A Theory of Monitors (Extended Abstract)*

Adrian Francalanza

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

Abstract. We develop a behavioural theory for *monitors* — software entities that passively analyse the runtime behaviour of systems so as to infer properties about them. First, we extend the monitor language and instrumentation relation of [17] to handle 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 more tractable, and prove full-abstraction for the latter alternative preorders with respect to the contextual preorders.

1 Introduction

Monitors (execution monitors [32]) are software entities that are *instrumented* to execute along side a program so as determine properties about it, inferred from the *runtime analysis* of the exhibited (program) execution; this basic monitor form is occasionally termed (sequence) *recognisers* [28]. In other settings, monitors go further and either *adapt* aspects of the monitored program [7, 11, 22] or *enforce* predefined properties by modifying the observable behaviour [15, 28, 4]. Monitors are central to software engineering techniques such as monitor-oriented programming [16] and fail-fast design patterns [8] used in fault-tolerant systems [19, 34]; they are also used extensively in runtime verification [27], a lightweight verification technique that attempts to mitigate state explosion problems associated with full-blown verification methods such as model checking.

Monitoring setups typically consist of three 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, $P \triangleleft M$. The latter gives rise to a software composition relation that has seldom been studied in its own right. This paper investigates compositional reasoning techniques for monitors performing *detections* (recognisers), composed using the instrumentation relation employed in [17], for programs expressed as piCalculus processes [29, 31], a well-studied concurrency model. We set out to develop monitor preorders

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

stating that, when instrumented in the *context* of an *arbitrary* process P, if $P \triangleleft M_1$ exhibits certain properties, then $P \triangleleft M_2$ exhibits them as well. Within

* The research was supported by the UoM research fund CPSRP05-04 and the research grant COST-STSM-ECOST-STSM-IC1201-170214-038253.

this setup, we formalise the possible instrumentation properties one may require from an instrumented (monitored) process, and show how these give rise to different monitoring preorders.

Example 1. Consider the monitors M_1 and M_2 below. M_1 monitors for output actions on channel a with a payload that is not in the set C. Apart from detecting the same outputs on channel a, M_2 also detects outputs with payload b on channels that are not in D (the + construct acts as an external choice [23]).

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

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 different criteria, the two monitors would not be related. Consider the case where $a \in D$ (or $b \in C$): for a process P exhibiting action $a!b$, monitor M_2 may non-deterministically *fail to detect* this behaviour; by contrast, M_1 *always* detects the behaviour $a!b$ and, in this sense, M_2 does not preserve all the properties of M_1. Monitors are also expected to *interfere minimally* with the execution of the analysed process, giving rise to other criteria for relating monitors, as we will see in the sequel. ■

There are various reasons why such preorders are useful. 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. Moreover, our preorders provide a formal understanding for when it is valid to *substitute one monitor implementation for another* while preserving elected monitoring properties. We consider a general model that allows monitors to behave non-deterministically; this permits us to study the cases where non-determinism is either tolerated or considered erroneous. Indeed, there are settings where determinism is unattainable (*e.g.,* distributed monitoring [18, 33]). Occasionally, non-determinism is also used to expresses under-specification in program refinement.

Although formal and intuitive, the preorders alluded to in (1) turn out to be hard to establish. One of the principal obstacles is the universal quantification over all possible processes 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 the alternative monitor preorders. More importantly, however, it also allows us to use the more tractable alternative relations as a 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.

The rest of the paper is structured as follows. Sec. 2 briefly overviews our process model 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 Sec. 5, where we also establish the correspondence with the other preorders. Sec. 6 concludes.

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

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

$$\textsc{pThn}\ \frac{}{I \triangleright \mathsf{if}\ c=c\ \mathsf{then}\ P\ \mathsf{else}\ Q \xrightarrow{\tau} P} \qquad \textsc{pEls}\ \frac{c \sharp d}{I \triangleright \mathsf{if}\ c=d\ \mathsf{then}\ P\ \mathsf{else}\ Q \xrightarrow{\tau} Q}$$

$$\textsc{pRec}\ \frac{}{I \triangleright \mathsf{rec}\ X.P \xrightarrow{\tau} P[\mathsf{rec}\ X.P/X]} \qquad \textsc{pPar}\ \frac{I \triangleright P \xrightarrow{\mu} P'}{I \triangleright P \parallel Q \xrightarrow{\mu} P' \parallel Q}$$

$$\textsc{pCom}\ \frac{I, d \triangleright P \xrightarrow{c!d} P' \qquad I, d \triangleright Q \xrightarrow{c?d} Q'}{I, d \triangleright P \parallel Q \xrightarrow{\tau} P' \parallel Q'} \qquad \textsc{pRes}\ \frac{I, d \triangleright P \xrightarrow{\mu} P' \qquad d \sharp \mu}{I \triangleright \mathsf{new}\ d.P \xrightarrow{\mu} \mathsf{new}\ d.P'}$$

