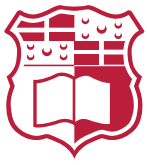
Refinement Preorders for Multi-Verdict Monitors

Adriana Baldacchino

Supervisor: Prof. Adrian Francalanza

June 2024

Submitted in partial fulfilment of the requirements
for the degree of B.Sc. (Hons) in Computing Science and Mathematics.



L-Università ta' Malta
Faculty of Information &
Communication Technology

Abstract

We study *multi-verdict monitors*, entities which execute alongside programs and analyse their behaviour in order to reach irrevocable judgements, referred to as *verdicts*. Monitors with multiple verdicts are used frequently for runtime verification, typically having acceptance and rejection verdicts which denote whether a program satisfies or violates a correctness property. We establish *refinement preorders* for these monitors, which allow us to determine whether a monitor preserves certain behavioural properties of another. This mechanism can be used when updating old monitors to new ones to determine that the updated monitor is at least as correct as the one it is replacing.

This framework is built up from an existing theory of refinement preorders for uni-verdict monitors. We adapt already established preorders to our system, and further investigate scenarios in which these preorders may give misleading results for a multi-verdict system. In particular, we consider monitor *inconsistency* which arises from detecting two conflicting verdicts for the same program execution. From this we further develop new preorders, which enforce consistency properties.

Our preorders are defined over all processes, which makes them a powerful tool for comparing monitors. However, this makes the task of establishing whether one monitor implements another computationally intensive, as this requires considering all processes. To circumvent this issue, we define alternative preorders which fully characterise our preorders, but are not quantified over all systems. By avoiding universal quantification over all process, these alternative preorders provide a tractable way to establish relationships between monitors.

Acknowledgements

I would like to thank my supervisor, Prof. Adrian Francalanza for his support and patience with me throughout this project, and for introducing me to this interesting problem. Moreover, I extend my thanks to the lovely people who taught me from the Faculty of ICT, for providing me a solid basis in computer science and programming. I would also like to thank Marietta, for her invaluable companionship throughout this course, and my family, without whom I would not have had this opportunity. Finally, I would like to thank my proofreader and best friend, Xandru.

Contents

1 Introduction

Monitors are computational entities which execute alongside a computer program so as to analyse its runtime behaviour, to determine correctness of the program. Once sufficient behaviour is observed, a monitor reaches an irrevocable judgement, referred to as a *verdict*. A monitoring setup typically consists of three main components: the program being monitored P, the *monitor* M and the *instrumentation* mechanism. This mechanism connects the execution of a program with the analysis performed by the monitor, determining how the monitor and program execute in relation to one other. The composition of these components $P \triangleleft M$ is referred to as the *monitored system*.

Monitors have been studied extensively, in particular for their application in *runtime verification* [1–6]. This lightweight verification technique checks whether a system's current execution satisfies or violates a given correctness property by generating monitors according to some formal specification. Monitors also play a major role in *monitor-oriented programming* (MOP) [7–10], in which software is organised as a layered architecture, with the original system at its core and monitors at the outer layers, which observe behaviour from the inner layers. In these systems, monitors play a more active role — modifying or suppressing the behaviour of the inner layers. This mechanism can be used in runtime enforcement, where monitors enforce that the system they are instrumented with always satisfies certain properties. This can also be used in runtime adaptation, in which monitors adapt the instrumented system to improve aspects of the system, such as efficiency and resilience.

In certain applications, it is enough to consider monitors which may only reach one verdict. We refer to these monitors as *uni-verdict* monitors. Uni-verdict monitors have been shown to be fully expressive for branching-time logic μHML when limiting to certain classes of properties [11]. However, in linear-time specifications, allowing for multiple verdicts increases the expressiveness of monitors [12]. Moreover, monitors utilising multiple verdicts, referred to as *multi-verdict* monitors are often utilised in runtime verification [3–6]. In this approach, programs are analysed in order to determine whether they satisfy or violate a property. For this purpose, the monitors used often have two verdicts: acceptance, denoting that the program satisfies the property being considered, and rejection, denoting that the program violates the property. Because of this, we broadly consider systems which use multiple verdicts.

Our aim in this work is to establish a framework for comparing and contrasting multi-verdict monitors. This should allow us to judge whenever one monitor can suitably be used in place of another, and judge whether or not two monitors behave the same given a set of criteria. To do this, we establish refinement preorders of the form $M \sqsubseteq N$. These allow us to establish whenever a monitor *extends* another, which is to say

implements the same behaviour when instrumented with any process P.

This approach to comparing computational entities is not new. Morris [13] first established the contextual refinement on lambda calculus terms. In this system, a process P extends Q, denoted $Q \leq P$ if for any context $C[\,]$, whenever $C[Q]$ returns a value, $C[P]$ must return the same value. This idea of refinements has been extended for various applications, including *testing refinements* [14, 15] which compare processes based on the tests they satisfy and *failure refinements* [16] which compare processes depending on the actions they fail to perform, and have various applications in verification [17–19].

Studying this type of relation is useful, as it allows us to replace one process with another with certain guarantees in place. This can for example be applied to program development, when an existing program is replaced by a new program, which should preserve the good properties of the existing program. If we know $Q \leq P$, then P is essentially stronger than Q, as it gives us the same answer as Q whenever Q returns a value, and can further return values when Q doesn't. However, using refinement preorders defined on processes may be limiting as it does not allow us to change the behaviour of processes.

Instead of studying processes directly, it would be ideal to compare entities which analyse processes. This approach was taken in [20, 21], where the authors establish client testing preorders which relate tests instead of processes. We take a similar approach, establishing refinement preorders on monitors instead of processes. We note that monitors differ significantly from tests, as they run alongside a process passively, while tests typically interact with the system, hence taking on a more active role. Nevertheless, parallels can be made as both tests and monitors are concerned with determining correctness properties of programs.

This approach is advantageous to us, as it allows us to compare processes which do not necessarily have the same behaviour but lead to the same verdicts when instrumented with a monitor. This approach is typically more efficient than comparing processes directly, as processes are often more complex than the monitors which analyse them. This study can also aid in development of better monitors to replace existing monitors while preserving the good behaviour of the extended monitor. In this case, the context in which we run our monitors becomes the processes being observed. This means that our contexts are of the form $C[\,] = P \triangleleft [\,]$. Applying this, we have that a monitor M *extends* another monitor N, denoted as $N \sqsubseteq M$ if whenever N reaches a verdict when instrumented with a process P, then M must reach the same verdict when instrumented process P.

Whenever we consider contextual preorders, we encounter one fundamental problem — the quantification over all contexts. To establish the contextual refinement $N \sqsubseteq M$ we must show that for every single context $C[\,] = P \triangleleft [\,]$, $C[M]$ reaches a verdict whenever $C[N]$ reaches a verdict. This means that we must show for every process P,

that instrumented with P, M reaches a verdict whenever N reaches a verdict instrumented with the same process. This is clearly unwieldy, as there are infinitely many contexts since there are infinitely many processes.

For this purpose, we define alternative preorders which rely wholly on the monitor itself, but fully characterise monitor preorders. We define and fully characterise our monitor preorders by building off of recent work concerning uni-verdict monitors [22]. In this work, preorders were developed for uni-verdict monitors and were then fully characterised by tractable alternative preorders. We adapt these previously established preorders, and analyse the suitability of these preorders for our multi-verdict system.

A notable consequence of considering multiple verdicts is that a monitor can be *inconsistent* — which means it can potentially detect different verdicts, even if instrumented with the same process which executes the exact same actions. Considering the application of monitors in runtime verification, this behaviour is clearly incorrect. If a monitor were to detect the acceptance and rejection verdict given the same process and observable actions, this would mean that the process' behaviour both satisfies and violates the considered property, which is a contradiction. This means we should attempt to avoid inconsistency as much as possible.

Since the already established preorders were defined for a uni-verdict system, they do not consider consistency. Consequentially, using only these preorders, we have that an inconsistent monitor may implement a consistent one. This is not ideal, as we would like to preserve good properties of the implemented monitor. For this purpose, we define new preorders which preserve consistency properties of monitors being implemented. These are designed to use alongside the already established preorders.

After defining our preorders, we still have to establish a tractable way to determine whether these preorders hold. As stated previously, we do this by defining alternative preorders which do not depend on all processes. We first adapt the already established preorders to fit into a multi-verdict system, noting that correctness of these alternative preorders follows from the proofs in [22]. Most notably, we define alternative preorders for our newly defined preorders and prove that these are both complete and sound — thereby fully characterising the defined preorders for a multi-verdict system. These alternative preorders allow us to compare monitors in a much more feasible manner.

Briefly put, in order to attain our goal of establishing a tractable way with which to compare our multi-verdict monitors, we must satisfy the following objectives:

(O1) Develop relevant criteria to compare multi-verdict monitors through examination of monitor behaviour in order to define contextual preorders.

(O2) Define tractable alternative preorders which do not depend on a universal quantification over all contexts, and prove that they fully characterise the preorders

established in (O1).

We justify the usefulness and expressiveness of the preorders in (O1) using a series of examples, which highlight the various use cases of the preorders. These examples are given as we are defining the preorders, as these examples motivate the definitions, as well as give the reader some familiarity with how these preorders work. Moreover, objective (O2) will be met by giving a formal mathematical proof. For these reasons, a separate evaluation section will not be included in this work, as we justify the fulfillment of these objectives throughout this work.

We proceed in Chapter 2 by defining the particular instrumentation system we will use in more detail, going through the process, monitor and instrumentation language. In Chapter 3 we fulfill objective (O1) by adapting the established uni-verdict monitors for multiple verdicts, and defining new monitor preorders through a series of examples. We further state characterisation results for these monitors in Chapter 4, and prove these results for our newly defined contextual preorders, fulfilling objective (O2).

2 Defining the monitored system

In specifying our monitored system, we follow closely the definitions in [22], noting whenever we modify definitions. We start in Section 2.1 by defining the process language, a version of piCalculus, which allows us to define programs. In Section 2.2, we further specify the language with which we define monitors, introducing multiple verdicts. Finally, we define the instrumentation language in Section 2.3, which specifies the way our processes and monitors run alongside each other.

2.1 Process language

For our process language, we use a standard version of the synchronous piCalculus as in [22], represented by Figure 2.1. We take channel names $a, b, c, d \in$ Chans, name variables $x, y, z, \cdots \in$ Vars and process variables $X, Y, \cdots \in$ PVars. We let $\mu \in \text{Act}_\tau$ range over all actions, including input actions $c?d$, output actions $c!d$ and a silent action, which we represent as τ. We let $\alpha \in$ Act represent all actions, except for the silent action τ. We also consider u, u' to be elements of Chans $\cup$ Vars.

The input $c?x.P$, recursion $\text{rec}\, X.P$ and scoping $\text{new}\, c.P$ constructs are taken to be binding constructs, binding the variable x, process variable X and channel c in P respectively. This allows us to talk about free and bound variables, denoting free and bound name variables as $\mathbf{fn}(P)$ and $\mathbf{bn}(P)$ and free and bound process variables as $\mathbf{fV}(P)$ and $\mathbf{bV}(P)$, letting $\mathbf{fv}(P) = \mathbf{fn}(P) \cup \mathbf{fV}(P)$ and $\mathbf{bv}(P) = \mathbf{bn}(P) \cup \mathbf{bV}(P)$. This further allows us to define substitution $P[u/x]$ as replacing every free occurrence of x in P with u. Note that we work up to alpha-conversion of bound variables.

We let $I \subseteq$ Chans denote an interface of names shared by both the processes and observer. The Labelled Transition System as described in Figure 2.1 allows us to reach judgements of the form $I \triangleright P \xrightarrow{\mu} P'$, where P is a closed term, and all free names in P are contained in I. These judgements denote that P is able to evolve into process P', while performing the action μ. In this scenario, P' denotes what is left to execute after action μ.

For example, if we consider a process that accepts an input on a channel c, then outputs that input on a different channel d, $P_1 = c?x.d!x.\text{nil}$, then if input b is sent over channel c, the process evolves into a process which outputs b on channel d, $d!b.\text{nil}$. In essence, P' becomes the continuation of the program. This would be derived by pIn, and written concretely as

$$I \triangleright c?x.d!x.\text{nil} \xrightarrow{c?b} d!b.\text{nil}.$$

As the interface I keeps track of the known names, it must also be updated given the actions taken. For example, in the above case after $c?b$ is observed, we are now

Syntax

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

Semantics

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

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

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

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

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

Figure 2.1 piCalculus syntax and semantics [22]

aware of a new channel b. We explicitly define the updated interface $\mathbf{aftr}(I, \mu)$ in I after a transition with action μ below.

