BIDIRECTIONAL RUNTIME ENFORCEMENT OF FIRST-ORDER BRANCHING-TIME PROPERTIES

LUCA ACETO, IAN CASSAR, ADRIAN FRANCALANZA, AND ANNA INGÓLFSDÓTTIR

[a] ICE-TCS, Department of Computer Science, Reykjavík University, Iceland and Gran Sasso Science Institute, L'Aquila, Italy
e-mail address: luca@ru.is

[b] Department of Computer Science, University of Malta, Malta and ICE-TCS, Department of Computer Science, Reykjavík University, Iceland
e-mail address: ian.cassar.10@um.edu.mt

[c] Department of Computer Science, University of Malta, Malta
e-mail address: adrian.francalanza@um.edu.mt

[d] ICE-TCS, Department of Computer Science, Reykjavík University, Iceland
e-mail address: anna@ru.is

Abstract. Runtime enforcement is a dynamic analysis technique that instruments a monitor with a system in order to ensure its correctness as specified by some property. This paper explores *bidirectional* enforcement strategies for properties describing the input and output behaviour of a system. We develop an operational framework for bidirectional enforcement and use it to study the enforceability of the safety fragment of Hennessy-Milner logic with recursion (sHML). We provide an automated synthesis function that generates correct monitors from sHML formulas, and show that this logic is enforceable via a specific type of bidirectional enforcement monitors called action disabling monitors.

Key words and phrases: runtime monitors, property enforcement, monitor synthesis, first-order safety properties, modal μ-calculus.

"TheoFoMon: Theoretical Foundations for Monitorability" (nr.163406-051) of the Icelandic Research Fund and the Italian MIUR project PRIN 2017FTXR7S IT MATTERS "Methods and Tools for Trustworthy Smart Systems".

"Developing Theoretical Foundations for Runtime Enforcement" (nr.184776-051) of the Icelandic Research Fund and the Endeavour Scholarship Scheme (Malta), part-financed by the European Social Fund (ESF) - Operational Programme II – 2014-2020.

'TheoFoMon: Theoretical Foundations for Monitorability" (nr.163406-051) and "MoVeMnt: Mode(l)s of Verification and Monitorability" (nr.217987-051) of the Icelandic Research Fund and the EU H2020 RISE programme under the Marie Skłodowska-Curie grant agreement nr. 778233.

"TheoFoMon: Theoretical Foundations for Monitorability" (nr.163406-051) of the Icelandic Research Fund.

Preprint submitted to
Logical Methods in Computer Science

1. Introduction

Runtime enforcement (RE) [LBW05, FFM08] is a dynamic verification technique that uses *monitors* to analyse the runtime behaviour of a system-under-scrutiny (SuS) and transform it in order to conform to some correctness *specification.* The earliest work in RE [LBW05, LBW09, Sak09, BM11a, KT12] models the behaviour of the SuS as a *trace* of *abstract* actions (*e.g.,* $\alpha, \beta, \ldots \in \textsc{Act}$). Importantly, it assumes that the monitor can either suppress or replace *any* of these (abstract) actions and, whenever possible, insert additional actions into the trace.

This work has been used as a basis to implement *unidirectional* enforcement approaches [KAB$^+$17, FFM12, ACFI18, Av11] that monitor the outputted trace of actions emitted by the SuS as illustrated by Figure 1(a). In this setup, the monitor is instrumented with the SuS to form a *composite system* (represented by the dashed enclosure in Figure 1) and is tasked with transforming the output behaviour of the SuS to ensure its correctness. For instance, an erroneous output β of the SuS is intercepted by the monitor and transformed into β', to stop the error from propagating to the surrounding environment.

Despite its merits, unidirectional enforcement disregards the fact that not all events originate from the SuS. For instance, protocol specifications describing the interaction of communicating computational entities include *input* actions, instigated by the environment in addition to output actions originating from the SuS. Arguably, these properties are harder to enforce. Since inputs are instigated by the environment, the SuS possesses only partial control over them and the capabilities to prevent or divert such actions can be curtailed. Moreover, in such a bidirectional setting, the properties to be enforced tend to be of a *first-order* nature [ACFI18, HRTZ18], describing relationships between the respective payload carried by input and output events. This means that even when the (monitored) SuS can control when certain inputs can be supplied (*e.g.,* by opening a communication port, or by reading a record from a database *etc.*), the environment still has control over the provided payload.

Broadly, there are two approaches to enforce bidirectional properties at runtime. Several bodies of work employ two monitors attached at the output side of each (diadic) interacting party [BCD$^+$17, JGP16, CBD$^+$12, **?**]. As shown in Figure 1(b), the extra monitor is attached to the *environment* to analyse its outputs before they are passed on as *inputs* to the SuS. While this approach is effective, it assumes that a monitor can actually be attached to the environment (which is often inaccessible).

By contrast, Figure 1(c) presents a less explored *bidirectional enforcement* approach where the monitor analyses the entire behaviour of the SuS without the need to instrument the environment. The main downside of this alternative setup is that it enjoys limited control over the SuS's inputs. As we already argued, the monitor may be unable to enforce a property that could be violated by an input action with an invalid payload value. In other cases, the monitor might need to adopt a different enforcement strategy to the ones that are conventionally used for enforcing output behaviour in a unidirectional one.

This paper explores how existing monitor transformations—namely, suppressions, insertions and replacements—can be repurposed to work for bidirectional enforcement, *i.e.,* the setup in Figure 1(c). Since inputs and outputs must be enforced differently, we find it essential to distinguish between the monitor's transformations and their resulting effect on the visible behaviour of the composite system. This permits us to study the enforceability of properties defined via a safety subset of the well-studied branching-time logic

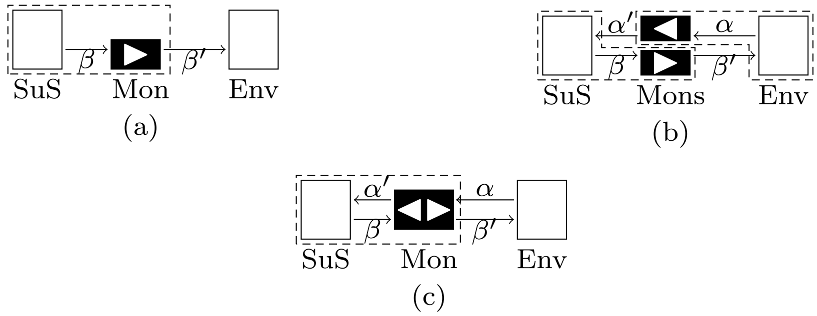

FIGURE 1. Enforcement instrumentation setups.

μHML [RH97, AILS07, Lar90] (a reformulation of the modal μ-calculus [Koz83]), called sHML. A crucial aspect of our investigation is the *synthesis* function that maps the *declarative* safety μHML specifications to *algorithmic* monitors that enforce properties concerning both the input and output behaviour of the SuS. Since monitors are part of the trusted computing base, it was imperative that we ensure that all synthesised monitors are correct [**?**, Fra21]. Our contributions are:

(*i*) A general instrumentation framework for bidirectional enforcement (Figure 4) that is parametrisable by any system whose behaviour can be modelled as a labelled transition system. The framework subsumes the one presented in previous work [ACFI18] and differentiates between the enforcement of input and output actions.

