A Theory of Monitors ⋆

Adrian Francalanza[a,1,*]

[a] *Department of Computer Science, Faculty of ICT, University of Malta, Msida, Malta.*

Abstract

We develop a behavioural theory for *monitors*, computational entities that passively analyse the runtime behaviour of systems so as to infer properties about them. First, we present a monitor language and an instrumentation relation used for piCalculus process monitoring. We then identify contextual behavioural preorders that allow us to relate monitors according to criteria defined over monitored executions of piCalculus processes. Subsequently, we develop alternative monitor preorders that are compositional, since they allow us to relate monitors without resorting to their composite behaviour when they instrumented with systems. Importantly, we show that the latter alternative preorders are sound and complete with respect to the contextual preorders. Finally, we demonstrate how these preorders can assist the development of correct monitor synthesis tools.

Keywords:
monitor correctness, behavioural preorders, runtime verification, process calculi, monitorability

1. Introduction

Monitors (execution monitors [2]) are software entities that are *instrumented* to execute along side a program so as to *observe* its runtime behaviour and determine properties about it, inferred from the *runtime analysis* of the exhibited (program) execution. This basic form of monitor is also occasionally referred to as (sequence) *recognisers* [3, 4], which are closely related to *partial-identity*

⋆The research was supported by the University of Malta Research Fund project "A Theory of Monitors" CPSRP05-04, the European Union COST Action BETTY with the short-term scientific mission COST-STSM-ECOST-STSM-IC1201-170214-038253, the Icelandic Research Fund RANNIS projects "Theoretical Foundations of Monitorability" TheoFoMon:163406-051 and "MoVeMnt: Mode(l)s of Verification and Monitorability" MoVeMnt:217987-051, and by the EU H2020 RISE programme under the Marie Skłodowska-Curie grant agreement No. 778233 "BehAPI: Behavioural Application Programming Interfaces".

*Corresponding author
Email address: adrian.francalanza@um.edu.mt (Adrian Francalanza)

[1] A shorter version of this work outlining the main results appeared as part of the conference proceedings for the 19th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS) [1]

monitors [5]. In other settings, monitors go even further, and either *adapt* aspects of the monitored program [6, 7, 8, 9, 10] or *enforce* predefined properties by modifying the observable behaviour [11, 3, 4, 12, 13, 14]. Monitors are central to software engineering techniques such as monitor-oriented programming (MOP) [15, 9], where a core system is augmented with layers of monitors that either restrict, extend or replace its behaviour. They also play a major role in fail-fast design patterns [16] used in fault-tolerant systems [17, 18, 19]. In the context of formal methods, monitors are used extensively in runtime verification [20, 21, 22], a lightweight verification technique that attempts to mitigate state explosion problems associated model checking, while also providing a means for post-deployment verification. In programming language design and implementation, monitors are occasionally used as a mechanism for dynamic type checking [23, 5, 10].

Monitoring setups typically consist of three main components: apart from the program being monitored, P, there is the *monitor* itself, M, and the *instrumentation*, the mechanism composing the monitor with the program as $P \triangleleft M$. This composite entity is referred to as the *monitored system* [21, 22] and exhibits behaviour that is dependent on both the program P and the monitor M. Remarkably, instrumentation composition relations have seldom been studied formally in their own right. This article investigates one such instrumentation relation employed throughout a body of work studying monitorability [24, 25, 26, 27, 28, 29, 30, 31, 32]. In particular, we develop *compositional* reasoning techniques for execution monitors (recognisers) that are composed with their respective programs using this instrumentation relation. Programs are here expressed as piCalculus processes [33, 34]—a well-studied concurrency model—where the monitorable events analysed by the monitors are precisely the external actions of the processes under scrutiny. We set out to develop monitor preorders of the form

$$M_1 \sqsubseteq M_2 \tag{1}$$

denoting the fact that, when instrumented in the *context* of any *arbitrary* process P, if the monitored system $P \triangleleft M_1$ exhibits certain behavioural properties, then the other monitored system $P \triangleleft M_2$ (instrumented with the *same* process P) exhibits the same behaviour as well. Within this setup, we consider various possible instrumentation properties one may require from a monitored system and show how these give rise to different monitor preorders.

Example 1. Consider the two monitors M_1 and M_2 defined below, both listening for *output actions* of the form $c!v$ (denoting an output on a channel named c with a payload v that can be a channel name itself). Monitor M_1 listens for output actions on channel a with a payload that is *not* in the set C and raises a detection flag, $\checkmark$, when it observes that the process under scrutiny exhibits such an action; the payload is bound dynamically to the variable x at runtime. Monitor M_2 augments this behaviour: apart from listening for the same outputs on channel a, it also detects outputs with payload b on channels (bound at runtime to variable y) that are not in set D: the $M' + M''$ construct used in M_2 acts as an *external choice* [35] that branches to either M' or M'' depending

on the behaviour observed.

$$M_1 = \mathsf{match}\ a!(x).\ \mathsf{if}\ x \notin C\ \mathsf{then}\ \checkmark$$
$$M_2 = (\mathsf{match}\ a!(x).\ \mathsf{if}\ x \notin C\ \mathsf{then}\ \checkmark)\ +\ (\ \mathsf{match}\ (y)!b.\ \mathsf{if}\ y \notin D\ \mathsf{then}\ \checkmark\ )$$

One can argue that M_2 is related to M_1, *i.e.,* $M_1 \sqsubseteq M_2$, since all the detections raised by M_1 are also raised by M_2. However, under more stringent criteria, the two monitors should not be related. Consider the case where $a \in D$ and $b \notin C$ (or $b \in C$ and $a \notin D$ for that matter). For a process P exhibiting the action $a!b$, monitor M_2 may non-deterministically *fail to detect* this behaviour because it may analyse the action along the right-hand branch, *i.e.,* along $\mathsf{match}\ (y)!b.\mathsf{if}\ y{\notin}D\ \mathsf{then}\ \checkmark$ since the pattern $(y)!b$ matches action $a!b$, and consequently not raise the flag. By contrast, monitor M_1 *always* detects the behaviour $a!b$ and, in this sense, M_2 does not preserve all the properties of M_1.

There are other potential behavioural properties to consider. For instance, monitors are also expected to *interfere minimally* with the execution of the analysed process, a property often termed as *transparency* [3, 4, 5, 14]. This gives rise to further criteria for relating monitors. For example, consider the monitor M_3 below denoting an internal choice, *i.e.,* the construct $\oplus$, that autonomously (without external stimuli) either branches to the submonitor $\mathsf{match}\ a!(x).\mathsf{if}\ x \notin C\ \mathsf{then}\ \checkmark$ or to the branch block; the latter construct blocks the process under scrutiny from performing further computation.

$$M_3 = (\mathsf{match}\ a!(x).\mathsf{if}\ x \notin C\ \mathsf{then}\ \checkmark) \oplus \mathsf{block}$$

Again, although M_1 and M_3 can detect the same behaviour, they should not be related under certain monitoring requirements since M_3 may occasionally affect the behaviour of the process under scrutiny by branching to block. Even though the construct block is clearly undesirable from a transparency perspective and should perhaps be ruled out at the monitor-language level, it turns out that the behaviour that it describes can manifest itself in subtle ways via combinations of other (useful) constructs. ■

The preorders suggested in E.g. 1 are worth investigating for a number of reasons. For a start, they act as notions of *refinement*: they allow us to formally specify properties that are expected of a monitor M by expressing them in terms of a monitor description, Spec_M, and then requiring that $\mathsf{Spec}_M \sqsubseteq M$ holds. In our setting, both Spec_M and M are expressed using the same monitor language, but our theory does not necessarily require this. Our preorders also provide a formal understanding for when it is admissible to *substitute one monitor implementation for another* while preserving the elected monitoring properties. For instance, the work in [36, 25, 28, 29, 14, 30, 37, 38] define monitor synthesis algorithms generating monitors from specification formulae that are shown to be correct; practical concerns occasionally require us to use a more efficient version of the synthesised monitors (*e.g.,* [39, 40, 41, 42, 43, 44]) and these preorders would allow us to show that the substituted versions are still correct. Finally, these preorders induce kernel equivalences, *i.e.,* $M_1 \cong M_2$ iff ($M_1 \sqsubseteq M_2$ and

$M_2 \sqsubseteq M_1$). This can then lead to equational theories where the equivalences can be used as a justifying underpinning for the development of an axiomatic semantics for monitors along the lines of [45, 35].

We note that our study considers a general model that, amongst other things, allows monitors to behave *non-deterministically*. This permits us to study the cases where non-determinism is either tolerated or considered erroneous. For instance, some monitor studies and tools rely explicitly on non-deterministic models for expressiveness reasons, such as Büchi automata [46, 40] and TOPL automata [47]. There are studies that employ non-determinism in their formalisms to keep certain constructions small and manageable when describing monitors [48, 28, 25, 29] whereas others use this aspect to asses the *cost* of determinising monitor descriptions [26, 31] and analyse the intricacies associated with establishing non-determinism [27, 49]. We would also like our theory to be applicable to practical settings and considering non-deterministic monitors brings it closer to more realistic scenarios. Non-determinism arises naturally in concurrent and distributed programming, which is increasingly being used for runtime monitoring [50, 51, 52, 53, 54, 55, 23]. A growing number of monitoring tools utilises automata-based specification languages similar to the monitor description language used in this work [56, 57, 58, 59]. These tools offer limited support for ruling out non-deterministic behaviour: their respective implementations are either thread-unsafe [58] or admit arbitrary code for transition-triggered actions [60, 56, 59]. In this setting of executable specifications, non-determinism may also arise from the composition of specifications even though, individually, each specification behaves deterministically. Note also that, in standard program refinement setting such as the one proposed in this paper, it is common and (economical) to describe *both* monitor specifications *and* implementations using the same (monitor) language formalism. In this case, the specification constructs expressing non-determinism are useful to denote *under-specification*.

Although formal and intuitive, the preorders alluded to in Eq. (1) turn out to be hard to determine. One of the obstacles making these preorders unwieldy is the *universal quantification over all possible processes* with which the respective monitors can be instrumented and for which the monitoring properties should hold. We therefore develop alternative characterisations for these preorders, $M_1 \leq M_2$, that do not rely on this universal quantification over process instrumentation. We show that such relations are sound *wrt.* the former monitor preorders, which serves as a semantic justification for these alternative monitor preorders. More importantly, however, it also allows us to use the more tractable alternative relations as a sound proof technique for establishing inequalities in the original preorders. We also show that these characterisations are complete, thereby obtaining full-abstraction for these alternative preorders. Concretely, the main contributions of the paper are:

1. The definition of three contextual monitor preorders, each requiring the preservation of different monitoring properties when composed with the systems to be monitored: Def. 3, Def. 4 and Def. 5.
2. The characterisation of these preorders in terms of alternative preorders

that do not not universally quantify over the systems to be monitored, making them more tractable: Thm. 1, Thm. 2 and Thm. 4.

3. A case study showcasing the utility of such monitor preorders, Sec. 8.

The rest of the article is structured as follows. Sec. 2 briefly overviews our process model, *i.e.,* a variant of the piCalculus, whereas Sec. 3 introduces our monitor language together with the instrumentation relation. In Sec. 4 we formalise our monitor preorder relations *wrt.* this instrumentation. We develop our alternative preorders in Secs. 5 to 7, where we also establish the correspondence with the other preorders. In Sec. 8 we demonstrate how our preorders can be used to analyse the monitors generated by an automated synthesis procedure. Sec. 9 concludes with a discussion about related work and future directions. Upon first reading, the more technical sections, *i.e.,* Secs. 5 to 7, can be safely skipped without substantially affecting the understanding of the remaining material discussed in other sections.

2. The Language

Fig. 1 presents our process language, a standard version of the synchronous piCalculus with name matching. It has the usual constructs (*e.g.,* input and output prefixing is used for communication, parallel composition is used to describe concurrent processes, and channel name scoping is used to model process link mobility) and assumes separate denumerable sets for channel names $a, b, c, d, \ldots \in \textsc{Chans}$, name variables $x, y, z, \ldots \in \textsc{Vars}$ and process variables, $X, Y, \ldots \in \textsc{PVars}$, and lets identifiers u, v range over the sets, $\textsc{Chans} \cup \textsc{Vars}$. The input construct, $c?x.P$, the recursion construct, $\mathsf{rec}\, X.P$, and the scoping construct, $\mathsf{new}\, c.P$, are *binders* where the free occurrences of the variable x, the process variable X, and the channel c *resp.*, are bound in the guarded body P. We write $\mathbf{fv}(P), \mathbf{fV}(P), \mathbf{fn}(P), \mathbf{bV}(P)$, $\mathbf{bv}(P)$ and $\mathbf{bn}(P)$ for the *resp.* free and bound variables, process variables and names in P. We use standard syntactic conventions from process calculi, *e.g.,* we identify processes up to alpha conversion of bound names and variables. For arbitrary syntactic objects o, o', we write $o \sharp o'$ when the free *names* mentioned in o and o' are disjoint *e.g.,* $P \sharp Q$ means $\mathbf{fn}(P) \cap \mathbf{fn}(Q) = \emptyset$.

The operational semantics of the language is given as a Labelled Transition System (LTS), defined by the rules in Fig. 1. LTS judgments are of the form

$$I \triangleright P \xrightarrow{\mu} P' \tag{2}$$

where $I \subseteq \textsc{Chans}$ denotes an *interface* of names *known* (*i.e.,* shared) by both the process and an implicit observer with which interactions occur, P is a closed term, and $\mathbf{fn}(P) \subseteq I$. We occasionally refer to $I \triangleright P$ as a *system* where $\mathbf{fn}(P) \subseteq I$ is assumed. Intuitively the judgment in Eq. (2) denotes that the process in state P carries out the interaction μ with an observer represented by I and transitions to the new state P'. We write I, c as a shorthand for $I \cup \{c\}$ whenever $c \notin I$, and

Syntax

$$
\begin{array}{llll}
P, Q \in \textsc{Proc} ::= u!v.P & \textit{(output)} & \mid u?x.P & \textit{(input)} \\
\mid \mathsf{nil} & \textit{(nil)} & \mid \mathsf{if}\ u{=}v\ \mathsf{then}\ P\ \mathsf{else}\ Q & \textit{(conditional)} \\
\mid \mathsf{rec}\, X.P & \textit{(recursion)} & \mid X & \textit{(process var.)} \\
\mid P \parallel Q & \textit{(parallel)} & \mid \mathsf{new}\, c.P & \textit{(scoping)}
\end{array}
$$

Semantics

$$\textsc{pOut}\ \frac{}{I \rhd c!d.P \xrightarrow{c!d} P} \qquad \textsc{pIn}\ \frac{}{I \rhd c?x.P \xrightarrow{c?d} P[d/x]}$$

$$\textsc{pThn}\ \frac{}{I \rhd \mathsf{if}\ c{=}c\ \mathsf{then}\ P\ \mathsf{else}\ Q \xrightarrow{\tau} P}$$

$$\textsc{pEls}\ \frac{c \sharp d}{I \rhd \mathsf{if}\ c{=}d\ \mathsf{then}\ P\ \mathsf{else}\ Q \xrightarrow{\tau} Q} \qquad \textsc{pRec}\ \frac{}{I \rhd \mathsf{rec}\, X.P \xrightarrow{\tau} P[\mathsf{rec}\, X.P/X]}$$

$$\textsc{pPar}\ \frac{I \rhd P \xrightarrow{\mu} P'}{I \rhd P \parallel Q \xrightarrow{\mu} P' \parallel Q} \qquad \textsc{pCom}\ \frac{I, d \rhd P \xrightarrow{c!d} P' \quad I, d \rhd Q \xrightarrow{c?d} Q'}{I, d \rhd P \parallel Q \xrightarrow{\tau} P' \parallel Q'}$$

$$\textsc{pRes}\ \frac{I, d \rhd P \xrightarrow{\mu} P' \quad d \sharp \mu}{I \rhd \mathsf{new}\, d.P \xrightarrow{\mu} \mathsf{new}\, d.P'}$$

$$\textsc{pCls}\ \frac{I \rhd P \xrightarrow{c!d} P' \quad I \rhd Q \xrightarrow{c?d} Q' \quad d \sharp I}{I \rhd P \parallel Q \xrightarrow{\tau} \mathsf{new}\, d.(P' \parallel Q')} \qquad \textsc{pOpn}\ \frac{I, d \rhd P \xrightarrow{c!d} P'}{I \rhd \mathsf{new}\, d.P \xrightarrow{c!d} P'}$$

Figure 1: piCalculus syntax and semantics

generally assume a version of the Barendregt convention whereby $\mathbf{bn}(P) \sharp I$.[2] For arbitrary names c, d, actions $\mu \in \textsc{Act}_\tau$ range over *input* actions, $c?d$, *output* actions, $c!d$, and a distinguished *silent* action, τ ($\alpha \in \textsc{Act}$ ranges over *external* actions, that exclude τ). The rules in Fig. 1 are fairly standard, using I for book-keeping purposes relating to free/bound names (we elide symmetric rules for pPar, pCom and pCls). Since we assume $\mathbf{fn}(P) \subseteq I$, we implicitly have that $c, d \in I$ in rule pOut and that $c \in I$ in rule pIn (but d in the action $c?d$ of pIn is not necessarily in I). Rule pIn employs a substitution, a partial map $\sigma \in (\textsc{Vars} \rightharpoonup \textsc{Chans})$, denoted as $[c_1, \ldots, c_n/x_1, \ldots, x_n]$; when a substitution σ is applied to a syntactic term o, denoted as $o\sigma$, it replaces every free occurrence in the term, $x \in \mathbf{fn}(o)$, that is covered by the substitution, $x \in \mathbf{dom}(\sigma)$, with

[2] The rules in Fig. 1 still check explicitly for this; see rules pRes, pCls and pOpn.

its respective channel name, $\sigma(x)$. We use $s, t \in \text{ACT}^*$ to denote *traces of external actions*. Although actions do not include explicit information relating to extruded names (*i.e.,* when a scoped name is communicated as a payload and the receiving process is outside the scope), this may be recovered using I as shown in Def. 1. As we will subsequently see, the absence of name binding in actions (as used in standard texts such as [34, 61]) simplifies our handling of traces. More concretely, the evolution of I after a transition is determined exclusively by the *resp.* action of the transition, defined as $\mathbf{aftr}(I, \mu)$ in Def. 1; note that both the process (through outputs) and the implicit observer (through inputs) may extend I.

Definition 1 (Extruded Names and Interface Evolution)**.**

$$\mathbf{ext}(I, \tau) \stackrel{\text{def}}{=} \emptyset \qquad \mathbf{ext}(I, c!d) \stackrel{\text{def}}{=} \{d\} \setminus I \qquad \mathbf{ext}(I, c?d) \stackrel{\text{def}}{=} \emptyset$$

$$\mathbf{aftr}(I, \tau) \stackrel{\text{def}}{=} I \qquad \mathbf{aftr}(I, c!d) \stackrel{\text{def}}{=} I \cup \{d\} \qquad \mathbf{aftr}(I, c?d) \stackrel{\text{def}}{=} I \cup \{d\} \quad \blacksquare$$

Example 2. Consider the process $\mathsf{new}\, d.(c!d.d?x.P)$ and the interface $I{=}\{c\}$. We can deduce the transition

$$I \triangleright \mathsf{new}\, d.c!d.d?x.P \xrightarrow{c!d} d?x.P \qquad \text{using rules } \textsc{pOut} \text{ and } \textsc{pOpn}$$

which scope extrudes the name d since $\mathbf{ext}(I, c!d) = \{d\}$. The new interface following the transition is $\mathbf{aftr}(I, c!d) = \{c, d\} = I'$, denoting the fact that now the observer is aware of the previously scoped name d as well. We can thus infer the following transition *wrt.* the (extended) interface I':

$$I' \triangleright d?x.P \xrightarrow{d?b} P[b/x] \qquad \text{using rule } \textsc{pIn}$$

Again, the interface is extended as a result of this transition, $\mathbf{aftr}(I', d?b) = \{c, d, b\}$, to denote the fact that the process is now aware of the fresh name b that was passed on by the observer. $\blacksquare$

Notice that whenever $\mathbf{fn}(P) \subseteq I$ and $I \triangleright P \xrightarrow{\mu} Q$ then $\mathbf{fn}(Q) \subseteq \mathbf{aftr}(I, \mu)$ as well. We write $I \triangleright P \stackrel{\mu}{\not\longrightarrow}$ to denote $\nexists P' \cdot I \triangleright P \xrightarrow{\mu} P'$ and lift the functions in Def. 1 to traces, *i.e.,* $\mathbf{ext}(I, s)$ and $\mathbf{aftr}(I, s)$, in the obvious way. Two successive transitions $I \triangleright P \xrightarrow{\mu_1} P_1$ and $\mathbf{aftr}(I, \mu_1) \triangleright P_1 \xrightarrow{\mu_2} P_2$ are denoted as $I \triangleright P \xrightarrow{\mu_1} \mathbf{aftr}(I, \mu_1) \triangleright P_1 \xrightarrow{\mu_2} P_2$. We write $I \triangleright P \stackrel{s}{\Longrightarrow} Q$ to denote the sequence of transitions $I_0 \triangleright P_0 \xrightarrow{\mu_1} I_1 \triangleright P_1 \xrightarrow{\mu_2} I_2 \triangleright P_2 \ldots \xrightarrow{\mu_n} P_n$ where $P_0 = P$, $P_n = Q$, $I_0 = I$, $I_i = \mathbf{aftr}(I_{i-1}, \mu_i)$ for $i \in 1..n$, and s is equal to $\mu_1 \ldots \mu_n$ after filtering τ labels. We call $I \triangleright P \stackrel{s}{\Longrightarrow} Q$ an *s-execution.*

Example 3. Consider the process $P_{\text{sv}}{=}\mathsf{rec}\, X.c?x.\mathsf{new}\, d.x!d.X$, modelling the idiomatic server that repeatedly waits for requests on channel c and answers back on the inputted channel (bound to variable x) with a *fresh* channel. We can derive the following behaviour *wrt.* $I = \{c\}$:

$$I \triangleright P_{\text{sv}} \xrightarrow{\tau} I \triangleright c?x.\mathsf{new}\, d.x!d.P_{\text{sv}} \xrightarrow{c?a} I, a \triangleright \mathsf{new}\, d.a!d.P_{\text{sv}} \xrightarrow{a!d}$$

$$I, a, d \triangleright P_{\text{sv}} \xrightarrow{c?a} I, a, d \triangleright \mathsf{new}\, d.a!d.P_{\text{sv}} \xrightarrow{a!d'} I, a, d, d' \triangleright P_{\text{sv}}$$

Syntax

$$p, q \in \textsc{Pat} ::= o?r \quad \textit{(input pattern)} \qquad | \; o!r \quad \textit{(output pattern)}$$
$$w \in \textsc{Verd} ::= \mathsf{end} \quad \textit{(termination)} \qquad | \; \checkmark \quad \textit{(detection)}$$

$$\begin{array}{llll} M, N \in \textsc{Mon} ::= w & \textit{(verdict)} & | \; p.M & \textit{(pattern match)} \\ \quad | \; M + N & \textit{(choice)} & | \; \mathsf{if}\ u = v\ \mathsf{then}\ M\ \mathsf{else}\ N & \textit{(branch)} \\ \quad | \; \mathsf{rec}\, X.M & \textit{(recursion)} & | \; X & \textit{(monitor var.)} \end{array}$$