Definition 2.1. (Interface evolution)

$$\mathbf{aftr}(I, \tau) \coloneqq I \qquad \mathbf{aftr}(I, c?d) \coloneqq I \cup \{d\} \qquad \mathbf{aftr}(I, c!d) \coloneqq I \cup \{d\}$$

It would be useful to keep track of how our systems evolves not just after one action, but after a sequence of actions. We use variables $s, t \in \mathsf{Act}^*$ to denote traces of external actions. Our traces exclude the silent action, as its occurrence does not appear to the observer. We can further extend interface evolution to traces inductively, as $\mathbf{aftr}(I, \alpha s) \coloneqq \mathbf{aftr}(\mathbf{aftr}(I, \alpha), s)$ and $\mathbf{aftr}(I, \varnothing) \coloneqq I$. We denote two successive transitions $I \triangleright P \xrightarrow{\mu_1} P_1$ and $\mathbf{aftr}(I, \mu_1) \triangleright P_1 \xrightarrow{\mu_2} P_2$ as $I \triangleright P \xrightarrow{\mu_1} \mathbf{aftr}(I, \mu_1) \triangleright P_1 \xrightarrow{\mu_2} P_2$. Moreover, we define an s-execution $I \triangleright P \xRightarrow{s} Q$ as the sequence of transitions

$$I_0 \triangleright P_0 \xrightarrow{\mu_1} I_1 \triangleright P_1 \xrightarrow{\mu_2} I_2 \triangleright P_2 \xrightarrow{\mu_3} I_3 \triangleright P_3 \ldots \xrightarrow{\mu_n} P_n$$

where $I_i = \mathbf{aftr}(I_{i-1}, \mu_{i-1})$ for $1 \leq i < n$, $I_0 = I$, $P_0 = P$, $P_n = Q$ and $s = \mu_1 \mu_2 \ldots \mu_n$,

after filtering out the silent actions.

Example 2.2. If we consider the same process as before $P_1 = c?x.d!x.\mathsf{nil}$, we know that $I \triangleright c?x.d!x.\mathsf{nil} \xrightarrow{c?b} d!b.\mathsf{nil}$ by pIn. Moreover, by rule pOut, $I \cup \{b\} \triangleright d!b.\mathsf{nil} \xrightarrow{d!b} \mathsf{nil}$. Hence combining the two together, we get the following judgement

$$I \triangleright c?x.d!x.\mathsf{nil} \xRightarrow{c?b.d!b} \mathsf{nil}$$

denoting that our process P_1 terminates (reaches nil) after receiving an input of b over channel c, and sending b over channel d.

Example 2.3. This reasoning can be extended to more complicated programs. Consider for example the process $P_2 = \mathsf{rec}\, X.c?x.(\mathsf{if}\ x = a\ \mathsf{then}\ \mathsf{nil}\ \mathsf{else}\ X)$, which accepts input indefinitely, terminating only if given a. We consider the interface I where $a, c \in I$.

By pRec, $I \triangleright P_2 \xrightarrow{\tau} c?x.(\mathsf{if}\ x = a\ \mathsf{then}\ \mathsf{nil}\ \mathsf{else}\ P_2) = P'$. Then P' can transition given either action $c?a$ or action $c?b$ where $b \neq a$. In the first case,

$$I \triangleright P' \xrightarrow{c?a} I \triangleright \mathsf{if}\ a = a\ \mathsf{then}\ \mathsf{nil}\ \mathsf{else}\ P_2 \xrightarrow{\tau} \mathsf{nil}$$

by pIn and pThn, and in the second case

$$I \triangleright P' \xrightarrow{c?b} I \cup \{b\} \triangleright \mathsf{if}\ b = a\ \mathsf{then}\ \mathsf{nil}\ \mathsf{else}\ P_2 \xrightarrow{\tau} P_2$$

by pIn and pEls. Combining these judgements together and filtering out the silent actions, we get that $I \triangleright P_2 \xRightarrow{s} \mathsf{nil}$, whenever $s = \mu_1\mu_2 \ldots \mu_n$ with $\mu_i = c?b_i$ for $1 \leq i < n$, $b_i \neq a$, and $\mu_n = c?a$.

2.2 Monitor language

We describe monitors $M, N \in \mathsf{Mon}$ by the syntax and semantics in Figure 2.2. We consider Verd to be a non-empty set of verdicts and take $w \in \mathsf{Verd} \cup \{\mathsf{end}\}$, where end is the inconclusive verdict. This differs from the original theory, as we consider any arbitrary set of verdicts instead of only the detection verdict ✓ and the inconclusive verdict end. Note that the inconclusive verdict is not contained in Verd in order to simplify notation at a later stage when defining our preorders.

We make use of patterns $p, q \in \mathsf{Pat}$, which range over input and output actions containing channel names, name variables, or variable binders of the form $(x), (y) \in \mathsf{Binds}$ which allow for more flexible matching, and are set to fixed values when they are matched. To provide a way to match patterns, we make use of function $\mathsf{match}(p, \alpha)$ that takes a closed pattern and an action and returns a substitution, as defined Definition A.1 in the appendix.

Syntax

$$p, q \in \mathsf{Pat} ::= o?r \quad \textit{(input pattern)} \qquad | \; o!r \quad \textit{(output pattern)}$$

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

Monitor Semantics

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

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

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

Figure 2.2 Monitor syntax, semantics [22]

Example 2.4. Consider the monitor $M_1 = c?(x).(\mathsf{if}\ x = a\ \mathsf{then\ yes\ else\ no})$, where $\mathsf{Verd} = \{\mathsf{yes}, \mathsf{no}\}$. M_1 is expected to return the positive verdict yes if it observes a as input over c, and the negative verdict no if it observes any other input over c. This is confirmed by the transition semantics. Considering action $c?b$,

$$\mathsf{match}(c?(x), c?b) = \mathsf{match}(c, c) \cup \mathsf{match}((x), b) = \varnothing \cup [{}^{b}/_{x}] = [{}^{b}/_{x}]$$

hence by mPat, $M_1 \xrightarrow{c?b}$ if $b = a\ \mathsf{then\ yes\ else\ no}$. If $b = a$, then by mThn, if $b = a\ \mathsf{then\ yes\ else\ no} \xrightarrow{\tau}$ yes, otherwise by mEls if $b = a\ \mathsf{then\ yes\ else\ no} \xrightarrow{\tau}$ no. At this point, if the monitor observes any more actions it can transition keeping the same verdict by mVer. This behaviour is observed by any verdict, and reflects the fact that verdicts are *irrevocable*, that is after a verdict is reached it is final.

As before, we reach judgements of the form $M \xrightarrow{\alpha} M'$, denoting that the monitor M evolves into the monitor M' when observing an action α. A monitor can also transition silently, denoted as $M \xrightarrow{\tau} M'$, independently of any actions observed. What we mean by observing an action is unclear in this section, as we have not yet defined how the monitor and process interact with each other. This behaviour is described in Section 2.3, where we specify the language for our instrumented system.

Instrumented System Semantics

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

$$\text{iTer}\ \frac{I \triangleright P \xrightarrow{\alpha} P' \quad M \xrightarrow{\alpha}\!\!\!\!\!/ \quad M \xrightarrow{\tau}\!\!\!\!\!/}{I \triangleright P \triangleleft M \xrightarrow{\alpha} P' \triangleleft \mathsf{end}} \qquad \text{iAsyM}\ \frac{M \xrightarrow{\tau} M'}{I \triangleright P \triangleleft M \xrightarrow{\tau} P \triangleleft M'}$$

Figure 2.3 Instrumentation Semantics [22]

2.3 Instrumented system semantics

We further define the instrumentation for the monitored system $P \triangleleft M$ by the labelled transition system in Figure 2.3. As before, we use the inferface I for bookkeeping. In this system, the monitor and process may transition simultaneously if they both transition with the same action, as specified in iMon.

The monitor and the process may also transition silently at any time by rules iAsyM and iAsyP respectively, without changing the execution of the other. In the case where a monitor cannot silently transition or match the action being executed by the process, by iTer it reaches the inconclusive verdict end.

We extend the concept of an s-execution to a *monitored s-execution* in the natural way, denoting two successive transitions $I \triangleright P \triangleleft M \xrightarrow{\mu_1} P_1 \triangleleft M_1$ and $\mathbf{aftr}(I, \mu_1) \triangleright P_1 \triangleleft M_1 \xrightarrow{\mu_2} P_2 \triangleleft M_2$ as $I \triangleright P \triangleleft M \xrightarrow{\mu_1} \mathbf{aftr}(I, \mu_1) \triangleright P_1 \triangleleft M_1 \xrightarrow{\mu_2} P_2 \triangleleft M_2$, and letting a *monitored s-execution* $I \triangleright P \triangleleft M \stackrel{s}{\Longrightarrow} Q \triangleleft N$ denote the sequence of transitions

$$I_0 \triangleright P_0 \triangleleft M_0 \xrightarrow{\mu_1} I_1 \triangleright P_1 \triangleleft M_1 \xrightarrow{\mu_2} I_2 \triangleright P_2 \triangleleft M_3 \xrightarrow{\mu_3} \ldots \xrightarrow{\mu_n} P_n \triangleleft M_n$$

where $I_i = \mathbf{aftr}(I_{i-1}, \mu_{i-1})$ for $1 \leq i < n$, $I_0 = I$, $P_0 = P$, $M_0 = M$, $P_n = Q$, $M_n = N$, and $s = \mu_1 \mu_2 \ldots \mu_n$, after filtering out the silent actions.

Example 2.5. Consider again the process $P_1 = c?x.d!x.\mathsf{nil}$ and monitor $M_1 = c?(x).\mathsf{if}\ x = a\ \mathsf{then\ yes\ else\ no}$. We know how these monitors transition individually, hence letting $I = \{a, c, d\}$ we can put this together alongside the instrumented system semantics to obtain

$$\begin{aligned} I \triangleright P_1 \triangleleft M_1 &\xrightarrow{c?b} I \cup \{b\} \triangleright d!b.\mathsf{nil} \triangleleft \mathsf{if}\ b = a\ \mathsf{then\ yes\ else\ no} \\ &\xrightarrow{\tau} I \cup \{b\} \triangleright d!b.\mathsf{nil} \triangleleft w \\ &\xrightarrow{d!b} \mathsf{nil} \triangleleft w, \end{aligned}$$

hence $I \triangleright P_1 \triangleleft M_1 \xrightarrow{c?b.d!b} \mathsf{nil} \triangleleft w$, where $w = \mathsf{yes}$ if $b = a$ and $w = \mathsf{no}$ otherwise. The first line follows by iMon as both the monitor and process transition given action $c?b$,

the second follows by iAsyM, as the monitor transitions silently based on b, and the last follows by iMon, noting that since w is a verdict w transitions given any action back to w.

Example 2.6. If instead of P_1 we had a process that begins with the output action, the monitor M_1 would reach the inconclusive verdict end, as it does not specify how to proceed in this scenario. For example considering a simple process $P_3 = c!a.\mathsf{nil}$, we have that $I \triangleright c!a.\mathsf{nil} \xrightarrow{c!a} \mathsf{nil}$, but $M_1 \not\xrightarrow{c!a}$ and $M_1 \not\xrightarrow{\tau}$, hence by iTer,

$$I \triangleright c!a.\mathsf{nil} \triangleleft M_1 \xrightarrow{c!a} \mathsf{nil} \triangleleft \mathsf{end}.$$

3 Defining Monitor preorders

After establishing the foundations of our system, we can now start tackling objective (O1) — the problem of establishing and motivating preorders for our multi-verdict monitors. We do this partly by adapting previous work and by defining new preorders based on the added challenges of a multi-verdict system.

3.1 Extending existing preorders

In this section, we take a closer look at the monitor preorders established for a uni-verdict context [22], in order to find proper adaptations for our multi-verdict system. To do so, we must first develop some terminology which can help us describe our monitors.

We have previously described s-executions of a monitored system $I \triangleright P \triangleleft M$. We highlight the fact that executions are indexed by a trace s, which describes the external actions that the monitor can observe. This is because a monitor will reach judgments on the process based on the actions it executes, hence its execution depends highly on the actions it observes. Given this, it makes sense to consider executions in light of the observable actions of the process.

We further motivate the necessity for a *maximal* s-execution, which we will refer to as an *s-computation*.

Example 3.1. Consider again the process $P_2 = \mathsf{rec}\, X.c?x.\mathsf{if}\ x = a\ \mathsf{then}\ \mathsf{nil}\ \mathsf{else}\ X$ and monitor $M_1 = c?(x).(\mathsf{if}\ x = a\ \mathsf{then}\ \mathsf{yes}\ \mathsf{else}\ \mathsf{no})$ from Examples 2.3 and 2.4, and trace $s = c?a$. Clearly, the following equations

$$I \triangleright P_2 \triangleleft M_1 \stackrel{s}{\Longrightarrow} \mathsf{nil} \triangleleft \mathsf{if}\ a = a\ \mathsf{then}\ \mathsf{yes}\ \mathsf{else}\ \mathsf{no} \tag{3.1}$$

$$I \triangleright P_2 \triangleleft M_1 \stackrel{s}{\Longrightarrow} \mathsf{nil} \triangleleft \mathsf{yes} \tag{3.2}$$

are both valid s-executions, but Equation 3.2 is more 'advanced' than 3.1 as

$$\mathsf{if}\ a = a\ \mathsf{then}\ \mathsf{yes}\ \mathsf{else}\ \mathsf{no} \xrightarrow{\tau} \mathsf{yes}.$$

Because of this, we would like to focus on s-executions which cannot be extended further using τ-transitions.

