposition. Third, we highlight the fact that the conditions specifying $\text{ens}(b)$ in Def. 24 yield a unique consolidated condition up to semantic equivalence meaning that, in an implementation of the framework, this can be defined as a function.

\begin{itemize}
  \item \textbf{Theorem 25} (Optimised Controllability). \(\langle \text{true}, \{m\} \rangle \in C_{\text{sym}}^{\text{opt}}\) iff \(\langle \text{true}, \{m_1\} \rangle \in C_{\text{sym}}\)
\end{itemize}

\begin{itemize}
  \item \textbf{Example 26}. As a result of Thms. 13, 21 and 25, we can show that $m_4$ of E.g. 2 is consistently-detecting by exhibiting the optimised symbolic relation $S_1$ below ($m'_4$ is the monitor defined earlier in E.g. 23). For expository purposes, we show how the consolidated boolean expressions are calculated. In particular, the first constrained monitor-set in $S_1$ denotes both the starting pair \(\langle \text{true}, \{m_4\} \rangle\), but also the pair \(\langle \text{ens}(x = 80 \land y = 81), \text{fv}((m_4)) \rangle, \{m_4\}\).

\[ S_1 = \{ \langle \text{true}, \{m_4\} \rangle, \langle \text{true}, \{,\} \rangle, \langle \text{true}, \{m'_4\} \rangle, \langle \text{true}, \{\} \rangle \} \]

The interested reader may check that the conditions of Def. 24 are satisfied by $S_1$.

\end{itemize}

Using standard techniques [42, 3], an algorithm constructing symbolically-controllable relations from Def. 24 can be extracted more easily. Moreover, the completeness aspect in Thms. 13, 21 and 25 should enable such an automation to infer a counter-example (system and trace) from failed attempts, thereby explaining why a monitor is not consistently-detecting.

\begin{itemize}
  \item \textbf{Example 27}. Recall monitor $m_2$ from E.g. 1. Assuming the shorthand abbreviations $m'_2 \triangleq \text{aut}(y) \cdot \text{ack}(y) \cdot T + \text{aut}(z) \cdot \text{if } z \not\in y \text{ then } \text{ack}(z') \cdot \bot$ and $M = \{ \text{if } z \not\in \text{enc}(x) \text{ then } \text{ack}(z') \cdot \bot, \text{ack}((\text{enc}(x)) \cdot T) \}$, compiling a relation satisfying Def. 24 fails because it needs to include:

\[ \langle \text{true}, [\text{let } y = \text{enc}(x) \text{ in } m'_2] \rangle \text{ since } \text{spa}((\text{true}, \{m_2\}), \text{chl}(x), \text{true}) \]
\[ \langle z = \text{enc}(x), M \rangle \text{ since } \text{spa}((\text{true}, [\text{let } y = \text{enc}(x) \text{ in } m'_2] \rangle, \text{aut}(z), z = \text{enc}(x)) \]
\[ \langle \text{true}, [ T, 0 ] \rangle \text{ since } \text{spa}((z = \text{enc}(x), M), \text{ack}(w), w = \text{enc}(x)) \]

The final pair \(\langle \text{true}, [ T, 0 ] \rangle\) violates Def. 24(i) and, via the symbolic events on right-hand column containing the $\text{spa}(\neg)$ assertions that lead to it, one can construct the counter-example inducing the inconsistent detection, i.e., a system $s$ producing the trace $\text{chl}(v) \cdot \text{aut}(u) \cdot \text{ack}(u)$ where $u = \text{enc}(v)$.