Monitor Semantics

$$\textsc{mVer}\ \frac{}{w \xrightarrow{\alpha} w} \qquad \textsc{mPat}\ \frac{\mathsf{match}(p, \alpha) = \sigma}{p.M \xrightarrow{\alpha} M\sigma} \qquad \textsc{mChL}\ \frac{M \xrightarrow{\mu} M'}{M + N \xrightarrow{\mu} M'}$$

$$\textsc{mRec}\ \frac{}{\mathsf{rec}\, X.M \xrightarrow{\tau} M[\mathsf{rec}\, X.M/X]} \qquad \textsc{mChR}\ \frac{N \xrightarrow{\mu} N'}{M + N \xrightarrow{\mu} N'}$$

$$\textsc{mThn}\ \frac{}{\mathsf{if}\ c = c\ \mathsf{then}\ M\ \mathsf{else}\ N \xrightarrow{\tau} M} \qquad \textsc{mEls}\ \frac{c \sharp d}{\mathsf{if}\ c = d\ \mathsf{then}\ M\ \mathsf{else}\ N \xrightarrow{\tau} N}$$

Instrumented System Semantics

$$\textsc{iMon}\ \frac{I \triangleright P \xrightarrow{\alpha} P' \quad M \xrightarrow{\alpha} M'}{I \triangleright P \triangleleft M \xrightarrow{\alpha} P' \triangleleft M'} \qquad \textsc{iAsyP}\ \frac{I \triangleright P \xrightarrow{\tau} P'}{I \triangleright P \triangleleft M \xrightarrow{\tau} P' \triangleleft M}$$

$$\textsc{iTer}\ \frac{I \triangleright P \xrightarrow{\alpha} P' \quad M \not\xrightarrow{\alpha} \quad M \not\xrightarrow{\tau}}{I \triangleright P \triangleleft M \xrightarrow{\alpha} P' \triangleleft \mathsf{end}} \qquad \textsc{iAsyM}\ \frac{M \xrightarrow{\tau} M'}{I \triangleright P \triangleleft M \xrightarrow{\tau} P \triangleleft M'}$$

Figure 2: Monitor syntax, semantics and Instrumentation Semantics

Notice that, whereas the first instance of $c?a$ extends the interface (and denotes a bound input [34, 61]) the second instance of $c?a$ denotes a standard input. Note also that to be able to derive the final transition, rule pOpn requires us to infer the premise $(I, a, d), d' \triangleright a!d'.P_{\mathrm{sv}} \xrightarrow{a!d'} P_{\mathrm{sv}}$ where the condition $d' \notin (I, a, d)$ implied by the notation $(I, a, d), d'$ of the premise, implicitly forces us to alpha-convert $\mathsf{new}\, d.a!d.P_{\mathrm{sv}}$ to $\mathsf{new}\, d'.a!d'.P_{\mathrm{sv}}$ before performing the transition. ■

3. Monitor Instrumentation

Monitors, $M, N \in \textsc{Mon}$, are syntactically described by the grammar in Fig. 2. They may reach either of *two* verdicts, namely detection, $\checkmark$, or termination, end, denoting an inconclusive verdict. Monitors are equipped with a *pattern matching* construct (used to observe external actions) and a name-comparison branching construct. In our case, patterns $p, q \in \textsc{Pat}$ range over

input and output actions[3] and may contain either names, CHANS, (free) variables, VARS, or variable binders of the form $(x), (y) \in$ BINDS that are bound to concrete values when pattern matched with an action. We let o, r range over CHANS $\cup$ VARS $\cup$ BINDS and call *closed* pattern those that do *not* contain free variables. The pattern matching function $\mathsf{match}(p,\alpha)$ is defined for *closed* patterns and actions as follows:

$$\mathsf{match}((x), c) = [c/x] \qquad \mathsf{match}(c, d) = \begin{cases} \emptyset & \text{if } c = d \\ \text{undefined} & \text{otherwise} \end{cases}$$

$$\mathsf{match}(o_1?o_2, c_1?c_2) = \begin{cases} \sigma_1 \cup \sigma_2 & \text{if } \sigma_1 = \mathsf{match}(o_1, c_1), \sigma_2 = \mathsf{match}(o_2, c_2) \\ & \text{and } \forall x{\in}\mathbf{dom}(\sigma_1){\cap}\mathbf{dom}(\sigma_2) \cdot \sigma_1(x){=}\sigma_2(x) \\ \text{undefined} & \text{otherwise} \end{cases}$$

$$\mathsf{match}(o_1!o_2, c_1!c_2) = \begin{cases} \sigma_1 \cup \sigma_2 & \text{if } \sigma_1 = \mathsf{match}(o_1, c_1), \sigma_2 = \mathsf{match}(o_2, c_2) \\ & \text{and } \forall x{\in}\mathbf{dom}(\sigma_1){\cap}\mathbf{dom}(\sigma_2) \cdot \sigma_1(x){=}\sigma_2(x) \\ \text{undefined} & \text{otherwise} \end{cases}$$

$$\mathsf{match}(o_1?o_2, c_1!c_2) = \mathsf{match}(o_1!o_2, c_1?c_2) = \text{undefined}$$

As in the case of processes (Sec. 2), in a pattern-match prefix $p.M$, any binding variables in p bind the *resp.* free variables in M; closed monitors are those with no free variables. The remaining constructs, *i.e.,* external branching, internal branching and recursion, are unremarkable.

The monitor semantics is defined in terms of an LTS (Fig. 2), modelling the analysis of the visible runtime execution of a process; unless otherwise stated, we assume closed monitors. Following [62, 11, 25, 27, 29], in rule MVER verdicts are able to analyse *any* external action but transition to the *same* verdict, *i.e.,* verdicts are *irrevocable*. By contrast, pattern-guarded monitors only transition when the action matches the pattern, binding pattern variables to the *resp.* action names, $\mathsf{match}(p, \alpha) = \sigma$, and substituting them in the continuation, $M\sigma$; see rule MPAT. Rules MCHL and MCHR allow a monitor to behave as either the left or right branch (possibly) depending on the action to be analysed, whereas MTHN and MELS allow the monitor to branch internally, depending on the analysis of an internal check (based on data). Rule MREC handles recursion in standard fashion.

A *monitored system*, $P \triangleleft M$, consists of a process, P, instrumented with a monitor, M, analysing its (external) behaviour. Fig. 2 defines the instrumentation semantics for configurations, $I \triangleright P \triangleleft M$, *i.e.,* systems augmented with an interface I, where again we assume P and M are closed and $\mathbf{fn}(P), \mathbf{fn}(M) \subseteq I$ for book-keeping purposes. The LTS semantics follows [25, 27, 29] and relies on the *resp.* process and monitor semantics of Fig. 1 and Fig. 2. In rule IMON,

[3] This may be extended in straight forward fashion to accommodate other forms of actions.

if the process exhibits the external action α *wrt.* I, and the monitor can analyse this action, they transition in lock-step in the instrumented system while exhibiting *same* action. If, however, a process exhibits an action that the monitor cannot analyse, the action is manifested at system level while the monitor is *terminated*; see rule iTer. Finally, iAsyP and iAsyM allow monitors and processes to transition independently *wrt.* internal moves, *i.e.*, our instrumentation forces process-monitor synchronisation for *external* actions only, which constitute our *monitorable* actions. We highlight the fact that, as is expected of recognisers, the process *drives* the behaviour of a monitored system: if the process cannot α-transition, the monitored system cannot α-transition either.

Example 4. Recall P_{sv} from Ex. 3. Using the semantics of Fig. 2, one can derive the monitored execution leading to a detection below, when composed with the monitor $M_1 = \big(c?(y).y!(z).\mathsf{if}\ z{=}c\ \mathsf{then\ end\ else}\ \checkmark\big)$, subject to $I' = \{c, a\}$:

$$I' \triangleright P_{\text{sv}} \triangleleft M_1 \xRightarrow{c?a} I' \triangleright \mathsf{new}\, d.a!d.P_{\text{sv}} \triangleleft a!(z).\mathsf{if}\ z{=}c\ \mathsf{then\ end\ else}\ \checkmark \tag{3}$$

$$\xrightarrow{a!d} I', d \triangleright P_{\text{sv}} \triangleleft \mathsf{if}\ d{=}c\ \mathsf{then\ end\ else}\ \checkmark \xrightarrow{\tau} I', d \triangleright P_{\text{sv}} \triangleleft \checkmark \tag{4}$$

Monitor M_1 detects any output with a payload that is *not* name c on a channel that is communicated as a payload on channel c in the previous action. Note how this channel, is learnt at runtime in Eq. (3) and then used for the pattern-matching of the second action in Eq. (4).

Contrastingly, for the same I', when monitoring with the monitor $M_2 = ((y)!(z).\mathsf{if}\ y{=}a\ \mathsf{then\ end\ else}\ \checkmark)$ does not lead to a detection for the same trace, because the first action, $c?a$ (an input) cannot pattern match with the output pattern, $(y)!(z)$. In fact, rule iTer terminates (prematurely) the monitor after transition $c?a$, so as to avoid erroneous detections.

$$\begin{aligned} I' \triangleright P_{\text{sv}} \triangleleft M_2 &\xrightarrow{\tau} c?x.\mathsf{new}\, d.x!d.P_{\text{sv}} \triangleleft M_2 \\ &\xrightarrow{c?a} \mathsf{new}\, d.a!d.P_{\text{sv}} \triangleleft \mathsf{end} \xRightarrow{a!d} P_{\text{sv}} \triangleleft \mathsf{end} \end{aligned} \tag{5}$$

Note that a monitor need not necessarily ever reach a verdict when monitoring a system. The monitor $M_3 = \mathsf{rec}\, X.c?(y).y!(z).\mathsf{if}\ z = c\ \mathsf{then}\ \checkmark\ \mathsf{else}\ X$ ensures that no payload that is outputted on the channel inputted on c is equal to channel c itself, since otherwise it flags a detection. When instrumented with P_{sv}, we are guaranteed never to reach a detection because all such payloads are fresh:

$$I' \triangleright P_{\text{sv}} \triangleleft M_3 \xRightarrow{c?a} I' \triangleright \mathsf{new}\, d.a!d.P_{\text{sv}} \triangleleft a!(z).\mathsf{if}\ z{=}c\ \mathsf{then}\ \checkmark\ \mathsf{else}\ M_3 \xRightarrow{a!d} I' \triangleright P_{\text{sv}} \triangleleft M_3$$

Finally, consider $M_4 = c?(y).\mathsf{if}\ y{=}c\ \mathsf{then\ end\ else}\ y!(z).c?z.\checkmark$, another monitor. We have the following (dissected) transition sequence for $I = \{c\}$ explaining

how instrumentation works:

$$I \triangleright P_{\mathrm{sv}} \triangleleft M_4 \overset{c?a}{\Longrightarrow} I, a \triangleright \left(\mathsf{new}\, d.a!d.P_{\mathrm{sv}}\right) \triangleleft \mathsf{if}\ a{=}c\ \mathsf{then}\ \mathsf{end}\ \mathsf{else}\ a!(z).c?z.\checkmark \quad (6)$$

$$\overset{\tau}{\longrightarrow} \quad I, a \triangleright \left(\mathsf{new}\, d.a!d.P_{\mathrm{sv}}\right) \triangleleft a!z.c?z.\checkmark \quad (7)$$

$$\overset{a!d}{\longrightarrow} \quad (I, a, d) \triangleright P_{\mathrm{sv}} \triangleleft c?d.\checkmark \quad (8)$$

$$\overset{c?b}{\Longrightarrow} \quad (I, a, d, b) \triangleright \mathsf{new}\, d'.b!d'.P_{\mathrm{sv}} \triangleleft \mathsf{end} \quad (9)$$

$$\overset{b!d'}{\longrightarrow} \quad (I, a, d, b, d') \triangleright P_{\mathrm{sv}} \triangleleft \mathsf{end} \quad (10)$$

In Eq. (6) the server (asynchronously) unfolds (pRec and iAsyP) and inputs on c the fresh name a (pIn). The monitor can analyse $c?a$ using mPat where $\mathsf{match}(c?(y), c?a) = [a/y]$, transitioning accordingly using the instrumentation rule iMon while learning the fresh name a (originating from the environment). At this stage, the instrumentation temporarily *stalls* the process, even though it can produce the (scope extruding) output $a!d$. More precisely, although the monitor cannot presently analyse $a!d$, the rule iTer—which terminates the monitor execution—*cannot* be applied, since the monitor can silently transition and potentially become capable of analysing the action. This is indeed the case, Eq. (7) using rule mEls, resulting in the second monitoring transition, Eq. (8) using iMon, where the monitor learns the second fresh name d, this time originating from the monitored process; we thus obtain $(c?z.\checkmark)[d/z] = c?d.\checkmark$. After another unfolding, the process is ready to input on c again. However, the monitor cannot match $c?b$ ($\mathsf{match}(c?d, c?b)$ is undefined) and since the monitor cannot silently transition either, it is terminated (iTer) while still allowing the process to proceed, Eq. (9). In Eq. (10), verdict monitors allow monitored processes to execute without any hindrance using rule mVer. ■

Ex. 4 highlights two conflicting instrumentation requirements. On the one hand, monitors should *interfere minimally* with the execution of a monitored process where, observationally, a monitored process should behave like the original one. On the other hand, instrumentation must also ensure *bona fide detections*, *e.g.*, in (5) and (9), terminating monitoring when the observed process behaviour does not correspond to that expected by the monitor using rule iTer. But in order to do this while avoiding premature termination, instrumentation needs to allow for monitor internal computation, *e.g.*, (7). Unfortunately, the premise caveat $M \overset{\tau}{\not\longrightarrow}$ in rule iTer—necessary to prevent this premature terminations—allows monitors to affect (indirectly) monitored process behaviour. This subtle aspect of instrumentation, illustrated in E.g. 5 below, will be revisited again in Sec. 4 when our monitor preorders are considered.

Example 5. The monitor Ω below:

$$\Omega \;=\; \mathsf{rec}\, X.(\mathsf{if}\ c{=}c\ \mathsf{then}\ X\ \mathsf{else}\ X) \qquad \Omega' \;=\; \mathsf{if}\ c{=}c\ \mathsf{then}\ \Omega\ \mathsf{else}\ \Omega \quad (11)$$

is divergent, *i.e.*, $\Omega \overset{\tau}{\longrightarrow} \Omega' \overset{\tau}{\longrightarrow} \Omega \overset{\tau}{\longrightarrow} \ldots$, and unresponsive, *i.e.*, $\forall \alpha \cdot \Omega \overset{\alpha}{\not\longrightarrow}$ and $\Omega' \overset{\alpha}{\not\longrightarrow}$. As a result, it suppresses *every* process external behaviour when

instrumented: for arbitrary $I \triangleright P$ we can show $I \triangleright P \triangleleft \Omega \overset{\alpha}{\not\longrightarrow}$ and $I \triangleright P \triangleleft \Omega' \overset{\alpha}{\not\longrightarrow}$ for any α, since rules iMon and iTer cannot ever be applied. ■

Remark 1. The model presented in this paper (Secs. 2 and 3) is a superset of the monitoring operational setup used to develop the monitorability results in [25, 30, 63, 32] and the deterministic correctness criteria in [27, 26, 31]. In this extended model, we allow name-passing in the monitored actions, which translates to the runtime extension of monitor-system boundary upon the scope extrusion of new names (*i.e.,* a monitor can learn new channel names at runtime and then monitor for events on these channels). ■

We conclude this section by proving a number of important properties about our model. Upon first reading, the reader may safely skip onto the next section. Prop. 1 is an important sanity check stating that verdicts are *irrevocable*.

Lemma 1. $M \xrightarrow{\tau}$ implies $M \neq w$

Proof. By the contrapositive, we assume $M = w$ and show, by case analysis of the monitor LTS that $M \overset{\tau}{\not\longrightarrow}$. □

Proposition 1 (Definite Verdicts)**.** $I \triangleright P \triangleleft w \overset{s}{\Longrightarrow} Q \triangleleft M$ implies $M = w$

Proof. By numerical induction on $(\xrightarrow{\mu_n})^n$ where $(\overset{s}{\Longrightarrow}) = (\xrightarrow{\mu_n})^n$. The base case is trivial. For the inductive case, $I \triangleright P \triangleleft w \xrightarrow{\mu_1} Q' \triangleleft M' (\xrightarrow{\mu_k})^k Q \triangleleft M$ we have two subcases:

$\mu_1 = \tau$**:** By a corollary of Lem. 1, we know that rule iAsyM could not have been used to derive the first transition, meaning that only iAsyP could have been used. Thus we know that $I \triangleright P \xrightarrow{\tau} Q'$ and that $M' = w$. The required result follows by $M' = w$ and the I.H. on $I \triangleright Q' \triangleleft w \ (\xrightarrow{\mu_k})^k \ Q \triangleleft M$.

$\mu_1 = \alpha$**:** We know that rule iTrm could never have been applied to derive the first transition, since $w \xrightarrow{\alpha} w$ for any external action α. Thus it could have only been derived using iMon, and by rule mVer we also know that $M' = w$. Similar to the previous case, the required result follows by the I.H. on $\mathbf{aftr}(I, \alpha) \triangleright Q' \triangleleft w \ (\xrightarrow{\mu_k})^k \ Q \triangleleft M$. □

Corollary 1. ($I \triangleright P \triangleleft M \overset{s}{\Longrightarrow} Q \triangleleft N$ and $N \neq w$) implies $M \neq w$ □

We also show that instrumenting a system with a verdict monitor does not affect its behaviour (Prop. 2) which justifies, in part, why we preempt monitoring to `end` when the monitor is under specified and does not know how to handle an action produced by the process under scrutiny in rule iTer in Fig. 2.

Proposition 2 (Verdict Non-Interference)**.**

$$I \triangleright P \overset{s}{\Longrightarrow} Q \text{ implies } I \triangleright P \triangleleft w \overset{s}{\Longrightarrow} Q \triangleleft w$$

Proof. By induction on the length of the transition sequence $I \triangleright P \overset{s}{\Longrightarrow} Q$. For the inductive case $I \triangleright P \xrightarrow{\mu} \cdot \overset{s}{\Longrightarrow} Q$:

- When $\mu = \tau$ we construct the first part of the required sequence using rule iAsyP.
- When $\mu = \alpha$ we construct the first part of the required sequence using rule iMon and rule mVer for the monitor. □

We close off this section with three lemmata deal with zipping and unzipping monitored transition sequences. These allow us to compose and decompose monitored computation and form the basis for our compositional analysis. In particular, we prove these lemmata for the special case where computations reach a verdict (which, by virtue of Prop. 1, means that it is persistent).

Lemma 2 (Verdict UnZipping)**.**

$$I \triangleright P \triangleleft M \stackrel{s}{\Longrightarrow} Q \triangleleft \checkmark \quad \text{implies} \quad (I \triangleright P \stackrel{s}{\Longrightarrow} Q \text{ and } M \stackrel{s}{\Longrightarrow} \checkmark).$$

Proof. By induction on the length of the transition sequence $I \triangleright P \triangleleft M \stackrel{s}{\Longrightarrow} Q \triangleleft \checkmark$.

- If $I \triangleright P \triangleleft M \xrightarrow{\tau} P' \triangleleft M' \stackrel{s}{\Longrightarrow} Q \triangleleft \checkmark$, by case analysis we know that the first transition was derived using either iAsyP or iAsyM. We therefore prepend either $I \triangleright P' \stackrel{s}{\Longrightarrow} Q$ or $M' \stackrel{s}{\Longrightarrow} \checkmark$ (obtained by the I.H.) accordingly, using the premise of either iAsyP or iAsyM.
- If $I \triangleright P \triangleleft M \xrightarrow{\alpha} P' \triangleleft M' \stackrel{s}{\Longrightarrow} Q \triangleleft \checkmark$ we know it could have only been derived using iMon; had it been derived using iTrm, then we would have $M' = \mathsf{end}$, which contradicts Prop. 1. We thus proceed by prepending both $I \triangleright P' \stackrel{s}{\Longrightarrow} Q$ or $M' \stackrel{s}{\Longrightarrow} \checkmark$ obtained by the I.H. by $I \triangleright P \xrightarrow{\alpha} P'$ and $M \xrightarrow{\alpha} M'$ *resp.* □

Lemma 3 (General Unzipping)**.** $I \triangleright P \triangleleft M \stackrel{s}{\Longrightarrow} Q \triangleleft N$ implies:

- $I \triangleright P \stackrel{s}{\Longrightarrow} Q$
- $M \stackrel{s}{\Longrightarrow} N$ or $(s = s_1 \alpha s_2$ and $M \stackrel{s_1}{\Longrightarrow} M' \not\xrightarrow{\tau}$ and $M' \not\xrightarrow{\alpha}$ and $N = \mathsf{end})$.

Proof. By induction on the structure of s. The proof is similar to that of Lem. 2, except for the inductive case of the form $I \triangleright P \triangleleft M \xrightarrow{\alpha} P' \triangleleft M' \stackrel{s}{\Longrightarrow} Q \triangleleft N$, for which we need to consider the possibility that rule iTrm from Fig. 2 was applied to derive $I \triangleright P \triangleleft M \xrightarrow{\alpha} P' \triangleleft M'$. In such a case, we know that $M \not\xrightarrow{\tau}$, $M' \not\xrightarrow{\alpha}$ and that $M' = \mathsf{end}$. Thus, by $\mathbf{aftr}(I, \alpha) \triangleright P' \triangleleft \mathsf{end} \stackrel{s}{\Longrightarrow} Q \triangleleft N$ and Prop. 1 we know that $N = \mathsf{end}$ as required. □

Lemma 4 (General Zipping)**.** $I \triangleright P \stackrel{s}{\Longrightarrow} Q$ and $M \stackrel{s}{\Longrightarrow} N$ implies $I \triangleright P \triangleleft M \stackrel{s}{\Longrightarrow} Q \triangleleft N$

Proof. By induction on the length of the transition sequence of $I \triangleright P \stackrel{s}{\Longrightarrow} Q$ and the length of the transition sequence of $M \stackrel{s}{\Longrightarrow} N$. For the inductive cases:

- When $I \triangleright P \xrightarrow{\tau} P' \stackrel{s}{\Longrightarrow} Q$ we construct the required monitored transition sequence using iAsyP and the resulting monitored transition sequence from the I.H.

- When $I \triangleright P \xrightarrow{\alpha} P' \stackrel{s}{\Longrightarrow} Q$ we have two subcases:
  - If $M \xrightarrow{\tau} M' \stackrel{s}{\Longrightarrow} N$ we construct the required monitored transition sequence using iAsyM and the resulting monitored transition sequence from the I.H.
  - If $M \xrightarrow{\alpha} M' \stackrel{s}{\Longrightarrow} N$ we construct the required monitored transition sequence using iMon and the subsequent monitored transition sequence from the I.H. □

4. Monitor Preorders

We can use the formal setting presented in Sec. 3 to develop the monitor preorders discussed in the Introduction. We start by defining the monitoring predicates we expect to be preserved by the preorder; a number of these predicates rely on *computations* and *detected* computations, defined below.