Example 3.2. It is possible that a monitored system can always be extended further using τ-transitions, hence we need to also consider this case when defining a maximal execution. Take for example the monitors Ω, Ω' defined as

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

Then, it is clear to see that $\Omega \xrightarrow{\tau} \Omega'$ and $\Omega' \xrightarrow{\tau} \Omega$, meaning that Ω can transition silently forever. Because of this property, we know there exist s-executions which can always be extended using τ-transitions. In this case, as we want our s-execution to be maximal, an infinite sequence of transitions must be included.

Putting these two observations together, we obtain the definition of a maximal s-execution, which leads to our definition of an s-computation. Moreover, we want to be able to discuss whether an s-computation *detects* a verdict w, which we do so by specifying that at some point in the transition sequence, the monitor reaches the verdict w.

Definition 3.3 (s-computation)**.** The transition sequence

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

is called an s-computation if it is *maximal* (*i.e.* if it is either infinite or finite and cannot be extended further using τ-transitions). Given a verdict $w \in$ Verd, an s-computation is called w-detecting if $\exists n \in \mathbb{N} \cdot M_n = w$. In this case we say that M detects w along trace s.

After having established the proper terminology, we now consider the already established preorders: *potential detection*, *deterministic detection* and *transparency*. As hinted at by their name, the first two depend on the detecting computations we have just defined. The last preorder, transparency, takes into account the fact that a program may end up in an endless loop, like our monitors in Example 3.2. Because of the different nature of these preorders, we first consider the detection preorders, then the transparency preorder. We further explore the limitations of these preorders in a multi-verdict context, establishing a need for more specialised monitors.

3.1.1 Detection preorders

There are two types of detection preorders: potential and deterministic. This reflects the non-deterministic properties of our process and monitor languages, specifically the parallel $P \parallel Q$ and choice $M + N$ operators. We illustrate this using a simple example.

Example 3.4. Let Verd $= \{\checkmark\}$ and consider the process $P_3 = c!a.\mathsf{nil}$ and the monitors $M_2 = c!a.\checkmark$ and $M_3 = c!a.\checkmark + c!a.\mathsf{end}$. In this case, with trace $s = c!a$ the only s-computation starting from $I \triangleright P_3 \triangleleft M_2$ detects the verdict $\checkmark$. However, considering the monitor M_3, both

$$I \triangleright P_3 \triangleleft M_3 \xRightarrow{s} \mathsf{nil} \triangleleft \checkmark \quad \text{and} \quad I \triangleright P_3 \triangleleft M_3 \xRightarrow{s} \mathsf{nil} \triangleleft \mathsf{end}$$

are valid s-computations, but only one of these detects $\checkmark$.

Because of this, it makes sense to differentiate those monitors that detect a verdict in some s-computations, and those that detect a verdict in every s-computation. Before we define potential and deterministic detection, it is first necessary to reconcile the definition of detecting computations in a uni-verdict context with our w-detecting computations. In this case, since Verd $= \{\checkmark\}$, an s-computation either detects or not. However, for our more general application, we need to distinguish *which* verdict our computation detects. To do this, we tie the definitions of potential and deterministic detection to a verdict w.

Definition 3.5 (Potential and deterministic w-detection)**.**

1. M *potentially detects* w for $I \triangleright P$ along trace s, denoted as $\mathsf{pd}_w(M, I, P, s)$, if and only if there *exists* a w-detecting s-computation from $I \triangleright P \triangleleft M$.
2. M *deterministically detects* w for $I \triangleright P$ along trace s, denoted as $\mathsf{dd}_w(M, I, P, s)$, if and only if *all* s-computations from $I \triangleright P \triangleleft M$ detect w.

We note that in a uni-verdict context the predicates $\mathsf{pd}_\checkmark, \mathsf{dd}_\checkmark$ are equivalent to the predicates pd, dd respectively in [22]. This definition leads to a fairly straightforward preorder: N implements M if whenever M detects the verdict w, N detects the verdict w. Because detection depends on the verdict, we quantify this over every $w \in$ Verd. This quantification does not introduce any significant overhead, as we take Verd to be a finite set.

Definition 3.6 (Potential and Deterministic Detection Preorders)**.**

1. $M \sqsubseteq_{\mathsf{pd}} N \coloneqq \forall I, P, s, w \cdot \mathsf{pd}_w(M, I, P, s)$ implies $\mathsf{pd}_w(N, I, P, s)$.
2. $M \sqsubseteq_{\mathsf{dd}} N \coloneqq \forall I, P, s, w \cdot \mathsf{dd}_w(M, I, P, s)$ implies $\mathsf{dd}_w(N, I, P, s)$.

Remark 3.7. When using the preorder $M \sqsubseteq N$, it is handy to define some notation for when we have equality or strict inequality. We consider the following shorthand:

- $M \not\sqsubseteq N$ denotes the negation of $M \sqsubseteq N$.
- $M \cong N$ denotes equality, that is $M \sqsubseteq N$ and $N \sqsubseteq M$.
- $M \sqsubset N$ denote strict inequality, that is $M \sqsubseteq N$ but $N \not\sqsubseteq M$.

Moreover, we will refer to *top* and *bottom* elements of our preorder. The monitor M is a *top (respectively, bottom)* element of the preorder $\sqsubseteq$ if for every monitor N, $M \not\sqsubset N$ (respectively, $N \not\sqsubset M$).

This means that our top elements are those that cannot be extended further. In the context of our preorders being used to create better and more advanced monitors,

this makes our top elements the ideal monitors, which achieve absolute correctness. On the flip side, our bottom elements are those which cannot strictly implement any monitor — meaning they will be the least useful monitors according to the specified behaviour. Referring to these monitors is helpful, as it gives us an idea of what our preorder is measuring.

3.1.2 Transparency preorder

Another aspect of monitor behaviour that we consider is whether they can become unresponsive and get stuck in infinite loops, hindering the advancement of the process they are instrumented with. As most programmers can attest to, unresponsive divergent programs arise frequently when programming, whether intentional or not, hence we would like to consider this behaviour in our comparison of monitors.

Our monitors can also keep executing forever using τ-transitions, as seen in Example 3.2. In this case, we say that a monitor *diverges*. This changes the way the instrumented system behaves, stopping the execution of the process, as the rule iTer cannot be applied. We present an example of another divergent monitor, highlighting how this behaviour affects our instrumented system.

Example 3.8. We consider again the monitor $M_2 = c!a.\checkmark$ and define the monitor $M_4 = \mathsf{rec}\, X.(X + c!a.\checkmark)$. These two monitors detect in exactly the same way, as in particular for any process that performs the action $c!a$ and trace starting with $c!a$, every s-computation instrumented with either monitor detects $\checkmark$.

However, they behave differently when instrumented with certain processes. Take for example process $P_4 = c!b.\mathsf{nil}$. When instrumented with M_2, we clearly get by iTer that

$$I \triangleright P_4 \triangleleft M_2 \xrightarrow{c!b} \mathsf{nil} \triangleleft \mathsf{end}.$$

However as $M_4 \xrightarrow{\tau} M_4 + c!a.\checkmark \xrightarrow{\tau} M_4$, the instrumented system $P_4 \triangleleft M_4$ is unable to transition on action $c!b$. This is because we cannot apply iMon, as M_4 and $M_4 + c!a.\checkmark$ do not transition with the action $c!a$, however we still cannot apply iTer, since these monitors transition silently, hence the condition $M \not\xrightarrow{\tau}$ is not met.

We note that while the divergence of M_4 may be obvious in this example, figuring out when a monitor diverges is not so clear cut when dealing with more complex monitors. This leads us to specify the *transparency* predicate, which is satisfied whenever an instrumented system is unable to transition with some trace s only if the process itself cannot transition with trace s. It is evident that this behaviour is independent to the number of verdicts of the system in question, hence we can safely take the exact definition from [22]. This definition states that a monitor M is transparent given a process P

and a trace s if whenever the instrumented system $I \triangleright P \triangleleft M$ fails to perform an external action, this is because the process itself fails to perform the action.

Definition 3.9 (Transparency Preorder)**.** M is *transparent* for $I \triangleright P$ along trace s, denoted as $\mathsf{tr}(M, I, P, s)$, if and only if

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

We further define the induced preorder as expected:

$$M \sqsubseteq_{\mathsf{tr}} N \coloneqq \forall I, P, s \cdot \ \mathsf{tr}(M, I, P, s) \text{ implies } \mathsf{tr}(N, I, P, s).$$

Example 3.10. We consider again $M_4 = \mathsf{rec}\, X.(X + c!a.\checkmark)$ and $P_4 = c!b.\mathsf{nil}$ defined in Example 3.8 in light of this new definition. We saw in the previous example how our instrumented system fails to transition because M_4 can transition silently forever, and so we would expect this system to fail at being transparent for some trace.

Confirming our intuition, for the empty trace ϵ, we have that M_4 is not transparent for $I \triangleright P_4$ along trace ϵ, as we have that $I \triangleright P_4 \triangleleft M_4 \stackrel{c!a}{\not\Longrightarrow}$, but $I \triangleright P_4 \stackrel{c!a}{\Longrightarrow}$, hence $\mathsf{tr}(M_4, P_4, I, \epsilon)$ does not hold.

3.1.3 Considering existing preorders in a multi-verdict setting

The previously defined preorders are highly effective in describing relationships between monitors in a uni-verdict setting, as shown in [22]. We further consider the behaviour of these preorders in a multi-verdict context in order to explore how they behave in this context and identify weaknesses of the established theory.

We first highlight how transparency remains the same in this context. Regardless of the number of verdicts considered in the system, divergent behaviour remains the same. It follows that the alternative preorder for transparency should also be the same as the one given in [22]. Because of this, we need not consider this preorder any further as it is already fully specified for our multi-verdict context, and we do not need to consider divergent behaviour any further.

The situation is different for the detecting preorders. In a multi-verdict context, we encounter situations in which potential and deterministic detection lead to counter-intuitive results. These scenarios involve *inconsistent monitors*, which may detect multiple verdicts given the same trace and process. We illustrate these deficiencies of our previously defined preorders by a series of examples.

Example 3.11. We consider $\mathsf{Verd} = \{\mathsf{yes}, \mathsf{no}\}$ and the monitors

$$M_{\mathsf{all}} = \mathsf{yes} + \mathsf{no} \quad \text{and} \quad M_5 = c!a.\mathsf{no} + \mathsf{end}.$$

Considering M_{all}, let $s = \alpha.t$ be a non-empty trace and consider a process P such that $P \stackrel{s}{\Longrightarrow} Q$. Then we have that by mVer, no $\stackrel{\alpha}{\longrightarrow}$ no, and hence by mChR we get that $M_{\mathsf{all}} \stackrel{\alpha}{\longrightarrow}$ no. From this, we get that $I \triangleright P \triangleleft M_{\mathsf{all}} \stackrel{s}{\Longrightarrow}$ no, and hence $\mathsf{pd}_{\mathsf{no}}(M_{\mathsf{all}}, I, P, s)$. We also can reason that M_5 only ever potentially detects the verdict no given a process that can perform the action $c!a$. Hence, we can deduce that M_{all} extends the monitor M_5 with respect to potential detection, that is $M_5 \sqsubseteq_{\mathsf{pd}} M_{\mathsf{all}}$.

Using the same reasoning as above, we have that for $s \neq \epsilon$, $P \stackrel{s}{\Longrightarrow} Q$, $I \triangleright P \triangleleft M_{\mathsf{all}} \stackrel{s}{\Longrightarrow}$ yes. Because we have both a yes-detecting s-computation here and a no-detecting s-computation, M_{all} does not deterministically detect given this process and trace. Neither does M_5, as $M_5 \stackrel{s}{\Longrightarrow}$ end, hence $\mathsf{dd}_w(M_5, I, P, s)$ does not hold. Both monitors will deterministically detect when given the empty trace ϵ, as they do not detect at all. Hence putting everything together we have that $M_5 \cong_{\mathsf{dd}} M_{\mathsf{all}}$, that is M_5 and M_{all} are equal with respect to deterministic detection.

Hence considering only potential and deterministic detection, M_{all} successfully implements M_5. However, if we consider $P_3 = c!a.\mathsf{nil}$ and $s = c!a$, we get a different picture. Instrumenting this process with M_{all}, we get

$$I \triangleright P_3 \triangleleft M_{\mathsf{all}} \stackrel{c!a}{\Longrightarrow} \mathsf{nil} \triangleleft \mathsf{yes} \quad \text{and} \quad I \triangleright P_3 \triangleleft M_{\mathsf{all}} \stackrel{c!a}{\Longrightarrow} \mathsf{nil} \triangleleft \mathsf{no}.$$

In runtime verification, we typically use the acceptance and rejection verdicts yes and no to denote whether a program belongs to the set of acceptable programs, or the set of erroneous programs. These sets are mutually exclusive, as you cannot say a program is correct and incorrect at the same time. In this context, detecting two different verdicts is conflicting. Examining this example through this light, we see that the behaviour of M_{all} is incorrect, as this monitor tells us to both accept and reject P_3.