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

Fig. 1. piCalculus syntax and semantics

2 The Language

Fig. 1 presents our process language, a standard version of the piCalculus. It has the usual constructs and assumes separate denumerable sets for channel names $c, d, a, b \in \textsc{Chans}$, variables $x, y, z \in \textsc{Vars}$ and process variables, $X, Y \in \textsc{PVars}$, 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/bound variables, process variables and names in P. We use standard syntactic conventions *e.g.,* we identify processes up to renaming of bound names and variables (alpha conversion). For arbitrary syntactic objects o, o', we write $o \sharp o'$ when the free *names* 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 defined by the Labelled Transition System (LTS) shown in Fig. 1. LTS judgements are of the form

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

where $I \subseteq \textsc{Chans}$ denotes an *interface* of names *known* (or 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 write I, c as a shorthand for $I \cup \{c\}$ where $c \notin I$, and generally assume a version of the Barendregt convention whereby $\mathbf{bn}(P) \sharp I$.[1] 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); implicitly, $c, d \in I$ in rule pOut and $c \in I$ in rule pIn (but d is not necessarily in I). We use $s, t \in \textsc{Act}^*$ to denote *traces of external actions*. Although actions do not include explicit information relating to extruded names, this may be retrieved using I as shown in Def. 1; the absence of action name binding (as in [31, 24]) simplifies subsequent handling of traces. 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).

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

$$\boldsymbol{aftr}(I, \tau) \stackrel{def}{=} I \qquad \boldsymbol{aftr}(I, c!d) \stackrel{def}{=} I \cup \{d\} \qquad \boldsymbol{aftr}(I, c?d) \stackrel{def}{=} I \cup \{d\}$$

We lift the functions in Def. 1 to traces, *e.g.*, $\mathbf{aftr}(I, s)$, in the obvious way and denote successive transitions $I \triangleright P \xrightarrow{\mu_1} P_1$ and $\mathbf{aftr}(I, \mu_1) \triangleright P_1 \xrightarrow{\mu_2} P_2$ as $I \triangleright P \xrightarrow{\mu_1} \mathbf{aftr}(I, \mu_1) \triangleright P_1 \xrightarrow{\mu_2} P_2$. We write $I \triangleright P \not\xrightarrow{\mu}$ to denote $\nexists P' \cdot I \triangleright P \xrightarrow{\mu} P'$ and $I \triangleright P \stackrel{s}{\Longrightarrow} Q$ to denote $I_0 \triangleright P_0 \xrightarrow{\mu_1} I_1 \triangleright P_1 \xrightarrow{\mu_2} I_2 \triangleright P_2 \ldots \xrightarrow{\mu_n} P_n$ 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.

Example 2. Consider $P_{\text{sv}} = \mathsf{rec}\, X.c?x.\mathsf{new}\, d.x!d.X$, modelling the idiomatic server that repeatedly waits for requests on c and answers back on the inputted channel with a *fresh* channel. We can derive the following behaviour *wrt.* $I = \{c\}$:

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

Above, bound outputs/inputs [31] are manifested as interface extensions. ■

3 Monitor Instrumentation

Monitors, $M, N \in \textsc{Mon}$, are syntactically defined by the grammar of Fig. 2. They may reach either of *two* verdicts, namely detection, ✓, or termination, end, denoting an inconclusive verdict. Our setting is a mild generalisation to that

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

Syntax

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

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

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

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

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

Fig. 2. Monitor syntax, semantics and Instrumentation Semantics

in [17] since monitors need to reason about communicated names so as to adequately monitor for piCalculus processes. They are thus equipped with a pattern matching construct (used to observe external actions) and a name-comparison branching construct. The remaining constructs, *i.e.,* external branching and recursion, are standard. Note that, whereas the syntax allows for monitors with free occurrences of monitor variables, monitors are always closed *wrt.* (value) variables, whereby the outermost occurrence of a variable acts as a binder. *E.g.,* in the monitor $(x?c.x!y.\mathsf{if}\ y = d\ \mathsf{then\ end\ else}\ \checkmark)$ pattern $x?c$ binds variable x in the continuation whereas pattern $x!y$ binds variable y.

The monitor semantics is defined in terms of an LTS (Fig. 2), modeling the analysis of the visible runtime execution of a process. Following [30, 15, 17], 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. The remaining transitions are unremarkable.

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 is closed and $\mathbf{fn}(P) \subseteq I$. The LTS semantics follows [17, 7] and relies on the *resp.* process and monitor semantics of Fig. 1 and Fig. 2. In rule iMon, 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 note 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 3. Recall P_{sv} from Ex. 2. 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 $I' = \{c, a\}$:

$$I' \triangleright P_{\text{sv}} \triangleleft M_1 \xRightarrow{c?a \cdot a!d} P_{\text{sv}} \triangleleft \checkmark$$

Contrastingly, for the same I', monitoring with $M_2 = (y!z.\mathsf{if}\ y = a\ \mathsf{then\ end\ else}\ \checkmark)$ 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 the monitor after transition $c?a$, so as to avoid erroneous detections.