Definition 2 (Detected Computations)**.** The transition sequence:

$$I \triangleright P \triangleleft M \stackrel{s}{\Longrightarrow} I_0 \triangleright P_0 \triangleleft M_0 \xrightarrow{\tau} I_1 \triangleright P_1 \triangleleft M_1 \xrightarrow{\tau} I_2 \triangleright P_2 \triangleleft M_2 \xrightarrow{\tau} \ldots$$

is called an *s-computation* if it is *maximal* (*i.e.*, either it is *infinite* or it is finite and *cannot be extended further* using τ-transitions). An s-computation is called *detected* (or a detected computation along s) iff $\exists n \in \mathbb{N} \cdot M_n = \checkmark$. ■

One criteria for comparing monitors considers the verdicts reached after observing a *specific execution trace* produced by the process under scrutiny. The semantics of Sec. 3 assigns a *passive role* to monitors, prohibiting them from influencing the branching execution of the monitored process. Def. 2 thus differentiates between the detected computations, identifying them by the *visible trace* of execution that is dictated by the process and over which the monitor should not have any control.

Example 6. Consider the process $P = \mathsf{new}\, d.(d!c \,\|\, d?x.c!a.\mathsf{nil} \,\|\, d?x.c!b.\mathsf{nil})$ and the interface $I = \{c, b, a\}$. The system can *non-deterministically* produce either of the following two behaviours: $I \triangleright P \stackrel{c!a}{\Longrightarrow} \mathsf{nil}$ or $I \triangleright P \stackrel{c!b}{\Longrightarrow} \mathsf{nil}$. Consider also the following monitors:

$$\begin{array}{llll} M_1 & = c!a.\checkmark + c!b.\checkmark & M_2 & = c!a.\checkmark + c!b.\mathsf{end} \qquad M_3 \quad = c!a.\checkmark \\ M_4 & = c!a.\checkmark + c!b.\checkmark + c!b.\mathsf{end} & M_5 & = c!a.\checkmark + c!b.\checkmark + c!a.\mathsf{end} + c!b.\mathsf{end} \end{array}$$

All configurations $I \triangleright P \triangleleft M_i$ for $i \in 1..5$ exhibit detecting computations along the trace $s = c!a.\epsilon$ (ϵ denotes the empty trace and we write $c!a.\epsilon$ instead of just $c!a$ to differentiate between the trace containing just that action and the action itself). For the trace $t = c!b.\epsilon$, configurations $I \triangleright P \triangleleft M_j$ for $j \in \{1, 4, 5\}$ detect t-computations as well, whereas the *resp.* configurations with M_2 and M_3 *do not*. Although the configuration $I \triangleright P \triangleleft M_1$ *always* detects along t (in fact, $I \triangleright P \triangleleft M_1 \stackrel{t}{\Rightarrow} P \triangleleft M_1'$ implies $M_1' = \checkmark$) those with M_4 and M_5

may fail to detect it along such a trace (for instance, we have the computation $I \triangleright P \triangleleft M_4 \stackrel{t}{\Longrightarrow} \mathsf{nil} \triangleleft \mathsf{end} \stackrel{\tau}{\not\longrightarrow}$). Similarly, the configurations with monitors M_1 and M_4 deterministically detect along trace s, but $I \triangleright P \triangleleft M_5$ does *not*. ■

Ex. 6 suggests two types of computation detections that a monitor may exhibit.

Definition 3 (Potential and Deterministic Detection)**.**

1. M *potentially detects* for $I \triangleright P$ along trace s, denoted as $\mathsf{pd}(M, I, P, s)$, iff there *exists* a detecting s-computation from $I \triangleright P \triangleleft M$.
2. M *deterministically detects* for $I \triangleright P$ along trace s, denoted as $\mathsf{dd}(M, I, P, s)$, iff *all* s-computation from $I \triangleright P \triangleleft M$ are detecting. ■

Remark 2. If a monitored process cannot produce trace s, *i.e.,* $I \triangleright P \stackrel{\hat{s}}{\not\Longrightarrow}$, then for any M we have $I \triangleright P \triangleleft M \stackrel{\hat{s}}{\not\Longrightarrow}$ as well (Lem. 3). If we have $I \triangleright P \triangleleft M \stackrel{\hat{s}}{\not\Longrightarrow}$, then $\mathsf{pd}(M, I, P, s)$ is trivially false and $\mathsf{dd}(M, I, P, s)$ is trivially true. ■

The detection predicates of Def. 3 induce the following monitor preorders (and equivalences), based on the *resp.* detection capabilities.

Definition 4 (Potential and Deterministic Detection Preorders)**.**

$$M \sqsubseteq_{\mathsf{pd}} N \quad \stackrel{\mathbf{def}}{=} \quad \forall I, P, s \cdot \ \mathsf{pd}(M, I, P, s) \ \text{ implies } \ \mathsf{pd}(N, I, P, s)$$

$$M \sqsubseteq_{\mathsf{dd}} N \quad \stackrel{\mathbf{def}}{=} \quad \forall I, P, s \cdot \ \mathsf{dd}(M, I, P, s) \ \text{ implies } \ \mathsf{dd}(N, I, P, s)$$

$M \cong_{\mathsf{pd}} N$ and $M \cong_{\mathsf{dd}} N$ denote the potential and deterministic detection kernel equivalences induced by the respective preorders, *i.e.,* $M \cong_{\mathsf{pd}} N \stackrel{\mathbf{def}}{=} (M \sqsubseteq_{\mathsf{pd}} N$ and $N \sqsubseteq_{\mathsf{pd}} M)$, and similarly for $M \cong_{\mathsf{dd}} N$. We write $M \sqsubset_{\mathsf{pd}} N$ as a shorthand notation for $(M \sqsubseteq_{\mathsf{pd}} N$ and $N \not\sqsubseteq_{\mathsf{pd}} M)$ and similarly for $M \sqsubset_{\mathsf{dd}} N$. ■

Example 7. Recall the monitors defined in Ex. 6. It turns out that

$$M_2 \quad \cong_{\mathsf{pd}} \quad M_3 \quad \sqsubset_{\mathsf{pd}} \quad M_5 \quad \cong_{\mathsf{pd}} \quad M_4 \quad \cong_{\mathsf{pd}} \quad M_1 \qquad (12)$$

$$M_5 \quad \sqsubset_{\mathsf{dd}} \quad M_2 \quad \cong_{\mathsf{dd}} \quad M_3 \quad \cong_{\mathsf{dd}} \quad M_4 \quad \sqsubset_{\mathsf{dd}} \quad M_1 \qquad (13)$$

Note that, whereas monitor M_5 can *potentially* detect more computations that monitors M_2 and M_3, Eq. (12), it can *deterministically* detect *less* computations than these two monitors, Eq. (13). In fact, M_5 *cannot* deterministically detect *any* computation, irrespective of the process and interface it is instrumented with (*i.e.,* it is a bottom element for the $\sqsubseteq_{\mathsf{dd}}$ preorder). ■

As opposed to related formal analysis on monitors such as [11, 36, 64, 38], the detection predicates in Def. 3 consider monitor behaviour within an instrumented system. Apart from acting as a continuation for the study in [25, 26, 28, 27, 29, 31], this setup also enables us to formally justify subtle monitor orderings, as exemplified in E.g. 8, and help us to analyse peculiarities brought about by the specific instrumentation relation, as illustrated in E.g. 10.

Example 8. Consider the shorthand notation $\tau.M$ for the monitor $\mathsf{rec}\,X.M$ whenever $X \notin \mathbf{fV}(M)$. We have the following (in)equalities:

$$\checkmark \cong_{\mathsf{pd}} \tau.\checkmark \qquad \text{but} \qquad \tau.\checkmark \sqsubset_{\mathsf{dd}} \checkmark$$

It should not be hard for the reader to see that both judgments $\mathsf{pd}(\checkmark, I, P, s)$ and $\mathsf{pd}(\tau.\checkmark, I, P, s)$ hold for any I, P and s, and thus $\checkmark \cong_{\mathsf{pd}} \tau.\checkmark$. The reason for the inequality to the right, *i.e.*, $\checkmark \sqsubset_{\mathsf{dd}} \tau.\checkmark$, is less obvious. Consider the processes $\Omega_P = \mathsf{rec}\,X.(\mathsf{if}\ c = c\,\mathsf{then}\,X\ \mathsf{else}\ X)$ and $\Omega'_P = \mathsf{if}\ c = c\,\mathsf{then}\,\Omega_P\ \mathsf{else}\ \Omega_P$, which are analogous to the divergent monitors Ω and Ω' defined earlier in Eq. (11). The predicate $\mathsf{dd}(\checkmark, I, P, s)$ holds trivially for the monitor $\checkmark$ for *any* process P, interface I, and trace s, since the monitor detects immediately, irrespective of the process' actions. This means that we have $\tau.\checkmark \sqsubseteq_{\mathsf{dd}} \checkmark$. It also means that, in particular, $\mathsf{dd}(\checkmark, I, \Omega_P, \epsilon)$ holds. But predicate $\mathsf{dd}(\tau.\checkmark, I, \Omega_P, \epsilon)$ *does not* hold, due to the non-detecting ϵ-computation

$$I \triangleright \Omega_P \triangleleft \tau.\checkmark \xrightarrow{\tau} I \triangleright \Omega'_P \triangleleft \tau.\checkmark \xrightarrow{\tau} I \triangleright \Omega_P \triangleleft \tau.\checkmark \xrightarrow{\tau} \ldots$$

This is enough to *refute* the inequality $\checkmark \sqsubseteq_{\mathsf{dd}} \tau.\checkmark$. Perhaps a more practical manifestation of the monitor behaviour just discussed is the strict inequality:

$$c!(x).\mathsf{if}\ x = d\,\mathsf{then}\,\checkmark\ \mathsf{else}\ \mathsf{end} \quad \sqsubset_{\mathsf{dd}} \quad c!d.\checkmark \tag{14}$$

were the apparently innocuous left hand monitor does *not* deterministically detect the trace $c!d.\epsilon$ when composed with the system $c!d.\Omega_P$, whereas the right hand monitor does. ■

E.g. 5 hinted that certain monitor inequalities are less straightforward than anticipated due to divergences. The following examples expand on this aspect.

Example 9. Recall the divergent monitor Ω from Eq. (11). We have the following ordering for $\sqsubseteq_{\mathsf{pd}}$ and $\sqsubseteq_{\mathsf{dd}}$ for the following pathological examples:

$$\big(\mathsf{end} \quad \cong_{\mathsf{pd}} \quad c!a.\mathsf{end} \quad \cong_{\mathsf{pd}} \quad \Omega\big) \quad \sqsubset_{\mathsf{pd}} \quad c!a.\checkmark \quad \sqsubset_{\mathsf{pd}} \quad \checkmark \tag{15}$$

$$\big(\mathsf{end} \quad \cong_{\mathsf{dd}} \quad c!a.\mathsf{end}\big) \quad \sqsubset_{\mathsf{dd}} \quad c!a.\checkmark \quad \sqsubset_{\mathsf{dd}} \quad \Omega \quad \sqsubset_{\mathsf{dd}} \quad \checkmark \tag{16}$$

As discussed already in E.g. 8, monitor $\checkmark$ satisfies $\mathsf{dd}(\checkmark, I, P, s)$ for *any* process P, interface I, and trace s. This is immediately apparent from the LTS behaviour of monitor $\checkmark$ (in isolation). For the same reasons, it also satisfies $\mathsf{pd}(\checkmark, I, P, s)$ for *any* process P, interface I, and trace s. As a result, monitor $\checkmark$ is a *top element* for both preorders, as evidenced in Eqs. (15) and (16). By analysing the LTS behaviour of both monitors end and $c!a.\mathsf{end}$, we can also see that no execution path leads to a detection, and it turns out that these two monitors are equated as bottom elements by both preorders, as shown in Eqs. (15) and (16). Similarly, it is not hard to see that monitor $c!a.\checkmark$ potentially and deterministically detects *any* system execution that starts with the action $c!a$ (and nothing more); this makes it neither a top nor a bottom element. Surprisingly, even though monitor Ω is a bottom element for the $\sqsubseteq_{\mathsf{pd}}$ preorder,

it is *not* a bottom element for the $\sqsubseteq_{\mathsf{dd}}$ preorder, even though it appears to be that case when simply looking at its LTS behaviour: there is *no* trace s such that $\Omega \stackrel{s}{\Longrightarrow} \checkmark$. More concretely, Ω prohibits any external action from the process it is instrumented with, limiting all computations to ϵ-computations, *i.e.,* irrespective of $I \triangleright P$, configuration $I \triangleright P \triangleleft \Omega$ exhibits no s-computations for any s where $s \neq \epsilon$, rendering $\mathsf{dd}(\Omega, I, P, s)$ for $s \neq \epsilon$ vacuously true (see Rem. 2). ■

Reasoning about monitor divergences becomes even less obvious in the presence of other monitor combinators.

Example 10. Recall the divergent monitor Ω from (11). We have:

$$\Omega + \mathsf{end} \quad \sqsubset_{\mathsf{dd}} \quad \big(\Omega \quad \cong_{\mathsf{dd}} \quad \Omega + \checkmark \quad \cong_{\mathsf{dd}} \quad \mathsf{rec}\, X.(\tau.X + \checkmark)\big) \tag{17}$$

$$\Omega + c!a.\checkmark \quad \sqsubset_{\mathsf{pd}} \quad \Omega + \checkmark \qquad \text{but} \qquad \Omega + c!a.\checkmark \quad \cong_{\mathsf{dd}} \quad \Omega + \checkmark \tag{18}$$

In Eq. (17), the monitor $\Omega + \mathsf{end}$ does *not* deterministically detect *any* computation: when composed with an arbitrary $I \triangleright P$, it clearly can never reach a detection, but it can neither prohibit the process from producing visible actions, as in the case of Ω (see rules mVer, mChR and iMon). Monitor $\Omega + \checkmark$ can either behave like Ω or transition to $\checkmark$ after one external action observed; in both cases, it deterministically detects all s-computation where $|s| > 0$. The monitor $\mathsf{rec}\, X.(\tau.X + \checkmark)$ first silently transitions to $(\tau.\mathsf{rec}\, X.(\tau.X + \checkmark)) + \checkmark$ and then either transitions back to the original state or else transitions to $\checkmark$ with an external action; in either case, when composed with *any* process $I \triangleright P$, it also deterministically detects all s-computation as long as $|s| > 0$.

In Eq. (18), monitor $\Omega + c!a.\checkmark$ potentially detects less computations than $\Omega + \checkmark$ (*e.g.,* for $I = \{c, a, b\}$, $P = c!b.\mathsf{nil}$ and $s = c!b.\epsilon$, the predicate $\mathsf{pd}(\Omega + \checkmark, I, P, s)$ holds but $\mathsf{pd}(\Omega + c!a.\checkmark, I, P, s)$ *does not*). However, both deterministically detect the same computations, *i.e.,* all s-computation where $|s| > 0$. Specifically, if a process being monitored can produce an action other than $c!a$, the instrumentation with monitor $\Omega + c!a.\checkmark$ restrains such an action, since the monitor *cannot* transition with that external action (it can only transition with $c!a$). Nevertheless, it can τ-transition (see rules iMon and iTrm in Fig. 2). ■

The preorders of Def. 4 are not as discriminating as one might expect.

Example 11. Consider the monitor $M_{\text{any}} = (x)?(y).\checkmark + (x)!(y).\checkmark$. Intuitively, the monitor M_{any} potentially and deterministically detects *any* s-computation when $|s| > 0$ since after any action (irrespective of whether it is an input or an output, or of the action subject and object values) the monitor always detects.

$$M_{\text{any}} \quad \cong_{\mathsf{pd}} \quad M_{\text{any}} + \Omega \qquad \text{and} \qquad M_{\text{any}} \quad \cong_{\mathsf{dd}} \quad M_{\text{any}} + \Omega \quad \cong_{\mathsf{dd}} \quad \Omega \tag{19}$$

Perhaps less intuitively, monitor $M_{\text{any}} + \Omega$ produces the same potential and deterministic detections, yielding the *resp.* equalities in Eq. (19). In the case of deterministic detections, monitor $M_{\text{any}} + \Omega$ may exhibit either of two behaviours: it can either detect after observing one action (just like in the case of M_{any}) or else silently transition to Ω, which prohibits any traces that are not equal to ϵ

from the process it is instrumented with. In either case, the monitor detects *all* traces s where $|s| > 0$ (in the case it transitions to Ω, this is trivially satisfied since there are none). In fact, the two monitors are deterministic detection equivalent to Ω.

$$\begin{array}{llllll} c!a.\mathsf{end} & \cong_{\mathsf{pd}} & c!a.\mathsf{end} + c!a.\Omega & \cong_{\mathsf{pd}} & \mathsf{end} \\ \text{and} \quad c!a.\mathsf{end} & \cong_{\mathsf{dd}} & c!a.\mathsf{end} + c!a.\Omega & \cong_{\mathsf{dd}} & \mathsf{end} \end{array} \tag{20}$$

In Eq. (20), neither monitor produces *any* potential or deterministic detections and they are thus equivalent according to the *resp.* kernel equivalences of Def. 4. In fact, they are potential and deterministic equivalent to the monitor end, which is a bottom element for both preorders. ■

There are settings where the equalities established in E.g. 11 are deemed too coarse. For instance, in Eq. (20), whereas monitor $c!a.\mathsf{end}$ is innocuous when instrumented with a process, monitor $c!a.\mathsf{end} + c!a.\Omega$ can change the observed behaviour of the process under scrutiny after the action $c!a$ is emitted (namely by suppressing external actions, as explained in E.g. 10); a similar argument applies for the monitors in Eq. (19).

We thus define a third monitor predicate called *transparency* [3, 11, 65, 14], stating that whenever a monitored process *cannot* perform an external action, it must be because the (unmonitored) process is unable to perform that action (*i.e.,* the monitor instrumentation does not prohibit that action).

Definition 5 (Transparency Preorder)**.** M is transparent for $I \triangleright P$ *wrt.* trace s, denoted as $\mathsf{tr}(M, P, I, s)$, iff

$$\left(I \triangleright P \triangleleft M \overset{s}{\Longrightarrow} Q \triangleleft N \text{ and } \mathbf{aftr}(I, s) \triangleright (Q \triangleleft N) \overset{\alpha}{\not\Longrightarrow}\right) \text{ implies } \mathbf{aftr}(I, s) \triangleright Q \overset{\alpha}{\not\Longrightarrow} .$$

We define the induced preorder as expected:

$$M \sqsubseteq_{\mathsf{tr}} N \quad \overset{\mathbf{def}}{=} \quad \forall I, P, s \cdot \ \mathsf{tr}(M, I, P, s) \text{ implies } \mathsf{tr}(N, I, P, s) \qquad ■$$

Although the preorders in Def. 4 and Def. 5 are interesting in their own right, we define the following relation as the finest monitor preorder in this paper.

Definition 6 (Monitor Preorder)**.**

$$M \sqsubseteq N \quad \overset{\mathbf{def}}{=} \quad M \sqsubseteq_{\mathsf{pd}} N \ \text{ and } \ M \sqsubseteq_{\mathsf{dd}} N \ \text{ and } \ M \sqsubseteq_{\mathsf{tr}} N \qquad ■$$

Example 12. Recall the monitors described in E.g. 11. Although it turns out that the inequality $M_{\text{any}} + \Omega \sqsubseteq M_{\text{any}}$ holds, we have $M_{\text{any}} \not\sqsubseteq M_{\text{any}} + \Omega$ because $M_{\text{any}} \not\sqsubseteq_{\mathsf{tr}} M_{\text{any}} + \Omega$. For instance, for $I = \{c, a\}$, $P = c!a.c!a.\mathsf{nil}$ and $s = c!a.\epsilon$ we have $\mathsf{tr}(M_{\text{any}}, I, P, s)$ but $\neg\mathsf{tr}(M_{\text{any}} + \Omega, I, P, s)$. Using analogous reasoning, we also have $c!a.\mathsf{end} \not\sqsubseteq c!a.\mathsf{end} + c!a.\Omega$. ■

Inequalities from the preorders of Def. 4 and Def. 5 are relatively easy to repudiate. For instance, recall

$$M_3 = c!a.\checkmark \qquad\qquad M_5 = c!a.\checkmark + c!b.\checkmark + c!a.\mathsf{end} + c!b.\mathsf{end}$$

from E.g. 6, together with the process $P = \mathsf{new}\, d.(d!c \,\|\, d?x.c!a.\mathsf{nil} \,\|\, d?x.c!b.\mathsf{nil})$, the interface $I = \{c, b, a\}$ and the trace $t = c!b.\epsilon$ (also defined in the same example). We can use P, I and t as counter examples to show that $\mathsf{pd}(M_5, I, P, t)$ and $\neg\mathsf{pd}(M_3, I, P, t)$, thus *disproving* $M_5 \sqsubseteq_{\mathsf{pd}} M_3$; similarly, P, I and $s = c!a.\epsilon$ can be used to show $\mathsf{dd}(M_3, I, P, s)$ and $\neg\mathsf{dd}(M_5, I, P, s)$ and refute $M_3 \sqsubseteq_{\mathsf{pd}} M_5$. However, it is much harder to show that an inequality from these preorders holds because we need to consider monitor behaviour *wrt. all* possible processes, interfaces and traces. As shown in Ex. 8, Ex. 10 and Ex. 11, this often requires intricate reasoning in terms of the three LTSs defined in Fig. 1 and Fig. 2, namely the process LTS, the monitor LTS and the instrumentation LTS.

5. Characterising the Potential Detection Preorder

We define three alternative monitor preorders that correspond to the preorders in Sec. 4 and for which positive statements about inequalities are easier to establish. The new preorders are defined exclusively in terms of the monitor operational semantics of Fig. 2, and do not consider how they are affected by arbitrary processes as in Defs. 4 to 6 (requiring the process and instrumentation LTSs of Figs. 1 and 2). Importantly, we show that the new preorders coincide with those in Sec. 4: apart from equipping us with an simpler mechanism for determining the inequalities of Sec. 4, the correspondence results provide further insight into the properties satisfied by the preorders of Def. 4 and Def. 5.

We start with the potential-detection preorder. We first define a restricted monitor LTS that disallows idempotent transitions from verdicts, $w \xrightarrow{\alpha} w$ since these are redundant when considering the monitor operational semantics in isolation.

Definition 7 (Restricted Monitor Semantics)**.** A *derived monitor transition*, $M \xrightarrow{\mu}_r N$, is the least relation satisfying the conditions $M \xrightarrow{\mu} N$ and $M \neq w$. $M \overset{s}{\Longrightarrow}_r N$ denotes a *transition sequence* in the restricted LTS. ■

Remark 3. Although Def. 7 prohibits rule mVer (Fig. 2) from being used as the root of a transition derivation, it can still be used *e.g.,* to derive:

$$\dfrac{\dfrac{\dfrac{}{\checkmark \xrightarrow{\alpha} \checkmark}\,\textsc{mVer}}{\checkmark + M \xrightarrow{\alpha} \checkmark}\,\textsc{mChL} \qquad \checkmark + M \neq w}{\checkmark + M \xrightarrow{\alpha}_r \checkmark}$$

■

The restricted LTS is used to limit the detecting transition sequences on the left of the implication in Def. 8. We still permit these transitions to be matched

by transition sequences in the *original* monitor LTS, so as to allow the monitor to the right of the inequality to match the sequence with a *prefix* of visible actions (which can then be padded by $\checkmark \xrightarrow{\alpha} \checkmark$ transitions as required).