The same problem is not encountered with M_5. Instrumenting P_3 with M_5 we get

$$I \triangleright P_3 \triangleleft M_5 \stackrel{c!a}{\Longrightarrow} \mathsf{nil} \triangleleft \mathsf{no} \quad \text{and} \quad I \triangleright P_3 \triangleleft M_5 \stackrel{c!a}{\Longrightarrow} \mathsf{nil} \triangleleft \mathsf{end}.$$

This means that sometimes M_5 tells us to reject P_3, and sometimes it does not come to a conclusive verdict. While M_5 is an unreliable monitor, in the sense that it may or may not reach a conclusive verdict, it is not an inconsistent monitor, as whenever it reaches a verdict it is always the same one. Considering this behaviour, we can claim that M_5 is more correct than M_{all}, as it does not give contradicting answers. When defining our preorders, we would like some way to preserve the correctness of the monitor being implemented in terms of consistency.

One way we could circumvent this issue completely is restricting our analysis to *consistent* monitors, that is monitors which given a trace, can only detect one verdict. We formally define these monitors as below.

Definition 3.12. (Monitor Consistency). A monitor M is *consistent* when there is no finite trace s and $w, v \in$ Verd such that $w \neq v$, $M \stackrel{s}{\Longrightarrow} w$ and $M \stackrel{s}{\Longrightarrow} v$.

This approach would work, and in this case our already defined preorders would suffice for a good description of the system. However, this approach greatly restricts the number of monitors we can formally analyse. In complex systems involving multiple people writing code, it is common to end up with some unexpected behaviour, which may include erroneous monitors. Inconsistency can easily arise in real world complex systems, hence we must define our preorders taking inconsistent monitors into account.

Instead of ignoring inconsistent monitors, we will define our preorders to enforce certain consistent behaviour. This will mean that whenever one monitor implements another, it is at least as consistent as the monitor it is implementing, and potentially more consistent. In this way, we define a method to quantify improvement of consistency of a program. This is especially useful in light of the main aim of these preorders: the development of better monitors to replace old ones, which preserve all the desired behaviours.

3.2 Defining consistency preorders

To analyse and compare the consistency of monitors, we introduce new preorders that depend on this concept. We first introduce a maximally inconsistent monitor $M_\perp$. This will act as a simple check for our established preorders, as the most inconsistent monitor should always be a bottom element for any preorder enforcing consistency properties.

Example 3.13. We define the always inconsistent monitor $M_\perp$ as

$$M_\perp = (\text{if } c{=}c \text{ then yes else nil}) + (\text{if } c{=}c \text{ then no else nil}).$$

Given any trace s, this monitor can always reach the verdict yes. This is because by mThn, if $c = c$ then yes else nil $\xrightarrow{\tau}$ yes, and hence by mChL, $M_\perp \xrightarrow{\tau}$ yes. Then, from mVerd we obtain yes $\stackrel{s}{\Longrightarrow}$ yes, hence putting everything together we get that $M_\perp \stackrel{s}{\Longrightarrow}$ yes. Similarly, we obtain that $M_\perp \stackrel{s}{\Longrightarrow}$ no, using mChR.

We note that for any verdict $w \in \{\text{yes}, \text{no}\}$ and any process P such that $P \stackrel{s}{\Longrightarrow} Q$, we can obtain an s-computation from $I \triangleright P \triangleleft M_\perp$ which detects w. We do this by first applying iAsyM, to get $I \triangleright P \triangleleft M_\perp \xrightarrow{\tau} P \triangleleft w$, then applying iMon and iAsyP as needed to get an s-computation.

In the above example, we do not use $M_{\text{all}} =$ yes $+$ no, as M_{all} does not fit the criteria of being maximally inconsistent. This is because it does not detect given the empty trace $s = \epsilon$ because the rule mChL (mChR, respectively) can only be applied if

there is a transition from yes (no, respectively), and verdicts do not transition with the silent action. Hence we needed to introduce the branching constructor, which allows for a τ-transition to the verdict, allowing mChL or mChR to be applied.

We further introduce a set of monitors which we use throughout this section in order to highlight the different aspects of consistency we will consider.

Example 3.14. We consider Verd $= \{\text{yes}, \text{no}\}$ and the monitors defined below.

$$M_{\text{all}} = \text{yes} + \text{no} \qquad M_7 = c!a.\text{yes} + c!a.\text{no} + c?b.\text{no}$$
$$M_6 = c?b.\text{no} \qquad M_8 = c!a.\text{yes} + c!a.\text{no} + c?b.\text{yes} + c?b.\text{no}$$

Clearly, these can be related using potential detection as $M_6 \sqsubset_{\text{pd}} M_7 \sqsubset_{\text{pd}} M_8 \sqsubset_{\text{pd}} M_{\text{all}}$. We encounter the same issue as in Example 3.11. Considering the monitors M_6 and M_8, M_8 implements M_6. However M_6 would reject a process that produces the action $c?b$, given a trace starting with $c?b$, but M_8 might accept this process given the same trace. The same problem occurs with M_7 and M_8 for the trace $c?b$.

In general, when considering potential detection, an inconsistent monitor can implement a consistent one. This is most obvious in the case of M_{all}, which can implement any monitor, so long as it does not reach any verdicts when given the empty trace. In this example, we also see that M_6, a consistent monitor, is implemented by the inconsistent monitors M_7, M_8 and M_{all}.

We can avoid this problem by specifying that if M implements N and N reaches only one verdict given a process and a trace, M must also reach only that same verdict. Using this reasoning, M_8 would not implement M_6 properly since M_6 only detects the verdict no given a process that performs the action $c?b$ and a trace starting with $c?b$, but M_8 would detect both verdicts yes and no. This is the basis on which we define *potential consistent detection*.

3.2.1 Potential consistent detection preorder

For this preorder, we differentiate between detecting a verdict w and consistently detecting a verdict w, which means detecting *only* the verdict w. This is a generalisation of the concept of consistency as in Definition 3.12, which allows us to consider consistency given a certain process and trace.

Definition 3.15. M *potentially consistently detects* w for $I \triangleright P$ along trace s, denoted as $\text{pcd}_w(M, I, P, s)$, if and only if there exists a w-detecting s-computation from $I \triangleright P \triangleleft M$, and for every $v \in \text{Verd}\backslash\{w\}$, there does not exist a v-detecting s-computation from

$I \triangleright P \triangleleft M$. More concisely,

$$\mathsf{pcd}_w(M, I, P, s) := \begin{cases} I \triangleright P \triangleleft M \overset{s}{\Longrightarrow} w \quad \wedge \\ \forall v \in \mathsf{Verd} \backslash \{w\} \cdot \neg(I \triangleright P \triangleleft M \overset{s}{\Longrightarrow} v) \end{cases}$$

Definition 3.16 (Potential consistent detection preorder)**.**

$$M \sqsubseteq_{\mathsf{pcd}} N := \forall I, P, s, w \cdot \mathsf{pcd}_w(M, I, P, s) \text{ implies } \mathsf{pcd}_w(N, I, P, s).$$

Remark 3.17. As desired, $M_\perp$ is indeed a bottom element of this preorder. The predicate $\mathsf{pcd}_{\mathsf{yes}}(M, I, P, s)$ does not hold because $I \triangleright P \triangleleft M \overset{s}{\Longrightarrow} \mathsf{no}$ is true, and hence the second condition of potential consistent detection does not hold. Moreover, $\mathsf{pcd}_{\mathsf{no}}(M, I, P, s)$ does not hold, because $I \triangleright P \triangleleft M \overset{s}{\Longrightarrow} \mathsf{yes}$ is true. Hence for any monitor N, the implication $\mathsf{pcd}_w(M_\perp, I, P, s)$ implies $\mathsf{pcd}_w(N, I, P, s)$ holds trivially, hence $M_\perp \sqsubseteq_{\mathsf{pcd}} N$ for every N.

In this case, the top elements of our preorder are the consistent monitors which always detect a verdict. An easy example is $M_{\mathsf{no}} = \mathsf{no}$. As long as $P \overset{s}{\Longrightarrow}$, we always have an s-computation that detects no from $I \triangleright P \triangleleft M_{\mathsf{no}}$ and moreover this detection is consistent, so $\mathsf{pcd}_{\mathsf{no}}(M_{\mathsf{no}}, I, P, s)$ always holds. Hence for any N such that $M_{\mathsf{no}} \sqsubseteq_{\mathsf{pcd}} N$, $\mathsf{pcd}_{\mathsf{no}}(N, I, P, s)$ must hold. By the second condition of potential consistent detection, $\mathsf{pcd}_{\mathsf{no}}(N, I, P, s)$ cannot hold, hence we get that $N \cong_{\mathsf{pcd}} M_{\mathsf{no}}$, meaning M_{no} is a top element of $\sqsubseteq_{\mathsf{pcd}}$.

Example 3.18. We consider the same setup as in Example 3.14, with monitors

$$M_{\mathsf{all}} = \mathsf{yes} + \mathsf{no} \qquad M_7 = c!a.\mathsf{yes} + c!a.\mathsf{no} + c?b.\mathsf{no}$$
$$M_6 = c?b.\mathsf{no} \qquad M_8 = c!a.\mathsf{yes} + c!a.\mathsf{no} + c?b.\mathsf{yes} + c?b.\mathsf{no}$$

to explore the different conclusions our new potential consistent detection preorder will reach. From this, we obtain the ordering

$$M_{\mathsf{all}} \cong_{\mathsf{pcd}} M_8 \sqsubset_{\mathsf{pcd}} M_6 \cong_{\mathsf{pcd}} M_7$$

noting that $M \cong N$ denotes $M \sqsubseteq N$ and $N \sqsubseteq M$.

This can be seen by analysing each monitor closely. The monitors M_6 and M_7 both consistently detect the verdict no given a process that performs the input action $c?b$ and a trace that starts with $c?b$. It is clear that M_6 does not detect any verdict in any other scenario, however this is not true for M_7, as M_7 detects the verdicts yes and no given a process and trace starting with $c!a$. Since this detection is not consistent, it is ignored for the potential consistent detection preorder, hence $M_6 \cong_{\mathsf{pcd}} M_7$.

Moreover, M_{all} and M_8 are less powerful with respect to potential consistent de-

tection, as given any process and trace, they do not consistently detect any verdict. Because of this we have the relation $M_{\mathsf{all}} \cong_{\mathsf{pcd}} M_8$, and since neither of them consistently detect the verdict no given a process that performs the action $c?b$ and trace starting with $c?b$, we obtain the final ordering.

We highlight one peculiarity of the potential consistent detection preorder by considering the process $P_3 = c!a.\mathsf{nil}$ instrumented with M_6 and M_7. The only possible outcome for trace $s = c!a$ for M_6 is inconclusive, as $I \triangleright P_3 \triangleleft M_6 \stackrel{c!a}{\Longrightarrow} \mathsf{nil} \triangleleft \mathsf{end}$ by iTer. However,

$$I \triangleright P_3 \triangleleft M_7 \stackrel{c!a}{\Longrightarrow} \mathsf{nil} \triangleleft \mathsf{yes} \quad \text{and} \quad I \triangleright P_3 \triangleleft M_7 \stackrel{c!a}{\Longrightarrow} \mathsf{nil} \triangleleft \mathsf{no}$$

both hold. In a sense, M_6 is consistent given the process P_3 and trace $c!a$, but M_7 is not. In fact, by Definition 3.12, M_6 is consistent, while M_7 is not.

The same situation also happens with M_{all} and M_8. Even though they are equal with respect to potential consistent detection, given process $P_4 = c!b.\mathsf{nil}$, M_8 fails to detect anything while M_{all} detects both verdicts.

This consequence of potential consistent detection may make sense in certain cases. If for example, detecting two verdicts means the judgement we reach is inconclusive, the equivalences we find above are perfectly reasonable. However in some cases it would be useful to distinguish being inconsistent and being inconclusive. Because of this, we introduce the consistent detection preorder, essentially enforcing that if a monitor is consistent given a trace and process, any monitor implementing it must also be consistent given the same trace and process.

3.2.2 Consistent Detection Preorder

To define our preorder, we define consistent detection. In order for a monitor M to consistently detect a verdict w given a process and trace, every detecting computation must detect w. This is different to potential consistent detection, as it is possible that there exists no detecting computation. Hence, if a monitor is inconclusive given a process and a trace, it can still be consistent, as we saw in the case of M_6 in Example 3.18.

We note the similarities between this definition and that of *deterministic detection* in Definition 3.5, which is defined over all s-computations, whereas consistent detection is defined over all *detecting* s-computations.

Definition 3.19. M *consistently detects* the verdict w for $I \triangleright P$ along trace s, denoted as $\mathsf{cd}_w(M, I, P, s)$, if and only if all detecting s-computations from $I \triangleright P \triangleleft M$ detect w.

Definition 3.20 (Consistent detection preorder)**.**

$$M \sqsubseteq_{\mathsf{cd}} N \coloneqq \forall I, P, s, w \cdot \mathsf{cd}_w(M, I, P, s) \text{ implies } \mathsf{cd}_w(N, I, P, s).$$

Example 3.21. We take a look again at the monitors introduced in Example 3.14 to contrast the three preorders we have now established.

$$M_{\mathsf{all}} = \mathsf{yes} + \mathsf{no} \qquad M_7 = c!a.\mathsf{yes} + c!a.\mathsf{no} + c?b.\mathsf{no}$$
$$M_6 = c?b.\mathsf{no} \qquad M_8 = c!a.\mathsf{yes} + c!a.\mathsf{no} + c?b.\mathsf{yes} + c?b.\mathsf{no}$$

From these monitors, we obtain the following relations

$$M_{\mathsf{all}} \sqsubset_{\mathsf{cd}} M_8 \sqsubset_{\mathsf{cd}} M_7 \sqsubset_{\mathsf{cd}} M_6.$$

As desired, this preorder allows us to differentiate between the monitors M_6 and M_7 and the monitors M_8 and M_{all}.