(*ii*) A novel formalisation describing what it means for a monitor to *adequately enforce* a property in a bidirectional setting (Definitions 4.1 and 4.9). These definitions are parametrisable with respect to an instrumentation relation, an instance of which is given by our enforcement framework of Figure 4.

(*iii*) A new result showing that the subclass of *disabling monitors*, Definition 3.1 (the counterpart to suppression monitors in unidirectional enforcement), suffices to bidirectionally enforce safety properties expressed as μHML formulas (Theorem 5.5). A by-product of this result is a synthesis function (Definition 5.3) that generates a disabling monitor from such safety formulas.

(*iv*) A preliminary investigation on the notion of monitor *optimality* (Definition 6.3). Our proposed definition assesses the level of intrusiveness of the monitor and guides in the search for the least intrusive one. We evaluate our monitor synthesis function of Definition 5.3 in terms of this optimality measure, Theorem 6.12.

This article is the extended version of the paper titled *"On Bidirectional Runtime Enforcement"* that appeared at FORTE 2021 [ACFI21]. In addition to the material presented in the conference version, this version contains extended examples, the proofs of the main results and new material on monitor optimality. The related work section has also been considerably expanded.

2. PRELIMINARIES

The Model. We assume a countably infinite set of communication ports $\mathsf{a},\mathsf{b},\mathsf{c} \in \textsc{Port}$, a set of values $v, w \in \textsc{Val}$, and partition the set of actions $\textsc{Act}$ into

Syntax

$$\begin{array}{llllllll}\varphi,\psi \in \mathrm{sHML} ::= \mathsf{tt} & \text{(truth)} & \mid \mathsf{ff} & \text{(falsehood)} & \mid \bigwedge_{i\in I}\varphi_i & \text{(conjunction)} \\ \quad\mid [p,c]\varphi & \text{(necessity)} & \mid \max X.\varphi & \text{(greatest fp.)} & \mid X & \text{(fp. variable)}\end{array}$$

Semantics

$$\begin{aligned}
[\![\mathsf{tt},\rho]\!] &\stackrel{\text{def}}{=} \textsc{Sys}\\
[\![\mathsf{ff},\rho]\!] &\stackrel{\text{def}}{=} \emptyset\\
[\![X,\rho]\!] &\stackrel{\text{def}}{=} \rho(X)\\
[\![\bigwedge_{i\in I}\varphi_i,\rho]\!] &\stackrel{\text{def}}{=} \bigcap_{i\in I}[\![\varphi_i,\rho]\!]\\
[\![\max X.\varphi,\rho]\!] &\stackrel{\text{def}}{=} \bigcup\{S \mid S\subseteq [\![\varphi,\rho[X\mapsto S]]\!]\}\\
[\![\,[p,c]\varphi,\rho]\!] &\stackrel{\text{def}}{=} \{s \mid \forall\alpha,r,\sigma\cdot(s\stackrel{\alpha}{\Longrightarrow} r \text{ and } \mathsf{match}(p,\alpha){=}\sigma \text{ and } c\sigma\Downarrow\mathsf{true}) \text{ implies } r\in[\![\varphi\sigma,\rho]\!]\}
\end{aligned}$$

FIGURE 2. The syntax and semantics for sHML, the safety fragment of the branching-time logic μHML [Lar90].

- *inputs*, $\mathsf{a}?v \in \textsc{iAct}$ *e.g.*, denoting an input by the system from the environment on port a carrying payload v; and
- *outputs*, $\mathsf{a}!v \in \textsc{oAct}$ originating from the system to the interacting environment on port a carrying payload v

where $\textsc{Act} = \textsc{iAct}\cup\textsc{oAct}$. Systems are described as *labelled transition systems* (LTSs); these are triples $\langle \textsc{Sys}, \textsc{Act}\cup\{\tau\},\rightarrow\rangle$ consisting of a set of *system states*, $s,r,q\in\textsc{Sys}$, a set of *visible actions*, $\alpha,\beta\in\textsc{Act}$, along with a distinguished silent action $\tau\notin\textsc{Act}$ (where $\mu\in\textsc{Act}\cup\{\tau\}$), and a *transition* relation, $\longrightarrow\subseteq(\textsc{Sys}\times(\textsc{Act}\cup\{\tau\})\times\textsc{Sys})$. We write $s\xrightarrow{\mu} r$ in lieu of $(s,\mu,r)\in\rightarrow$, and $s\stackrel{\alpha}{\Longrightarrow} r$ to denote weak transitions representing $s(\xrightarrow{\tau})^*\cdot\xrightarrow{\alpha} r$ where r is called the α-derivative of s. For convenience, we use the syntax of the regular fragment of value-passing CCS [HL96] to concisely describe LTSs. Traces $t,u\in\textsc{Act}^*$ range over (finite) sequences of *visible* actions. We write $s\stackrel{t}{\Longrightarrow} r$ to denote a sequence of *weak* transitions $s\stackrel{\alpha_1}{\Longrightarrow}\ldots\stackrel{\alpha_n}{\Longrightarrow} r$ where $t=\alpha_1\ldots\alpha_n$ for some $n\geq 0$; when $t=\varepsilon$, $s\stackrel{\varepsilon}{\Longrightarrow} r$ means $s\xrightarrow{\tau}{}^* r$. Additionally, we represent system runs as *explicit traces* that include τ-actions, $t_\tau,u_\tau\in(\textsc{Act}\cup\{\tau\})^*$ and write $s\xrightarrow{\mu_1\ldots\mu_n} r$ to denote a sequence of *strong* transitions $s\xrightarrow{\mu_1}\ldots\xrightarrow{\mu_n} r$. The function $\mathsf{sys}(t_\tau)$ returns a system that produces exclusively the sequence of actions defined by t_τ, modulo the data carried by input actions in t_τ that cannot be controlled by the receiving process. For instance, $\mathsf{sys}(\mathsf{a}?3.\tau.\mathsf{a}!5)$ produces the process $\mathsf{a}?x.\tau.\mathsf{a}!5.\mathsf{nil}$. We consider states in our system LTS modulo the classic notion of *strong bisimilarity* [HL96, San11] and write $s\sim r$ when states s and r are bisimilar.

The Logic. The behavioral properties we consider are described using sHML [AI99, FAI17], a subset of the value passing μHML [RH97, HL95] that uses *symbolic actions* of the form (p,c) consisting of an action pattern p and a condition c. Symbolic actions facilitate reasoning about LTSs with infinitely many actions (*e.g.*, inputs or outputs carrying data from infinite domains). They abstract over concrete actions using *data variables* $x,y,z\in\textsc{DVar}$ that occur free in the constraint c or as binders in the pattern p. Patterns are subdivided into input $(x)?(y)$ and output $(x)!(y)$ patterns where (x) binds the information about the port on which the interaction has occurred, whereas (y) binds the payload; $\mathbf{bv}(p)$ denotes the set of binding variables in p whereas $\mathbf{fv}(c)$ represents the set of free variables in condition c. We assume a (partial) *matching function* $\mathsf{match}(p, \alpha)$ that (when successful) returns the (smallest) substitution $\sigma : \text{DVar} \rightharpoonup (\text{Port} \cup \text{Val})$, mapping bound variables in p to the corresponding values in α; by replacing every occurrence (x) in p with $\sigma(x)$ we get the matched action α. The *filtering condition*, c, is evaluated *wrt.* the substitution returned by successful matches, written as $c\sigma \Downarrow v$ where $v \in \{\mathsf{true}, \mathsf{false}\}$.