Definition 8 (Alternative Potential Detection Preorder)**.**

$$M \preceq_{\mathsf{pd}} N \stackrel{\text{def}}{=} \forall s \cdot M \stackrel{s}{\Rightarrow}_r \checkmark \quad \text{implies} \quad N \stackrel{s}{\Rightarrow} \checkmark$$

The new preorder $M \preceq_{\mathsf{pd}} N$ of Def. 8 fully characterises the contextual preorder $M \sqsubseteq_{\mathsf{pd}} N$ of Def. 4. We proceed to show this in Thm. 1, which relies on a useful property about the restricted monitor LTS of Def. 7.

Lemma 5 (Restricted Monitor Operational Semantics)**.**

1. $M \stackrel{s}{\Rightarrow}_r N$ implies $M \stackrel{s}{\Rightarrow} N$.
2. $M \stackrel{s}{\Rightarrow} N$ and $N \neq w$ implies $M \stackrel{s}{\Rightarrow}_r N$.
3. $M \stackrel{s}{\Rightarrow} w$ implies $\exists t \leq s \cdot M \stackrel{t}{\Rightarrow}_r w$

Proof. By induction on the number of transitions. For the second clause, we make use of Cor. 1. □

The proof of Thm. 1 is split into two parts. The *if* part, *i.e.,* if $M \preceq_{\mathsf{pd}} N$ then $M \sqsubseteq_{\mathsf{pd}} N$, which justifies the use of $\preceq_{\mathsf{pd}}$ as a *sound* method for determining when two monitors are related according to $\sqsubseteq_{\mathsf{pd}}$, follows from Lem. 6. The *only-if* part, *i.e., M* $\preceq_{\mathsf{pd}}$ *N* only if $M \sqsubseteq_{\mathsf{pd}} N$, shows that such a characterisation is *complete* and follows from Lem. 8 below.

Lemma 6 (Soundness)**.** $M \preceq_{\mathsf{pd}} N$ implies $M \sqsubseteq_{\mathsf{pd}} N$

Proof. From Def. 4, we prove the implication by assuming $\mathsf{pd}(M, I, P, s)$ for an arbitrary I,P and s, and then showing that $\mathsf{pd}(N, I, P, s)$ holds. The assumption $\mathsf{pd}(M, I, P, s)$ implies that $I \triangleright P \triangleleft M \stackrel{s}{\Rightarrow} Q \triangleleft \checkmark$ for some Q, and by verdict unzipping, Lem. 2, we obtain:

$$I \triangleright P \stackrel{s}{\Rightarrow} Q \tag{21}$$

$$M \stackrel{s}{\Rightarrow} \checkmark. \tag{22}$$

From Eq. (22) and Lem. 5.3 we have $M \stackrel{t}{\Rightarrow}_r \checkmark$ where t is a prefix of s. Thus by $M \preceq_{\mathsf{pd}} N$ we obtain $N \stackrel{t}{\Rightarrow} \checkmark$ which can be extended to $N \stackrel{s}{\Rightarrow} \checkmark$ using rule MVER from Fig. 2 for the additional padding. Thus, by Eq. (21) and verdict zipping, Lem. 4, we construct $I \triangleright P \triangleleft N \stackrel{s}{\Rightarrow} Q \triangleleft \checkmark$, which implies $\mathsf{pd}(N, I, P, s)$. □

Completeness, Lem. 8, requires us to define a generic way how to construct a *characteristic process* (and interface) from a trace. The resulting characteristic process, $\mathbf{prc}(s)$ defined in Def. 10, can produce the required trace *wrt.* a maximal interface (approximation), $\mathbf{nm}(s)$ extracted via Def. 9.

Definition 9 (Trace Names)**.**

$$\mathbf{nm}(c?d.s) \stackrel{\text{def}}{=} \{c, d\} \cup \mathbf{nm}(s) \quad \mathbf{nm}(c!d.s) \stackrel{\text{def}}{=} \{c, d\} \cup \mathbf{nm}(s) \quad \mathbf{nm}(\epsilon) \stackrel{\text{def}}{=} \emptyset \quad \blacksquare$$

Definition 10 (Trace-characterising Process)**.**

$$\mathbf{prc}(c?d.s) \stackrel{\text{def}}{=} c?x.\mathbf{prc}(s) \qquad \mathbf{prc}(c!d.s) \stackrel{\text{def}}{=} c!d.\mathbf{prc}(s) \qquad \mathbf{prc}(\epsilon) \stackrel{\text{def}}{=} \mathsf{nil} \quad \blacksquare$$

Lemma 7 (Properties of the characteristic process)**.** For all traces s,t:

1. $\mathbf{nm}(s) \triangleright \mathbf{prc}(s) \stackrel{t}{\Rightarrow} P$ implies $s = tt'$ for some t' where $P = \mathbf{prc}(t')$
2. $\mathbf{nm}(st) \triangleright \mathbf{prc}(st) \stackrel{s}{\Rightarrow} \mathbf{prc}(t)$ where $\mathbf{aftr}(\mathbf{nm}(st), s) = \mathbf{nm}(st)$

Proof. By induction on the structure of s. □

Lemma 8 (Completeness)**.** $M \sqsubseteq_{\mathsf{pd}} N$ implies $M \preceq_{\mathsf{pd}} N$

Proof. From Def. 8, assume $M \sqsubseteq_{\mathsf{pd}} N$ and pick an s such that $M \stackrel{s}{\Rightarrow}_r \checkmark$. We need to show that $N \stackrel{s}{\Rightarrow} \checkmark$. From $M \stackrel{s}{\Rightarrow}_r \checkmark$ and Lem. 5.1, we obtain $M \stackrel{s}{\Rightarrow} \checkmark$. Thus, by Lem. 7.2 and Lem. 4 (verdict zipping) we deduce:

$$\mathbf{nm}(s) \triangleright \mathbf{prc}(s) \triangleleft M \stackrel{s}{\Rightarrow} \mathsf{nil} \triangleleft \checkmark$$

which implies $\mathsf{pd}(M, \mathbf{nm}(s), \mathbf{prc}(s), s)$. From the assumption $M \sqsubseteq_{\mathsf{pd}} N$ we obtain the predicate $\mathsf{pd}(N, \mathbf{nm}(s), \mathbf{prc}(s), s)$ which, by Def. 3, means that

$$\mathbf{nm}(s) \triangleright \mathbf{prc}(s) \triangleleft N \stackrel{s}{\Rightarrow} Q \triangleleft \checkmark \tag{23}$$

for some Q. Thus, by Eq. (23) and Lem. 2 (verdict unzipping) we obtain $N \stackrel{s}{\Rightarrow} \checkmark$ as required. □

Theorem 1 (Potential-Detection Preorders)**.** $M \sqsubseteq_{\mathsf{pd}} N \quad \text{iff} \quad M \preceq_{\mathsf{pd}} N$

Proof. The *if* case follows from Lem. 6 whereas the *only-if* case is a direct consequence of Lem. 8. □

Example 13. Recall $\Omega + c!a.\checkmark$ and $\Omega + \checkmark$ from Eq. (18) of E.g. 10. By virtue of Thm. 1, to show that $\Omega + c!a.\checkmark \sqsubseteq_{\mathsf{pd}} \Omega + \checkmark$ holds, it suffices to show that $\Omega + c!a.\checkmark \preceq_{\mathsf{pd}} \Omega + \checkmark$, which allows us to focus *exclusively* on the LTS of the *resp.* monitors $\Omega + c!a.\checkmark$ and $\Omega + \checkmark$. Moreover, the bi-implication of the full abstraction result in Thm. 1 guarantees that this can always be done.

This can alleviate the proof burden substantially and facilitate the automation of the process substantially. For instance, by Def. 8, we only need to consider $\Omega + c!a.\checkmark \stackrel{c!a}{\Longrightarrow}_r \checkmark$, which can be easily matched by $\Omega + \checkmark \stackrel{c!a}{\Longrightarrow} \checkmark$. Similarly, recall the monitor $c!(x).\mathsf{if}\ x = d\ \mathsf{then}\ \checkmark\ \mathsf{else}\ \mathsf{end}$ from Eq. (14). To show that $c!(x).\mathsf{if}\ x = d\ \mathsf{then}\ \checkmark\ \mathsf{else}\ \mathsf{end} \sqsubseteq_{\mathsf{pd}} \checkmark$, we only need to consider $c!(x).\mathsf{if}\ x = d\ \mathsf{then}\ \checkmark\ \mathsf{else}\ \mathsf{end} \stackrel{c!d}{\Longrightarrow}_r \checkmark$, which can be matched by $\checkmark \xrightarrow{c!a} \checkmark$. ■

6. Characterising the Deterministic Detection Preorder

The characterisation of the remaining preorders is more involved. For a start, both characterisations rely on divergence judgments.

Definition 11 (Divergence and Strong Divergence)**.**

1. $M \uparrow$ denotes that M *diverges*, meaning that it can produce an infinite transition sequence of τ-actions $M \xrightarrow{\tau} M' \xrightarrow{\tau} M'' \xrightarrow{\tau} \ldots$
2. $M \Uparrow$ denotes that M *strongly diverges*, meaning that it *cannot* produce finite transition sequence of τ-actions $M \xrightarrow{\tau} M' \xrightarrow{\tau} \ldots M'' \not\xrightarrow{\tau}$. ■

The alternative preorder for deterministic detections, Def. 13 below, is based on three predicates describing the behaviour of a monitor M along a trace s. The predicate $\mathsf{blk}(M, s)$ describes the potential for monitor M to *block* before it can complete trace s. The predicate $\mathsf{fl}(M, s)$ describes the potential for monitor M to *fail* after monitoring trace s: an s-derivative of M either reaches a non-detecting state from which no further τ actions are possible, or it diverges (implicitly, by Lem. 1, this also implies that the monitors along the diverging sequences are never detecting). Finally the predicate $\mathsf{nd}(M, s)$ states the existence of a non-detecting s-derivative of monitor M.

Definition 12 (Monitor Block, Fail and Non-Detecting)**.**

$$\begin{aligned}
\mathsf{blk}(M, s) &\stackrel{\text{def}}{=} \exists s_1, \alpha, s_2, N \cdot s = s_1 \alpha s_2 \text{ and } M \stackrel{s_1}{\Longrightarrow} N \not\xrightarrow{\tau} \text{ and } N \not\xrightarrow{\alpha} \\
\mathsf{fl}(M, s) &\stackrel{\text{def}}{=} \exists N \cdot M \stackrel{s}{\Longrightarrow} N \text{ and } \big((N \neq \checkmark \text{ and } N \not\xrightarrow{\tau}) \text{ or } N \uparrow\big) \\
\mathsf{nd}(M, s) &\stackrel{\text{def}}{=} \exists N \cdot M \stackrel{s}{\Longrightarrow} N \text{ and } N \neq \checkmark
\end{aligned}$$ ■

Note that the predicate $\mathsf{blk}(M, s)$ of Def. 12 implicitly requires $|s| \geq 1$ for it to hold. It is also closed under extensions, as stated by the following corollary.

Corollary 2. $\mathsf{blk}(M, s) \quad \text{implies} \quad \forall t \cdot \mathsf{blk}(M, st)$ □

More importantly, the three monitor predicates in Def. 12 capture all the possible scenarios why a monitor might not able to deterministically detect a trace when composed with an arbitrary system. This intuition is formalised in the following three lemmata, namely Lems. 9 to 11.

Lemma 9. $\mathsf{blk}(M, s)$ and $I \triangleright P \stackrel{s}{\Longrightarrow} \quad \text{implies} \quad \neg\mathsf{dd}(M, I, P, s)$

Proof. Assume $\mathsf{blk}(M, s)$ and $I \triangleright P \stackrel{s}{\Longrightarrow}$. By Def. 12 we know

$$\exists s_1, \alpha, s_2, N \text{ such that } s = s_1 \alpha s_2 \tag{24}$$

$$\text{and } M \stackrel{s_1}{\Longrightarrow} N \not\xrightarrow{\tau} \text{ and } N \not\xrightarrow{\alpha} \tag{25}$$

From Eq. (24), we can also decomposed $I \triangleright P \stackrel{s}{\Longrightarrow}$ and express it as

$$\exists P', P'' \text{ such that } I \triangleright P \stackrel{s_1}{\Longrightarrow} P' \xrightarrow{\alpha} P'' \stackrel{s_2}{\Longrightarrow} P''' \tag{26}$$

By Eq. (25), Eq. (26) and Lem. 4 (zipping) we obtain $I \triangleright M \triangleleft P \stackrel{s_1}{\Longrightarrow} P' \triangleleft N$. Again, by Eqs. (25) and (26) and iTrm from Fig. 2 we can deduce the monitored system transition $\mathbf{aftr}(I, s_1) \triangleright P' \triangleleft N \xrightarrow{\alpha} P'' \triangleleft \mathsf{end}$. Finally, by (26) and Prop. 2 (verdict zipping) we deduce $\mathbf{aftr}(I, (s_1\alpha)) \triangleright P'' \triangleleft \mathsf{end} \stackrel{s_2}{\Longrightarrow} P'' \triangleleft \mathsf{end}$. The three transition sequences can be conjoined together for form

$$I \triangleright M \triangleleft P \stackrel{s_1}{\Longrightarrow} P' \triangleleft N \xrightarrow{\alpha} P'' \triangleleft \mathsf{end} \stackrel{s_2}{\Longrightarrow} P'' \triangleleft \mathsf{end}$$

which can then be extended to a full computation (which is either finite or infinite, depending on the sequence of τ-transitions that can be generated from P''). By Eq. (24), since $s = s_1\alpha s_2$, this constructs a non-detecting s-computation, which is the evidence required to obtain $\neg\mathsf{dd}(M, I, P, s)$. □

Lemma 10. $\mathsf{fl}(M, s)$ and $I \triangleright P \stackrel{s}{\Longrightarrow}$ implies $\neg\mathsf{dd}(M, I, P, s)$

Proof. Assume $\mathsf{fl}(M, s)$ and $I \triangleright P \stackrel{s}{\Longrightarrow}$. By Def. 12 we have

$$M \stackrel{s}{\Longrightarrow} N \tag{27}$$

$$(N \neq \checkmark \text{ and } N \stackrel{\tau}{\not\longrightarrow}) \text{ or } N \uparrow \tag{28}$$

By the assumption $I \triangleright P \stackrel{s}{\Longrightarrow} P'$ (for some P'), Eq. (27) and Lem. 4 (zipping) we obtain $I \triangleright P \triangleleft M \stackrel{s}{\Longrightarrow} P' \triangleleft N$. At this point, by Eq. (28), we have two cases to consider:

- If $N \uparrow$ we can easily construct a non-terminating s-computation. Importantly, by Lem. 1, we also know that this computation is non-detecting since the monitors along the diverging sequences can never be detecting (*i.e.*, verdict cannot produce τ-transitions).
- If we have $N \neq \checkmark$ and $N \stackrel{\tau}{\not\longrightarrow}$, then we can still extend this to a full s-computation (whether this is finite or infinite depends on the τ-transitions from P'); we note that this computation would be non-detecting as well, and hence obtain $\neg\mathsf{dd}(M, I, P, s)$.[4] □

Lemma 11. $\mathsf{nd}(M, s)$ and $I \triangleright P \stackrel{s}{\Longrightarrow} P'$ and $P' \uparrow$ implies $\neg\mathsf{dd}(M, I, P, s)$

Proof. Assume $\mathsf{nd}(M, s)$ and $I \triangleright P \stackrel{s}{\Longrightarrow} P'$ where $P' \uparrow$. From Def. 12 we know $M \stackrel{s}{\Longrightarrow} N$ for some N where $N \neq \checkmark$ and by Lem. 4 (zipping) we obtain:

$$I \triangleright P \triangleleft M \stackrel{s}{\Longrightarrow} P' \triangleleft N \tag{29}$$

Note that, by Cor. 1, every intermediate monitor in the transition sequence Eq. (29) cannot be detecting. This same transition sequence can be extended to

[4] Note that no interactions can happen between P' and N at this stage (using rules iMon and iTer), because this would necessarily generate an external action α and make it an $s\alpha$-computation instead of an s-computation.

a *non-detecting* derivation sequence using the (infinite) τ-transitions from $P' \uparrow$ and repeated applications of the rule iAsyP (*i.e.,* N never progresses). Since this transition sequence is infinite, it constitutes an s-computation, and hence we obtain the witness required to conclude $\neg\mathsf{dd}(M, I, P, s)$. □

The relationship between the deterministic detection predicate of Def. 3 and the monitor LTS predicates of Def. 12 is even stronger. In fact, Lem. 13 below justifies the need for the three (and only those three) predicates in Def. 12 to operationally characterise the violation of the deterministic detection predicate. Lem. 12 is a technical lemma used by Lem. 13 and allows us to retrace the cause of a divergence in a monitored system to either the process itself or the monitor.

Lemma 12. $I \triangleright P \triangleleft M \uparrow \quad$ implies $\quad I \triangleright P \uparrow$ or $M \uparrow$

Proof. By contradiction. Assume $I \triangleright P \triangleleft M \uparrow$ and consider the possibility of having $\neg(I \triangleright P \uparrow$ or $M \uparrow)$: this means that both $\neg(I \triangleright P \uparrow)$ and $\neg(M \uparrow)$. Thus we can place a *finite* upper bound on the maximum number of τ-transition sequences from $I \triangleright P$ and M (irrespective of the computation chosen); let us denote the *resp.* integer upper-bounds as k_1 and k_2. Now, for any diverging transition sequence in $I \triangleright P \triangleleft M$ (with an *infinite* number of transitions), we know that each transition is derived using either rule iAsyP, which means that a τ^*-derivative of P induced the transition, or by using rule iAsyM, meaning that a τ^*-derivative of M induced the transition. But the diverging transition sequence could not have more than $k_1 + k_2$ transitions, which yields a contradiction. □

Lemma 13 (Deterministic Detection Violation)**.** $\neg\mathsf{dd}(M, I, P, s)$ implies

1. $(\mathsf{blk}(M, s)$ and $I \triangleright P \overset{s}{\Longrightarrow})$ or;
2. $(\mathsf{fl}(M, s)$ and $I \triangleright P \overset{s}{\Longrightarrow})$ or;
3. $(\mathsf{nd}(M, s)$ and $\exists P' \cdot I \triangleright P \overset{s}{\Longrightarrow} P' \uparrow)$.

Proof. Pick an M, an I, a P and a s such that $\neg\mathsf{dd}(M, I, P, s)$. By Def. 3, this means that there exists a non-detecting s-computation from $I \triangleright P \triangleleft M$. We have two subcases to consider:

- If this non-detecting s-computation is *finite*, we have the transition sequence $I \triangleright P \triangleleft M \overset{s}{\Longrightarrow} Q \triangleleft N \overset{\tau}{\not\longrightarrow}$ for some Q and N. Clearly, we have

$$N \neq \checkmark \text{ and } N \overset{\tau}{\not\longrightarrow}. \tag{30}$$

  since, otherwise, we would contradict $\mathbf{aftr}(I, s) \triangleright Q \triangleleft N \overset{\tau}{\not\longrightarrow}$ using the rule iAsyM. By Lem. 3 (unzipping) we have

$$I \triangleright P \overset{s}{\Longrightarrow} Q \tag{31}$$

$$\text{and either } M \overset{s}{\Longrightarrow} N \tag{32}$$

$$\text{or } s = s_1 \alpha s_2 \text{ where } M \overset{s_1}{\Longrightarrow} M' \overset{\tau}{\not\longrightarrow}, M' \overset{\alpha}{\not\longrightarrow}, N = \mathsf{end} \tag{33}$$

In case of Eq. (32), by Eq. (30) we obtain $\mathsf{fl}(M,s)$ and since Eq. (31) gives us $I \triangleright P \stackrel{s}{\Longrightarrow}$) which satisfied the second alternative of the claimed conclusion disjunction. Analogously, in case of Eq. (33) we obtain the first alternative, namely ($\mathsf{blk}(M,s)$ and $I \triangleright P \stackrel{s}{\Longrightarrow}$).

- If this non-detecting s-computation is *infinite* we can split the computation into two parts

$$I \triangleright P \triangleleft M \stackrel{s}{\Longrightarrow} Q \triangleleft N \tag{34}$$

$$I' \triangleright Q \triangleleft N \uparrow \text{ where } I' = \mathbf{aftr}(I,s) \tag{35}$$

We note that, by Def. 2, it must be that $N \neq \checkmark$. Again, by Eq. (34) and Lem. 3, we have:

$$I \triangleright P \stackrel{s}{\Longrightarrow} Q \tag{36}$$

$$\text{and either } M \stackrel{s}{\Longrightarrow} N \tag{37}$$

$$\text{or } s = s_1 \alpha s_2 \text{ where } M \stackrel{s_1}{\Longrightarrow} M' \stackrel{\tau}{\not\longrightarrow}, M' \stackrel{\alpha}{\not\longrightarrow}, N = \mathsf{end} \tag{38}$$

In case of Eq. (38), by Eq. (36) we obtain ($\mathsf{blk}(M,s)$ and $I \triangleright P \stackrel{s}{\Longrightarrow}$). In case of Eq. (37), we use Eq. (35) and Lem. 12 to obtain two subcases:

  - If $N \uparrow$, then by Eq. (37) and Eq. (36) we obtain $\mathsf{fl}(M,s)$ and $I \triangleright P \stackrel{s}{\Longrightarrow}$.
  - If $I' \triangleright Q \uparrow$, by Eq. (37) and $N \neq \checkmark$ we satisfy the third alternative of the claimed conclusion disjunction, namely $\mathsf{nd}(M,s)$ and $\exists P' \cdot I \triangleright P \stackrel{s}{\Longrightarrow} P'$ and $P' \uparrow$. More precisely, $P' = Q$. □

Based on these results, we define our alternative deterministic detection preorder in terms of the predicates of Def. 12.

Definition 13 (Alternative Deterministic Detection Preorder)**.**

$$M \preceq_{\mathsf{dd}} N \stackrel{\mathbf{def}}{=} \forall s \cdot \begin{cases} \mathsf{blk}(N,s) & \text{implies} \quad \mathsf{blk}(M,s) \text{ or } \mathsf{fl}(M,s) \\ \mathsf{fl}(N,s) & \text{implies} \quad \mathsf{blk}(M,s) \text{ or } \mathsf{fl}(M,s) \\ \mathsf{nd}(N,s) & \text{implies} \quad \mathsf{nd}(M,s) \text{ or } \mathsf{blk}(M,s) \end{cases}$$

We write $M \simeq_{\mathsf{dd}} N$ to denote the kernel equality of this preorder (*i.e.,* $M \preceq_{\mathsf{dd}} N$ and $N \preceq_{\mathsf{dd}} M$). ■

In Lem. 14 we show that the new alternative preorder is sound *wrt.* its contextual counterpart from Def. 4. Note how the proof for this lemma works on the *contrapositive* of the preorder implication of Def. 4, and thus relies on the *violation* of the deterministic detection predicate of Def. 3, *i.e.,* $\neg\mathsf{dd}(M,I,P,s)$.