We analyse the behaviour of these monitors more closely, in order to explain how we got to these relations. Firstly, we have that M_{all} does not consistently detect any verdict given a process P and trace $s \neq \epsilon$, as it can always reach the verdicts yes and no. Because of this, M_{all} is implemented by M_6, M_7 and M_8 when considering consistent detection.

As we saw in Example 3.18, M_8 does consistently detect any verdict given process $P_4 = c!b.\mathsf{nil}$ and trace $c!b$, as there is no detecting $c!b$-computation starting from $I \triangleright P_4 \triangleleft M_8$. Moreover, the same logic can be extended to show that M_8 consistently detects any verdict given process and trace which do not start with $c!a$ or $c?b$. This means we have the strict relationship, $M_{\mathsf{all}} \sqsubset_{\mathsf{cd}} M_8$.

We still have that M_6 and M_7 consistently detect the verdict no when given a process that can produce the action $c?b$ and trace starting with $c?b$. However, M_6 consistently detects for every verdict and trace, while M_7 does not. For example, considering $P_3 = c!a.\mathsf{nil}$ and $s = c!a$, there is no s-computation from $I \triangleright P_3 \triangleleft M_6$, hence M_6 consistently detects every verdict for $I \triangleright P_3$ along s. This is not true for M_7, as $I \triangleright P_3 \triangleleft M_7 \stackrel{s}{\Longrightarrow} \mathsf{yes}$ and $I \triangleright P_3 \triangleleft M_7 \stackrel{s}{\Longrightarrow} \mathsf{no}$, hence M_7 does not consistently detect any verdict in this case. This establishes that the relation $M_7 \sqsubset_{\mathsf{cd}} M_6$ is strict. We also have that M_6 and M_7 strictly implement the other monitors, establishing our relations.

We contrast this to the results obtained by potential consistent detection, where M_6 and M_7 are seen to have the same behaviour. The difference with this preorder is that we care about preserving consistency not only when the implemented monitor detects a verdict, but even when it doesn't. M_6 did not detect any verdict $I \triangleright P_3 \triangleleft M_7 \stackrel{s}{\Longrightarrow} \mathsf{yes}$, but the consistent detection preorder still ensures that its behaviour given this trace and process is consistent. Because of this, we have that if a monitor implements a consistent monitor with respect to consistent detection, then the monitor itself must also be consistent. This is evident from the definition of consistent detection.

In fact, for any consistent monitor M, given I, P, s, the predicate $\mathsf{cd}_w(M, I, P, s)$ is

true for some w. If M does not have a detecting s-computation starting from $I \triangleright P \triangleleft M$, then $\mathsf{cd}_w(M, I, P, s)$ is true for all w. We see this clearly for the monitor M_6, where given process P that performs the action $c?x$ and trace s starting with action $c?b$, the predicate $\mathsf{cd}_{\mathsf{no}}(M_6, I, P, s)$ holds. For any other P and s, $\mathsf{cd}_w(M_6, I, P, s)$ holds for every verdict w, as there does not exist any detecting s-computation starting from $I \triangleright P \triangleleft M_6$.

Remark 3.22. Similarly for potential consistent detection, we again have that $M_\perp$ is a bottom element for consistent detection. This follows from the fact that whenever $P \stackrel{s}{\Longrightarrow} Q$, $I \triangleright P \triangleleft M_\perp \stackrel{w}{\Longrightarrow}$ for both $w = \mathsf{yes}$ and $w = \mathsf{no}$, hence $\mathsf{cd}_w(M_\perp, I, P, s)$ never holds.

More interestingly, our top elements are different here. As a side effect of the property that no detection is consistent detection, the top elements are counter intuitively the ones that reach no verdict. For example, $M_{\mathsf{end}} = \mathsf{end}$ is a top element, as the predicate $\mathsf{cd}_w(M_{\mathsf{end}}, I, P, s)$ is always true, as every s-computation must be consistently detecting, since it does not detect at all. This observation highlights the need to use this consistent detection preorder in conjunction with the already defined preorders if one would like to preserve the judgements attained by the implemented monitor, such as potential detection.

One might notice that the behaviour of our consistent detection is very similar to deterministic detection. In fact, if a verdict is deterministically detected, then its detection must be consistent, as if all s-computations detect w, then it must be the case that they do not detect any other verdict.

The key difference between the two is that consistent detection rules out detecting any other verdict, but does not eliminate the case where no verdict is detected. We illustrate this in the example below.

Example 3.23. Consider the monitor $M_9 = c!a.\mathsf{yes} + c!a.c?a.\mathsf{yes}$ and the process $P_3 = c!a.\mathsf{nil}$. In this setup, with trace $s = c!a$, M_9 can transition both by mChL, eventually leading to the verdict yes being detected, or by mChR, leading to no verdict being detected. In this case, every detecting s-computation detects the verdict yes, so we get that $\mathsf{cd}_{\mathsf{yes}}(M_9, I, P_3, c!a)$. However not every $c!a$-computation is detecting, so $\mathsf{dd}_w(M_9, I, P, c!a)$ does not hold for any verdict w.

Note that while we have taken $\mathsf{Verd} = \{\mathsf{yes}, \mathsf{no}\}$ throughout these examples, these predicates have a natural interpretation in systems with more than two verdicts. To show this, we will give an example with three verdicts, considering consistent detection.

Example 3.24. Consider a system with $\mathsf{Verd} = \{\mathsf{A}, \mathsf{B}, \mathsf{C}\}$, representing a system in which programs are judged to have property A, B or C, where a program can only satisfy one of these properties. Consider the monitors

$$M_{10} = c!a.\mathsf{A} + c!a.\mathsf{B} \quad M_{11} = c!a.\mathsf{B} + c!a.\mathsf{C}.$$

In this case, M_{10} and M_{11} exhibit the same behaviour in terms of consistency, and in fact both $M_{10} \cong_{\mathsf{pcd}} M_{11}$ and $M_{10} \cong_{\mathsf{cd}} M_{11}$. This is because they are both inconsistent on the trace $c!a$. With consistent and potential consistent detection, whenever a monitor is inconsistent we do not care exactly how it is inconsistent, as this behaviour is erroneous regardless.

This arises from the definition of the consistent detection and potential consistent detection preorders. For example, in consistent detection we specify that $M \sqsubseteq_{\mathsf{cd}}$ if $\forall I, P, s, w \cdot \mathsf{cd}_w(M, I, P, s)$ implies $\mathsf{cd}_w(N, I, P, s)$. If $\mathsf{cd}_w(M, I, P, s)$ does not hold, there are no restrictions to the behaviour of N. However, if we wish to differentiate between these two scenarios, we could always use potential detection. In fact $M_{10} \not\sqsubseteq_{\mathsf{pd}} M_{11}$ and $M_{11} \not\sqsubseteq_{\mathsf{pd}} M_{10}$, as M_{11} does not detect A and M_{10} does not detect C.

Remark 3.25. We highlight that throughout this analysis, we have made use of very simple monitors. Most monitors we have established preorders for can detect a verdict immediately when observing an external action. Considering again our running example of monitors,

$$M_{\mathsf{all}} = \mathsf{yes} + \mathsf{no} \qquad M_7 = c!a.\mathsf{yes} + c!a.\mathsf{no} + c?b.\mathsf{no}$$
$$M_6 = c?b.\mathsf{no} \qquad M_8 = c!a.\mathsf{yes} + c!a.\mathsf{no} + c?b.\mathsf{yes} + c?b.\mathsf{no}$$

it is clear that only one observable action has to be observed for the monitors to come to a conclusion. In fact for any external action α, and for each monitor defined here, $M \stackrel{\alpha}{\Longrightarrow} w$ for some $w \in \{\mathsf{end}, \mathsf{yes}, \mathsf{no}\}$. Hence when deciding whether these monitors reach a verdict given a process, we only need to consider whether the process can perform an action α. These monitors give a very simplistic view of how preorders may be established, and may give the wrong impression when it comes to the difficulty of establishing preorders.

In a real-world system, monitors are far more complex, meaning that we cannot typically partition processes into simple categories as we do for the example monitors we have considered. Moreover, our preorders are incredibly powerful, in the sense that they hold over every system the monitors may be instrumented with. Confirming that a preorder does not hold is still straightforward, as to do this, one only needs to find a counterexample process. However, establishing that a preorder holds becomes an unwieldy task, as every processes needs to be considered. When doing this with a complex monitors, the shortcuts we use in our examples are no longer feasible, hence we must resort to iterating over all processes.

4 Alternative Monitor Preorders

The preorders defined in Chapter 3 give us a thorough way to compare and contrast monitors, however in practice they are unwieldy to use. This is because our preorders are defined over every system P, hence to establish these preorders, we must consider every process. In this chapter, we aim to remedy this problem by introducing alternative preorders which are quantified only over all verdicts and traces, but are equivalent to the original preorders. We further prove these equivalences formally, fulfilling our second objective (O2).

This process was already done for uni-verdict monitors in [22] for the transparency, deterministic detection and potential detection preorders. As noted previously, since the transparency preorder remains exactly the same in a multi-verdict context, we need not consider it any further as the already established alternative preorder is enough.

For the deterministic and potential detection preorders, we can easily adapt the existing alternative preorders, noting that proving the equivalence of these alternative preorders to the originals follows almost identically to [22]. Moreover, we find alternative preorders for our consistency preorders, making use of similar techniques.

4.1 Extending existing alternative preorders

Our alterations to the definitions of potential and deterministic detection allow us to make use of the original alternative preorders with only minor changes. This arises naturally from the fact that $\mathsf{pd}_{\checkmark}(M, I, P, s)$ coincides with $\mathsf{pd}(M, I, P, s)$ and $\mathsf{dd}_{\checkmark}(M, I, P, s)$ coincides with $\mathsf{dd}(M, I, P, s)$ in a uni-verdict setting. We present these modified preorders, emphasizing where we are required to differ from the original work.

4.1.1 Extending the potential detection preorder

The alternative potential detection preorder makes use of a restricted monitor semantics, which we specify below. Using the restricted semantics is not necessary for the equivalence to hold, however it is a stronger result, as by restricting the ways to transition, we are reducing the number of possibilities to check. This makes it easier to check whether $M \sqsubseteq_{\mathsf{pd}} N$ holds, as we have fewer traces to consider.

Definition 4.1 (Restricted monitor semantics [22])**.** A derived monitor transition $M \xrightarrow{\mu}_r N$ is the least relation satisfying the conditions $M \xrightarrow{\mu} N$ and $M \neq w$. $M \overset{s}{\underset{r}{\Longrightarrow}} N$ denotes a *transition sequence* in the restricted LTS.

Definition 4.2 (Alternative potential detection preorder)**.**

$$M \preceq_{\mathsf{pd}} N \coloneqq \forall w \forall s \cdot M \overset{s}{\underset{r}{\Longrightarrow}} w \quad \text{implies} \quad N \overset{s}{\Longrightarrow} w.$$

We state below soundness and completeness of the modified alternative preorder.

Theorem 4.3 (Potential detection preorder equivalence)**.**

$$M \sqsubseteq_{pd} N \quad \textit{if and only if} \quad M \preceq_{pd} N.$$

Proof. Analogous to [22]. □

4.1.2 Deterministic detection preorder

For deterministic detection, we are required to define a few more terms. We start by defining monitor divergence, an idea first introduced in Examples 3.2 and 3.8. Divergence plays a part in deterministic detection, as a monitor diverging may keep it from detecting a verdict.

Definition 4.4 (Monitor divergence [22])**.** $M \uparrow$ denotes that M *diverges*, meaning that it can produce an infinite transition sequence of τ-actions $M \xrightarrow{\tau} M' \xrightarrow{\tau} M'' \xrightarrow{\tau} \ldots$

Next we obtain three predicates which will help us in characterising our alternative deterministic detection preorder. We consider the different ways a monitor might end up not deterministically detecting a verdict, by considering various diverging and non-detecting behaviours. We highlight these through a series of examples.

Example 4.5.

1. We first consider the case when a monitor blocks, that is fails to transition with a particular trace. Consider the simple monitor $M_2 = c!a.\checkmark$, and the trace $b!a$. In this case M_2 does not transition given this action, hence we say that the monitor M_2 *blocks* when given trace $b!a$.

2. We also consider the possibility a monitor may fail to detect a verdict, which means it does not lead to the verdict with trace s. Considering again the monitor $M_9 = c!a.\mathsf{yes} + c!a.c?a.\mathsf{yes}$, with trace $c!a$. In Example 3.23, we deduce that $M_9 \overset{c!a}{\Longrightarrow} c?a.\mathsf{yes}$. In this case M_9 *fails* to detect the verdict yes, as the monitor can no longer transition.