$$\begin{aligned} I' \triangleright P_{\text{sv}} \triangleleft M_2 &\xrightarrow{\tau} c?x.\mathsf{new}\, d.x!d.P_{\text{sv}} \triangleleft M_2 \\ &\xrightarrow{c?a} \mathsf{new}\, d.a!d.P_{\text{sv}} \triangleleft \mathsf{end} \xRightarrow{a!d} P_{\text{sv}} \triangleleft \mathsf{end} \end{aligned} \tag{2}$$

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

$$I \triangleright P_{\text{sv}} \triangleleft N \xrightarrow{\tau} \cdot \xrightarrow{c?a} I, a \triangleright \big(\mathsf{new}\, d.a!d.P_{\text{sv}}\big) \triangleleft \mathsf{if}\ a = c\ \mathsf{then\ end\ else}\ a!z.c?z.\checkmark \tag{3}$$

$$\xrightarrow{\tau} I, a \triangleright \big(\mathsf{new}\, d.a!d.P_{\text{sv}}\big) \triangleleft a!z.c?z.\checkmark \tag{4}$$

$$\xrightarrow{a!d} (I, a, d) \triangleright P_{\text{sv}} \triangleleft c?d.\checkmark \tag{5}$$

$$\xrightarrow{\tau} \cdot \xrightarrow{c?b} (I, a, d, b) \triangleright \mathsf{new}\, d'.b!d'.P_{\text{sv}} \triangleleft \mathsf{end} \tag{6}$$

$$\xrightarrow{b!d'} (I, a, d, b, d') \triangleright P_{\text{sv}} \triangleleft \mathsf{end} \tag{7}$$

In (3) the server (asynchronously) unfolds (pRec and iAsyP) and inputs on c the fresh name a (pIn); the monitor can analyse $c?a$ (mPat where $\mathsf{match}(c?y, c?a) = \{y \mapsto a\}$), transitioning accordingly (iMon) while learning the fresh name a for future monitoring. 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, (4) using mEls, resulting in the second monitoring transition, (5) using iMon, where the monitor learns the second fresh name d, this time originating from the monitored process. 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, (6). In (7), verdicts allow monitored processes to execute without any hindrance. ■

Ex. 3 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 (2) and (6), terminating monitoring when the observed process behaviour does not correspond, (through rule iTer). But in order to do this while avoiding premature termination, instrumentation needs to allow for monitor internal computation, *e.g.,* (4). Unfortunately, the premise caveat $M \overset{\tau}{\not\longrightarrow}$ in rule iTer — necessary to prevent premature terminations — allows monitors to affect (indirectly) process behaviour. For instance the monitor Ω below:

$$\Omega = \mathsf{rec}\, X.(\mathsf{if}\ c{=}c\ \mathsf{then}\ X\ \mathsf{else}\ X) \qquad \Omega' = \mathsf{if}\ c{=}c\ \mathsf{then}\ \Omega\ \mathsf{else}\ \Omega \tag{8}$$

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 be applied. We revisit this point in Sec. 4. We conclude with the property stating that verdicts are irrevocable.

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

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*

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

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 detected computations, identifying them by the visible trace that is dictated by the process (over which the monitor should not have any control).

Example 4. Consider $P = \mathsf{new}\, d.(d! \,\|\, d?.c!a \,\|\, d?.c!b)$, $I = \{c, b, a\}$ and monitors:

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

Configurations $I \triangleright M_i \triangleleft P$ for $i \in 1..5$ exhibit detecting computations along $s = c!a.\epsilon$. For trace $t = c!b.\epsilon$, configurations $I \triangleright M_j \triangleleft P$ 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 with M_1 *always* detects along t, those with M_4 and M_5 may fail to detect it along such a trace. Similarly, configuration with M_1 and M_4 deterministically detect along trace s, but $I \triangleright M_5 \triangleleft P$ does not. ■

Ex. 4 suggests two types of computation detections that a monitor may exhibit.

Definition 3 (Potential and Deterministic Detection). 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* $I \triangleright P \triangleleft M$*.* 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 1. If a monitored process cannot produce trace s, *i.e.*, $I \triangleright P \not\stackrel{\hat{s}}{\Longrightarrow}$, then $\mathsf{pd}(M, I, P, s)$ is trivially false and $\mathsf{dd}(M, I, P, s)$ is trivially true for any M. ■

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).

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

$M \cong_{pd} N$ *and* $M \cong_{dd} N$ *are the kernel equivalences induced by the* resp. *preorders, i.e.,* $M \cong_{pd} N \stackrel{def}{=} (M \sqsubseteq_{pd} N$ *and* $N \sqsubseteq_{pd} M)$*, and similarly for* $M \cong_{dd} N$*. We write* $M \sqsubset_{pd} N$ *in lieu of* $(M \sqsubseteq_{pd} N$ *and* $N \not\sqsubseteq_{pd} M)$ *and similarly for* $M \sqsubset_{dd} N$*.* ■