Lemma 14 (Soundness)**.** $M \preceq_{\mathsf{dd}} N$ implies $M \sqsubseteq_{\mathsf{dd}} N$

Proof. By Def. 4 we show the implication by the contrapositive. We assume $M \preceq_{\mathsf{dd}} N$ and pick I, P and s such that $\neg\mathsf{dd}(N,I,P,s)$. Then we show that $\neg\mathsf{dd}(M,I,P,s)$. From $\neg\mathsf{dd}(N,I,P,s)$ and Lem. 13 we obtain three subcases:

- If $\mathsf{blk}(N, s)$ and $I \triangleright P \stackrel{s}{\Longrightarrow}$, then by Def. 13 we know that:
  - Either $\mathsf{fl}(M, s)$, and by $I \triangleright P \stackrel{s}{\Longrightarrow}$ and Lem. 10 we obtain $\neg\mathsf{dd}(M, I, P, s)$;
  - Or $\mathsf{blk}(M, s)$. By $I \triangleright P \stackrel{s}{\Longrightarrow}$ and Lem. 9 we get $\neg\mathsf{dd}(M, I, P, s)$ as well.
- If $\mathsf{fl}(N, s)$ and $I \triangleright P \stackrel{s}{\Longrightarrow}$, then by Def. 13 we know that:
  - Either $\mathsf{fl}(M, s)$: again, by $I \triangleright P \stackrel{s}{\Longrightarrow}$ and Lem. 10 we get $\neg\mathsf{dd}(M, I, P, s)$;
  - Or $\mathsf{blk}(M, s)$: similar to the previous case, by $I \triangleright P \stackrel{s}{\Longrightarrow}$ and Lem. 9 we obtain $\neg\mathsf{dd}(M, I, P, s)$ as well.
- If $\mathsf{nd}(N, s)$ and $\exists P' \cdot I \triangleright P \stackrel{s}{\Longrightarrow} P' \uparrow$, then by Def. 13 we know that:
  - Either $\mathsf{nd}(M, s)$, and by $\exists P' \cdot I \triangleright P \stackrel{s}{\Longrightarrow} P' \uparrow$ and Lem. 11 we obtain $\neg\mathsf{dd}(M, I, P, s)$;
  - Or $\mathsf{blk}(M, s)$, in which case, by $I \triangleright P \stackrel{s}{\Longrightarrow} P'$ and Lem. 9 we obtain $\neg\mathsf{dd}(M, I, P, s)$ as well. □

We can also show that the alternative preorder of Def. 13 is complete, Lem. 17. This requires us to define new properties and prove a number of supporting lemmata. Concretely, Def. 14 defines a predicate stating that a process does *not* diverge after producing a trace s. Cor. 3 states that the characteristic process for trace s (Def. 9 and Def. 10) does not diverge after performing s. Cor. 4 states that a non-deterministically-detecting monitor along s for $I \triangleright P$ when $I \triangleright P$ never diverges after s implies that either the monitor blocks along trace s or it fails after analysing s. Finally, Lem. 15 states that $\mathsf{fl}(M, s)$ is a stronger predicate than $\mathsf{nd}(M, s)$.

Definition 14 (Non-Diverging After s)**.**

$$\mathsf{ndiv}(I, P, s) \stackrel{\mathbf{def}}{=} I \triangleright P \stackrel{s}{\Longrightarrow} P' \text{ implies } \neg\big(\mathbf{aftr}(I, s) \triangleright P' \uparrow\big) \qquad \blacksquare$$

Corollary 3. For all traces s we have $\mathsf{ndiv}(\mathbf{nm}(s), \mathbf{prc}(s), s)$

Proof. Follows from Lem. 7. □

Corollary 4. $\neg\mathsf{dd}(M, I, P, s)$ and $\mathsf{ndiv}(I, P, s)$ implies $\mathsf{blk}(M, s)$ or $\mathsf{fl}(M, s)$

Proof. Follows from Lem. 13, where $\mathsf{ndiv}(I, P, s)$ precludes the last option. □

Lemma 15. $\mathsf{fl}(M, s)$ implies $\mathsf{nd}(M, s)$

Proof. From Def. 12, we have two cases to consider for $\mathsf{fl}(M, s)$:

- Case $M \stackrel{s}{\Longrightarrow} N \stackrel{\tau}{\not\longrightarrow}$ and $N \neq \checkmark$ trivially implies the weaker conditions of $\mathsf{nd}(M, s)$.
- Case $M \stackrel{s}{\Longrightarrow} N \uparrow$ implies $N \stackrel{\tau}{\longrightarrow}$. From Lem. 1 we know that $N \neq w$ meaning that $N \neq \checkmark$ as well, from which $\mathsf{nd}(M, s)$ follows. □

We are now in a position to prove completeness. Before, however, we need to define a slightly different characterising process for a particular trace that diverges once the external trace is produced. This is used for the last subcase of the proof for Lem. 17.

Definition 15 (Trace-characterising Diverging Process)**.**

$$\mathbf{prcd}(c?d.s) \stackrel{\text{def}}{=} c?x.\mathbf{prcd}(s) \quad \mathbf{prcd}(c!d.s) \stackrel{\text{def}}{=} c!d.\mathbf{prcd}(s) \quad \mathbf{prcd}(\epsilon) \stackrel{\text{def}}{=} \Omega \quad \blacksquare$$

Lemma 16. $\mathbf{nm}(s) \triangleright \mathbf{prcd}(s) \stackrel{s}{\Longrightarrow} \Omega$

Proof. By structural induction on s. □

Lemma 17 (Completeness)**.** $M \sqsubseteq_{\mathsf{dd}} N$ implies $M \preceq_{\mathsf{dd}} N$

Proof. By Def. 13 we have to assume $M \sqsubseteq_{\mathsf{dd}} N$ and:

- Pick an s where $\mathsf{blk}(N, s)$, and show that $\mathsf{blk}(M, s)$ or $\mathsf{fl}(M, s)$. From $\mathsf{blk}(N, s)$, Lems. 7 and 9 we obtain $\neg\mathsf{dd}(N, \mathbf{nm}(s), \mathbf{prc}(s), s)$. From $M \sqsubseteq_{\mathsf{dd}} N$, it must be the case that $\neg\mathsf{dd}(M, \mathbf{nm}(s), \mathbf{prc}(s), s)$ as well, and from Cors. 3 and 4 we obtain $\mathsf{blk}(M, s)$ or $\mathsf{fl}(M, s)$ as required.
- Pick an s where $\mathsf{fl}(N, s)$ and show that $\mathsf{fl}(M, s)$ or $\mathsf{blk}(M, s)$. The proof is analogous to the previous case.
- Pick an s where $\mathsf{nd}(N, s)$ and show that $\mathsf{nd}(M, s)$ or $\mathsf{blk}(M, s)$. From $\mathsf{nd}(N, s)$, Lems. 11 and 16 we obtain $\neg\mathsf{dd}(N, \mathbf{nm}(s), \mathbf{prc}(s), s)$. From $M \sqsubseteq_{\mathsf{dd}} N$, it must be the case that $\neg\mathsf{dd}(M, \mathbf{nm}(s), \mathbf{prc}(s), s)$ as well, and from Lem. 13 we obtain $\mathsf{blk}(M, s)$, $\mathsf{fl}(M, s)$, or $\mathsf{nd}(M, s)$. In the case where we have $\mathsf{fl}(M, s)$, we know by Lem. 15 that we also have $\mathsf{nd}(M, s)$. The other two possible cases, *i.e.,* $\mathsf{blk}(M, s)$ or $\mathsf{nd}(M, s)$, are immediately what is required by Def. 13. □

Theorem 2 (Deterministic-Detection Preorders)**.** $M \sqsubseteq_{\mathsf{dd}} N$ iff $M \preceq_{\mathsf{dd}} N$

Proof. Follows from Lem. 14 and Lem. 17. □

In the following examples we show how to put Thm. 2 to good use.

Example 14. Consider the monitors $M = c?a.\mathsf{end} + (x)!b.\mathsf{end}$ and $N = c?a.\mathsf{end}$. By virtue of Thm. 2, in order to determine that $M \sqsubseteq_{\mathsf{dd}} N$ it suffices to prove $M \preceq_{\mathsf{dd}} N$. We proceed as follows:

1. We have $\mathsf{blk}(N, s)$ whenever $s = \alpha t$ and $\alpha \neq c?a$. We have two subcases:
   - If $\mathsf{match}((x)!b, \alpha)$ is undefined, we show $\mathsf{blk}(M, \alpha t)$ by first showing that $\mathsf{blk}(M, \alpha t)$ for the specific case where $t = \epsilon$ and then generalising the result for arbitrary t using Cor. 2.
   - If $\exists\sigma{\cdot}\mathsf{match}((x)!b, \alpha) = \sigma$, we show $\mathsf{fl}(M, \alpha t)$ by first showing $\mathsf{fl}(M, \alpha t)$ for the specific case where $t = \epsilon$ since $M \stackrel{\alpha}{\Longrightarrow} \mathsf{end}$; clearly $\mathsf{end} \neq \checkmark$ and $\mathsf{end} \stackrel{\tau}{\not\longrightarrow}$. We then generalise the result for arbitrary t using Prop. 1.

2. For any s where either $s = \epsilon$ or $s = c?a.t$ for some t, we have $\mathsf{fl}(N, s)$:
   - When $s = \epsilon$ we can easily show $\mathsf{fl}(M, \epsilon)$.
   - When $s = c?a.t$ for some t, we can first show $\mathsf{fl}(M, c?a.\epsilon)$ and then use Prop. 1 again, for all extensions to alleviate the proof burden.
3. For any s where either $s = \epsilon$ or $s = c?a.t$ for some t, we also have $\mathsf{nd}(N, s)$: the required proof is analogous to the previous case. We can also use (the technical lemma) Lem. 10 and the previous case for $\mathsf{fl}(N, s)$ to simplify the proof further. ■

Example 15. Recall the statement $\checkmark \not\sqsubseteq_{\mathsf{dd}} \tau.\checkmark$ from E.g. 8. The inequality was repudiated by *"guessing"* the process context $\Omega_P = \mathsf{rec}\, X.(\mathsf{if}\ c{=}c\ \mathsf{then}\ X\ \mathsf{else}\ X)$ that drives the monitor along the trace ϵ. Although intuitive, this method is arguably hard to automate because there is no immediately apparent connection between the process Ω_P and the two monitors $\checkmark$ and $\tau.\checkmark$.

Full abstraction (*i.e.,* completeness, Lem. 17), provides an alternative way how to disprove $\checkmark \sqsubseteq_{\mathsf{dd}} \tau.\checkmark$, by showing that $\checkmark \not\preceq_{\mathsf{dd}} \tau.\checkmark$. Specifically, we can readily argue that whereas $\mathsf{nd}(\tau.\checkmark, \epsilon)$, we can neither show $\mathsf{nd}(\checkmark, \epsilon)$ nor $\mathsf{blk}(\checkmark, \epsilon)$. This reasoning is easier to automate and infer from the structure of the two monitors. ■

Example 16. Recall the equalities $\Omega \cong_{\mathsf{dd}} \Omega + \checkmark \cong_{\mathsf{dd}} \mathsf{rec}\, X.(\tau.X + \checkmark)$ claimed in Eq. (17) of E.g. 10. It is easier to determine these equalities by considering only the LTSs of the respective monitors to show that $\Omega \simeq_{\mathsf{dd}} \Omega + \checkmark \simeq_{\mathsf{dd}} \mathsf{rec}\, X.(\tau.X + \checkmark)$ since:

1. For any $s \neq \epsilon$ we have the negated predicates $\neg\mathsf{blk}(\Omega, s)$, $\neg\mathsf{blk}(\Omega + \checkmark, s)$ and $\neg\mathsf{blk}(\mathsf{rec}\, X.(\tau.X + \checkmark), s)$. For instance, whenever $s \neq \epsilon$, it is not hard to prove that $\mathsf{rec}\, X.(\tau.X + \checkmark) \stackrel{s}{\Longrightarrow} N$ implies $N = \checkmark$ (and similarly for $\Omega + \checkmark$); Ω, on the other hand can never produce such a transition.
2. We only have $\mathsf{fl}(\Omega, \epsilon)$, $\mathsf{fl}(\Omega{+}\checkmark, \epsilon)$ and $\mathsf{fl}(\mathsf{rec}\, X.(\tau.X{+}\checkmark), \epsilon)$ and by Lem. 15 we also have $\mathsf{nd}(\Omega, \epsilon)$, $\mathsf{nd}(\Omega + \checkmark, \epsilon)$ and $\mathsf{nd}(\mathsf{rec}\, X.(\tau.X + \checkmark), \epsilon)$. ■

Remark 4. The alternative preorder in Def. 13 can be optimised further using refined versions of the predicates $\mathsf{fl}(M, s)$ and $\mathsf{nd}(M, s)$ that are defined in terms of the restricted monitor transitions of Def. 7, as in the case of Def. 3. ■

7. Characterising the Transparency Preorder

We can also characterise the contextual transparency preorder of Def. 5. The alternative transparency preorder, Def. 17 below, is defined in terms of *divergence refusals* which, in turn, rely on *strong* divergences from Def. 11. Intuitively, divergence refusals are the set of actions that cannot be performed whenever a monitor reaches a strongly divergent state following the analysis of trace s. These actions turn out to be precisely those actions that are suppressed on a process after producing trace s, when instrumented with the respective monitor.

Definition 16 (Divergence Refusals)**.**

$$\mathsf{dref}(M,s) \stackrel{\mathbf{def}}{=} \left\{\alpha \mid \exists N \cdot M \stackrel{s}{\Longrightarrow} N \text{ and } N \Uparrow \text{ and } N \stackrel{\alpha}{\not\Longrightarrow}\right\} \qquad \blacksquare$$

Definition 17 (Alternative Transparency Preorder)**.**

$$M \preceq_{\mathsf{tr}} N \stackrel{\mathbf{def}}{=} \forall s \cdot \mathsf{dref}(N,s) \subseteq \mathsf{dref}(M,s) \qquad \blacksquare$$

As before, it turns out that the alternative transparency preorder of Def. 17 coincides with its contextual counterpart from Def. 5. But before we embark on the proof for the characterisation of the transparency preorder, we define a *second* contextual preorder for transparency that turns out to be equivalent to that of Def. 5. Importantly, this second preorder facilitates the proof of the characterisation for the alternative preorder.

Definition 18 (Action-Specific Transparency Preorder)**.** M is transparent for action α, *wrt.* the system $I \triangleright P$ and trace s, denoted as $\mathsf{atr}(M,P,I,s,\alpha)$, iff

$$\left(I \triangleright P \triangleleft M \stackrel{s}{\Longrightarrow} Q \triangleleft N \text{ and } \mathbf{aftr}(I,s) \triangleright (Q \triangleleft N) \stackrel{\alpha}{\not\Longrightarrow}\right) \text{ implies } \mathbf{aftr}(I,s) \triangleright Q \stackrel{\alpha}{\not\Longrightarrow} .$$

The action-specific transparency predicate induces the *resp.* preorder:

$$M \sqsubseteq_{\mathsf{atr}} N \quad \stackrel{\mathbf{def}}{=} \quad \forall I,P,s,\alpha \cdot \ \mathsf{atr}(M,I,P,s,\alpha) \text{ implies } \mathsf{atr}(N,I,P,s,\alpha) \qquad \blacksquare$$

The two transparency preorders, namely $M \sqsubseteq_{\mathsf{tr}} N$ of Def. 5 and $M \sqsubseteq_{\mathsf{atr}} N$ of Def. 18, turn out to be equivalent, as shown in Thm. 3. This theorem relies on the following lemmata, the most important of which is Lem. 20 which, in turn, relies on the following two technical lemmata.

Lemma 18. $\left(I \triangleright P \xrightarrow{\alpha} P' \text{ and } M \stackrel{\alpha}{\not\Longrightarrow} \text{ and } M \Uparrow\right)$ implies $I \triangleright P \triangleleft M \stackrel{\alpha}{\not\Longrightarrow}$

Proof. We proceed by contradiction. We assume $I \triangleright P \triangleleft M \stackrel{\alpha}{\Longrightarrow}$ and then show that we either contradict $M \stackrel{\alpha}{\not\Longrightarrow}$ or $M \Uparrow$.

From $I \triangleright P \triangleleft M \stackrel{\alpha}{\Longrightarrow}$ we know that there exist P', M' such that

$$I \triangleright P \triangleleft M \Longrightarrow P' \triangleleft M' \tag{39}$$

$$\mathbf{aftr}(I,\epsilon) = I \text{ and } I \triangleright P' \triangleleft M' \xrightarrow{\alpha} \tag{40}$$

By case analysis, only two rules could have been used Eq. (40):

iMon: This means that $I \triangleright P' \xrightarrow{\alpha}$ and, more importantly,

$$M' \xrightarrow{\alpha} \tag{41}$$

Now from Eq. (39) and Lem. 3 (unzipping) we have two further subcases:

1. Either $M \Longrightarrow M'$, which, by Eq. (41), would allow us to derive $M \stackrel{\alpha}{\Longrightarrow}$ (thus contradicting $M \stackrel{\alpha}{\not\Longrightarrow}$).

2. Or $M \Longrightarrow M'' \stackrel{\tau}{\not\longrightarrow}$ for some M'' where $M' = \mathsf{end}$. In particular, $M \Longrightarrow M'' \stackrel{\tau}{\not\longrightarrow}$ would contradict $M \Uparrow$.

iTrm: This means that $I \triangleright P' \stackrel{\alpha}{\longrightarrow}$ and, more importantly, $M' \stackrel{\tau}{\not\longrightarrow}$ and $M' \stackrel{\alpha}{\not\longrightarrow}$. Similar to the previous case, we use Eq. (39) and Lem. 3 to get two further subcases, but in either case we always end up contradicting $M \Uparrow$. □

Lemma 19. $\big(I \triangleright P \stackrel{\alpha}{\longrightarrow} P'$ and $I \triangleright P \triangleleft M \stackrel{\alpha}{\not\Longrightarrow}\big)$ implies $\big(M \stackrel{\alpha}{\not\Longrightarrow}$ and $M \Uparrow\big)$

Proof. First, we obtain $M \stackrel{\alpha}{\not\Longrightarrow}$ by contradiction: if we assume $M \stackrel{\alpha}{\Longrightarrow}$, then by $I \triangleright P \stackrel{\alpha}{\longrightarrow} P'$ and Lem. 4 (zipping) we would obtain $I \triangleright P \triangleleft M \stackrel{\alpha}{\Longrightarrow}$ which contradicts our initial assumption.

Second, we need to show $M \Uparrow$; we do this by also assuming $M \stackrel{\alpha}{\not\Longrightarrow}$ as part of our assumptions (we just derived it above). Again, we proceed by contradiction. We assume $M \not\Uparrow$ and then show that this would imply the contradicting conclusion $I \triangleright P \triangleleft M \stackrel{\alpha}{\Longrightarrow}$. If $M \not\Uparrow$ holds, this means that there exist a number of finite transition sequences from M and, moreover, the maximum number of τ-transitions from M can be capped by some integer; we denote this (positive) integer as $\mathsf{maxtau}(M)$. We proceed by strong numerical induction on $\mathsf{maxtau}(M)$ to show that for *all* of these τ-transition sequences, it is always the case that we obtain the contradicting conclusion that $I \triangleright P \triangleleft M \stackrel{\alpha}{\Longrightarrow}$; hence our assumption $M \not\Uparrow$ must be false.

$\mathsf{maxtau}(M) = 0$: We have $M \stackrel{\tau}{\not\longrightarrow}$. By assumption, we also have $M \stackrel{\alpha}{\not\longrightarrow}$ and $I \triangleright P \stackrel{\alpha}{\longrightarrow} P'$. Hence by rule iTrm we deduce $I \triangleright P \triangleleft M \stackrel{\alpha}{\longrightarrow} P' \triangleleft \mathsf{end}$.

$\mathsf{maxtau}(M) = k+1$: For all M' such that $M \stackrel{\tau}{\longrightarrow} M'$, we have $\mathsf{maxtau}(M') < \mathsf{maxtau}(M)$. Thus by I.H. we obtain $I \triangleright P \triangleleft M' \stackrel{\alpha}{\Longrightarrow}$, from which we can then derive $I \triangleright P \triangleleft M \stackrel{\alpha}{\Longrightarrow}$ using $M \stackrel{\tau}{\longrightarrow} M'$ and rule iAsyM. □

Lemma 20 (Non-transparency and Divergence refusal)**.**

$$\neg\mathsf{atr}(M, I, P, s, \alpha) \text{ iff } \begin{cases} \exists Q \cdot I \triangleright P \stackrel{s}{\Longrightarrow} Q \text{ and } \mathbf{aftr}(I, s) \triangleright Q \stackrel{\alpha}{\Longrightarrow} \\ \text{and } \exists N \cdot M \stackrel{s}{\Longrightarrow} N \text{ and } N \Uparrow \text{ and } N \stackrel{\alpha}{\not\Longrightarrow} \end{cases}$$

Proof. The proof is split into two parts.

***If*:** By $I \triangleright P \stackrel{s}{\Longrightarrow} Q$, $M \stackrel{s}{\Longrightarrow} N$ and Lem. 4 (zipping) we obtain

$$I \triangleright P \triangleleft M \stackrel{s}{\Longrightarrow} Q \triangleleft N \tag{42}$$

By $\mathbf{aftr}(I, s) \triangleright Q \stackrel{\alpha}{\Longrightarrow}$, $N \Uparrow$, $N \stackrel{\alpha}{\not\Longrightarrow}$ and Lem. 18 we obtain

$$I \triangleright Q \triangleleft N \stackrel{\alpha}{\not\Longrightarrow} \tag{43}$$

From Eqs. (42) and (43) and $\mathbf{aftr}(I, s) \triangleright Q \stackrel{\alpha}{\Longrightarrow}$ we get $\neg\mathsf{atr}(M, I, P, s, \alpha)$.

***Only-If*:** By Def. 18 $\neg\mathsf{atr}(M, I, P, s, \alpha)$ means that there exists Q and N where

$$I \triangleright P \triangleleft M \stackrel{s}{\Longrightarrow} Q \triangleleft N \tag{44}$$

$$\mathbf{aftr}(I, s) \triangleright (Q \triangleleft N) \stackrel{\alpha}{\not\Longrightarrow} \tag{45}$$

$$\mathbf{aftr}(I, s) \triangleright Q \stackrel{\alpha}{\Longrightarrow} \tag{46}$$

From Eqs. (45) and (46) and the contrapositive of Prop. 2 we know that $N \neq \mathsf{end}$. Thus by Eq. (44) and Lem. 3 we know that

$$I \triangleright P \stackrel{s}{\Longrightarrow} Q \tag{47}$$

$$M \stackrel{s}{\Longrightarrow} N \tag{48}$$

Eq. (46) means that there exists some Q' such that $\mathbf{aftr}(I, s) \triangleright Q \Longrightarrow Q' \xrightarrow{\alpha}$ and by Eq. (45), it must also be the case that $\mathbf{aftr}(I, s) \triangleright (Q' \triangleleft N) \stackrel{\alpha}{\not\Longrightarrow}$. Thus, by the technical lemma Lem. 19, we obtain

$$N \stackrel{\alpha}{\not\Longrightarrow} \text{ and } N \Uparrow \tag{49}$$

The required result follows from Eqs. (46) to (49). □

Lem. 20 allows us to prove a number of derived lemmata that will then be used towards proving Thms. 3 and 4.