   A monitor can *fail* to detect a verdict in two ways: either it cannot transition silently further, or it can enter an infinite loop in which it does not detect the verdict. To

highlight this, consider $M_{12} = c!a.\mathsf{rec}\, X.(\mathsf{yes} + X)$. In this case, given trace $c!a$ we can get the infinite transition sequence

$$M_{12} \xRightarrow{c!a.} \mathsf{rec}\, X.(\mathsf{yes} + X) \xrightarrow{\tau} \mathsf{yes} + \mathsf{rec}\, X.(\mathsf{yes} + X) \xrightarrow{\tau} \mathsf{rec}\, X.(\mathsf{yes} + X) \xrightarrow{\tau} \ldots$$

which does not detect yes, hence M_{12} fails to reach the verdict yes.

3. We also broadly consider the case when a monitor has an s derivative which does not detect a verdict w. We do this because even though the monitor could potentially transition silently to the verdict, this behaviour may be suppressed by a divergent process. This predicate clearly holds for M_9 and M_{12} with the verdict yes and same traces as given above. It is also true if the monitor considered can still transition silently, for example if $M_{13} = c!a.(\mathsf{if}\ c = c\ \mathsf{then\ yes\ else\ yes})$, $M_{13} \xRightarrow{c!a} \mathsf{yes} + \mathsf{yes} \neq \mathsf{yes}$, and so M_{13} has an $c!a$ derivative that does not reach yes, even though M_{13} clearly does not fail at detecting yes with this trace.

Given these examples, we now define explicitly the monitor block, fail and non-detecting predicates. These are as following:

1. A monitor *blocks* if it fails to transition given a trace;
2. A monitor *fails* to detect a verdict if it does not detect the verdict and it either cannot transition any further, or diverges;
3. A monitor is *non-detecting* for a verdict and trace s if there exists an s-derivative of the monitor which does not reach the verdict.

We note that since we are now in a multi-verdict setting, failure and non-detecting behaviours now require an extra parameter w which specifies the verdict in question. Formally, we have the definitions below.

Definition 4.6 (Monitor block, fail and non-detecting [22])**.**

1. $\mathsf{blk}(M, s) \coloneqq \exists s_1, \alpha, s_2, N \cdot s = s_1 \alpha s_2$ and $M \xRightarrow{s_1} N \not\xrightarrow{\tau}$ and $N \not\xrightarrow{\alpha}$
2. $\mathsf{fl}_w(M, s) \coloneqq \exists N \cdot M \xRightarrow{s} N$ and $\big((N \neq w$ and $N \not\xrightarrow{\tau}) \text{ or } N \uparrow\big)$
3. $\mathsf{nd}_w(M, s) \coloneqq \exists N \cdot M \xRightarrow{s} N$ and $N \neq w$

Using these predicates, we can now specify our alternative deterministic detection preorder which relies only on the monitor, trace and verdict.

Definition 4.7 (Alternative deterministic detection preorder)**.**

$$M \preceq_{\mathsf{dd}} N \;\coloneqq\; \forall w \forall s \cdot \begin{cases} \mathsf{blk}(N,s) & \text{implies} \quad \mathsf{blk}(M,s) \text{ or } \mathsf{fl}_w(M,s) \\ \mathsf{fl}_w(N,s) & \text{implies} \quad \mathsf{blk}(M,s) \text{ or } \mathsf{fl}_w(M,s) \\ \mathsf{nd}_w(N,s) & \text{implies} \quad \mathsf{nd}_w(M,s) \text{ or } \mathsf{blk}(M,s) \end{cases}$$

Theorem 4.8 (Deterministic detection preorder equivalence)**.**

$$M \sqsubseteq_{dd} N \quad \textit{if and only if} \quad M \preceq_{dd} N.$$

Proof. Analogous to [22]. □

4.2 Characterising our consistency preorders

We now turn to characterising our newly defined consistency preorders. To do so, we draw heavily on the techniques used in the proofs in [22]. Before we start, we first equip ourselves with some basic lemmas which we will use throughout our proofs.

The following two lemmas, zipping and unzipping, allow us to obtain information about how a process and monitor will execute from the execution of the instrumented system. The outline of these proofs is given in [22], and is identical here.

Lemma 4.9 (Verdict unzipping)**.**

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

Lemma 4.10 (General zipping)**.**

$$I \triangleright P \stackrel{s}{\Longrightarrow} Q \textit{ and } M \stackrel{s}{\Longrightarrow} N \quad \textit{implies} \quad I \triangleright P \triangleleft M \stackrel{s}{\Longrightarrow} Q \triangleleft N.$$

We also state that our verdicts are irrevocable, meaning that once a verdict is reached it cannot be changed.

Lemma 4.11 (Irrevocable Verdicts)**.** *For all* $w \in \mathsf{Verd} \cup \{\textit{end}\}$,

$$I \triangleright P \triangleleft w \stackrel{s}{\Longrightarrow} Q \triangleleft M \quad \textit{implies} \quad M = w.$$

Proof. By definition, $I \triangleright P \triangleleft w \stackrel{s}{\Longrightarrow} Q \triangleleft M$ means there exists a sequence of transitions

$$I_0 \triangleright P_0 \triangleleft M_0 \xrightarrow{\mu_1} I_1 \triangleright P_1 \triangleleft M_1 \xrightarrow{\mu_2} I_2 \triangleright P_2 \triangleleft M_3 \xrightarrow{\mu_3} \ldots \xrightarrow{\mu_n} P_n \triangleleft M_n$$

with $I_0 = I$, $M_0 = w$, $P_0 = P$, $M_n = M$, $P_n = Q$ and $s = \mu_1 \mu_2 \ldots \mu_n$ after filtering out

the silent actions. We prove this lemma by induction on n. For $n = 0$, our result holds trivially as $M = M_n = M_0 = w$. Let $k \geq 0$ be fixed, and suppose $I \triangleright P \triangleleft w \stackrel{s}{\Longrightarrow} Q \triangleleft M$ is obtained by

$$I_0 \triangleright P_0 \triangleleft M_0 \xrightarrow{\mu_1} I_1 \triangleright P_1 \triangleleft M_1 \xrightarrow{\mu_2} \ldots \xrightarrow{\mu_k} I_k \triangleright P_k \triangleleft M_k \xrightarrow{\mu_{k+1}} P_{k+1} \triangleleft M_{k+1}.$$

Then from this we can derive the t computation $I \triangleright P \triangleleft w \stackrel{t}{\Longrightarrow} P_k \triangleleft M_k$, where $t = \mu_1 \mu_2 \ldots \mu_k$ after filtering out the silent actions. Here we can apply the inductive hypothesis to conclude that $M_k = w$.

Hence we have that $I_k \triangleright P_k \triangleleft w \xrightarrow{\mu_{k+1}} P_{k+1} \triangleleft M_{k+1}$. If $\mu_{k+1} = \alpha$, then this transition could only be obtained by iMon, from $w \xrightarrow{\alpha} w$, hence we conclude that $M_{k+1} = w$ as required. Otherwise if $\mu_{k+1} = \tau$, this transition is either obtained by iAsyP or iAsyM. By a case analysis of the rules, we can conclude that $w \stackrel{\tau}{\not\longrightarrow}$, hence iAsyP must have been applied. From this we can conclude that $M_{k+1} = w$. □

From this lemma, we obtain straightforward but nevertheless useful property of s-computations in a multi-verdict environment.

Corollary 4.11.1. *A detecting s-computation can only detect one verdict.*

Proof. Consider an s-computation

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

that detects two verdicts, that is there exist n, m such that $M_n = w$ and $M_m = v$ where $v \neq w$. Without loss of generality we take $n < m$.

Then, we can conclude that $I \triangleright P_n \triangleleft M_n \stackrel{\epsilon}{\Longrightarrow} P_m \triangleleft M_m$. However by Lemma 4.11, since $M_n = w$, then $M_m = w$, hence $v = w$. □

4.2.1 Alternative consistent detection preorder

We first start with the consistent detection preorder, as it is simpler to characterise. We begin our characterization by defining the conflicting detection of a verdict w, which is true whenever we can detect a different verdict $v \neq w$.

Definition 4.12. We define the *conflicting detection* of a verdict w given a monitor M on trace s to be

$$\mathsf{cnf}_w(M, s) := \exists v \in \mathsf{Verd} \cdot M \stackrel{s}{\Longrightarrow} v \text{ and } v \neq w.$$

Example 4.13. Consider the monitor $M_7 = c!a.\mathsf{yes} + c!a.\mathsf{no} + c?b.\mathsf{no}$. Clearly, on trace $c!a$, $M_7 \stackrel{c!a}{\Longrightarrow} \mathsf{no}$, hence by the definition above we have that $\mathsf{cnf}_{\mathsf{yes}}(M_7, c!a)$, as M_7 detects no. By the same reasoning, we get that $\mathsf{cnf}_{\mathsf{no}}(M_7, c!a)$ and $\mathsf{cnf}_{\mathsf{yes}}(M_7, c?b)$. However, $\mathsf{cnf}_{\mathsf{no}}(M_7, c?b)$ does *not* hold, because given this trace the monitor can only detect no.

Taking advantage of a simple relationship between conflicting and consistent detection, we can then define our consistent detection preorder in terms of conflicting detection in a very straightforward way.

Definition 4.14 (Alternative consistent detection preorder)**.**

$$M \preceq_{\mathsf{cd}} N \coloneqq \forall s, w \cdot \mathsf{cnf}_w(N, s) \text{ implies } \mathsf{cnf}_w(M, s).$$

To prove soundness of this preorder, we first prove a lemma that makes the relationship between conflicting detection and consistent detection clear.

Lemma 4.15. *For all M, I, P, s, w, we have that*

$$\neg \mathit{cd}_w(M, I, P, s) \quad \textit{if and only if} \quad \mathit{cnf}_w(M, s) \textit{ and } \exists Q \cdot I \triangleright P \stackrel{s}{\Longrightarrow} Q.$$

Proof. For the first direction, assume that $\neg\mathsf{cd}_w(M, I, P, s)$. This means that there exists a detecting s-computation starting from $I \triangleright P \triangleleft M$ that does not detect w, hence detects some $v \neq w$. By definition, there exists an n such that $I \triangleright P \triangleleft M \stackrel{s}{\Longrightarrow} I_n \triangleright P_n \triangleleft M_n$ and $M_n = w$. By verdict unzipping we have that $I \triangleright P \stackrel{s}{\Longrightarrow} P_n$ and $M \stackrel{s}{\Longrightarrow} v$ for $v \neq w$, *i.e.* $\mathsf{cnf}_w(M, s)$ holds.

Conversely assume that there exists Q such that $\mathsf{cnf}_w(M, s)$ and $I \triangleright P \stackrel{s}{\Longrightarrow} Q$. Hence there exists $v \neq w$ such that $M \stackrel{s}{\Longrightarrow} v$. Then by zipping we have that $I \triangleright P \triangleleft M \stackrel{s}{\Longrightarrow} Q \triangleleft v$. We can further extend this to an s-computation that detects v, which by Corollary 4.2, does not detect w. □

Lemma 4.16 (Soundness)**.**

$$M \preceq_{\mathit{cd}} N \quad \textit{implies} \quad M \sqsubseteq_{\mathit{cd}} N.$$

Proof. Assume that $M \preceq_{\mathsf{cd}} N$, and let I, P, s, w be given. We assume that $\neg\mathsf{cd}_w(N, I, P, s)$ and show that $\neg\mathsf{cd}_w(M, I, P, s)$, proving the contrapositive of the desired result. By Lemma 4.15 we have that $\mathsf{cnf}_w(N, s)$ and $I \triangleright P \stackrel{s}{\Longrightarrow} Q$. Since $M \preceq_{\mathsf{cd}} N$, we know $\mathsf{cnf}_w(M, s)$ holds, and so applying Lemma 4.15 again we get that $\neg\mathsf{cd}_w(M, I, P, s)$ holds. □

To prove completeness, we make use of *characteristic processes* and their respective interfaces, defined inductively as in [22]. We define these using a trace, such that these processes transition in a way we expect when given this trace.

Definition 4.17 (Trace names [22])**.**

$$\mathbf{nm}(c?d.s) \coloneqq \{c, d\} \cup \mathbf{nm}(s) \qquad \mathbf{nm}(c!d.s) \coloneqq \{c, d\} \cup \mathbf{nm}(s) \qquad \mathbf{nm}(\epsilon) \coloneqq \emptyset$$

Definition 4.18 (Trace characterising processes [22])**.**

$$\textbf{prc}(c?d.s) \coloneqq c?x.\textbf{prc}(s) \qquad \textbf{prc}(c!d.s) \coloneqq c!d.\textbf{prc}(s) \qquad \textbf{prc}(\epsilon) \coloneqq \mathsf{nil}$$

Proposition 4.19 (Properties of characteristic processes [22])**.** *For all traces* s, t*:*

(i) $\textbf{nm}(s) \triangleright \textbf{prc}(s) \stackrel{t}{\Longrightarrow} P$ *implies* $s = tt'$ *for some* t' *where* $P = \textbf{prc}(t')$

(ii) $\textbf{nm}(st) \triangleright \textbf{prc}(st) \stackrel{s}{\Longrightarrow} \textbf{prc}(t)$ *where* $\textbf{aftr}(\textbf{nm}(st), s) = \textbf{nm}(st)$

Proof. These properties follow by induction on s. □

Lemma 4.20 (Completeness)**.**

$$M \sqsubseteq_{cd} N \quad \textit{implies} \quad M \preceq_{cd} N.$$

Proof. Let $M \sqsubseteq_{\mathsf{cd}} N$, and let s, w be given. Assume that $\mathsf{cnf}_w(N, s)$. By Proposition 4.19 (ii), we have that $\textbf{nm}(s) \triangleright \textbf{prc}(s) \stackrel{s}{\Longrightarrow} \mathsf{nil}$. Hence by Lemma 4.15, we have that $\neg cd_w(N, \textbf{nm}(s), \textbf{prc}(s), s)$. By $M \sqsubseteq_{\mathsf{cd}} N$, we get that $\neg cd_w(M, \textbf{nm}(s), \textbf{prc}(s), s)$, hence applying Lemma 4.15 again we get that $\mathsf{cnf}_w(M, s)$, as required. □

Theorem 4.21 (Consistent detection preorder equivalence)**.**

$$M \sqsubseteq_{cd} N \quad \textit{if and only if} \quad M \preceq_{cd} N.$$

Proof. Follows from Lemma 4.16 and Lemma 4.20. □

4.2.2 Alternative potential consistent detection preorder

We move on to characterising the potential consistent detection preorder. To do this, we build off of our previous work on consistent detection by first finding an equivalent definition for consistent detection using potential and consistent detection.