Whenever a symbolic action (p, c) is *closed*, *i.e.,* $\mathbf{fv}(c) \subseteq \mathbf{bv}(p)$, it denotes the *set* of actions $[\![(p, c)]\!] \stackrel{\text{def}}{=} \{\ \alpha \mid \exists \sigma \cdot \mathsf{match}(p, \alpha) = \sigma \textit{ and } c\sigma \Downarrow \mathsf{true}\ \}$. For example, we can have $[\![\big((x)!(y),\ (x = \mathsf{a} \vee x = \mathsf{b})\ \wedge\ y \geq 3\big)]\!] = \{\mathsf{a}!3, \mathsf{b}!3, \mathsf{a}!4, \mathsf{b}!4, \mathsf{a}!5, \mathsf{b}!5, \mathsf{a}!6, \mathsf{b}!6, \ldots\}$. Following standard (concrete) value-passing LTS semantics [MPW92, HL96], our systems have *no control* over the data values supplied via inputs. Accordingly, we assume a well-formedness constraint where the condition c of an input symbolic action, $((x)?(y),c)$, *cannot* restrict the values of binder y, *i.e.,* $y \notin \mathbf{fv}(c)$. As a shorthand, whenever a condition in a symbolic action equates a bound variable to a specific value we embed the equated value within the pattern, *e.g.,* $((x)!(y), x = \mathsf{a} \wedge y = 3)$, $((x)?(y), x = \mathsf{a})$ and $((x)?(y), x = z)$ become $(\mathsf{a}!3, \mathsf{true})$, $(\mathsf{a}?(y), \mathsf{true})$ and $(z?(y), \mathsf{true})$ *resp.*; we also elide true conditions, and occasionally just write $(\mathsf{a}!3)$ and $(\mathsf{a}?(y))$ in lieu of $(\mathsf{a}!3, \mathsf{true})$ and $(\mathsf{a}?(y), \mathsf{true})$ when the meaning of this shorthand can be inferred from the context.

Figure 2 presents the sHML syntax for some countable set of logical variables $X, Y \in \text{LVar}$. The construct $\bigwedge_{i \in I} \varphi_i$ describes a *compound* conjunction, $\varphi_1 \wedge \ldots \wedge \varphi_n$, where $I = \{1, .., n\}$ is a finite set of indices. The syntax also permits recursive properties using greatest fixpoints, $\mathsf{max}\, X.\varphi$, which bind free occurrences of X in φ. The central construct is the (symbolic) universal modal operator, $[p, c]\varphi$, where the binders $\mathbf{bv}(p)$ bind the free data variables in c and φ. We occasionally use the notation $(\_)$ to denote "don't care" binders in the pattern p, whose bound values are not referenced in c and φ. We also assume that all fixpoint variables, X, are guarded by modal operators.

Formulas in sHML are interpreted over the system powerset domain where $S \in \mathcal{P}(\text{Sys})$. The semantic definition of Figure 2, $[\![\varphi, \rho]\!]$, is given for *both* open and closed formulas. It employs a valuation from logical variables to sets of states, $\rho \in (\text{LVar} \rightarrow \mathcal{P}(\text{Sys}))$, which permits an inductive definition on the structure of the formulas; $\rho' = \rho[X \mapsto S]$ denotes a valuation where $\rho'(X) = S$ and $\rho'(Y) = \rho(Y)$ for all other $Y \neq X$. The only non-standard case is that for the universal modality formula, $[p, c]\varphi$, which is satisfied by any system that either *cannot* perform an action α that matches p while satisfying condition c, or for any such matching action α with substitution σ, its derivative state satisfies the continuation $\varphi\sigma$. We consider formulas modulo associativity and commutativity of $\wedge$, and unless stated explicitly, we assume *closed* formulas, *i.e.,* without free logical and data variables. Since the interpretation of a closed φ is independent of the valuation ρ we write $[\![\varphi]\!]$ in lieu of $[\![\varphi, \rho]\!]$. A system s *satisfies* formula φ whenever $s \in [\![\varphi]\!]$, and a formula φ is *satisfiable*, when $[\![\varphi]\!] \neq \emptyset$.

We find it convenient to define the function *after*, describing how an sHML formula *evolves* in reaction to an action μ. Note that, for the case $\varphi = [p, c]\psi$, the formula returns $\psi\sigma$ when μ matches successfully the symbolic action (p, c) with σ, and tt otherwise, to signify a trivial satisfaction.

Definition 2.1. We define the function *after* : $(\text{sHML} \times \textsc{Act} \cup \{\tau\}) \rightarrow \text{sHML}$ as:

$$\mathit{after}(\varphi,\alpha) \stackrel{\text{def}}{=} \begin{cases} \varphi & \text{if } \varphi \in \{\mathsf{tt},\mathsf{ff}\} \\ \mathit{after}(\varphi'\{^{\varphi}/_{X}\},\alpha) & \text{if } \varphi = \mathsf{max}\,X.\varphi' \\ \bigwedge_{i\in I} \mathit{after}(\varphi_i,\alpha) & \text{if } \varphi = \bigwedge_{i\in I}\varphi_i \\ \psi\sigma & \text{if } \varphi = [p,c]\psi \textit{ and } \exists\sigma\cdot(\mathsf{match}(p,\alpha){=}\sigma \wedge c\sigma \Downarrow \mathsf{true}) \\ \mathsf{tt} & \text{if } \varphi = [p,c]\psi \textit{ and } \nexists\sigma\cdot(\mathsf{match}(p,\alpha){=}\sigma \wedge c\sigma \Downarrow \mathsf{true}) \end{cases}$$

$$\mathit{after}(\varphi,\tau) \stackrel{\text{def}}{=} \varphi$$ ■

We abuse notation and lift the *after* function to (explicit) traces in the obvious way, *i.e.,* $\mathit{after}(\varphi,t_\tau)$ is equal to $\mathit{after}(\mathit{after}(\varphi,\mu),u_\tau)$ when $t_\tau = \mu u_\tau$ and to φ when $t_\tau = \varepsilon$. Our definition of *after* is justified vis-a-vis the semantics of Figure 2 via Proposition 2.3; it will play a role later on when defining our notion of enforcement in Section 4.