Lemma 21. $\neg\mathsf{atr}(M, P, I, s, \alpha)$ implies $\neg\mathsf{atr}(M, \mathbf{prc}(s\alpha), \mathbf{nm}(s\alpha), s, \alpha)$

Proof. Assume $\neg\mathsf{atr}(M, P, I, s, \alpha)$. From Lem. 20 we know:

$$\begin{aligned} &\exists Q \cdot I \triangleright P \stackrel{s}{\Longrightarrow} Q \text{ and } \mathbf{aftr}(I, s) \triangleright Q \stackrel{\alpha}{\Longrightarrow} \\ &\exists N \cdot M \stackrel{s}{\Longrightarrow} N \text{ and } N \Uparrow \text{ and } N \stackrel{\alpha}{\not\Longrightarrow} \end{aligned} \tag{50}$$

By Lem. 7 we obtain

$$\mathbf{nm}(s\alpha) \triangleright \mathbf{prc}(s\alpha) \stackrel{s}{\Longrightarrow} \mathbf{prc}(\alpha) \stackrel{\alpha}{\Longrightarrow} \mathsf{nil} \tag{51}$$

By Eqs. (50) and (51) and Lem. 20 we get $\neg\mathsf{atr}(M, \mathbf{prc}(s\alpha), \mathbf{nm}(s\alpha), s, \alpha)$ □

Lemma 22. $(\neg\mathsf{atr}(M, \mathbf{prc}(s\alpha), \mathbf{nm}(s\alpha), s, \alpha)$ and $I \triangleright P \stackrel{s\alpha}{\Longrightarrow})$ implies $\neg\mathsf{atr}(M, P, I, s, \alpha)$

Proof. Analogous to Lem. 21. □

The following result, Lem. 23, establishes a correspondence between the contextual preorder predicate of Def. 5 and that of Def. 18.

Lemma 23. $\neg\mathsf{tr}(M, \mathbf{prc}(s\alpha), \mathbf{nm}(s\alpha), s)$ implies $\neg\mathsf{atr}(M, \mathbf{prc}(s\alpha), \mathbf{nm}(s\alpha), s, \alpha)$

Proof. From Def. 5 and Def. 18 we know that $\exists\beta.\neg\mathsf{atr}(M, \mathbf{prc}(s\alpha), \mathbf{nm}(s\alpha), s, \beta)$ and subsequently, by Lem. 20, we obtain:

$$\exists\beta, Q \cdot \mathbf{nm}(s\alpha) \triangleright \mathbf{prc}(s\alpha) \stackrel{s}{\Longrightarrow} Q \text{ and } \mathbf{aftr}(\mathbf{nm}(s\alpha), s) \triangleright Q \stackrel{\beta}{\Longrightarrow} \tag{52}$$

$$\exists N \cdot M \stackrel{s}{\Longrightarrow} N \text{ and } N \Uparrow \text{ and } N \stackrel{\beta}{\not\Longrightarrow} \tag{53}$$

By Eq. (52) and Lem. 7 we know that $Q = \mathbf{prc}(\alpha)$ and again by Lem. 7 we know that $\beta = \alpha$. We thus use Eqs. (52) and (53), $\beta = \alpha$ and Lem. 20 to obtain the required result, namely $\neg\mathsf{atr}(M, \mathbf{prc}(s\alpha), \mathbf{nm}(s\alpha), s, \alpha)$. □

We are now in a position to prove the two main results of the section. Thm. 3 establishes a bi-directional correspondence between the transparency preorder of Def. 5 and the one defined in Def. 18. We then prove soundness and completeness for the alternative transparency preorder of Def. 17 in terms of the preorder in Def. 18. This turns out to be easier to do because the preorder in Def. 18 is more specific which allows us to define the characteristic driving system and violating trace.

Theorem 3 (Transparency Preorder Equivalence)**.** $M \sqsubseteq_{\mathsf{tr}} N$ iff $M \sqsubseteq_{\mathsf{atr}} N$

Proof. The proof is split into two parts for the *if* and *only-if* implications; the *only-if* case is less straightforward because it is more specific.

***If*:** Assume $M \sqsubseteq_{\mathsf{atr}} N$ and pick some I, P and s such that $\neg\mathsf{tr}(N, P, I, s)$. By the contrapositive of the implication in Def. 5, we need to show that $\neg\mathsf{tr}(M, P, I, s)$. From $\neg\mathsf{tr}(N, P, I, s)$, Def. 5 and Def. 18 we know that $\exists\alpha \cdot \neg\mathsf{atr}(N, P, I, s, \alpha)$. By $M \sqsubseteq_{\mathsf{atr}} N$, Def. 18 and the contrapositive we obtain $\neg\mathsf{atr}(M, P, I, s, \alpha)$ which, by Def. 5, implies $\neg\mathsf{tr}(M, P, I, s)$.

***Only-If*:** Assume $M \sqsubseteq_{\mathsf{tr}} N$ and $\neg\mathsf{atr}(N, P, I, s, \alpha)$ for an arbitrary I, P, s and action α. By the contrapositive and Def. 18, we need to show that $\neg\mathsf{atr}(M, P, I, s, \alpha)$. From $\neg\mathsf{atr}(N, P, I, s, \alpha)$ and Lem. 20 we know that

$$P \stackrel{s\alpha}{\Longrightarrow} \tag{54}$$

and by Lem. 21 we subsequently obtain

$$\neg\mathsf{atr}(N, \mathbf{prc}(s\alpha), \mathbf{nm}(s\alpha), s, \alpha) \tag{55}$$

for the characteristic process defined in Defs. 9 and 10. From Eq. (55), we know $\neg\mathsf{tr}(N, \mathbf{prc}(s\alpha), \mathbf{nm}(s\alpha), s)$ and by $M \sqsubseteq_{\mathsf{tr}} N$ and the contrapositive of Def. 5 we obtain $\neg\mathsf{tr}(M, \mathbf{prc}(s\alpha), \mathbf{nm}(s\alpha), s)$. By Lem. 23, we derive the stronger predicate $\neg\mathsf{atr}(M, \mathbf{prc}(s\alpha), \mathbf{nm}(s\alpha), s, \alpha)$ and by Lem. 20 we obtain

$$\exists M' \cdot M \stackrel{s}{\Longrightarrow} M' \text{ and } M' \Uparrow \text{ and } M' \stackrel{\alpha}{\not\Longrightarrow}. \tag{56}$$

Hence, by Eqs. (54) and (56) and Lem. 20 we obtain $\neg\mathsf{atr}(M, P, I, s, \alpha)$ as required. □

As mentioned earlier we prove the soundness and completeness lemmata for $\preceq_{\mathsf{tr}}$ of Def. 17 with respect to $\sqsubseteq_{\mathsf{atr}}$ of Def. 18. The required result thus follows by transitivity as a result of Thm. 3.

Lemma 24 (Soundness)**.** $M \preceq_{\mathsf{tr}} N$ implies $M \sqsubseteq_{\mathsf{atr}} N$

Proof. Assume $M \preceq_{\mathsf{tr}} N$. By the contrapositive of Def. 18, pick an I,P,s and α such that $\neg\mathsf{atr}(N, I, P, s, \alpha)$. We need to show that $\neg\mathsf{atr}(M, I, P, s, \alpha)$ holds. By $\neg\mathsf{atr}(N, I, P, s, \alpha)$, Lem. 20 and Def. 16 we have:

$$I \triangleright P \stackrel{s}{\Longrightarrow} Q \text{ for some } Q \text{ and } \mathbf{aftr}(I, s) \triangleright Q \stackrel{\alpha}{\Longrightarrow} \tag{57}$$

$$\alpha \in \mathsf{dref}(N, s) \tag{58}$$

By Def. 17 we know that $\mathsf{dref}(N, s) \subseteq \mathsf{dref}(M, s)$; by Eq. (58), this means that:

$$\alpha \in \mathsf{dref}(M, s) \tag{59}$$

Thus, by Eqs. (57) and (59), Def. 16, and Lem. 20 we deduce $\neg\mathsf{atr}(M, I, P, s, \alpha)$ as required. □

Lemma 25 (Completeness)**.** $M \sqsubseteq_{\mathsf{atr}} N$ implies $M \preceq_{\mathsf{tr}} N$

Proof. Assume $M \sqsubseteq_{\mathsf{atr}} N$. By Def. 17, to prove the required conclusion, we have to show that $\mathsf{dref}(N, s) \subseteq \mathsf{dref}(M, s)$. Pick an arbitrary $\alpha \in \mathsf{dref}(N, s)$. By Def. 16, $\alpha \in \mathsf{dref}(N, s)$ means that

$$\exists N' \cdot N \stackrel{s}{\Longrightarrow} N' \text{ and } N' \Uparrow \text{ and } N' \stackrel{\alpha}{\not\Longrightarrow} \tag{60}$$

By Lems. 7 and 20 and Eq. (60) we can deduce $\neg\mathsf{atr}(N, \mathbf{nm}(s\alpha), \mathbf{prc}(s\alpha), s, \alpha)$. Thus, by the assumption $M \sqsubseteq_{\mathsf{atr}} N$, we obtain $\neg\mathsf{tr}(M, \mathbf{nm}(s\alpha), \mathbf{prc}(s\alpha), s, \alpha)$. Again, by Lem. 20 this means that

$$\exists M' \cdot M \stackrel{s}{\Longrightarrow} M' \text{ and } M' \Uparrow \text{ and } M' \stackrel{\alpha}{\not\Longrightarrow}$$

which, by Def. 16, implies $\alpha \in \mathsf{dref}(M, s)$ as required. □

Theorem 4 (Transparency Preorders)**.** $M \sqsubseteq_{\mathsf{tr}} N$ iff $M \preceq_{\mathsf{tr}} N$

Proof. The result follows from Lems. 24 and 25 and Thm. 3 and transitivity. □

We can revisit the earlier examples and assess them in terms of the alternative transparency preorder $\preceq_{\mathsf{tr}}$ of Def. 17, which gives us a better sense why this preorder is easier to use and automate.

Example 17. Recall monitors $c!a.\mathsf{end}$ and $c!a.\mathsf{end} + c!a.\Omega$ from (20) of E.g. 11. The inequality $c!a.\mathsf{end} + c!a.\Omega \sqsubseteq_{\mathsf{tr}} c!a.\mathsf{end}$ follows immediately from Thm. 4, since for any trace s we have $\mathsf{dref}(c!a.\mathsf{end}, s) = \emptyset$, which means that the inclusion $\mathsf{dref}(c!a.\mathsf{end}, s) \subseteq \mathsf{dref}(c!a.\mathsf{end} + c!a.\Omega, s)$ holds trivially.

The symmetric case, $c!a.\mathsf{end} \sqsubseteq_{\mathsf{tr}} c!a.\mathsf{end} + c!a.\Omega$, can also be readily repudiated by applying Thm. 4. For instance, $\mathsf{dref}((c!a.\mathsf{end} + c!a.\Omega), c!a.\epsilon) = \textsc{Act}$ (and $\mathsf{dref}(c!a.\mathsf{end}, c!a.\epsilon) = \emptyset$) we trivially obtain a violation of the set inclusion requirements of Def. 17. This means that $c!a.\mathsf{end} \not\preceq_{\mathsf{tr}} c!a.\mathsf{end} + c!a.\Omega$ which, in turn, implies that $c!a.\mathsf{end} \not\sqsubseteq_{\mathsf{tr}} c!a.\mathsf{end} + c!a.\Omega$. ■

Example 18. Recall the two pathological monitors $\Omega + \checkmark$ and $\mathsf{rec}\, X.(\tau.X + \checkmark)$ from Eq. (17). In E.g. 10 we argued that they are deterministic-detection equivalent, *i.e.*, $\Omega + \checkmark \;\cong_{\mathsf{dd}}\; \mathsf{rec}\, X.(\tau.X + \checkmark)$. It is not hard to see that they are also potential-detection equivalent as well,*i.e.*, $\Omega + \checkmark \;\cong_{\mathsf{pd}}\; \mathsf{rec}\, X.(\tau.X + \checkmark)$. However, at first glance, it is hard to tell how these monitors relate in terms of the transparency preorder of Def. 5. One of the reasons complicating this analysis is that both monitors produce diverging computations when composed with any arbitrary system, even though these diverging computations impact transparency differently. Arguably, we can differentiate between these monitors from a transparency perspective more easily using $\preceq_{\mathsf{tr}}$ of Def. 17 because we only need to consider the respective monitor LTSs which are also finite state (in this case, at least).

More concretely, we can readily calculate that $\mathsf{dref}((\Omega + \checkmark), \epsilon) = \textsc{Act}$ since $\Omega + \checkmark \xrightarrow{\tau} \Omega$ and $\mathsf{dref}(\Omega, \epsilon) = \textsc{Act}$. By contrast, we can deduce that $\mathsf{dref}((\mathsf{rec}\, X.(\tau.X + \checkmark)), \epsilon) = \emptyset$. In fact, it is not hard to see that whenever $\mathsf{rec}\, X.(\tau.X + \checkmark) \stackrel{\epsilon}{\Longrightarrow} M$, we either have $M = \tau.\big(\mathsf{rec}\, X.(\tau.X + \checkmark)\big) + \checkmark$ or $M = \mathsf{rec}\, X.(\tau.X + \checkmark)$; in either case, for any $\alpha \in \textsc{Act}$, we can show that $M \stackrel{\alpha}{\Longrightarrow}$ since $M \stackrel{\epsilon}{\Longrightarrow} \checkmark$ and $\checkmark \xrightarrow{\alpha} \checkmark$ by mVer of Fig. 2. For all other traces s where $|s| \geq 1$ (*i.e.*, $s = \alpha t$ for some α and t) we obtain empty divergence refusal sets for both monitors since we have $\Omega + \checkmark \stackrel{\alpha}{\Longrightarrow} \checkmark$ and $\mathsf{rec}\, X.(\tau.X + \checkmark) \stackrel{\alpha}{\Longrightarrow} \checkmark$ for any $\alpha \in \textsc{Act}$. We thus can positively conclude that $\Omega + \checkmark \sqsubseteq_{\mathsf{tr}} \mathsf{rec}\, X.(\tau.X + \checkmark)$ while refuting $\mathsf{rec}\, X.(\tau.X + \checkmark) \sqsubseteq_{\mathsf{tr}} \Omega + \checkmark$. ■

Equipped with the three alternative preorders of Defs. 8, 13 and 17, we are now in a position to provide an alternative complete proof method for determining the strongest monitor preorder presented in this article, *i.e.*, the contextual preorder $\sqsubseteq$ of Def. 6. This is given as the alternative preorder $\preceq$ in Def. 19, and is shown to be fully abstract to $\sqsubseteq$ in Thm. 5.

Definition 19 (Alternative Monitor Preorder)**.**

$$M \preceq N \quad \stackrel{\mathbf{def}}{=} \quad M \preceq_{\mathsf{pd}} N \text{ and } M \preceq_{\mathsf{dd}} N \text{ and } M \preceq_{\mathsf{tr}} N$$

■

Theorem 5 (Full Abstraction)**.** $M \sqsubseteq N \quad \text{iff} \quad M \preceq N$

Proof. Follows immediately as a result of Thms. 1, 2 and 4. □

8. Possible Applications

Our preorders and their characterisations can be used in a number of settings. One obvious application would be that of ensuring the correct translation from monitors described in the modelling language of Fig. 2 to corresponding monitor implementations in an actual programming language. This would however require a formal operational semantics for the target programming language, which can be quite substantial, is often unavailable, and is certainly

Logic Syntax

$$\varphi, \phi \in \text{sHML} ::= \quad \mathsf{tt} \quad | \ \mathsf{ff} \quad | \ \varphi \wedge \phi \quad | \ [p, g]\varphi \quad | \ \mathsf{max}\, X.\varphi \quad | \ X$$

Logic Semantics

$$[\![\mathsf{tt}]\!] \stackrel{\text{def}}{=} \text{Sys} \qquad [\![\mathsf{ff}]\!] \stackrel{\text{def}}{=} \emptyset \qquad [\![\varphi \wedge \phi]\!] \stackrel{\text{def}}{=} [\![\varphi]\!] \cap [\![\phi]\!]$$

$$[\![[p, g]\varphi]\!] \stackrel{\text{def}}{=} \left\{ I \triangleright P \;\middle|\; \left(\begin{array}{c} I \triangleright P \stackrel{\alpha}{\Longrightarrow} Q \\ \text{and } \mathsf{match}(p, \alpha) = \sigma \\ \text{and } g\sigma \Downarrow \mathsf{true} \end{array} \right) \text{ implies } \mathbf{aftr}(I, \alpha) \triangleright Q \in [\![\varphi\sigma]\!] \right\}$$

$$[\![\mathsf{max}\, X.\varphi]\!] \stackrel{\text{def}}{=} \bigcup \{ S \mid S \subseteq [\![\varphi[^{X}/_{S}]]\!] \}$$

Monitor Synthesis

$$\langle\!\langle \mathsf{tt} \rangle\!\rangle \stackrel{\text{def}}{=} \mathsf{end} \qquad \langle\!\langle \mathsf{ff} \rangle\!\rangle \stackrel{\text{def}}{=} \checkmark \qquad \langle\!\langle \varphi \wedge \phi \rangle\!\rangle \stackrel{\text{def}}{=} \langle\!\langle \varphi \rangle\!\rangle + \langle\!\langle \phi \rangle\!\rangle$$

$$\langle\!\langle [p, g]\varphi \rangle\!\rangle \stackrel{\text{def}}{=} \begin{cases} p.\langle\!\langle \varphi \rangle\!\rangle & \text{if } g = \mathsf{true} \\ p.\mathsf{if}\ g\ \mathsf{then}\ \langle\!\langle \varphi \rangle\!\rangle\ \mathsf{else\ end} & \text{otherwise} \end{cases}$$

$$\langle\!\langle \mathsf{max}\, X.\varphi \rangle\!\rangle \stackrel{\text{def}}{=} \mathsf{rec}\, X.\langle\!\langle \varphi \rangle\!\rangle \qquad \langle\!\langle X \rangle\!\rangle \stackrel{\text{def}}{=} X$$

Figure 3: The syntax and semantics of sHML.

beyond the scope of this work. Even when this operational semantics is available, one would also need to ensure that the instrumentation mechanism used coincides with that of Fig. 2 before the characterisations of Defs. 8, 13 and 17 can be used in their present form; see the related work discussion in Sec. 9.

In this section, we showcase the utility of our preorders through a case study involving an automated synthesis procedure originally proposed for a body of work studying the monitorability of program specifications. The study was conducted for a variant of the highly expressive modal μ-calculus called μHML [66, 28, 25, 29, 30]: the study identifies a maximally-expressive[5] logical fragment for runtime violation detections called sHML (safety μHML), and the automated synthesis works on this logical fragment. The synthesis procedure has also been implemented as a tool called detectEr, and has been used to runtime verify concurrent systems written in Erlang [67, 68, 69, 44, 70]. Conveniently, the implementation of detectEr is based on a model that uses the same instrumentation relation formalised in Fig. 2.

Fig. 3 restates the syntax of sHML and adapts it to systems defined in terms of the piCalculus variant of Sec. 2. It assumes a countable set of logical variables $X, Y \in \text{LVar}$, that are bound by recursive formulae of the form $\mathsf{max}\, X.\varphi$, expressing largest fixpoints. In addition to the standard constructs for truth, tt, falsehood, ff, and conjunction, $\varphi \wedge \phi$, the logic includes the universal modality

[5]This means that every property that can be, in some sense, adequately monitored for violations, can also be expressed in the logical fragment.

construct, $[p, g]\varphi$, where the action patterns p are matched to concrete actions using the function $\mathsf{match}(p,\alpha)$ from Sec. 3. Action matching dynamically maps variables to names, which are then used to evaluate *pattern guards* $g \in \text{PGRD}$; these take the following form[6] for our target piCalculus systems (recall that $u, v \in (\text{CHANS} \cup \text{VARS})$):

$$\text{PGRD} ::= \mathsf{true} \mid u = v \mid u \neq v$$

Closed pattern guards (*i.e.,* without free variables) have the expected semantics:

$$\mathsf{true} \Downarrow \mathsf{true} \qquad c = c \Downarrow \mathsf{true} \qquad c \neq d \Downarrow \mathsf{true} \quad (\text{whenever } c \sharp d)$$

Closed formulae $\varphi \in$ sHML are interpreted over systems, $I \triangleright P \in \text{SYS}$ as $[\![\varphi]\!]$, defined inductively on the structure of φ in Fig. 3:

$$\text{SYS} \stackrel{\text{def}}{=} \{I \triangleright P \mid I \subset \text{CHANS}, P \in \text{PROC}, \mathbf{fv}(P) = \mathbf{fV}(P) = \emptyset, \mathbf{fn}(P) \subseteq I\}\,.$$

Formula tt is satisfied by all systems, ff is satisfied by none, whereas conjunctions bear the standard set-theoretic meaning. The universal modality formulae $[p, g]\varphi$ state that *whenever* a satisfying system produces an action α matching the pattern p yielding σ, and the instantiated guard $g\sigma$ holds, the resultant system that is transitioned to must satisfy $\varphi\sigma$ (*i.e.,* φ instantiated with the bindings in σ). To capture maximal fixpoints, the recursive formula $\mathsf{max}\, X.\varphi$ is defined as the union of all the post-fixpoint solutions $S \subseteq \text{SYS}$ of φ; see [71] for more details.

Fig. 3 also presents the synthesis function, $\langle\!\langle - \rangle\!\rangle$, from sHML formulae to monitors given in [66, 25, 28, 29], but adapted to our monitors. The synthesis assumes a one-to-one mapping between logical variables, LVAR, and process variables, PVARS; for convenience, LVAR and PVARS are assumed to coincide. A key insight to understand the synthesis for a formula φ is that the monitor generated detects execution traces pertaining systems belonging to the *dual* of the property, *i.e.,* $\text{SYS} \setminus [\![\varphi]\!]$. Thus, ff translates to a detection, $\checkmark$, whereas monitor for a conjunction $\varphi \wedge \phi$ should behave as *either* of the monitors for the constituent subformulae, $\langle\!\langle\varphi\rangle\!\rangle + \langle\!\langle\phi\rangle\!\rangle$. Analogously, although maximal fixpoints can express infinite behaviour, the synthesised monitor need only detect a violating behaviour of finite length. A core part of the synthesis function deals with universal modality formulae, $\langle\!\langle[p, g]\varphi\rangle\!\rangle$. Intuitively, it translate to a pattern-match monitor prefix, $p...$, followed by monitor branch on g that proceeds with the respective monitoring of the guarded subformula φ, *i.e.,* $\mathsf{if}\ g\ \mathsf{then}\ \langle\!\langle\varphi\rangle\!\rangle\ \mathsf{else}\ \mathsf{end}$. In order to keep the monitor overheads low, the synthesis presented in Fig. 3 short-circuits the branching condition whenever it holds trivially, *i.e.,* when g is true. The synthesis function is inherently compositional, and this aspect facilitated any correctness analysis conducted in prior work where the synthesis was first devised (*e.g.,* [25, 29, 67]).