Example 5. Recall the monitors defined in Ex. 4. It turns out that

$$M_2 \cong_{\mathsf{pd}} M_3 \sqsubset_{\mathsf{pd}} M_5 \cong_{\mathsf{pd}} M_4 \cong_{\mathsf{pd}} M_1 \tag{9}$$
$$M_5 \sqsubset_{\mathsf{dd}} M_2 \cong_{\mathsf{dd}} M_3 \cong_{\mathsf{dd}} M_4 \sqsubset_{\mathsf{dd}} M_1 \tag{10}$$

Note that, whereas M_5 can *potentially* detect more computations that M_2 and M_3, (9), it can *deterministically* detect less computations than these monitors (10); in fact, M_5 cannot deterministically detect *any* computation. ■

As opposed to prior work on monitors [15, 2, 10], the detection predicates in Def. 3 consider monitor behaviour within an instrumented system. Apart from acting as a continuation for the study in [17], this setup also enables us to formally justify subtle monitor orderings, Ex. 6, and analyse peculiarities brought about by the instrumentation relation, Ex. 7.

Example 6. Using the shorthand $\tau.M$ for $\mathsf{rec}\, X.M$ where $X \notin \mathbf{fV}(M)$, we have:

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

For process $P = \Omega$, defined as in (8), predicate $\mathsf{dd}(\checkmark, I, P, \epsilon)$ holds trivially, but predicate $\mathsf{dd}(\tau.\checkmark, I, P, \epsilon)$ *does not*, due of the non-detecting ϵ-computation $I \triangleright \Omega \triangleleft \tau.\checkmark \xrightarrow{\tau} \cdot \xrightarrow{\tau} \Omega \triangleleft \tau.\checkmark \xrightarrow{\tau} \cdot \xrightarrow{\tau} \ldots$, refuting the inequality $\checkmark \sqsubseteq_{\mathsf{dd}} \tau.\checkmark$. ■

Example 7. Recalling the divergent monitor Ω from (8), we have:

$$\big(\mathsf{end} \quad \cong_{\mathsf{dd}} \quad c!a.\mathsf{end}\big) \quad \sqsubset_{\mathsf{dd}} \quad c!a.\checkmark \quad \sqsubset_{\mathsf{dd}} \quad \Omega \quad \sqsubset_{\mathsf{dd}} \quad \checkmark \tag{11}$$

$$\Omega + \mathsf{end} \quad \sqsubset_{\mathsf{dd}} \quad \big(\Omega \quad \cong_{\mathsf{dd}} \quad \Omega + \checkmark \quad \cong_{\mathsf{dd}} \quad \mathsf{rec}\, X.(\tau.X + \checkmark)\big) \tag{12}$$

$$\Omega + c!a.\checkmark \quad \sqsubset_{\mathsf{pd}} \quad \Omega + \checkmark \qquad \text{but} \qquad \Omega + c!a.\checkmark \quad \cong_{\mathsf{dd}} \quad \Omega + \checkmark \tag{13}$$

In (11), *every* computation starting with monitor $\checkmark$ is trivially detected. Monitor Ω limits 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 $|s| > 0$, rendering $\mathsf{dd}(\Omega, I, P, s)$ for $|s| > 0$ vacuously true (see Rem. 1). By contrast, $\mathsf{dd}(c!a.\checkmark, I, P, s)$ holds only whenever $s = c!a.t$ (for arbitrary t) and $I \triangleright P \stackrel{s}{\Longrightarrow}$.

In (12), 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 for $|s| > 0$.

In (13), although 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*), both deterministically detect the same computations, *i.e.,* all s-computation where $|s| > 0$. Specifically, if a process being monitored, say $I \triangleright P$, 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 $c!a$) but, at the same time, it can τ-transition (see rules iMon and iTrm). ■

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

Example 8. Consider the monitor $M_{\text{any}} = x?y.\checkmark + x!y.\checkmark$.

$$M_{\text{any}} \cong_{\mathsf{pd}} M_{\text{any}} + \Omega \quad \text{and} \quad M_{\text{any}} \cong_{\mathsf{dd}} M_{\text{any}} + \Omega \tag{14}$$

$$c!a.\mathsf{end} \cong_{\mathsf{pd}} c!a.\mathsf{end} + c!a.\Omega \quad \text{and} \quad c!a.\mathsf{end} \cong_{\mathsf{dd}} c!a.\mathsf{end} + c!a.\Omega \tag{15}$$

Intuitively, M_{any} potentially and deterministically detects any s-computation when $|s| > 0$. It turns out that $M_{\text{any}} + \Omega$ produces the same potential and deterministic detections, yielding the *resp.* equalities in (14). In (15), neither monitor produces *any* potential or deterministic detections and they are thus equivalent according to the *resp.* kernel equivalences of Def. 4. ■