Remark 2.2. The function *after* is well-defined due to our assumption that formulas are guarded, guaranteeing that $\varphi'\{^{\varphi}/_{X}\}$ has fewer top level occurrences of greatest fixpoint operators than $\mathsf{max}\,X.\varphi'$. ■

Proposition 2.3. *For every system state s, formula φ and action α, if $s \in [\![\varphi]\!]$ and $s \stackrel{\alpha}{\Longrightarrow} s'$ then $s' \in [\![\mathit{after}(\varphi,\alpha)]\!]$.* □

Example 2.4. The *safety* property φ_1 *repeatedly* requires that *every* input request that is made on a port that is *not* b, cannot be followed by another input on the same port in succession. However, following this input it allows a *single* output answer on the same port in response, followed by the logging of the serviced request by outputting a notification on a dedicated port b. We note how the channel name bound to x is used to constrain sub-modalities. Similarly, values bound to y_1 and y_2 are later referenced in condition $y_3 = (\mathsf{log}, y_1, y_2)$.

$$\varphi_1 \stackrel{\text{def}}{=} \mathsf{max}\,X.[((x)?(y_1), x{\neq}\mathsf{b})](([(x?(\_))]\mathsf{ff} \wedge [(x!(y_2))]\varphi_1')$$
$$\varphi_1' \stackrel{\text{def}}{=} ([(x!(\_))]\mathsf{ff} \wedge [(\mathsf{b}!(y_3), y_3{=}(\mathsf{log}, y_1, y_2))]X)$$

Consider the systems $s_{\mathbf{a}}$, $s_{\mathbf{b}}$ and $s_{\mathbf{c}}$:

$$s_{\mathbf{a}} \stackrel{\text{def}}{=} \mathsf{rec}\,X.((\mathsf{a}?x.y := \mathsf{ans}(x).\mathsf{a}!y.\mathsf{b}!(\mathsf{log},x,y).X) + s_{\mathbf{cls}})$$
$$\text{(where } s_{\mathbf{cls}} \stackrel{\text{def}}{=} (\mathsf{b}?z.\mathsf{if}\ z{=}\mathsf{cls}\ \mathsf{then}\ \mathsf{nil}\ \mathsf{else}\ X))$$
$$s_{\mathbf{b}} \stackrel{\text{def}}{=} \mathsf{rec}\,X.((\mathsf{a}?x.y := \mathsf{ans}(x).\mathsf{a}!y.(\underline{\mathsf{a}!y}.\mathsf{b}!(\mathsf{log},x,y).s_{\mathbf{a}} + \mathsf{b}!(\mathsf{log},x,y).X)) + s_{\mathbf{cls}})$$
$$s_{\mathbf{c}} \stackrel{\text{def}}{=} \mathsf{a}?y.s_{\mathbf{a}}$$

The system $s_{\mathbf{a}}$ implements a *request-response* server that repeatedly inputs values (for some domain $\textsc{Val}$) on port a, $\mathsf{a}?x$, for which it internally computes an answer and assigns it to the data variable y, $y := \mathsf{ans}(x)$. Subsequently, it outputs the answer on port a in response to each request, $\mathsf{a}!y$, and then logs the serviced request pair of values by outputting the triple (log,x,y) on port b, $\mathsf{b}!(\mathsf{log},x,y)$. It terminates whenever it receives a close request cls from port b, *i.e.,* $\mathsf{b}?z$ when $z = \mathsf{cls}$. Systems $s_{\mathbf{b}}$ and $s_{\mathbf{c}}$ are similar to $s_{\mathbf{a}}$ but define additional behaviour: $s_{\mathbf{c}}$ requires a startup input, $\mathsf{a}?y$, before behaving as $s_{\mathbf{a}}$, whereas $s_{\mathbf{b}}$ occasionally provides a redundant (underlined) answer prior to logging a serviced request.

Using the semantics of Figure 2, one can verify that the first system satisfies our correctness property φ_1, *i.e.,* $s_{\mathbf{a}} \in [\![\varphi_1]\!]$. However the second system $s_{\mathbf{b}}$ does not satisfy

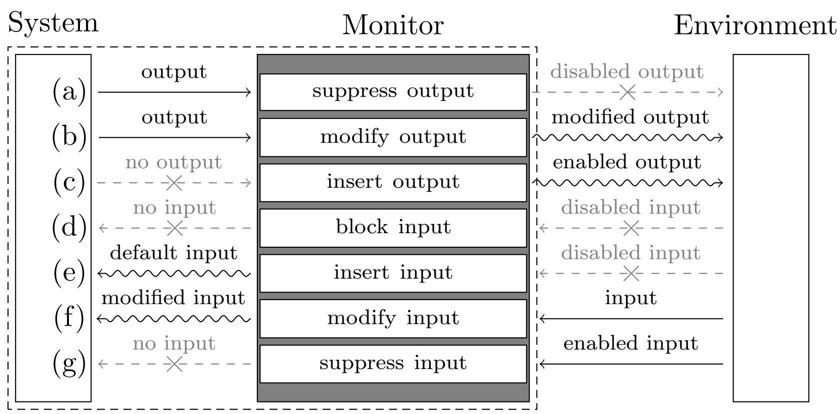

FIGURE 3. Disabling, enabling and adapting bidirectional runtime enforcement via suppressions, insertions and replacements.

this property because it can inadvertently answer twice a request, *i.e.*, $s_{\mathbf{b}} \notin [\![\varphi_1]\!]$ since we have $s_{\mathbf{b}} \xRightarrow{\mathsf{a}?v_1.\mathsf{a}!\mathsf{ans}(v_1).\mathsf{a}!\mathsf{ans}(v_1)}$ (for some value v_1). Analogously, the third system $s_{\mathbf{c}}$ violates property φ_1 because it can accept two consecutive inputs on port a (without answering the preliminary request first), *i.e.*, $s_{\mathbf{c}} \notin [\![\varphi_1]\!]$ since $s_{\mathbf{c}} \xRightarrow{\mathsf{a}?v_1.\mathsf{a}?v_2}$ (for any pair of values v_1 and v_2). ■

3. A Bidirectional Enforcement Model

Bidirectional enforcement seeks to transform the entire (visible) behaviour of the SuS in terms of output actions (instigated by the SuS itself, which in turn controls the payload values being communicated) and input actions (originating from the interacting environment which chooses the payload values); this contrasts with unidirectional approaches that only modify output traces. In this richer setting, it helps to differentiate between the transformations performed by the monitor (*i.e.*, insertions, suppressions and replacements), and the way they can be used to affect the resulting behaviour of the composite system. In particular, we say that:

- an action that can be performed by the SuS has been *disabled* when it is no longer visible in the resulting composite system (consisting of the SuS and the monitor);
- an action is *enabled* when the composite system can execute it while the SuS cannot;
- an action is *adapted* when either its payload differs from that of the composite system, or when the action is rerouted through a different port.