\section{Conclusion}

Monitors should provide guarantees that they will operate correctly when instrumented with any system [35]. At the same time, for this requirement to be scalable, the corresponding correctness analysis that determines it must also be compositional: monitors should be verified separately, independent of the systems they may be instrumented with. The fact that monitors tend to be considerably smaller (in size and complexity) than the systems they observe further justifies this point. This paper provides two definitions that formalise deterministic monitoring behaviour, Def. 6 (consistently-detecting monitors) and Def. 11 (controllable monitors), that address these seemingly conflicting concerns; it also shows that the two definitions coincide, Thm. 13. In addition, the paper also studies alternative definitions for controllability, Defs. 19 and 24, that enable the implementation of sound and complete symbolic analyses, Thms. 21 and 25.

Our methods provide a systematic way for factoring out auxiliary reasoning on data from the analysis relating to the branching structure of the monitors; the former kind of reasoning can be
Consistently-Detecting Monitors

determined by calling on an independent satisfiability solver. In fact, for the specific case of our expository monitor language in Fig. 1, one can show that our methods yield finite symbolic transition graphs, making the latter reasoning decidable modulo the expression and boolean language used. The results obtained in this work should also be general enough to be applicable to other monitoring systems. For instance, Defs. 6, 11, 19 and 24 are independent of the kind of systems monitored, the syntax of the monitor language, and the event value domains and expression languages used. Instead, they are defined in terms of generic characteristics such as their LTS semantics. As a result, extending the monitor language with constructs such as parallel composition would not affect the existing framework. The instrumentation relation one adopts for composing monitors with systems necessarily affects the compositional properties and the correspondence between the respective definitions. However, these changes do not impinge on the general structure of our definitions and should be local to the detection condition in the resp. controllability definitions, namely Def. 11(i), 19(i) and 24(i), and the reachability-set definitions aft(−), saft(−) and osaft(−) of Defs. 10, 18 and 24.

Future work We will further investigate the implementability aspects of our analysis, possibly as an extension to existing model-checking tools. This may raise further issues and adjustments to our definitions e.g., it may be more efficient to batch the satisfiability checks in Def. 24. We plan to apply this to existing transition-based monitor specifications such as [5, 46] and validate its feasibility as an automated specification assistant.

Related Work The need for determinising monitor syntheses from logical specifications is frequently discussed in the literature [7, 23]. In [1], the authors employ a trace-based definition of deterministic monitors that takes into consideration verdicts (similar to our definition for consistent detections of Def. 6 but without considering universal quantification over system instrumentations) and establish complexity bounds for determinising monitors wrt. this definition. Set-simulations, which are related to our monitor-sets, have been used as a proof technique for testing preorders in [15] but do not consider symbolic analyses. Acceptor ambiguity [29] is closely related to our notion of consistent detection with respect to the three outcomes of acceptance, rejection and withholding, as specified in Def. 6. Crucially, however, our definition universally quantifies over all possible system compositions. Subsequently, the main endeavour of our work was that of developing sound and complete compositional techniques to alleviate the analysis for consistent detection; we are unaware of any compositional or coinductive techniques used for determining acceptor ambiguity. Symbolic LTSs were studied extensively for value-passing CCS in [27, 28], but their use in controllability for reasoning about consistent monitor detections is, to our knowledge, novel. The particular setting where it is used, namely the instrumentation composition relation and the use of monitor-sets, also require new technical machinery, such as that of Def. 18. Our definition of controllability, Def. 11, is related to viability (usability) for clients in contract compliance [39] and must-testing [9]. Particularly, in the case of compliance, viability is defined coinductively and is satisfied whenever there exists a server that can engage with the client so as to lead it to success whenever interaction terminates. Apart from the universal quantifications over systems (viability existentially quantifies over servers), our work differs from [39, 9] wrt. the treatment of verdicts considered, the composition relation used (i.e., instrumentation), and the development of a symbolic analysis for handling of action/event data.

Acknowledgements The author acknowledges the Dagstuhl seminar 17051 and would like to thank Luca Aceto, Antonis Achilleos, Duncan Attard, Giovanni Bernardi, Ian Cassar, Anna Ingólfsdóttir, Giles Reger and anonymous reviewers for their help and suggestions.
4:16 Consistently-Detecting Monitors