There are however settings where the equalities established in Ex. 8 are deemed too coarse. *E.g.,* in (15), whereas monitor $c!a.\mathsf{end}$ is innocuous when instrumented with a process, monitor $c!a.\mathsf{end} + c!a.\Omega$ may potentially change the observed behaviour of the process under scrutiny after the action $c!a$ is emitted (by suppressing external actions, as explained in Ex. 7); a similar argument applies for the monitors in (14). We thus define a third monitor predicate called *transparency* [28, 15, 5], 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 monitoring 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 $\left(I \triangleright P \triangleleft M \stackrel{s}{\Longrightarrow} Q \triangleleft N$ and $\boldsymbol{aftr}(I, s) \triangleright (Q \triangleleft N) \not\stackrel{\alpha}{\Longrightarrow}\right)$ implies $\boldsymbol{aftr}(I, s) \triangleright Q \not\stackrel{\alpha}{\Longrightarrow}$. We define the induced preorder as expected:*

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

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).

$$M \sqsubseteq N \quad \stackrel{def}{=} \quad M \sqsubseteq_{pd} N \text{ and } M \sqsubseteq_{dd} N \text{ and } M \sqsubseteq_{tr} N$$ ■

Example 9. We have $M_{\text{any}} \not\sqsubseteq M_{\text{any}} + \Omega$ because $M_{\text{any}} \not\sqsubseteq_{\mathsf{tr}} M_{\text{any}} + \Omega$, since $\neg\mathsf{tr}(M_{\text{any}} + \Omega, I, P, s)$ for $I = \{c, a\}$, $P = c!a.c!a.\mathsf{nil}$ and $s = c!a.\epsilon$. Similarly, 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, we can use P, I and t from Ex. 4 as counter examples to show that $\mathsf{pd}(M_5, I, P, t)$ and $\neg\mathsf{pd}(M_3, I, P, t)$, thus *disproving* $M_5 \sqsubseteq_{\mathsf{pd}} M_3$. 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. 6, Ex. 7 and Ex. 8, this often requires intricate reasoning in terms of the three LTSs defined in Fig. 1 and Fig. 2.

5 Characterisation

We define alternative monitor preorders for which positive statements about their inequalities are easier to establish. The new preorders are defined exclusively in terms of the monitor operational semantics of Fig. 2, as opposed to how they are affected by arbitrary processes as in Def. 6 (which considers also the process and instrumentation LTSs). We show that the new preorders coincide with those in Sec. 4. Apart from equipping us with an easier 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, $w \xrightarrow{\alpha} w$: these are redundant when considering the monitor operational semantics in isolation. Note, however, that we can still used rule mVer, *e.g.*, to derive $\checkmark + M \xrightarrow{\alpha}_r \checkmark$.

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.* ■

We use the restricted LTS to limit the detecting transition sequences on the left of the implication of Def. 8. However, we 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).

$$M \preceq_{pd} N \overset{def}{=} \forall s \cdot M \overset{s}{\Longrightarrow}_r \checkmark \quad \textit{implies} \quad N \overset{s}{\Longrightarrow} \checkmark$$

Theorem 2 (Potential-Detection Preorders). $M \sqsubseteq_{pd} N$ *iff* $M \preceq_{pd} N$

Example 10. By virtue of Thm. 2, to show that $\Omega + c!a.\checkmark \sqsubseteq_{\mathsf{pd}} \Omega + \checkmark$ from (13) of Ex. 7 holds, we only need to consider $\Omega + c!a.\checkmark \overset{c!a}{\Longrightarrow}_r \checkmark$, which can be matched by $\Omega + \checkmark \overset{c!a}{\Longrightarrow} \checkmark$. Similarly, to show $(x!a.\mathsf{if}\ x = c\ \mathsf{then}\ \checkmark\ \mathsf{else\ end}) \sqsubseteq_{\mathsf{pd}} \checkmark$, we only need to consider $(x!a.\mathsf{if}\ x = c\ \mathsf{then}\ \checkmark\ \mathsf{else\ end}) \overset{c!a}{\Longrightarrow}_r \checkmark$, matched by $\checkmark \xrightarrow{c!a} \checkmark$. ■

For the remaining characterisations, we require two divergence judgements.

Definition 9 (Divergence and Strong Divergence).

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

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

The alternative preorder for deterministic detections, Def. 11 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 M to *block* before it can complete trace s. Predicate $\mathsf{fl}(M, s)$ describes the potential for *failing* after monitoring trace s, *i.e.,* an s-derivative of M 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 $\mathsf{nd}(M, s)$ states the existence of a non-detecting s-derivative of M.

Definition 10 (Monitor Block, Fail and Non-Detecting).

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

Note that $\mathit{blk}(M, s)$ *implicitly requires* $|s| \geq 1$ *for the predicate to hold.* ■

Corollary 1. $\mathit{blk}(M, s) \quad \mathit{implies} \quad \forall t \cdot \mathit{blk}(M, st)$

Definition 11 (Alternative Deterministic Detection Preorder).