We argue that implementing action enabling, disabling and adaptation differs according to whether the action is an input or an output; see Figure 3. Enforcing actions instigated by the SuS—such as outputs—is more straightforward. Figure 3(a), (b) and (c) *resp.* state that disabling an output can be achieved by suppressing it, adapting an output amounts to replacing the payload or redirecting it to a different port, whereas output enabling can be attained via an insertion. However, enforcing actions instigated by the environment such as inputs is harder. In Figure 3(d), we propose to *disable* an input by concealing the input port. Since this may block the SuS from progressing, the instrumented monitor may additionally

Syntax

$$m, n \in \textsc{Trn} ::= (p, c, p').m \quad | \; \textstyle\sum_{i\in I} m_i \; (I \text{ is a finite index set}) \quad | \; \mathsf{rec}\, X.m \quad | \; X$$

Dynamics

$$\textsc{eSel}\,\frac{m_j \xrightarrow{\gamma\blacktriangleright\gamma'} n_j}{\sum_{i\in I} m_i \xrightarrow{\gamma\blacktriangleright\gamma'} n_j}\; j\in I \qquad \textsc{eRec}\,\frac{m\{^{\mathsf{rec}\,X.m}/_X\} \xrightarrow{\gamma\blacktriangleright\gamma'} n}{\mathsf{rec}\,X.m \xrightarrow{\gamma\blacktriangleright\gamma'} n}$$

$$\textsc{eTrn}\,\frac{\mathsf{match}(p,\gamma) = \sigma \qquad c\sigma \Downarrow \mathsf{true} \qquad \gamma' = \pi\sigma}{(p, c, \pi).m \xrightarrow{\gamma\blacktriangleright\gamma'} m\sigma}$$

Instrumentation

$$\textsc{biTrnO}\,\frac{s \xrightarrow{\mathsf{b}!w} s' \qquad m \xrightarrow{(\mathsf{b}!w)\blacktriangleright(\mathsf{a}!v)} n}{m[s] \xrightarrow{\mathsf{a}!v} n[s']} \qquad \textsc{biTrnI}\,\frac{m \xrightarrow{(\mathsf{a}?v)\blacktriangleright(\mathsf{b}?w)} n \qquad s \xrightarrow{\mathsf{b}?w} s'}{m[s] \xrightarrow{\mathsf{a}?v} n[s']}$$

$$\textsc{biDisO}\,\frac{s \xrightarrow{\mathsf{a}!v} s' \qquad m \xrightarrow{(\mathsf{a}!v)\blacktriangleright\bullet} n}{m[s] \xrightarrow{\tau} n[s']} \qquad \textsc{biDisI}\,\frac{m \xrightarrow{\bullet\blacktriangleright(\mathsf{a}?v)} n \qquad s \xrightarrow{\mathsf{a}?v} s'}{m[s] \xrightarrow{\tau} n[s']}$$

$$\textsc{biEnO}\,\frac{m \xrightarrow{\bullet\blacktriangleright(\mathsf{a}!v)} n}{m[s] \xrightarrow{\mathsf{a}!v} n[s]} \qquad \textsc{biEnI}\,\frac{m \xrightarrow{(\mathsf{a}?v)\blacktriangleright\bullet} n}{m[s] \xrightarrow{\mathsf{a}?v} n[s]} \qquad \textsc{biAsy}\,\frac{s \xrightarrow{\tau} s'}{m[s] \xrightarrow{\tau} m[s']}$$

$$\textsc{biDef}\,\frac{s \xrightarrow{\mathsf{a}!v} s' \qquad m \not\xrightarrow{\mathsf{a}!v} \qquad \forall\, \mathsf{b}\in\textsc{Port}, w\in\textsc{Val}\cdot m \not\xrightarrow{\bullet\blacktriangleright\mathsf{b}!w}}{m[s] \xrightarrow{\mathsf{a}!v} \mathsf{id}[s']}$$

FIGURE 4. A bidirectional instrumentation model for enforcement monitors.

insert a default input to unblock the system waiting to input on the channel used for the insertion, Figure 3(e), in cases where the environment fails to provide the corresponding output. Input *adaptation*, Figure 3(f), is also attained via a *replacement*, albeit applied in the opposite direction to the output case. Inputs can also be *enabled* whenever the SuS is unable to carry them out, Figure 3(g), by having the monitor accept the input in question and then *suppress* it. Note that, from the perspective of the environment, the input would still be effected.

Figure 4 presents an operational model for the bidirectional instrumentation proposal of Figure 3 in terms of (symbolic) transducers. A variant of these transducers was originally introduced in [ACFI18] for unidirectional enforcement. Transducers, $m, n \in \textsc{Trn}$, are monitors that define *symbolic transformation triples*, (p,c,π), consisting of an action *pattern* p, *condition* c, and a *transformation action* π. Conceptually, the action pattern and condition determine the range of system (input or output) actions upon which the transformation should be applied, while the transformation action specifies the transformation that should be applied. The symbolic transformation pattern p is an extended version of those definable in symbolic actions, that may also include $\bullet$; when $p = \bullet$, it means that the monitor can act independently from the system to insert the action specified by the transformation action. Transformation actions are possibly open actions (*i.e.,* actions with possibly free variable such as $x?v$ or $\mathsf{a}!x$) or the special action $\bullet$; the latter represents the suppression of the action specified by p. We assume a well-formedness constraint where, for every $(p, c, \pi).m$, p and π cannot both be $\bullet$, and when neither is, they are of the *same* type *i.e.,* an input (*resp.* output) pattern and action. Examples of well-formed symbolic transformations are:

- $(\bullet, \mathsf{true}, \mathsf{a}?v)$, inserting an input on port a with value v;
- $((x)!(y), y \geq 5, \bullet)$, suppressing an output action carrying a payload that is greater or equal to 5; and
- $((x)!(y), x = \mathsf{b}, \mathsf{a}!y)$, redirecting (*i.e.,* adapting) outputs on port b carrying the payload y (learnt dynamically at runtime) to port a.

The monitor transition rules in Figure 4 assume closed terms, *i.e.,* every *transformation-prefix transducer* of the form $(p, c, \pi).m$ must obey the closure constraint stating that $\big(\mathbf{fv}(c) \cup \mathbf{fv}(\pi) \cup \mathbf{fv}(m)\big) \subseteq \mathbf{bv}(p)$. A similar closure requirement is assumed for recursion variables X and $\mathsf{rec}\, X.m$. Each transformation-prefix transducer yields an LTS with labels of the form $\gamma \blacktriangleright \gamma'$, where $\gamma, \gamma' \in (\textsc{Act} \cup \{\bullet\})$. Intuitively, transition $m \xrightarrow{\gamma \blacktriangleright \gamma'} n$ denotes the way that a transducer in state m *transforms* the action γ into γ' while transitioning to state n. The transducer action $\alpha \blacktriangleright \beta$ represents the *replacement* of α by β, $\alpha \blacktriangleright \alpha$ denotes the *identity* transformation, whereas $\alpha \blacktriangleright \bullet$ and $\bullet \blacktriangleright \alpha$ respectively denote the *suppression* and *insertion* transformations of action α. The key transition rule in Figure 4 is eTrn. It states that the transformation-prefix transducer $(p, c, \pi).m$ transforms action γ into a (potentially) different action γ' and reduces to state $m\sigma$, whenever γ matches pattern p, *i.e.,* $\mathsf{match}(p, \gamma) = \sigma$, and satisfies condition c, *i.e.,* $c\sigma \Downarrow \mathsf{true}$. Action γ' results from instantiating the free variables in π as specified by σ, *i.e.,* $\gamma' = \pi\sigma$. The remaining rules for selection (eSel) and recursion (eRec) are standard. We employ the shorthand notation $m \overset{\gamma}{\not\rightarrow}$ to mean $\nexists \gamma', m'$ such that $m \xrightarrow{\gamma \blacktriangleright \gamma'} m$. Moreover, for the semantics of Figure 4, we can encode the identity transducer/monitor, id, as follows

$$\mathsf{id} \stackrel{\text{def}}{=} \mathsf{rec}\, Y.((x)!(y), \mathsf{true}, x!y).Y + ((x)?(y), \mathsf{true}, x?y).Y. \tag{3.1}$$

When instrumented with any arbitrary system, the identity monitor id leaves its behaviour unchanged. As a shorthand notation, we write $(p, c).m$ instead of $(p, c, \pi).m$ when all the binding occurrences (x) in p correspond to free occurrences x in π, thus denoting an identity transformation. Similarly, we elide c whenever $c = \mathsf{true}$.