[6] Even though the guard true can be easily encoded as the guard $c = c$ (for some default channel name c), we use true to give a refined monitor synthesis; see Fig. 3.

The preorders developed in this paper can help us analyse the monitors produced by the synthesis in Fig. 3. For instance, consider the behavioural analysis of a system that interacts on two channels named in and out. The sHML variant presented in Fig. 3 allows us to specify that *"a system cannot (immediately) produce outputs on channel* out*"* in at least two different ways:

$$\varphi_1 = [\mathsf{out}!(y), \mathsf{true}]\mathsf{ff} \qquad \text{and} \qquad \varphi_2 = [(x)!(y), x = \mathsf{out}]\mathsf{ff}$$

It is not hard to see from the semantics of Fig. 3 that φ_1 and φ_2 are semantically equivalent. However, our synthesis function yields slightly different monitors for the two formulae:

$$\langle\!\langle \varphi_1 \rangle\!\rangle = m_1 = \mathsf{out}!(y).\checkmark \qquad \langle\!\langle \varphi_2 \rangle\!\rangle = m_2 = (x)!(y).\mathsf{if}\ x = \mathsf{out}\ \mathsf{then}\ \checkmark\ \mathsf{else}\ \mathsf{end}$$

These monitors are not only syntactically different, but may also exhibit different runtime behaviour when composed with an arbitrary system. Using the preorder characterisations of Secs. 5 and 6, we can determined that the two monitors are potential-detection equivalent (for any system), $m_1 \cong_{\mathsf{pd}} m_2$, by employing Thm. 1 and simply analysing the LTS of the respective monitor descriptions, $m_1 \simeq_{\mathsf{pd}} m_2$. But we can also determined that they are *not* deterministic-detection equivalent, $m_1 \ncong_{\mathsf{pd}} m_2$, by analysing their respective LTS semantics (and not considering any system instrumentation). This follows from Thm. 2 since, by Def. 13, we can show that $m_1 \not\preceq_{\mathsf{dd}} m_2$ because $\mathsf{nd}(m_2, \mathsf{out}!c)$ holds for some arbitrary channel c, but we can neither show $\mathsf{nd}(m_1, \mathsf{out}!c)$ nor $\mathsf{blk}(m_1, \mathsf{out}!c)$.

As a secondary example of the utility of our preorders, consider the specification requiring *"systems not to input (on channel* in*) and consecutively output (on channel* out*) with different payloads, and also not to input twice in succession (on channel* in*)"*. This can be expressed in our adapted sHML as

$$\varphi_3 = [\mathsf{in}?(x), \mathsf{true}][\mathsf{out}!(y), y \neq x]\mathsf{ff} \wedge [\mathsf{in}?(x), \mathsf{true}][\mathsf{in}?(y), \mathsf{true}]\mathsf{ff}$$

and from φ_3 we can automatically synthesise the monitor

$$\langle\!\langle \varphi_3 \rangle\!\rangle = m_3 = \begin{cases} \mathsf{in}?(x).\mathsf{out}!(y).\mathsf{if}\ y \neq x\ \mathsf{then}\ \checkmark\ \mathsf{else}\ \mathsf{end} \\ \quad + \mathsf{in}?(x).\mathsf{in}?(y).\checkmark \end{cases}$$

For the sake of our example, assume that detecting successive inputs on channel in (violating the subformula $[\mathsf{in}?(x), \mathsf{true}][\mathsf{in}?(y), \mathsf{true}]\mathsf{ff}$) is critical. Therefore, we would required that monitor m_3 detects this violating behaviour *both* potentially *and* deterministically (we are less concerned about the other detections carried out by m_3). One way to enforce this additional requirement is to describe the monitor specification

$$m_4 = \mathsf{in}?(x).\mathsf{in}?(y).\checkmark$$

which clearly detects consecutive inputs on channel in potentially *and* deterministically, and then require that m_3 is a refinement of m_4 following Def. 4:

$$m_4 \sqsubseteq_{\mathsf{pd}} m_3 \qquad \text{and} \qquad m_4 \sqsubseteq_{\mathsf{dd}} m_3$$

Although we can show that the first inequality holds, we can prove that the second inequality does not, $m_4 \not\sqsubseteq_{\mathsf{dd}} m_3$. In fact, since $\mathsf{blk}(m_3, s)$ for the trace $s = \mathsf{in}?c.\mathsf{in}?d$ with arbitrary names c and d, but neither $\mathsf{blk}(m_4, s)$ nor $\mathsf{fl}(m_4, s)$, we deduce $m_4 \not\preceq_{\mathsf{dd}} m_3$ and, by Thm. 2, $m_4 \not\sqsubseteq_{\mathsf{dd}} m_3$ follows.

We consider a third and final example showcasing the utility of our monitor preorders. Sampling (or event polling) has been frequently proposed as a technique for lowering the runtime overhead of monitors [72, 73, 74]. We can model event sampling in terms of our monitor language of Sec. 3, by using the code excerpt $\mathsf{rec}\, X.(\tau^n.X + p.m)$ in lieu of the monitor $p.m$ that blocks listening for events that match with pattern p; the alternative code excerpt introduces a periodic delay of some pre-determined n (internal) τ-steps between event listens. Thus, for the sHML formula $\varphi_1 = [\mathsf{out}!(y), \mathsf{true}]\mathsf{ff}$ discussed earlier, a modified monitor synthesis would instead generate the monitor

$$m_1' = \mathsf{rec}\, X.(\tau^n.X + \mathsf{out}!(y).\checkmark)$$

Although m_1' is an adequate monitor refinement for m_1 in terms of both potential and deterministic detections, *i.e.,* $m_1 \sqsubseteq_{\mathsf{pd}} m_1'$ and $m_1 \sqsubseteq_{\mathsf{dd}} m_1'$, it is *not* a valid refinement for transparency, $m_1 \not\sqsubseteq_{\mathsf{tr}} m_1'$. This can easily be shown via the alternative characterisation of Def. 17 following Thm. 4 since we can show that $\mathsf{in}?c \in \mathsf{dref}(m_1', \epsilon)$ for some arbitrary payload c whereas $\mathsf{dref}(m_1, \epsilon) = \emptyset$.

9. Conclusion

We have presented a theory for (recogniser) monitors based on refinement preorders. It allows us to substitute a monitor M_1 in a monitored process $P \triangleleft M_1$ by another monitor M_2 while guaranteeing the preservation of a number of monitoring properties relating to (behaviour) detection and monitor interference; see Defs. 4 and 5. We content that our theory can serve as a basis for defining rigorous notions of monitor correctness, as advocated by various other work such as [75, 51, 21, 22, 76, 63, 77, 78, 32]. A distinguishing feature of our approach is that any correctness definitions based on our preorders would automatically inherit the universal quantification over all (system) contexts: such a powerful criterion guarantees the preservation of the monitor correctness properties asserted, irrespective of where the monitor is employed. Universal quantification over contexts is useful in practice since limited access to the system under scrutiny is usually provided while the monitor is being constructed. Our approach also promotes a holistic approach to monitor correctness, as in the case of [27, 49]. We have shown, through various pathological examples such as E.g. 4, 5 and 8 to 11, how subtle interdependencies between systems and monitors arise, affecting the overall monitored computation. In Sec. 8, we argue that these are not just edge cases, but can be pertinent issues that engineers need to content with when developing their analysis tools. The examples discussed in this paper also make a strong case that a proper definition of monitor correctness needs to take into consideration system instrumentation.

Despite these stronger guarantees, our monitor theory also provides mechanisms to substantially alleviate the effort of proving monitors correct. More

concretely, as a result of the alternative preorders developed, the theory is also *compositional*, since it enables us to ensure the preservation of properties from say $P \triangleleft M_1$ to $P \triangleleft M_2$ by analysing the *resp.* monitors M_1 and M_2 *in isolation*, without the need to consider the process being monitored, P. In principle, processes may be arbitrarily complex; in fact, they are usually far more complex than the monitors that analyse them. It is thus reasonable to expect that such compositional techniques are effective and yield considerable efficiency gains when assessing monitor correctness.

9.1. Related and Future Work.

The instrumentation relation considered here has been used extensively in other studies that investigate various foundational aspects of runtime monitoring, from monitorability [25, 30, 63, 32, 79] to monitor deterministic behaviour [27, 26, 49, 31]. This composition relation embodies *synchronous* instrumentation, where the external actions naturally constitute the monitorable actions. Variants of this relation have also been studied: some consider the instrumentation of silent actions as monitorable or opaque[7] [28] whereas others extend the visibility to aspects of the system under scrutiny such as action refusals [29]. The variant considered here constitutes a mild extension to the instrumentation relation used in previous work such as [25, 27, 30]: in this work, monitors may learn *new* channel names (that are scope extruded) at runtime and subsequently extend their analysis to any actions occurring on these channels. This mechanism extends what usually happens in runtime verification settings, where system instrumentation typically listens for specific actions/events from a fixed set (*e.g.,* method names within a specific codebase of classes); these events are statically linked to the monitor prior execution via mechanisms such as aspect orientation(*e.g.,* [80, 81, 82, 83]). However, there exist more *dynamic* instrumentation approaches that can handle the generality described by our setting (*e.g.,* [84, 85, 86, 87, 88]) and these have already been used in the context of runtime monitoring (*e.g.,* [67, 89, 90]).

Synchronous instrumentation is the most prevalent method used in monitoring tools (*e.g.,* [91, 60, 92, 57]) because it carries benefits such as timely detections. It can however lead to high runtime overheads and there are variants such as asynchronous instrumentation (*e.g.,* [93, 51, 94, 67]) or hybrid variations that mix synchornous and asynchronous instrumentation (*e.g.,* [60, 62, 6, 9, 95, 96]). Our theory should be applicable, at least in part, to these instrumentation variants. Monitor instrumentation can also be either inlined [97] within the system code or outlined and kept as a separate unit of code. Our techniques clearly address outline monitors, which are used in various settings (*e.g.,* [23, 55, 5]). Despite the advantages that inlined monitors carry (*e.g.,* they yield lower overheads and are generally more expressive because monitors have full access of the system code) they introduce further dependencies with the system under

[7] A sequence of silent actions can be opaquely observed by the monitor when their occurrence can be observed but the precise number of τ-transitions cannot be determined.

scrutiny and it is substantially harder to attain compositional techniques for them.

In runtime verification, multi-verdict monitors [36, 11, 64, 98, 38] are often considered, where detections are partitioned (or refined) into verdicts such as (maybe)acceptances and (maybe)rejections. The monitors studied here express generic (*i.e.,* uni-verdict) detections only. They are nevertheless maximally expressive for branching-time properties [24, 25] and have recently been shown to also be maximally expressive for certain classes of linear-time properties [30, 63]. The uni-verdict monitors expressed in this work also facilitate comparisons with other linear-time behavioural preorders (see next paragraph). We nevertheless expect our theory to extend smoothly to more general settings that include multi-verdicts such as acceptances and rejections.

Our potential and deterministic detection preorders are reminiscent of the classical may and must preorders of [45, 35] and, more recently (for the deterministic detection preorder), of the subcontract relations in [99, 100]. These relations differ from those presented in Def. 4 in a number of respects. For starters, the monitor instrumentation relation of Fig. 2 assigns monitors a *passive* role where a monitored system produces an external action only if the system produces that action. In contrast, the parallel composition relation composing processes (servers in [99, 100]) with tests (clients in [99, 100]) is more symmetric and invites tests to *interact* with the process being probed. Another important difference is that testing preorders typically relate processes (*e.g.,* see [45, 35]), whereas our preorders are defined over the adjudicating entities *i.e.,* the monitors.

The closest work in this regard is that of [99, 101], where the authors develop a must and compliance theory for clients (or tests) *i.e.,* processes with a special *"success"* action. Still, there are significant discrepancies between this theory and our deterministic detection preorder (further to the differencies between the detected (monitored) computations of Def. 2 and the successful computations under tests of [45, 35, 99] as outlined above) — success in the compliance relation of [100] is even more disparate. Concretely, in our setting we have equalities such as $c!a.\checkmark \cong_{\mathsf{dd}} c!a.\checkmark + b!a.\mathsf{end}$ (a slight variant on Eq. (13) of E.g. 8), but this equality would not hold in the setting of [99, 101] since their client preorder is sensitive to external choices. In fact, whereas $c!a.\checkmark$ would pass a must-test with the process $c?x.\mathsf{nil} + b?x.\mathsf{nil}$, the test equivalent of the monitor $c!a.\checkmark + b!a.\mathsf{end}$ would not, since we would have the following computations where one of them (see Eq. (62)) is unsuccessful:

$$c?x.\mathsf{nil} + b?x.\mathsf{nil} \parallel c!a.\checkmark + b!a.\mathsf{end} \xrightarrow{\tau} \mathsf{nil} \parallel \checkmark \tag{61}$$

$$c?x.\mathsf{nil} + b?x.\mathsf{nil} \parallel c!a.\checkmark + b!a.\mathsf{end} \xrightarrow{\tau} \mathsf{nil} \parallel \mathsf{end} \tag{62}$$

Note that, in the case of $\sqsubseteq_{\mathsf{dd}}$, the unsuccessful computation analogous to Eq. (62) would be distinguished from the one analogous to Eq. (61) by its visible trace

(the interface $I{=}\{c, b, \ldots\}$ is omitted for uniformity):

$$c!a.\mathsf{nil} + b!a.\mathsf{nil} \triangleleft c!a.\checkmark + b!a.\mathsf{end} \xrightarrow{c!a} \mathsf{nil} \triangleleft \checkmark$$
$$c!a.\mathsf{nil} + b!a.\mathsf{nil} \triangleleft c!a.\checkmark + b!a.\mathsf{end} \xrightarrow{b!a} \mathsf{nil} \triangleleft \mathsf{end}$$

The two relations are in fact incomparable, since divergent tests are bottom elements in the client must preorder of [99, 101], but they are not in $\sqsubseteq_{\mathsf{dd}}$. In fact, we have seen that $\Omega \not\sqsupseteq_{\mathsf{dd}} \Omega + \mathsf{end}$ in Eq. (17) of E.g. 10. Moreover, for any arbitrary external action α, we also have

$$\alpha.\checkmark \not\sqsupseteq_{\mathrm{mst}} \Omega \sqsubseteq_{\mathrm{mst}} \alpha.\checkmark$$

according to [99, 101] whereas in our setting we have the opposite:

$$\alpha.\checkmark \sqsubseteq_{\mathsf{dd}} \Omega \not\sqsupseteq_{\mathsf{dd}} \alpha.\checkmark .$$

At an intuitive level, this is because the instrumentation relation of Fig. 2 prioritises silent actions over external actions that cannot be matched by the monitor. The interested reader should consult [102] for a complementary discussion relating process testing with runtime monitoring.

Transparency is usually a concern for enforcement monitors whereby the visible behaviour of a monitored process should not be modified unless it violates some specified property [3, 23, 14]. We have adapted this concept to recognisers in Def. 5, to express different degrees to which the process behaviour is suppressed by the monitor. In [14], a slightly different notion of transparency was defined for a branching-time setting using transducers that are composed using a relation that is very similar to our instrumentation relation. It would certainly be worthwhile to adapt our transparency preorder to that setting.

There a number of studies that investigate monitoring for the piCalculus: in particular, most of them seem concerned with , and focusses on synthesising adaptation/enforcement monitors from session types [7, 8, 23, 5]. The closest to our work is that by Bocchi *et al.* [23] and Gommerstadt *et al.* [5]. The definitions of monitor correctness by Bocchi *et al.* [23] are distinctly different from ours since they they are based on branching-time (bisimulation) equivalences. They also employ compositional techniques as in our case, but their decomposition methods for decoupling the monitor analysis from that of processes rely on type information provided by the session types used. The correctness definitions of Gommerstadt *et al.* target partial-identity monitors that, in addition to analysing the behaviour of a system, are also tasked with acting as forwarders of messages to and from the system under scrutiny. They attain a compositional analysis for determining monitor correctness by devising a dedicated type system for these partial-identity monitors. However, since these entities exist in a linear type setting where channels are used once, many guarantees related to deterministic behaviour come for free.

The potential- and deterministic-detection monitor preorders of Defs. 4, 8 and 13 complement well recent work targetting the development of formal tools

for the analysis of (observationally) deterministic behaviour of monitors [27, 49]. In fact, for monitors that are deemed to be consistently-detecting/controllable [27], potential detection should imply deterministic detection. This investigation is left for future work. Should this be the case, this result could be exploited to further alleviate the burden of proving our contextual inequalities since the proofs for showing potential-detection preorders are less onerous than those for determining deterministic-detection preservation.

Acknowledgements.. The paper benefited from discussions with Luca Aceto, Giovanni Bernardi, Matthew Hennessy and Anna Ingólfsdóttir.

References

[1] A. Francalanza, A Theory of Monitors (Extended Abstract), in: B. Jacobs, C. Löding (Eds.), Foundations of Software Science and Computation Structures (FoSSaCS), Vol. 9634 of LNCS, 2016, pp. 145–161. doi:10.1007/978-3-662-49630-5_9.

[2] F. B. Schneider, Enforceable security policies, ACM Trans. Inf. Syst. Secur. 3 (1) (2000) 30–50. doi:10.1145/353323.353382.
URL http://doi.acm.org/10.1145/353323.353382

[3] J. Ligatti, L. Bauer, D. Walker, Edit automata: enforcement mechanisms for run-time security policies, Int. J. Inf. Secur. 4 (1-2) (2005) 2–16. doi:10.1007/s10207-004-0046-8.
URL http://dx.doi.org/10.1007/s10207-004-0046-8

[4] N. Bielova, F. Massacci, Do you really mean what you actually enforced?: Edited automata revisited, Int. J. Inf. Secur. 10 (4) (2011) 239–254. doi:10.1007/s10207-011-0137-2.
URL http://dx.doi.org/10.1007/s10207-011-0137-2

[5] H. Gommerstadt, L. Jia, F. Pfenning, Session-typed concurrent contracts, in: ESOP, Vol. 10801 of LNCS, Springer, 2018, pp. 771–798. doi:10.1007/978-3-319-89884-1_27.
URL https://doi.org/10.1007/978-3-319-89884-1_27

[6] I. Cassar, A. Francalanza, Runtime Adaptation for Actor Systems, in: RV, Vol. 9333 of LNCS, Springer, 2015, pp. 38–54.

[7] M. Coppo, M. Dezani-Ciancaglini, B. Venneri, Self-adaptive monitors for multiparty sessions, in: PDP, IEEE Computer Society, 2014, pp. 688–696.

[8] C. D. Giusto, J. A. Perez, Disciplined Structured Communications with Disciplined Runtime Adaptation, Sci. of Computer Programming 97 (2) (2015) 235–265. doi:http://dx.doi.org/10.1016/j.scico.2014.04.017.
URL http://www.sciencedirect.com/science/article/pii/S0167642314002512

[9] I. Cassar, A. Francalanza, On Implementing a Monitor-Oriented Programming Framework for Actor Systems, in: integrated Forma Methods (iFM), LNCS, Springer, 2016, pp. 176–192.

[10] A. Francalanza, C. A. Mezzina, E. Tuosto, Reversible choreographies via monitoring in erlang, in: Distributed Applications and Interoperable Systems - 18th IFIP WG 6.1 International Conference, DAIS 2018, Held as Part of the 13th International Federated Conference on Distributed Computing Techniques, DisCoTec 2018, Madrid, Spain, June 18-21, 2018, Proceedings, Vol. 10853 of LNCS, Springer, 2018, pp. 75–92. doi:10.1007/978-3-319-93767-0_6.
URL https://doi.org/10.1007/978-3-319-93767-0_6

[11] Y. Falcone, J.-C. Fernandez, L. Mounier, What can you verify and enforce at runtime?, STTT 14 (3) (2012) 349–382.

[12] E. Dolzhenko, J. Ligatti, S. Reddy, Modeling runtime enforcement with mandatory results automata, Int. J. Inf. Sec. 14 (1) (2015) 47–60. doi:10.1007/s10207-014-0239-8.
URL https://doi.org/10.1007/s10207-014-0239-8

[13] S. Pinisetty, V. Preoteasa, S. Tripakis, T. Jéron, Y. Falcone, H. Marchand, Predictive runtime enforcement, Formal Methods in System Design 51 (1) (2017) 154–199. doi:10.1007/s10703-017-0271-1.
URL https://doi.org/10.1007/s10703-017-0271-1

[14] L. Aceto, I. Cassar, A. Francalanza, A. Ingólfsdóttir, On runtime enforcement via suppressions, in: S. Schewe, L. Zhang (Eds.), 29th International Conference on Concurrency Theory (CONCUR 2018), LIPIcs, Schloss Dagstuhl, Dagstuhl, Germany, 2018, pp. 34:1–8:19.

[15] Formal Systems Laboratory, Monitor Oriented Programming, University of Illinois at Urbana Champaign, http://fsl.cs.illinois.edu/index.php/Monitoring-Oriented_Programming.

[16] F. Cesarini, S. Thompson, Erlang Programming, O'Reilly, 2009.

[17] A. Francalanza, M. Hennessy, A Theory for Observational Fault Tolerance, J. Log. Algebraic Methods Program. (JLAP) 73 (1–2) (2007) 22 – 50. doi:http://dx.doi.org/10.1016/j.jlap.2007.03.003.

[18] P. Verissimo, L. Rodrigues, Distributed Systems for System Architects, Kluwer Academic Publishers, 2001.

[19] I. Cassar, A. Francalanza, C. A. Mezzina, E. Tuosto, Reliability and fault-tolerance by choreographic design, in: Proceedings Second International Workshop on Pre- and Post-Deployment Verification Techniques, Pre-Post@iFM 2017, Torino, Italy, 19 September 2017, Vol. 254 of EPTCS,

2017, pp. 69–80. doi:10.4204/EPTCS.254.6.
URL https://doi.org/10.4204/EPTCS.254.6

[20] M. Leucker, C. Schallhart, A brief account of Runtime Verification, JLAP 78 (5) (2009) 293 – 303. doi:http://dx.doi.org/10.1016/j.jlap.2008.08.004.
URL http://www.sciencedirect.com/science/article/pii/S1567832608000775

[21] A. Francalanza, L. Aceto, A. Achilleos, D. P. Attard, I. Cassar, D. D. Monica, A. Ingólfsdóttir, A Foundation for Runtime Monitoring, in: Runtime Verification: 17th International Conference, RV 2017, Seattle, WA, USA, September 2017, Vol. 10548 of LNCS, Springer, 2017, pp. 8–29. doi:10.1007/978-3-319-67531-2_2.