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

We write $M \simeq_{\mathit{dd}} N$ *to denote the kernel equality* ($M \preceq_{\mathit{dd}} N$ *and* $N \preceq_{\mathit{dd}} M$). ■

Theorem 3 (Deterministic-Detection Preorders). $M \sqsubseteq_{\mathit{dd}} N$ *iff* $M \preceq_{\mathit{dd}} N$

Example 11. Consider $M = c?a.\mathsf{end} + x!b.\mathsf{end}$ and $N = c?a.\mathsf{end}$. By virtue of Thm. 3, to determine $M \sqsubseteq_{\mathsf{dd}} N$ we prove $M \preceq_{\mathsf{dd}} N$ as follows:

1. We have $\mathsf{blk}(N, s)$ whenever $s = \alpha s'$ and $\alpha \neq c?a$. We have two subcases:
   - If $\mathsf{match}(x!b, \alpha)$ is undefined, we show $\mathsf{blk}(M, \alpha s')$ by first showing that $\mathsf{blk}(M, \alpha\epsilon)$ and then generalising the result for arbitrary s' using Cor. 1.
   - If $\exists \sigma \cdot \mathsf{match}(x!b, \alpha) = \sigma$, we show $\mathsf{fl}(M, \alpha s')$ by first showing $\mathsf{fl}(M, \alpha\epsilon)$ and then generalising the result for arbitrary s' using Thm. 1.
2. For any s, we have $\mathsf{fl}(N, c?a.s)$: we can show $\mathsf{fl}(M, c?a.s)$, again using Thm. 1 to alleviate the proof burden.
3. For any s, we have $\mathsf{nd}(N, c?a.s)$: the required proof is analogous to the previous case. ■

Example 12. Due to full abstraction (*i.e.,* completeness), we can alternatively disprove $\checkmark \sqsubseteq_{\mathsf{dd}} \tau.\checkmark$ from Ex. 6 by showing that $\checkmark \not\preceq_{\mathsf{dd}} \tau.\checkmark$: we can readily argue that whereas $\mathsf{nd}(\tau.\checkmark, \epsilon)$, we *cannot* show either $\mathsf{nd}(\checkmark, \epsilon)$ or $\mathsf{blk}(\checkmark, \epsilon)$. ■

Example 13. Recall the equalities $\Omega \cong_{\mathsf{dd}} \Omega + \checkmark \cong_{\mathsf{dd}} \mathsf{rec}\, X.(\tau.X + \checkmark)$ claimed in (12) of Ex. 7. It is arguably easier to determine these equalities by considering only the monitor LTSs 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 $\neg\mathsf{blk}(\Omega, s)$, $\neg\mathsf{blk}(\Omega{+}\checkmark, s)$ and $\neg\mathsf{blk}(\mathsf{rec}\, X.(\tau.X{+}\checkmark), s)$.
2. We only have $\mathsf{fl}(\Omega, \epsilon)$, $\mathsf{fl}(\Omega + \checkmark, \epsilon)$ and $\mathsf{fl}(\mathsf{rec}\, X.(\tau.X + \checkmark), \epsilon)$.
3. Similarly, we only have $\mathsf{nd}(\Omega, \epsilon)$, $\mathsf{nd}(\Omega + \checkmark, \epsilon)$ and $\mathsf{nd}(\mathsf{rec}\, X.(\tau.X + \checkmark), \epsilon)$. ■

Remark 2. The alternative preorder in Def. 11 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. ■

The alternative transparency preorder, Def. 13 below, is defined in terms of *divergence refusals* which, in turn, rely on strong divergences from Def. 9. 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 those that are suppressed on a process when instrumented with the *resp.* monitor.

Definition 12 (Divergence Refusals).

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

Definition 13 (Alternative Transparency Preorder).

$$M \preceq_{\mathit{tr}} N \stackrel{\mathit{def}}{=} \forall s \cdot \mathit{dref}(N, s) \subseteq \mathit{dref}(M, s)$$

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

Example 14. Recall monitors $c!a.\mathsf{end}$ and $c!a.\mathsf{end} + c!a.\Omega$ from (15) of Ex. 8. The inequality $c!a.\mathsf{end} + c!a.\Omega \sqsubseteq_{\mathsf{tr}} c!a.\mathsf{end}$ follows trivially from Thm. 4, since $\forall s \cdot \mathsf{dref}(c!a.\mathsf{end}, s) = \emptyset$. The symmetric case, $c!a.\mathsf{end} \sqsubseteq_{\mathsf{tr}} c!a.\mathsf{end} + c!a.\Omega$, can also be readily repudiated by plying Thm. 4. Since $\mathsf{dref}((c!a.\mathsf{end}{+}c!a.\Omega), c!a.\epsilon) = \textsc{Act}$ (and $\mathsf{dref}(c!a.\mathsf{end}, c!a.\epsilon) = \emptyset$) we trivially obtain a violation of the set inclusion requirements of Def. 13. ■

Example 15. Recall again $\Omega + \checkmark$ and $\mathsf{rec}\, X.(\tau.X + \checkmark)$ from (12). We can differentiate between these monitors from a transparency perspective, and Thm. 4 permits us to do this with relative ease. In fact, whereas $\mathsf{dref}((\Omega + \checkmark), \epsilon) = \textsc{Act}$ (since $\Omega{+}\checkmark \xrightarrow{\tau} \Omega$ and $\mathsf{dref}(\Omega, \epsilon) = \textsc{Act}$) we have $\mathsf{dref}((\mathsf{rec}\, X.(\tau.X{+}\checkmark)), \epsilon) = \emptyset$; for all other traces $|s| \geq 1$ we obtain empty divergence refusal sets for both monitors. We thus can positively conclude that $\Omega + \checkmark \sqsubseteq_{\mathsf{tr}} \mathsf{rec}\, X.(\tau.X + \checkmark)$ while refuting $\mathsf{rec}\, X.(\tau.X + \checkmark) \sqsubseteq_{\mathsf{tr}} \Omega + \checkmark$. ■

Definition 14 (Alternative Monitor Preorder).