The first contribution of this work lies in the new *instrumentation relation* of Figure 4, linking the behaviour of the SuS s with that of a monitor m: the term $m[s]$ denotes their composition as a *monitored system*. Crucially, the instrumentation rules in Figure 4 give us a semantics in terms of an LTS over the actions $\textsc{Act} \cup \{\tau\}$, in line with the LTS semantics of the SuS. Following Figure 3(b), rule biTrnO states that if the SuS transitions with an output $\mathsf{b}!w$ to s' and the transducer can *replace* it with $\mathsf{a}!v$ and transition to n, the *adapted* output can be externalised so that the composite system $m[s]$ transitions over $\mathsf{a}!v$ to $n[s']$. Rule biDisO states that if s performs an output $\mathsf{a}!v$ that the monitor *can suppress*, the instrumentation withholds this output and the composite system silently transitions; this amounts to action *disabling* as outlined in Figure 3(a). Rule biEnO is dual, and it *enables* the output $\mathsf{a}!v$ on the SuS as outlined in Figure 3(c): it augments the composite system $m[s]$ with an output $\mathsf{a}!v$ whenever m can *insert* $\mathsf{a}!v$, independently of the behaviour of s. Rules biDisO, biTrnO and biEnO therefore correspond to items (a), (b) and (c) in Figure 3 respectively.

Rule biDef is analogous to standard rules for premature monitor termination [Fra21, FAI17, Fra17, AAFI18a], and accounts for underspecification of transformations. We, however, restrict defaulting (termination) to output actions performed by the SuS exclusively, *i.e.,* a monitor only defaults to id when it cannot react to or enable a system output. By forbidding the monitor from defaulting upon unspecified inputs, the monitor is able to *block* them from becoming part of the composite system's behaviour. Hence, any input that the monitor is unable to react to, *i.e.,* $m \stackrel{\mathsf{a}?v\blacktriangleright\gamma}{\not\longrightarrow}$, is considered as being *invalid and blocked* by default. This technique is thus used to implement Figure 3(d). To avoid disabling valid inputs unnecessarily, the monitor must therefore explicitly define symbolic transformations that cover *all* the valid inputs of the SuS. Note, that rule biAsy still allows the SuS to silently transition independently of m. Following Figure 3(f), rule biTrnI adapts inputs, provided the SuS can accept the adapted input. Similarly, rule biEnI *enables* an input on a port a as described in Figure 3(g): the composite system accepts the input while suppressing it from the SuS. Rule biDisI allows the monitor to generate a default input value v and forward it to the SuS on a port a, thereby unblocking it whenever the environment is unable to provide the corresponding output on channel a (carrying v); from the environment's perspective, the composite system silently transitions to some state, following Figure 3(e). It is worth comparing rule biDisI with the other instrumentation rule biEnO discussed earlier, since they both handle outputs inserted by the monitor. In the case of rule biEnO, whenever the monitor inserts an output to be consumed *by the environment*, this is expressed at the level of the composite system as an external output (see conclusion of rule biEnO) since, in our LTS, the actions represent the interaction between the (composite) system and the environment. Contrastingly, whenever the monitor inserts an output to be input *by the SuS*, then this is expressed at the level of the composite system as a silent action (see conclusion of rule biDisI) since no interaction occurs between the (composite) system and the environment. We conclude our discussion of the instrumenation rules in Figure 4 by remarking that rules biDisI, biTrnI and biEnI respectively implement items (e), (f) and (g) of Figure 3.

Definition 3.1. We call disabling monitors/transducers those monitors that only perform disabling actions. The same applies to enabling and adapting monitors/transducers. ■

Example 3.2. Consider the following action disabling transducer $m_{\mathbf{d}}$, that repeatedly disables every output performed by the system via the branch $((\_)!(\_), \bullet).Y$. In addition, it limits inputs to those on port b via the input branch $(\mathsf{b}?(\_)).Y$; inputs on other ports are disabled since none of the relevant instrumentation rules in Figure 4 can be applied.

$$m_{\mathbf{d}} \stackrel{\mathsf{def}}{=} \mathsf{rec}\, Y.(\mathsf{b}?(\_)).Y + ((\_)!(\_), \bullet).Y$$

Recall the two systems below from Example 2.4:

$$s_{\mathbf{b}} \stackrel{\mathsf{def}}{=} \mathsf{rec}\, X.((\mathsf{a}?x.y := \mathsf{ans}(x).\mathsf{a}!y.(\underline{\mathsf{a}!y}.\mathsf{b}!(\mathsf{log}, x, y).s_{\mathbf{a}} + \mathsf{b}!(\mathsf{log}, x, y).X)) + s_{\mathbf{cls}})$$
$$s_{\mathbf{c}} \stackrel{\mathsf{def}}{=} \mathsf{a}?y.s_{\mathbf{a}}$$

where

$$s_{\mathbf{a}} \stackrel{\mathsf{def}}{=} \mathsf{rec}\, X.((\mathsf{a}?x.y := \mathsf{ans}(x).\mathsf{a}!y.\mathsf{b}!(\mathsf{log}, x, y).X) + s_{\mathbf{cls}}) \qquad \text{and}$$
$$s_{\mathbf{cls}} \stackrel{\mathsf{def}}{=} (\mathsf{b}?z.\mathsf{if}\ z = \mathsf{cls}\ \mathsf{then}\ \mathsf{nil}\ \mathsf{else}\ X)$$

When instrumented with the system $s_{\mathbf{c}}$, monitor $m_{\mathbf{d}}$ blocks its initial input, *i.e.*, we have $m_{\mathbf{d}}[s_{\mathbf{c}}] \not\xrightarrow{\alpha}$ for any α. In the case of $s_{\mathbf{b}}$, the composite system $m_{\mathbf{d}}[s_{\mathbf{b}}]$ can only input requests on port b, such as the termination request $m_{\mathbf{d}}[s_{\mathbf{b}}] \xrightarrow{\mathsf{b?cls}} m_{\mathbf{d}}[\mathsf{nil}]$.