[22] E. Bartocci, Y. Falcone, A. Francalanza, G. Reger, Introduction to Runtime Verification, Springer, 2018, Ch. 1, pp. 1–33.

[23] L. Bocchi, T. Chen, R. Demangeon, K. Honda, N. Yoshida, Monitoring networks through multiparty session types, TCS 669 (2017) 33–58.

[24] A. Francalanza, L. Aceto, A. Ingólfsdóttir, On Verifying Hennessy-Milner Logic with Recursion at Runtime, in: RV, Vol. 9333 of LNCS, Springer, 2015, pp. 71–86.

[25] A. Francalanza, L. Aceto, A. Ingolfsdottir, Monitorability for the Hennessy–Milner logic with recursion, Formal Methods in System Design (FMSD) (2017) 1–30.

[26] L. Aceto, A. Achilleos, A. Francalanza, A. Ingólfsdóttir, S. Ö. Kjartansson, On the Complexity of Determinizing Monitors, in: CIAA, Vol. 10329 of LNCS, 2017, pp. 1–13. doi:10.1007/978-3-319-60134-2_1.

[27] A. Francalanza, Consistently-detecting monitors, in: R. Meyer, U. Nestmann (Eds.), 28$^{\text{th}}$ International Conference on Concurrency Theory (CONCUR 2017), Vol. 85 of LIPIcs, Schloss Dagstuhl, Dagstuhl, Germany, 2017, pp. 8:1–8:19. doi:10.4230/LIPIcs.CONCUR.2017.8.

[28] L. Aceto, A. Achilleos, A. Francalanza, A. Ingólfsdóttir, Monitoring for silent actions, in: FSTTCS, Vol. 93 of LIPIcs, 2017, pp. 7:1–7:14.

[29] L. Aceto, A. Achilleos, A. Francalanza, A. Ingólfsdóttir, A Framework for Parametrized Monitorability, in: FoSSaCS, Vol. 10803 of LNCS, Springer, 2018, pp. 203–220.

[30] L. Aceto, A. Achilleos, A. Francalanza, A. Ingólfsdóttir, K. Lehtinen, Adventures in Monitorability: From Branching to Linear Time and Back Again, Proceedings of the ACM on Programming Languages 3 (POPL) (2019) 52:1–52:29.
URL https://dl.acm.org/citation.cfm?id=3290365

[31] L. Aceto, A. Achilleos, A. Francalanza, A. Ingólfsdóttir, S. Ö. Kjartansson, Determinizing monitors for HML with recursion, J. Log. Algebraic Methods Program. (JLAMP) 111 (2020) 100515. doi:10.1016/j.jlamp.2019.100515.
URL https://doi.org/10.1016/j.jlamp.2019.100515

[32] L. Aceto, A. Achilleos, A. Francalanza, A. Ingólfsdóttir, K. Lehtinen, An Operational Guide to Monitorability with Applications to Regular Properties, Software and Systems Modeling (SOSYM), 2021 (to appear).

[33] R. Milner, Communication and concurrency, Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1989.

[34] D. Sangiorgi, D. Walker, PI-Calculus: A Theory of Mobile Processes, Cambridge University Press, 2001.

[35] M. Hennessy, Algebraic Theory of Processes, MIT Press, 1988.

[36] A. Bauer, M. Leucker, C. Schallhart, Runtime verification for LTL and TLTL, TOSEM 20 (4) (2011) 14.

[37] A. Francalanza, C. A. Mezzina, E. Tuosto, Towards Choreographic-Based Monitoring, in: Reversible Computation: Extending Horizons of Computing - Selected Results of the COST Action IC1405, Vol. 12070 of LNCS, Springer, 2020, pp. 128–150. doi:10.1007/978-3-030-47361-7_6.
URL https://doi.org/10.1007/978-3-030-47361-7_6

[38] A. Francalanza, C. Cini, Computer says no: Verdict explainability for runtime monitors using a local proof system, J. Log. Algebraic Methods Program. (JLAMP) 119 (2021) 100636. doi:https://doi.org/10.1016/j.jlamp.2020.100636.
URL http://www.sciencedirect.com/science/article/pii/S2352220820301218

[39] K. Sen, A. Vardhan, G. Agha, G. Rosu, Efficient Decentralized Monitoring of Safety in Distributed Systems, in: ICSE, IEEE, 2004, pp. 418–427.

[40] M. d'Amorim, G. Roşu, Efficient monitoring of ω-languages, in: CAV, 2005, pp. 364 – 378.

[41] L. Kuhtz, B. Finkbeiner, Efficient parallel path checking for linear-time temporal logic with past and bounds, Log. Methods Comput. Sci. 8 (4). doi:10.2168/LMCS-8(4:10)2012.
URL https://doi.org/10.2168/LMCS-8(4:10)2012

[42] I. Cassar, A. Francalanza, S. Said, Improving runtime overheads for detecter, in: Proceedings 12th International Workshop on Formal Engineering approaches to Software Components and Architectures, FESCA 2015, London, United Kingdom, April 12th, 2015, Vol. 178 of EPTCS, 2015, pp. 1–8. doi:10.4204/EPTCS.178.1.
URL https://doi.org/10.4204/EPTCS.178.1

[43] K. Chatterjee, T. A. Henzinger, J. Otop, Quantitative monitor automata, in: X. Rival (Ed.), Static Analysis - 23rd International Symposium, SAS 2016, Edinburgh, UK, September 8-10, 2016, Proceedings, Vol. 9837 of LNCS, Springer, 2016, pp. 23–38. doi:10.1007/978-3-662-53413-7_2.
URL https://doi.org/10.1007/978-3-662-53413-7_2

[44] D. P. Attard, A. Francalanza, Trace Partitioning and Local Monitoring for Asynchronous Components, in: SEFM, Vol. 10469 of LNCS, Springer, 2017, pp. 219–235. doi:10.1007/978-3-319-66197-1_14.
URL https://doi.org/10.1007/978-3-319-66197-1_14

[45] R. De Nicola, M. C. B. Hennessy, Testing equivalences for processes, Theoretical Computer Science (TCS) 34 (1-2) (1984) 83–133.
URL http://www.sciencedirect.com/science/article/pii/0304397584901130

[46] M. Y. Vardi, P. Wolper, Reasoning about infinite computations, Inf.& Comp. 115 (1) (1994) 1–37.

[47] R. Grigore, D. Distefano, R. L. Petersen, N. Tzevelekos, Runtime verification based on register automata, in: TACAS, Vol. 7795 of LNCS, 2013, pp. 260–276.

[48] Y. Yamagata, C. Artho, M. Hagiya, J. Inoue, L. Ma, Y. Tanabe, M. Yamamoto, Runtime monitoring for concurrent systems, in: RV, 2016, pp. 386–403.

[49] A. Francalanza, J. Xuereb, On implementing symbolic controllability, in: S. Bliudze, L. Bocchi (Eds.), Coordination Models and Languages - 22nd IFIP WG 6.1 International Conference, COORDINATION 2020, Held as Part of the 15th International Federated Conference on Distributed Computing Techniques, DisCoTec 2020, Valletta, Malta, June 15-19, 2020, Proceedings, Vol. 12134 of LNCS, Springer, 2020, pp. 350–369. doi:10.1007/978-3-030-50029-0_22.
URL https://doi.org/10.1007/978-3-030-50029-0_22

[50] Q. Luo, G. Roşu, EnforceMOP: A Runtime Property Enforcement System for Multithreaded Programs, in: ISSTA, ACM, New York, NY, USA, 2013, pp. 156–166. doi:10.1145/2483760.2483766.
URL http://doi.acm.org/10.1145/2483760.2483766

[51] A. Francalanza, A. Seychell, Synthesising Correct concurrent Runtime Monitors, Formal Methods in System Design (FMSD) 46 (3) (2015) 226–261. doi:10.1007/s10703-014-0217-9.
URL http://dx.doi.org/10.1007/s10703-014-0217-9

[52] P. Fraigniaud, S. Rajsbaum, C. Travers, On the number of opinions needed for fault-tolerant run-time monitoring in distributed systems, in: RV, 2014, pp. 92–107.

[53] S. Berkovich, B. Bonakdarpour, S. Fischmeister, Runtime verification with minimal intrusion through parallelism, FMSD 46 (3) (2015) 317–348.

[54] B. Bonakdarpour, P. Fraigniaud, S. Rajsbaum, D. A. Rosenblueth, C. Travers, Decentralized asynchronous crash-resilient runtime verification, in: CONCUR, 2016, pp. 16:1–16:15.

[55] L. Jia, H. Gommerstadt, F. Pfenning, Monitors and blame assignment for higher-order session types, in: POPL, 2016, pp. 582–594.

[56] C. Colombo, A. Francalanza, R. Mizzi, G. J. Pace, polylarva: Runtime verification with configurable resource-aware monitoring boundaries, in: SEFM, 2012, pp. 218–232.

[57] H. Barringer, Y. Falcone, K. Havelund, G. Reger, D. E. Rydeheard, Quantified Event Automata: Towards Expressive and Efficient Runtime Monitors, in: FM, Vol. 7436 of LNCS, Springer, 2012, pp. 68–84. doi:10.1007/978-3-642-32759-9_9.

[58] G. Reger, H. C. Cruz, D. E. Rydeheard, MarQ: Monitoring at Runtime with QEA, in: TACAS, 2015, pp. 596–610.

[59] S. Debois, T. Hildebrandt, T. Slaats, Safety, liveness and run-time refinement for modular process-aware systems with dynamic sub processes, in: FM, 2015, pp. 143–160.

[60] F. Chen, G. Roşu, MOP: An Efficient and Generic Runtime Verification Framework, in: OOPSLA, ACM, 2007, pp. 569–588. doi:10.1145/1297027.1297069.

[61] M. Hennessy, A Distributed Pi-Calculus, Cambridge University Press, 2007.

[62] G. Roşu, K. Havelund, Rewriting-based techniques for runtime verification, Automated Software Engg. 12 (2) (2005) 151–197. doi:10.1007/s10515-005-6205-y.
URL http://dx.doi.org/10.1007/s10515-005-6205-y

[63] L. Aceto, A. Achilleos, A. Francalanza, A. Ingólfsdóttir, K. Lehtinen, An Operational Guide to Monitorability, in: P. C. Ölveczky, G. Salaün (Eds.), Software Engineering and Formal Methods (SEFM), Vol. 11724 of LNCS, Springer, 2019, pp. 433–453.

[64] C. Cini, A. Francalanza, An LTL Proof System for Runtime Verification, in: TACAS, Vol. 9035 of LNCS, Springer, 2015, pp. 581–595.

[65] L. Bocchi, T.-C. Chen, R. Demangeon, K. Honda, N. Yoshida, Monitoring networks through multiparty session types, in: FMOODS/FORTE 2013, Vol. 7892 of LNCS, 2013, pp. 50–65.

[66] A. Francalanza, L. Aceto, A. Ingólfsdóttir, On verifying hennessy-milner logic with recursion at runtime, in: E. Bartocci, R. Majumdar (Eds.), Runtime Verification - 6th International Conference, RV 2015 Vienna, Austria, September 22-25, 2015. Proceedings, Vol. 9333 of LNCS, Springer, 2015, pp. 71–86. doi:10.1007/978-3-319-23820-3_5.
URL https://doi.org/10.1007/978-3-319-23820-3_5

[67] D. P. Attard, A. Francalanza, A Monitoring Tool for a Branching-Time Logic, in: RV, Vol. 10012 of LNCS, Springer, Cham, 2016, pp. 473–481.

[68] D. P. Attard, I. Cassar, A. Francalanza, L. A. A. Ingofsdottir, A runtime monitoring tool for actor-based systems, in: Behavioural Types: from Theory to Tools, Automation, Control and Robotics, River Publishers, 2017, pp. 49–76. doi:10.13052/rp-9788793519817.
URL https://doi.org/10.13052/rp-9788793519817

[69] detectEr Project, http://www.cs.um.edu.mt/svrg/Tools/detectEr/.

[70] I. Cassar, A. Francalanza, D. P. Attard, L. Aceto, A. Ingólfsdóttir, A suite of monitoring tools for erlang, in: G. Reger, K. Havelund (Eds.), RV-CuBES 2017. An International Workshop on Competitions, Usability, Benchmarks, Evaluation, and Standardisation for Runtime Verification Tools, September 15, 2017, Seattle, WA, USA, Vol. 3 of Kalpa Publications in Computing, EasyChair, 2017, pp. 41–47.
URL http://www.easychair.org/publications/paper/cSzb

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

[72] L. Fei, S. P. Midkiff, Artemis: practical runtime monitoring of applications for execution anomalies, in: M. I. Schwartzbach, T. Ball (Eds.), Proceedings of the ACM SIGPLAN 2006 Conference on Programming Language Design and Implementation, Ottawa, Ontario, Canada, June 11-14, 2006, ACM, 2006, pp. 84–95. doi:10.1145/1133981.1133992.
URL https://doi.org/10.1145/1133981.1133992

[73] B. Bonakdarpour, S. Navabpour, S. Fischmeister, Sampling-based runtime verification, in: M. J. Butler, W. Schulte (Eds.), FM 2011: Formal Methods - 17th International Symposium on Formal Methods, Limerick, Ireland, June 20-24, 2011. Proceedings, Vol. 6664 of LNCS, Springer, 2011, pp. 88–102. doi:10.1007/978-3-642-21437-0_9.
URL https://doi.org/10.1007/978-3-642-21437-0_9

[74] E. Bartocci, R. Grosu, A. Karmarkar, S. A. Smolka, S. D. Stoller, E. Zadok, J. Seyster, Adaptive runtime verification, in: S. Qadeer, S. Tasiran (Eds.), Runtime Verification, Third International Conference, RV 2012, Istanbul, Turkey, September 25-28, 2012, Revised Selected Papers, Vol. 7687 of LNCS, Springer, 2012, pp. 168–182. doi:10.1007/

978-3-642-35632-2_18.
URL https://doi.org/10.1007/978-3-642-35632-2_18

[75] J. Laurent, A. Goodloe, L. Pike, Assuring the Guardians, in: RV, Vol. 9333 of LNCS, Springer, 2015, pp. 87–101.

[76] T. Ferrère, T. A. Henzinger, N. E. Saraç, A theory of register monitors, in: A. Dawar, E. Grädel (Eds.), Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018, ACM, 2018, pp. 394–403. doi:10.1145/3209108.3209194.
URL https://doi.org/10.1145/3209108.3209194

[77] D. A. Basin, T. Dardinier, L. Heimes, S. Krstic, M. Raszyk, J. Schneider, D. Traytel, A formally verified, optimized monitor for metric first-order dynamic logic, in: Automated Reasoning - 10th International Joint Conference, IJCAR 2020, Paris, France, July 1-4, 2020, Proceedings, Part I, Vol. 12166 of LNCS, Springer, 2020, pp. 432–453. doi:10.1007/978-3-030-51074-9_25.
URL https://doi.org/10.1007/978-3-030-51074-9_25

[78] B. Finkbeiner, S. Oswald, N. E. Passing, M. Schwenger, Verified rust monitors for lola specifications, in: Runtime Verification - 20th International Conference, RV 2020, Los Angeles, CA, USA, October 6-9, 2020, Proceedings, Vol. 12399 of LNCS, Springer, 2020, pp. 431–450. doi:10.1007/978-3-030-60508-7_24.
URL https://doi.org/10.1007/978-3-030-60508-7_24

[79] L. Aceto, A. Achilleos, A. Francalanza, A. Ingólfsdóttir, K. Lehtinen, The best a monitor can do, in: C. Baier, J. Goubault-Larrecq (Eds.), 29th EACSL Annual Conference on Computer Science Logic, CSL 2021, January 25-28, 2021, Ljubljana, Slovenia (Virtual Conference), Vol. 183 of LIPIcs, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, pp. 7:1–7:23. doi:10.4230/LIPIcs.CSL.2021.7.
URL https://doi.org/10.4230/LIPIcs.CSL.2021.7

[80] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, W. G. Griswold, An overview of aspectj, in: Proceedings of the 15th European Conference on Object-Oriented Programming, ECOOP '01, Springer-Verlag, London, UK, UK, 2001, pp. 327–353.
URL http://dl.acm.org/citation.cfm?id=646158.680006

[81] A. Colyer, A. Clement, G. Harley, M. Webster, Eclipse Aspectj: Aspect-oriented Programming with Aspectj and the Eclipse Aspectj Development Tools, 1st Edition, Addison-Wesley Professional, 2004.

[82] J. Seyster, K. Dixit, X. Huang, R. Grosu, K. Havelund, S. A. Smolka, S. D. Stoller, E. Zadok, Interaspect: aspect-oriented instrumentation with GCC, Formal Methods in System Design 41 (3) (2012) 295–320. doi:

10.1007/s10703-012-0171-3.
URL https://doi.org/10.1007/s10703-012-0171-3

[83] I. Cassar, A. Francalanza, L. Aceto, A. Ingólfsdóttir, eAOP: an aspect oriented programming framework for Erlang, in: N. Chechina, S. L. Fritchie (Eds.), Proceedings of the 16th ACM SIGPLAN International Workshop on Erlang, ACM, 2017, pp. 20–30. doi:10.1145/3123569.3123570.
URL https://doi.org/10.1145/3123569.3123570

[84] S. Liang, G. Bracha, Dynamic Class Loading in the Java Virtual Machine, in: Proceedings of the 13th ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications, OOPSLA, ACM, New York, NY, USA, 1998, pp. 36–44. doi:10.1145/286936.286945.
URL http://doi.acm.org/10.1145/286936.286945

[85] T. Arts, L.-A. Fredlund, Trace analysis of erlang programs, SIGPLAN Not. 37 (12) (2002) 18–24. doi:10.1145/636517.636524.
URL http://doi.acm.org/10.1145/636517.636524

[86] J. Bonér, What are the key issues for commercial AOP use: how does AspectWerkz address them?, in: G. C. Murphy, K. J. Lieberherr (Eds.), Proceedings of the 3rd International Conference on Aspect-Oriented Software Development, AOSD, ACM, 2004, pp. 5–6. doi:10.1145/976270.976273.
URL https://doi.org/10.1145/976270.976273

[87] A. R. Bernat, B. P. Miller, Anywhere, Any-time Binary Instrumentation, in: J. Foster, L. L. Pollock (Eds.), Proceedings of the 10th ACM SIGPLAN-SIGSOFT workshop on Program Analysis for Software Tools, PASTE, ACM, 2011, pp. 9–16. doi:10.1145/2024569.2024572.
URL https://doi.org/10.1145/2024569.2024572

[88] N. Nethercote, J. Seward, Valgrind: A Framework for Heavyweight Dynamic Binary Instrumentation, in: Proceedings of the 28th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI, ACM, New York, NY, USA, 2007, pp. 89–100. doi:10.1145/1250734.1250746.
URL http://doi.acm.org/10.1145/1250734.1250746

[89] N. Grech, G. Fourtounis, A. Francalanza, Y. Smaragdakis, Heaps don't lie: countering unsoundness with heap snapshots, PACMPL 1 (OOPSLA) (2017) 68:1–68:27. doi:10.1145/3133892.
URL https://doi.org/10.1145/3133892

[90] N. Grech, G. Fourtounis, A. Francalanza, Y. Smaragdakis, Shooting from the heap: ultra-scalable static analysis with heap snapshots, in: F. Tip, E. Bodden (Eds.), Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 2018, Amsterdam,

The Netherlands, July 16-21, 2018, ACM, 2018, pp. 198–208. doi:10.1145/3213846.3213860.
URL https://doi.org/10.1145/3213846.3213860

[91] M. Kim, M. Viswanathan, S. Kannan, I. Lee, O. Sokolsky, Java-MaC: A run-time assurance approach for Java programs, FMSD 24 (2) (2004) 129–155. doi:10.1023/B:FORM.0000017719.43755.7c.

[92] N. Decker, M. Leucker, D. Thoma, jUnitRV - Adding Runtime Verification to jUnit, in: NASA FM, Vol. 7871 of LNCS, Springer, 2013, pp. 459–464. doi:10.1007/978-3-642-38088-4_34.

[93] B. D'Angelo, S. Sankaranarayanan, C. Sánchez, W. Robinson, B. Finkbeiner, H. B. Sipma, S. Mehrotra, Z. Manna, Lola: Runtime monitoring of synchronous systems, in: TIME, IEEE, 2005, pp. 166–174.

[94] D. Basin, G. Caronni, S. Ereth, M. Harvan, F. Klaedtke, H. Mantel, Scalable Offline Monitoring of Temporal Specifications, Form. Methods Syst. Des. 49 (1-2) (2016) 75–108. doi:10.1007/s10703-016-0242-y.
URL http://dx.doi.org/10.1007/s10703-016-0242-y

[95] T. Zhang, P. Gebhard, O. Sokolsky, SMEDL: combining synchronous and asynchronous monitoring, in: Y. Falcone, C. Sánchez (Eds.), Runtime Verification - 16th International Conference, RV, Vol. 10012 of LNCS, Springer, 2016, pp. 482–490. doi:10.1007/978-3-319-46982-9_32.
URL https://doi.org/10.1007/978-3-319-46982-9_32

[96] C. Sánchez, Online and offline stream runtime verification of synchronous systems, in: C. Colombo, M. Leucker (Eds.), Runtime Verification - 18th International Conference, (RV), Vol. 11237 of LNCS, Springer, 2018, pp. 138–163. doi:10.1007/978-3-030-03769-7_9.
URL https://doi.org/10.1007/978-3-030-03769-7_9

[97] U. Erlingsson, The Inlined Reference Monitor approach to Security Policy Enforcement, Ph.D. thesis, Cornell University (2004).

[98] S. Pinisetty, T. Jron, S. Tripakis, Y. Falcone, H. Marchand, V. Preoteasa, Predictive runtime verification of timed properties, J. Syst. Softw. 132 (C) (2017) 353–365. doi:10.1016/j.jss.2017.06.060.
URL https://doi.org/10.1016/j.jss.2017.06.060

[99] G. Bernardi, M. Hennessy, Mutually testing processes, Logical Methods in Computer Science 11 (2). doi:10.2168/LMCS-11(2:1)2015.
URL https://doi.org/10.2168/LMCS-11(2:1)2015

[100] G. Castagna, N. Gesbert, L. Padovani, A theory of contracts for web services, ACM Trans. Program. Lang. Syst. 31 (5) (2009) 19:1–19:61. doi:10.1145/1538917.1538920.
URL http://doi.acm.org/10.1145/1538917.1538920

[101] G. T. Bernardi, A. Francalanza, Full-abstraction for Client Testing Preorders, Sci. Comput. Program. 168 (2018) 94–117. doi:10.1016/j.scico.2018.08.004.
URL https://doi.org/10.1016/j.scico.2018.08.004

[102] L. Aceto, A. Achilleos, A. Francalanza, A. Ingólfsdóttir, K. Lehtinen, Testing Equivalence vs. Runtime Monitoring, in: Models, Languages, and Tools for Concurrent and Distributed Programming - Essays Dedicated to Rocco De Nicola on the Occasion of His 65th Birthday, Vol. 11665 of LNCS, Springer, 2019, pp. 28–44. doi:10.1007/978-3-030-21485-2_4.
URL https://doi.org/10.1007/978-3-030-21485-2_4