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

■

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

6 Conclusion

We have presented a theory for (recogniser) monitors that 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. The theory is *compositional*, since it enables us to ensure the preservation of properties by analysing the *resp.* monitors M_1 and M_2 *in isolation*, without needing to consider the process being monitored, P (which may be arbitrarily complex). To the best of our knowledge, it is the first monitor theory of its kind and could be used to alleviate efforts for proving monitors correct *e.g.,* [26]. The concrete contributions are:

1. The definition of three monitor preorders, each requiring the preservation of different monitoring properties: Def. 3, Def. 4 and Def. 5.
2. The characterisation of these preorders in terms of alternative preorders that are more tractable, Thm. 2, Thm. 3 and Thm. 4.

Related and Future Work. The instrumentation relation we consider is adopted from [17] and embodies *synchronous* instrumentation (where the external actions constituted the monitorable actions). Synchronous instrumentation is the most prevalent method used in monitoring tools (*e.g.,* [25, 9, 14, 1]) because it carries benefits such as timely detections. There are however variants such as asynchronous instrumentation (*e.g.,* [12, 20]) as well as hybrid variations (*e.g.,* [9, 30, 6, 7]). Our theory should be applicable, at least in part, to these variants.

In runtime verification, three-verdict monitors [2, 15, 10, 17] are often considered, where detections are partitioned into acceptances and rejections. The monitors studied here express generic detections only; they are nevertheless maximally expressive for branching-time properties [17]. They also facilitate comparisons with other linear-time preorders (see below). We also expect our theory to extend smoothly to settings with acceptances and rejections.

Our potential and deterministic detection preorders are reminiscent of the classical may and must preorders of [13, 23] and, more recently (for the deterministic detection preorder), of the subcontract relations in [3, 21]. However, these relations differ from ours in a number of respects. For starters, the monitor instrumentation relation of Fig. 2 assigns monitors a *passive* role whereas the parallel composition relation composing processes (servers in [3, 21]) with tests (clients in [3, 21]) invites tests to *interact* with the process being probed. Another important difference is that testing preorders typically relate processes, whereas our preorders are defined over the adjudicating entities *i.e.,* the monitors. The closest work in this regard is that of [3], where the authors develop a must theory for clients. Still, there are significant discrepancies between this must 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 [13, 23, 3] as outlined above — success in the compliance relation of [21] is even more disparate). Concretely, in our setting we have equalities such as $c!a.\checkmark \cong_{\mathsf{dd}} c!a.\checkmark + c!b.\mathsf{end}$ (see (10) of Ex. 6), which would not hold in the

setting of [3] since their client preorder is sensitive to external choices ($\sqsubseteq_{\mathsf{dd}}$ is not because monitored executions are distinguished by their visible trace). The two relations are in fact incomparable, since divergent processes are bottom elements in the client must preorder of [3], but they are not in $\sqsubseteq_{\mathsf{dd}}$. In fact, we have $\Omega \not\sqsubseteq_{\mathsf{dd}} \Omega + \mathsf{end}$ in (12) of Ex. 7 or, more clearly, $\Omega \not\sqsubseteq_{\mathsf{dd}} \Omega + \alpha.\mathsf{end}$; 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.

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 [28, 15, 5]. We adapted this concept to recognisers, whereby the process behaviour should never be suppressed by the monitor.

To our knowledge, the only body of work that studies monitoring for the piCalculus is [5, 11, 22], and focusses on synthesising adaptation/enforcement monitors from session types. The closest to our work is [5]: their definitions of monitor correctness are however distinctly different (*e.g.*, they are based on branching-time equivalences) and their decomposition methods for decoupling the monitor analysis from that of processes rely on static type-checking.