$$m_{\mathbf{dt}} \stackrel{\text{def}}{=} \mathsf{rec}\, X.(((x)?(y_1), x{\neq}\mathsf{b}).(((x_1)?(\_), x_1 \neq x).\mathsf{id} + (x!(y_2)).m'_{\mathbf{dt}}) + (\mathsf{b}?(\_)).\mathsf{id})$$
$$m'_{\mathbf{dt}} \stackrel{\text{def}}{=} (x!(\_),\bullet).m_{\mathbf{d}} + ((\_)?(\_)).\mathsf{id} + (\mathsf{b}!(y_3), y_3{=}(\mathsf{log}, y_1, y_2)).X$$

By defining branch $(\mathsf{b}?(\_)).\mathsf{id}$, the more elaborate monitor $m_{\mathbf{dt}}$ (above) allows the SuS to immediately input on port b (possibly carrying a termination request). At the same time, the branch prefixed by $((x)?(y_1), x{\neq}\mathsf{b})$ permits the SuS to input the first request via any port $x \neq \mathsf{b}$, subsequently blocking inputs on the same port x (without deterring inputs on other ports) via the input branch $((x_1)?(\_), x_1 \neq x).\mathsf{id}$. In conjunction to this branch, $m_{\mathbf{dt}}$ defines another branch $(x!(y_2)).m'_{\mathbf{dt}}$ to allow outputs on the port bound to variable x. The continuation monitor $m'_{\mathbf{dt}}$ then defines the suppression branch $(x!(\_),\bullet).m_{\mathbf{d}}$ by which it disables any *redundant* response that is output following the first one. Since it also defines branches $(\mathsf{b}!(y_3), y_3{=}(\mathsf{log}, y_1, y_2)).X$ and $((\_)?(\_)).\mathsf{id}$, it does not affect log events or further inputs that occur immediately after the first response.

When instrumented with system $s_{\mathbf{c}}$ from Example 2.4, $m_{\mathbf{dt}}$ allows the composite system to perform the first input but then blocks the second one, permitting only input requests on channel b, *e.g.*,

$$m_{\mathbf{dt}}[s_{\mathbf{c}}] \xrightarrow{\mathsf{a}?v} \cdot \xrightarrow{\mathsf{b?cls}} \mathsf{id}[\mathsf{nil}].$$

It also disables the first redundant response of system $s_{\mathbf{b}}$ while transitioning to $m_{\mathbf{d}}$, which proceeds to suppress every subsequent output (including log actions) while blocking every other port except b, *i.e.*,

$$m_{\mathbf{dt}}[s_{\mathbf{b}}] \xrightarrow{\mathsf{a}?v} \cdot \xRightarrow{\mathsf{a}!w} \cdot \xrightarrow{\tau} m_{\mathbf{d}}[\mathsf{b}!(\mathsf{log}, v, w).s_{\mathbf{a}}] \xrightarrow{\tau} m_{\mathbf{d}}[s_{\mathbf{a}}] \not\xrightarrow{\mathsf{a}?v}$$

(for every port a where $\mathsf{a}{\neq}\mathsf{b}$ and any value v). Rule iDef allows it to default when handling unspecified outputs, *e.g.*, for system $\mathsf{b}!(\mathsf{log}, v, w).s_{\mathbf{a}}$ the composite system can still perform the logging output, *i.e.*,

$$m_{\mathbf{dt}}[\mathsf{b}!(\mathsf{log}, v, w).s_{\mathbf{a}}] \xrightarrow{\mathsf{b}!(\mathsf{log},v,w)} \mathsf{id}[s_{\mathbf{a}}].$$

Consider one further monitor, defined below:

$$m_{\mathbf{det}} \stackrel{\text{def}}{=} \mathsf{rec}\, X.(((x)?(y_1), x{\neq}\mathsf{b}).m'_{\mathbf{det}} + (\mathsf{b}?(\_)).\mathsf{id})$$
$$m'_{\mathbf{det}} \stackrel{\text{def}}{=} \mathsf{rec}\, Y_1.\underline{(\bullet, x?v_{\text{def}}).Y_1} + (x!(y_2)).m''_{\mathbf{det}} + ((x_1)?(\_), x_1 \neq x).\mathsf{id}$$
$$m''_{\mathbf{det}} \stackrel{\text{def}}{=} \mathsf{rec}\, Y_2.\big(\underline{(x!(\_), x \neq \mathsf{b},\bullet).Y_2} + (\mathsf{b}!(y_3), y_3{=}(\mathsf{log}, y_1, y_2)).X + ((\_)?(\_)).\mathsf{id}\big)$$

Monitor $m_{\mathbf{det}}$ (above) behaves similarly to $m_{\mathbf{dt}}$ but instead employs a loop of suppressions (underlined in $m''_{\mathbf{det}}$) to disable further responses until a log or termination input is made. When composed with $s_{\mathbf{b}}$, it permits the log action to go through:

$$m_{\mathbf{det}}[s_{\mathbf{b}}] \xrightarrow{\mathsf{a}?v} \cdot \xRightarrow{\mathsf{a}!w} \cdot \xrightarrow{\tau} m''_{\mathbf{det}}[\mathsf{b}!(\mathsf{log}, v, w).s_{\mathbf{b}}] \xrightarrow{\mathsf{b}!(\mathsf{log},v,w)} m_{\mathbf{det}}[s_{\mathbf{b}}].$$

$m_{\mathbf{det}}$ also defines a branch prefixed by the insertion transformation $(\bullet, x?v_{\text{def}})$ (underlined in $m'_{\mathbf{det}}$) where v_{def} is a default input domain value. This permits the instrumentation to silently unblock the SuS when this is waiting for a request following an unanswered one. In fact, when instrumented with $s_{\mathbf{c}}$, $m_{\mathbf{det}}$ not only forbids invalid input requests, but it also (internally) unblocks $s_{\mathbf{c}}$ by supplying the required input via the added insertion branch. This allows the composite system to proceed, as shown below (where $s'_{\mathbf{a}} \stackrel{\text{def}}{=} y := \mathsf{ans}(v_{\text{def}}).\mathsf{a}!y.\mathsf{b}!(\mathsf{log}, v_{\text{def}}, y).s_{\mathbf{a}}$):

$$\begin{aligned} m_{\mathbf{det}}[s_{\mathbf{c}}] &\xrightarrow{\mathsf{a}?v} \mathsf{rec}\, Y.((\bullet, \mathsf{a}?v_{\text{def}}).Y + (\mathsf{a}!(y_2)).m''_{\mathbf{det}} + (\mathsf{b}?(\_)).\mathsf{id})[s_{\mathbf{a}}] \\ &\xrightarrow{\tau} \mathsf{rec}\, Y.((\bullet, \mathsf{a}?v_{\text{def}}).Y + (\mathsf{a}!(y_2)).m''_{\mathbf{det}} + (\mathsf{b}?(\_)).\mathsf{id})[s'_{\mathbf{a}}] \\ &\xRightarrow{\mathsf{a}!\mathsf{ans}(v_{\text{def}}).\mathsf{b}!(\mathsf{log}, v_{\text{def}}, y)} m_{\mathbf{det}}[s_{\mathbf{a}}] \end{aligned}$$ ■

Although in this paper we mainly focus on action disabling monitors, using our model one can also define action enabling and adaptation monitors.