Lemma 4.22. *For all* M, I, P, s, w*, we have that*

$$pcd_w(M, I, P, s) \quad \textit{if and only if} \quad pd_w(M, I, P, s) \textit{ and } cd_w(M, I, P, s)$$

Proof. Suppose that $\mathsf{pcd}_w(M, I, P, s)$ holds. Then, by definition, we know that there exists a w-detecting s-computation from $I \triangleright P \triangleleft M$ and for all $v \neq w$, there does not exist a v-detecting s-computation starting from $I \triangleright P \triangleleft M$. The first condition is exactly potential detection, hence $\mathsf{pd}_w(M, I, P, s)$ holds. Moreover, from the second condition we can deduce that every detecting s-computation must necessarily detect w, hence $\mathsf{cd}_w(M, I, P, s)$.

Conversely, assume that $\mathsf{pd}_w(M, I, P, s)$ and $\mathsf{cd}_w(M, I, P, s)$. By $\mathsf{pd}_w(M, I, P, s)$, we have that there exists a w-detecting s-computation from $I \triangleright P \triangleleft M$. Moreover, from $\mathsf{cd}_w(M, I, P, s)$ we know that every detecting s-computation from $I \triangleright P \triangleleft M$ detects w, hence by Corollary 4.2, there does not exist a v-detecting s-computation for $v \neq w$, as desired. □

Note that from Lemma 4.22, we can easily deduce that if $M \sqsubseteq_{\mathsf{pd}} N$ and $M \sqsubseteq_{\mathsf{cd}} N$, then $M \sqsubseteq_{\mathsf{pcd}} N$. However, the converse is not true, as seen in Examples 3.18 and 3.21 with monitors M_6 and M_7, defined as

$$M_6 = c?b.\mathsf{no} \quad \text{and} \quad M_7 = c!a.\mathsf{yes} + c!a.\mathsf{no} + c?b.\mathsf{no}.$$

In Example 3.18, we showed that $M_6 \sqsubseteq_{\mathsf{pcd}} M_7$ as whenever M_6 detects consistently, M_7 detects consistently. However in Example 3.21, we established that $M_6 \not\sqsubseteq_{\mathsf{cd}} M_7$, since M_6 is consistent given a process that performs the external action $c!a$ and trace $c!a$, while M_7 is not.

This means that we cannot simply use the alternative potential detection and consistent detection preorders to characterise potential consistent detection. However, we will make heavy use of Lemma 4.22 as this will allow us to prove things more easily, using previously proved properties of potential and consistent detection.

We further define the exclusive detection of a verdict, meaning that given a trace, a monitor detects that verdict and only that verdict.

Definition 4.23. We define the exclusive detection of a verdict w given a monitor M on trace s to be

$$\mathsf{excl}_w(M, s) \coloneqq M \stackrel{s}{\Longrightarrow} w \wedge \neg \mathsf{cnf}_w(M, s).$$

Example 4.24. Considering again the monitor $M_7 = c!a.\mathsf{yes} + c!a.\mathsf{no} + c?b.\mathsf{no}$, we have that $\mathsf{excl}_{\mathsf{no}}(M_7, c?b)$ is true, as in Example 4.13 we see that $\mathsf{cnf}_{\mathsf{no}}(M_7, c?b)$ does not hold, and clearly $M_7 \stackrel{c?b}{\Longrightarrow} \mathsf{no}$.

Definition 4.25 (Alternative potential consistent detection preorder)**.**

$$M \preceq_{\mathsf{pcd}} N \coloneqq \forall w, s \cdot \mathsf{excl}_w(M, s) \text{ implies } \mathsf{excl}_w(N, s).$$

From here onwards, soundness and completeness of this alternative preorder follow from our already established results and lemmas.

Lemma 4.26 (Soundness)**.**

$$M \preceq_{pcd} N \quad \textit{implies} \quad M \sqsubseteq_{pcd} N.$$

Proof. Let M, N be two monitors such that $M \preceq_{\mathsf{pcd}} N$, and let $\mathsf{pcd}_w(M, I, P, s)$ hold for some arbitrary I, P, s, w. By Lemma 4.22, we know that $\mathsf{pd}_w(M, I, P, s)$, hence there exist I', Q such that $I \triangleright P \triangleleft M \stackrel{s}{\Longrightarrow} I' \triangleright Q \triangleleft w$. By verdict unzipping, we have that $I \triangleright P \stackrel{s}{\Longrightarrow} Q$ and $M \stackrel{s}{\Longrightarrow} w$ holds. Moreover $\mathsf{cd}_w(M, I, P, s)$ holds, so we have by Lemma 4.15 that either $\neg\mathsf{cnf}_w(M, s)$ or $I \triangleright P \stackrel{s}{\not\Longrightarrow}$. Since $I \triangleright P \stackrel{s}{\Longrightarrow} Q$ holds, it must be the case that $\neg\mathsf{cnf}_w(M, s)$. From $\neg\mathsf{cnf}_w(M, s)$ and $\stackrel{s}{\Longrightarrow} w$ we have by definition that $\mathsf{excl}_w(M)$ holds. Hence by the hypothesis, $\mathsf{excl}_w(N)$ holds, that is $\neg\mathsf{cnf}_w(N, s)$ and $N \stackrel{s}{\Longrightarrow} w$ both hold.

Applying Lemma 4.15 again, we have that $\neg\mathsf{cnf}_w(N, s)$ implies that $\mathsf{cd}_w(M, I, P, s)$ holds. Moreover since $N \stackrel{s}{\Longrightarrow} w$ and $I \triangleright P \stackrel{s}{\Longrightarrow} Q$, we can use zipping to conclude that $I \triangleright P \triangleleft M \stackrel{s}{\Longrightarrow} Q \triangleleft M$, which we can extend to an w-detecting s-computation. Hence $\mathsf{pd}_w(N, I, P, s)$ and $\mathsf{cd}_w(N, I, P, s)$ both hold, so by Lemma 4.22, $\mathsf{pcd}_w(N, I, P, s)$ holds as desired. □

Lemma 4.27 (Completeness)**.**

$$M \sqsubseteq_{\mathit{pcd}} N \quad \textit{implies} \quad M \preceq_{\mathit{pcd}} N.$$

Proof. Let M, N be monitors such that $M \sqsubseteq_{\mathsf{pcd}} N$, and let $\mathsf{excl}_w(M, s)$ hold for arbitrary s and w. By Lemma 4.15, and by $\neg\mathsf{cnf}_w(M, s)$, we can conclude that $\mathsf{cd}_w(M, I, P, s)$ holds. By Lemma 4.19 (ii), we know $\mathbf{nm}(s) \triangleright \mathbf{prc}(s) \stackrel{s}{\Longrightarrow} \mathsf{nil}$. Combining this with $M \stackrel{s}{\Longrightarrow} w$, we have by zipping that

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

which we can further extend to a w-detecting s-computation. Hence we can apply Lemma 4.22 to conclude that $\mathsf{pcd}_w(M, \mathbf{nm}(s), \mathbf{prc}(s), s)$ holds.

By our hypothesis, from this we obtain that $\mathsf{pcd}_w(N, \mathbf{nm}(s), \mathbf{prc}(s), s)$ holds, hence by Lemma 4.22, $\mathsf{pd}_w(N, \mathbf{nm}(s), \mathbf{prc}(s), s)$ holds. This means that exists a w-detecting s-computation, so for some I, Q, $\mathbf{nm}(s) \triangleright \mathbf{prc}(s) \triangleleft N \stackrel{s}{\Longrightarrow} I \triangleright Q \triangleleft w$. By verdict unzipping, $\mathbf{nm}(s) \triangleright \mathbf{prc}(s) \stackrel{s}{\Longrightarrow} Q$ and $N \stackrel{s}{\Longrightarrow} w$, proving the first part of our desired equation. To see that $\neg\mathsf{cnf}_w(N, s)$, we note that $\mathsf{cd}_w(N, \mathbf{nm}(s), \mathbf{prc}(s), s)$ holds by Lemma 4.22. Moreover by Lemma 4.15, this implies that either $\neg\mathsf{cnf}_w(N, s)$, or $\mathbf{nm}(s) \triangleright \mathbf{prc}(s) \stackrel{s}{\not\Longrightarrow}$. However the second equation does not hold, as we proved that $\mathbf{nm}(s) \triangleright \mathbf{prc}(s) \stackrel{s}{\Longrightarrow} Q$, hence our result follows. □

Theorem 4.28 (Potential consistent detection preorder equivalence)**.**

$$M \sqsubseteq_{\mathit{pcd}} N \quad \textit{if and only if} \quad M \preceq_{\mathit{pcd}} N.$$

Proof. Follows from Lemma 4.26 and Lemma 4.27. □

5 Conclusion

Throughout this work we have explored the various ways a multi-verdict monitor can properly implement another by considering a series of examples. This led to us establishing the refinement preorders of *potential detection*, *deterministic detection* and *transparency*, which follow from previous work on uni-verdict monitors [22]. By analysing the behaviour of these preorders in a multi-verdict setting, we uncover a need for new preorders which take into account the additional possibilities in a multi-verdict context, namely regarding consistency.

Multi-verdict monitors may be inconsistent, meaning they may come to conflicting conclusions given the same process and trace. In most scenarios, inconsistent behaviour is best avoided as it indicates that there is an error in the specification of the monitor. Hence, we define a way to specify that for a monitor to implement another, it has to be at least as consistent as the monitor it is implementing. This is done by introducing the *consistent detection* and *potential consistent detection* preorders, which enforce consistency of the implementing monitor. The potential consistent detection preorder enforces this property whenever the implemented monitor reaches a consistent verdict, while the consistent detection preorder is stronger, enforcing that consistency is preserved for every trace and process.

These preorders enforce behavioural properties over all possible processes, which gives us strong guarantees when using them. Moreover, we can establish whether one monitor implements another without referring to the processes being monitored, as we have characterised these refinement preorders by defining tractable alternative preorders. These alternative preorders do not depend on a universal quantification over all processes, which means that it is not necessary to consider every possible process to establish these preorders. This is ideal for comparing monitors and establishing correctness criteria, especially since monitors are often simpler than the processes they are meant to test.

This theory allows us to tractably compare multi-verdict monitors based on a variety of correctness criteria. These preorders can be used in conjunction to enforce stricter behaviours according to the specification required, or separately for a coarser comparison, allowing for diverse applications. In particular, we define two monitors which enforce consistency, one being stronger than the other. This allows for more flexibility in the choice of behaviour we would like to preserve. Using these preorders, we can effectively replace one monitor with another, while keeping certain guarantees in place. We can also equate different monitors using these preorders, which allows us to analyse when two monitors behave the same. Overall, these preorders provide us with versatile yet efficient methods for comparing multi-verdict monitors.

5.1 Related and Future Work

Our work is an extension of similar work establishing refinement preorders on uni-verdict monitors [22] to be applicable for multi-verdict monitors. In this work, the potential detection, deterministic detection and transparent preorders were established for uni-verdict monitors, along with tractable alternative preorders. Moreover, this work highlights the possible applications of these preorders through a case study utilising an automated synthesis procedure which works on sHML (safety μHML) [23–26]. We note that by building on top of this theory, the examples established in this work still hold for our adapted preorders. In fact, our adapted preorders are equivalent to the ones already defined if we consider the verdict set Verd $= \{\checkmark\}$.

By extending this theory, we aim to broaden the possible applications for these monitor preorders, as multi-verdict monitors appear often in runtime verification [3–6]. During runtime verification, monitors are used to determine whether a process is obeying certain correctness criteria, and for this reason typically two verdicts are defined — acceptance, meaning the program has behaved correctly, and rejection, meaning the program is violating some correctness criteria and should be rejected. Our monitors give us a way to compare these multi-verdict monitors, keeping in mind the new challenges faced in a multi-verdict context. In particular, multi-verdict monitors may be inconsistent, hence we need to take into consideration the consistency of our monitors. This property was studied in work establishing a theory of monitorability for μHML [12], in which some properties of consistent monitors were established.