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

References

1. H. Barringer, Y. Falcone, K. Havelund, G. Reger, and D. E. Rydeheard. Quantified Event Automata: Towards Expressive and Efficient Runtime Monitors. In *FM*, volume 7436 of *LNCS*, pages 68–84. Springer, 2012.
2. A. Bauer, M. Leucker, and C. Schallhart. Runtime verification for LTL and TLTL. *TOSEM*, 20(4):14, 2011.
3. G. Bernardi and M. Hennessy. Mutually testing processes. *LMCS*, 11(2:1), 2015.
4. N. Bielova and F. Massacci. Do you really mean what you actually enforced?: Edited automata revisited. *Int. J. Inf. Secur.*, 10(4):239–254, Aug. 2011.
5. L. Bocchi, T.-C. Chen, R. Demangeon, K. Honda, and N. Yoshida. Monitoring networks through multiparty session types. In *FMOODS/FORTE 2013*, volume 7892 of *LNCS*, pages 50–65, 2013.
6. I. Cassar and A. Francalanza. On Synchronous and Asynchronous Monitor Instrumentation for Actor Systems. In *FOCLASA*, volume 175, pages 54–68, 2014.
7. I. Cassar and A. Francalanza. Runtime Adaptation for Actor Systems. In *RV*, volume 9333 of *LNCS*, pages 38–54. Springer, 2015.
8. F. Cesarini and S. Thompson. *Erlang Programming*. O'Reilly, 2009.
9. F. Chen and G. Roşu. MOP: An Efficient and Generic Runtime Verification Framework. In *OOPSLA*, pages 569–588. ACM, 2007.
10. C. Cini and A. Francalanza. An LTL Proof System for Runtime Verification. In *TACAS*, volume 9035, pages 581–595. Springer, 2015.
11. M. Coppo, M. Dezani-Ciancaglini, and B. Venneri. Self-adaptive monitors for multiparty sessions. In *PDP*, pages 688–696. IEEE Computer Society, 2014.
12. B. D'Angelo, S. Sankaranarayanan, C. Sánchez, W. Robinson, B. Finkbeiner, H. B. Sipma, S. Mehrotra, and Z. Manna. Lola: Runtime monitoring of synchronous systems. In *TIME*. IEEE, 2005.

13. R. De Nicola and M. C. B. Hennessy. Testing equivalences for processes. *TCS*, 34(1-2):83–133, 1984.
14. N. Decker, M. Leucker, and D. Thoma. jUnitRV - Adding Runtime Verification to jUnit. In *NASA FM*, volume 7871 of *LNCS*, pages 459–464. Springer, 2013.
15. Y. Falcone, J.-C. Fernandez, and L. Mounier. What can you verify and enforce at runtime? *STTT*, 14(3):349–382, 2012.
16. Formal Systems Laboratory. Monitor Oriented Programming. University of Illinois at Urbana Champaign. `http://fsl.cs.illinois.edu/index.php/Monitoring-Oriented_Programming`.
17. A. Francalanza, L. Aceto, and A. Ingólfsdóttir. On Verifying Hennessy-Milner Logic with Recursion at Runtime. In *RV*, volume 9333 of *LNCS*, pages 71–86. Springer, 2015.
18. A. Francalanza, A. Gauci, and G. J. Pace. Distributed System Contract Monitoring. *JLAP*, 82(5-7):186–215, 2013.
19. A. Francalanza and M. Hennessy. A Theory for Observational Fault Tolerance. *JLAP*, 73(12):22 – 50, 2007.
20. A. Francalanza and A. Seychell. Synthesising Correct concurrent Runtime Monitors. *FMSD*, 46(3):226–261, 2015.
21. N. G. Giuseppe Castagna and L. Padovani. A theory of contracts for web services. *ACM Trans. Program. Lang. Syst.*, 31(5), 2009.
22. C. D. Giusto and J. A. Perez. Disciplined Structured Communications with Disciplined Runtime Adaptation. *Sci. of Computer Programming*, 97(2):235–265, 2015.
23. M. Hennessy. *Algebraic Theory of Processes*. MIT Press, 1988.
24. M. Hennessy. *A Distributed Pi-Calculus*. Cambridge University Press, 2007.
25. M. Kim, M. Viswanathan, S. Kannan, I. Lee, and O. Sokolsky. Java-MaC: A run-time assurance approach for Java programs. *FMSD*, 24(2):129–155, 2004.
26. J. Laurent, A. Goodloe, and L. Pike. Assuring the Guardians. In *RV*, volume 9333 of *LNCS*, pages 87–101. Springer, 2015.
27. M. Leucker and C. Schallhart. A brief account of Runtime Verification. *JLAP*, 78(5):293 – 303, 2009.
28. J. Ligatti, L. Bauer, and D. Walker. Edit automata: enforcement mechanisms for run-time security policies. *Int. J. Inf. Secur.*, 4(1-2):2–16, 2005.
29. R. Milner. *Communication and concurrency*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1989.
30. G. Roşu and K. Havelund. Rewriting-based techniques for runtime verification. *Automated Software Engg.*, 12(2):151–197, Apr. 2005.
31. D. Sangiorgi and D. Walker. *PI-Calculus: A Theory of Mobile Processes*. Cambridge University Press, 2001.
32. F. B. Schneider. Enforceable security policies. *ACM Trans. Inf. Syst. Secur.*, 3(1):30–50, Feb. 2000.
33. K. Sen, A. Vardhan, G. Agha, and G. Rosu. Efficient Decentralized Monitoring of Safety in Distributed Systems. In *ICSE*, pages 418–427. IEEE, 2004.
34. P. Verissimo and L. Rodrigues. *Distributed Systems for System Architects*. Kluwer Academic Publishers, 2001.