Another relevant work considers *consistently-detecting* monitors and provides a more tractable alternate definition for these monitors based on controllability [27]. This analysis considers two conclusive verdicts, acceptances and rejections, as well as the inconclusive verdict. The focus of this work is on deterministic behaviour, hence the approach differs from ours significantly in the definition of consistent detection. In this work, a monitor *consistently detects* if it deterministically detects a verdict, or if it does not detect at all. Our definition of consistent detection is weaker, as we allow for non-deterministic detection of a verdict, so long as no other conclusive verdicts are detected. For example, if we consider

$$M_5 = c!a.\mathsf{no} + \mathsf{end},$$

we have that M_5 consistently detects the verdict no, but would not be a consistently-detecting monitor with respect to the referenced work. The definition of consistently-detecting monitors is closer to our definition of deterministic detection, however we still have that with respect to our definition of consistent detection, every consistently-detecting monitor consistently detects some verdict.

In our work, we make use of a well-established instrumentation relation which

was used in various other works exploring the foundations of runtime monitoring [11, 12, 22, 27–31], considering a varying verdict set Verd. This instrumentation is *synchronous*, meaning the monitor actions correspond to the observable actions carried out by the process. This method is typically used for monitoring tools as it allows for immediate detection of verdicts, and hence more control over the system being observed. However, asynchronous behaviour is more efficient and less intrusive on the monitored system, and so asynchronous instrumentation variants have also been used, along with variants that mix synchronous and asynchronous behaviour [32]. Further work can be done in adapting our theory for these asynchronous monitors — some aspects of our work should be easily extendable to this context.

Similar work has also been done in considering refinement preorders on processes. Considering the testing refinements in [14, 15], we note a similarity of our work to the may and must preorders, defined on maybe satisfying a test o, and always satisfying a test o. We compare this with our definitions of potentially (maybe) detecting and deterministically (always) detecting. These properties arise from an aspect of nondeterminism. A notable departure of our work from testing preorders is that testing preorders are defined on processes, making use of contexts which are tests, while our work considers preorders on monitors, making use of contexts which are processes.

The most similar work to ours is on client preorders [20, 21], in which the authors consider preorders on tests (clients), over processes (servers) as contexts. Our work differs from this work as we define our preorders differently, and moreover we are considering monitors, which are passive entities, as opposed to tests, which are active. Similar work has also been done to establish a general framework for studying equivalences between programs considering contextual equivalences [33]. In this work, contextual equivalences on tests were considered separately to those on processes, highlighting the need for correctness criteria for tests.

Our work defines preorders which act generally, enforcing behaviour over all verdicts. There is scope for expanding this work for more refined preorders $\sqsubseteq^w$, which enforce detection properties based on a particular verdict, or a subset of verdicts. This would allow more flexibility in the application of our preorders. We expect our proof techniques and alternative preorders to be easily adaptable to this more refined theory, as the involvement of the quantification over all verdicts is minimal throughout our proofs.

Given these preorders, it would be worthwhile to create an automated mechanism with which we can determine whether one monitor implements another. While our alternative preorders are definitely more tractable than the original definition quantified over all processes, it has not yet been studied whether these monitor preorders are decidable. It would be useful to explore this problem further, as being able to establish these preorders algorithmically would aid in the development of monitors.

References

[1] E. Bartocci, Y. Falcone, A. Francalanza, and G. Reger, "Introduction to runtime verification," in *Lectures on Runtime Verification: Introductory and Advanced Topics*. Springer, 2018, ch. 1, pp. 1–33.

[2] M. Leucker and C. Schallhart, "A brief account of runtime verification," *The Journal of Logic and Algebraic Programming*, vol. 78, no. 5, pp. 293–303, 2009, The 1st Workshop on Formal Languages and Analysis of Contract-Oriented Software (FLACOS'07), issn: 1567-8326. doi: https://doi.org/10.1016/j.jlap.2008.08.004. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1567832608000775.

[3] Y. Falcone, J.-C. Fernandez, and L. Mounier, "What can you verify and enforce at runtime?" *International Journal on Software Tools for Technology Transfer*, vol. 14, Jun. 2011. doi: 10.1007/s10009-011-0196-8.

[4] C. Cini and A. Francalanza, "An ltl proof system for runtime verification," in *Tools and Algorithms for the Construction and Analysis of Systems*, C. Baier and C. Tinelli, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 581–595, isbn: 978-3-662-46681-0.

[5] S. Pinisetty, T. Jéron, S. Tripakis, Y. Falcone, H. Marchand, and V. Preoteasa, "Predictive runtime verification of timed properties," *Journal of Systems and Software*, vol. 132, Jun. 2017. doi: 10.1016/j.jss.2017.06.060.

[6] A. Bauer, M. Leucker, and C. Schallhart, "Runtime verification for ltl and tltl," *ACM Trans. Softw. Eng. Methodol.*, vol. 20, no. 4, Sep. 2011, issn: 1049-331X. doi: 10.1145/2000799.2000800. [Online]. Available: https://doi.org/10.1145/2000799.2000800.

[7] F. Chen and G. Roşu, "Towards monitoring-oriented programming: A paradigm combining specification and implementation," *Electronic Notes in Theoretical Computer Science*, vol. 89, pp. 108–127, Jan. 2003.

[8] I. Cassar and A. Francalanza, "On implementing a monitor-oriented programming framework for actor systems," in *Integrated Formal Methods*, E. Ábrahám and M. Huisman, Eds., Cham: Springer International Publishing, 2016, pp. 176–192, isbn: 978-3-319-33693-0.

[9] P. Meredith, D. Jin, D. Griffith, F. Chen, and G. Roşu, "An overview of the mop runtime verification framework," *International Journal on Software Tools for Technology Transfer - STTT*, vol. 14, pp. 1–41, Jun. 2011. doi: 10.1007/s10009-011-0198-6.

[10] I. Cassar and A. Francalanza, "Runtime adaptation for actor systems," in *Runtime Verification*, E. Bartocci and R. Majumdar, Eds., Cham: Springer International Publishing, 2015, pp. 38–54, isbn: 978-3-319-23820-3.

[11] A. Francalanza, L. Aceto, and A. Ingolfsdottir, "Monitorability for the hennessy–milner logic with recursion," *Formal Methods in System Design*, vol. 51, Aug. 2017. doi: 10.1007/s10703-017-0273-z.

[12] L. Aceto, A. Achilleos, A. Francalanza, A. Ingólfsdóttir, and K. Lehtinen, "Adventures in monitorability: From branching to linear time and back again," *Proceedings of the ACM on Programming Languages*, vol. 3, no. POPL, pp. 1–29, Jan. 2019, issn: 2475-1421. doi: 10.1145/3290365. [Online]. Available: http://dx.doi.org/10.1145/3290365.

[13] J. H. Morris, "Lambda-calculus models of programming languages.," 1969. [Online]. Available: https://api.semanticscholar.org/CorpusID:50159501.

[14] R. De Nicola and M. Hennessy, "Testing equivalences for processes," *Theoretical Computer Science*, vol. 34, no. 1, pp. 83–133, 1984, issn: 0304-3975. doi: https://doi.org/10.1016/0304-3975(84)90113-0. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0304397584901130.

[15] M. Hennessy, *Algebraic Theory of Processes* (Current Studies in Linguistics Series). MIT Press, 1988, isbn: 9780262081719. [Online]. Available: https://books.google.com.mt/books?id=Rl8EAQAAIAAJ.

[16] A. Cavalcanti and M.-C. Gaudel, "Testing for refinement in csp," in *Formal Methods and Software Engineering*, M. Butler, M. G. Hinchey, and M. M. Larrondo-Petrie, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 151–170, isbn: 978-3-540-76650-6.

[17] Y. Isobe and M. Roggenbach, "A generic theorem prover of csp refinement," in *Tools and Algorithms for the Construction and Analysis of Systems*, N. Halbwachs and L. D. Zuck, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 108–123, isbn: 978-3-540-31980-1.

[18] B. Buth and M. Schrönen, "Model-checking the architectural design of a fail-safe communication system for railway interlocking systems," in *FM'99 — Formal Methods*, J. M. Wing, J. Woodcock, and J. Davies, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 1869–1869, isbn: 978-3-540-48118-8.

[19] S. Schneider, "Verifying authentication protocol implementations," vol. 81, Mar. 2002, isbn: 978-1-4757-5268-7. doi: 10.1007/978-0-387-35496-5_2.

[20] G. Bernardi and A. Francalanza, "Full-abstraction for client testing preorders," *Science of Computer Programming*, vol. 168, pp. 94–117, 2018, issn: 0167-6423. doi: https://doi.org/10.1016/j.scico.2018.08.004. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167642318303277.

[21] G. Bernardi and M. Hennessy, "Mutually testing processes," *Log. Methods Comput. Sci.*, vol. 11, no. 2, 2015. doi: 10.2168/LMCS-11(2:1)2015. [Online]. Available: https://doi.org/10.2168/LMCS-11(2:1)2015.

[22] A. Francalanza, "A theory of monitors," *Information and Computation*, vol. 281, p. 104 704, 2021, issn: 0890-5401. doi: https://doi.org/10.1016/j.ic.2021.104704. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0890540121000195.

[23] L. Aceto and A. Ingolfsdottir, "Testing hennesy-milner logic with recursion," English, in *Foundations of Software Science and Computation Structures : Second International Conference, FOSSACS ' 99 : Held as Part of the Joint European Conference on Theory and Practice of Software, ETAPS '99, Amsterdam, the Netherlands, March 22-28, 1999*, Thomas, W. (ed.), Ed., ISSN ; 0302-9743; Testing Hennesy-Milner Logic with Recursion ; Conference date: 19-05-2010, United States: IEEE Computer Society Press, 1999, pp. 41–55, isbn: 3540657193.

[24] L. Aceto, I. Cassar, A. Francalanza, and A. Ingólfsdóttir, "Comparing controlled system synthesis and suppression enforcement," *International Journal on Software Tools for Technology Transfer*, vol. 23, no. 4, pp. 601–614, Aug. 2021, issn: 1433-2787. doi: 10.1007/s10009-021-00624-0. [Online]. Available: https://doi.org/10.1007/s10009-021-00624-0.

[25] A. Francalanza, L. Aceto, and A. Ingolfsdottir, "On verifying hennessy-milner logic with recursion at runtime," in *Runtime Verification*, E. Bartocci and R. Majumdar, Eds., Cham: Springer International Publishing, 2015, pp. 71–86, isbn: 978-3-319-23820-3.

[26] L. Aceto, I. Cassar, A. Francalanza, and A. Ingólfsdóttir, "Comparing controlled system synthesis and suppression enforcement," *International Journal on Software Tools for Technology Transfer*, vol. 23, Aug. 2021. doi: 10.1007/s10009-021-00624-0.

[27] A. Francalanza, "Consistently-detecting monitors," Sep. 2017, p. 16. doi: 10.4230/LIPIcs.CONCUR.2017.8.

[28] L. Aceto, A. Achilleos, A. Francalanza, A. Ingólfsdóttir, and K. Lehtinen, *An operational guide to monitorability*, 2019. arXiv: 1906.00766 [cs.LO].

[29] L. Aceto, A. Achilleos, A. Francalanza, A. Ingólfsdóttir, and S. Ö. Kjartansson, "Determinizing monitors for HML with recursion," *CoRR*, vol. abs/1611.10212, 2016. arXiv: 1611.10212. [Online]. Available: http://arxiv.org/abs/1611.10212.

[30] L. Aceto, A. Achilleos, A. Francalanza, A. Ingólfsdóttir, and S. Ö. Kjartansson, "On the complexity of determinizing monitors," in *Implementation and Application of Automata*, A. Carayol and C. Nicaud, Eds., Cham: Springer International Publishing, 2017, pp. 1–13, isbn: 978-3-319-60134-2.

[31] A. Francalanza and J. Xuereb, "On implementing symbolic controllability," in Jun. 2020, pp. 350–369, isbn: 978-3-030-50028-3. doi: 10.1007/978-3-030-50029-0_22.

[32] I. Cassar, A. Francalanza, L. Aceto, and A. Ingólfsdóttir, "A survey of runtime monitoring instrumentation techniques," *Electronic Proceedings in Theoretical Computer Science*, vol. 254, pp. 15–28, Aug. 2017, issn: 2075-2180. doi: 10.4204/eptcs.254.2. [Online]. Available: http://dx.doi.org/10.4204/EPTCS.254.2.

[33] C. Aubert and D. Varacca, "Processes against tests: On defining contextual equivalences," *Journal of Logical and Algebraic Methods in Programming*, vol. 129, p. 100 799, 2022, issn: 2352-2208. doi: https://doi.org/10.1016/j.jlamp.2022.100799. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2352220822000529.

Appendix A Supplementary Material

A.1 Definitions

Definition A.1. (Pattern matching)

We define $\mathsf{match}(p, \alpha)$ inductively as follows:

(i) $\mathsf{match}((x), c) = [c/x]$

(ii) $\mathsf{match}(c, d) = \begin{cases} \emptyset \text{ if } c = d \\ \text{undefined} \quad \text{otherwise} \end{cases}$

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

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

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