Example 3.3. Consider now the transducers $m_{\mathbf{e}}$ and $m_{\mathbf{a}}$ below:

$$\begin{aligned} m_{\mathbf{e}} &\stackrel{\text{def}}{=} ((x)?(y), x{\neq}\mathsf{b}, \bullet).(\bullet, x!\mathsf{ans}(y)).(\bullet, \mathsf{b}!(\mathsf{log}, y, \mathsf{ans}(y))).\mathsf{id} \\ m_{\mathbf{a}} &\stackrel{\text{def}}{=} \mathsf{rec}\, X.(\mathsf{b}?(y), \mathsf{a}?y).X + (\mathsf{a}!(y), \mathsf{b}!y).X. \end{aligned}$$

Once instrumented, $m_{\mathbf{e}}$ first uses a suppression to enable an input on any port $x \neq \mathsf{b}$ (but then gets discarded). It then automates a response by inserting an answer followed by a log action. Concretely, when composed with the systems $r \in \{s_{\mathbf{b}}, s_{\mathbf{c}}\}$ from Example 2.4 (restated in Example 3.2), the execution of the composite system can only start as follows, for some channel name $\mathsf{c} \neq \mathsf{b}$, values v and $w = \mathsf{ans}(v)$:

$$m_{\mathbf{e}}[r] \xrightarrow{\mathsf{c}?v} (\bullet, \mathsf{c}!w).(\bullet, \mathsf{b}!(\mathsf{log}, v, w)).\mathsf{id}[r] \xRightarrow{\mathsf{c}!w} (\bullet, \mathsf{b}!(\mathsf{log}, v, w)).\mathsf{id}[r] \xrightarrow{\mathsf{b}!(\mathsf{log}, v, w)} \mathsf{id}[r].$$

By contrast, $m_{\mathbf{a}}$ uses action adaptation to redirect the inputs and outputs from the SuS through port b: it allows the composite system to exclusively input values on port b forwarding them to the SuS on port a, and dually allowing outputs from the SuS on port a to reroute them to port b. As a result, from an external viewpoint, the resulting composite system can only be seen to communicate on port b with its environment. For instance, for the systems $s_{\mathbf{c}}$ and $s_{\mathbf{b}}$ restated earlier, we can observe the following behaviour:

$$m_{\mathbf{a}}[s_{\mathbf{c}}] \xrightarrow{\mathsf{b}?v_1} m_{\mathbf{a}}[s_{\mathbf{a}}] \xRightarrow{\mathsf{b}?v_2.\mathsf{b}!w_2.\mathsf{b}!(\mathsf{log}, v_2, w_2)} m_{\mathbf{a}}[s_{\mathbf{a}}] \quad \text{and}$$

$$m_{\mathbf{a}}[s_{\mathbf{b}}] \xRightarrow{\mathsf{b}?v_1.\mathsf{b}!w_1.\mathsf{b}!(\mathsf{log}, v_1, w_1)} m_{\mathbf{a}}[s_{\mathbf{b}}].$$ ■

4. Enforcement

We are concerned with extending the enforceability result obtained in prior work [ACFI18] to the extended setting of bidirectional enforcement. The *enforceability* of a logic rests on the relationship between the semantic behaviour specified by the logic on the one hand, and the ability of the operational mechanism (that of Section 3 in this case) to enforce the specified behaviour on the other. This is captured by the predicate "(monitor) m *adequately enforces* (property) φ" in Definition 4.1 below. In fact, the definitions of formula and logic enforceability in Definition 4.1 are parametric with respect to the precise meaning of such a predicate. In what follows, we will explore the design space for formalising this predicate.

Definition 4.1 (Enforceability [ACFI18])**.** A formula φ is *enforceable* iff there *exists* a transducer m such that m *adequately enforces* φ. A logic $\mathcal{L}$ is enforceable iff *every* formula $\varphi \in \mathcal{L}$ is *enforceable.* ■

Since we have limited control over the SuS that a monitor is composed with, "m *adequately enforces* φ" should hold for *any* (instrumentable) system. In [ACFI18] we stipulate that any notion of adequate enforcement should at least entail soundness.

Definition 4.2 (Sound Enforcement [ACFI18])**.** Monitor m *soundly enforces* a formula φ, denoted as $\mathsf{senf}(m,\varphi)$, iff, whenever φ is satisfiable, *i.e.*, $[\![\varphi]\!] \neq \emptyset$, then for every state $s \in \textsc{Sys}$, it is the case that $m[s] \in [\![\varphi]\!]$. ■

Example 4.3. Although showing that a monitor soundly enforces a formula should consider *all* systems, we give an intuition based on $s_{\mathbf{a}}$, $s_{\mathbf{b}}$, $s_{\mathbf{c}}$ for formula φ_1 from Example 2.4 (restated below) where $s_{\mathbf{a}} \in [\![\varphi_1]\!]$ (hence $[\![\varphi_1]\!] \neq \emptyset$) and $s_{\mathbf{b}}, s_{\mathbf{c}} \notin [\![\varphi_1]\!]$.

$$\varphi_1 \stackrel{\text{def}}{=} \mathsf{max}\, X.[((x)?(y_1), x{\neq}\mathsf{b})]([(x?(\_))]\mathsf{ff} \wedge [(x!(y_2))]\varphi_1')$$
$$\varphi_1' \stackrel{\text{def}}{=} ([(x!(\_))]\mathsf{ff} \wedge [(\mathsf{b}!(y_3), y_3{=}(\mathsf{log}, y_1, y_2))]X)$$

Recall the transducers $m_{\mathbf{e}}$, $m_{\mathbf{a}}$, $m_{\mathbf{d}}$, $m_{\mathbf{dt}}$ and $m_{\mathbf{det}}$ from Examples 3.2 and 3.3, restated below:

$$m_{\mathbf{e}} \stackrel{\text{def}}{=} ((x)?(y), x{\neq}\mathsf{b}, \bullet).(\bullet, x!\mathsf{ans}(y)).(\bullet, \mathsf{b}!(\mathsf{log}, y, \mathsf{ans}(y))).\mathsf{id}$$
$$m_{\mathbf{a}} \stackrel{\text{def}}{=} \mathsf{rec}\, X.(\mathsf{b}?(y), \mathsf{a}?y).X + (\mathsf{a}!(y), \mathsf{b}!y).X$$
$$m_{\mathbf{d}} \stackrel{\text{def}}{=} \mathsf{rec}\, Y.(\mathsf{b}?(\_)).Y + ((\_)!(\_), \bullet).Y$$
$$m_{\mathbf{dt}} \stackrel{\text{def}}{=} \mathsf{rec}\, X.(((x)?(y_1), x{\neq}\mathsf{b}).(((x_1)?(\_), x_1 \neq x).\mathsf{id} + (x!(y_2)).m_{\mathbf{dt}}') + (\mathsf{b}?(\_)).\mathsf{id})$$
$$m_{\mathbf{det}} \stackrel{\text{def}}{=} \mathsf{rec}\, X.(((x)?(y_1), x{\neq}\mathsf{b}).m_{\mathbf{det}}' + (\mathsf{b}?(\_)).\mathsf{id})$$

where

$$m_{\mathbf{dt}}' \stackrel{\text{def}}{=} (x!(\_),\bullet).m_{\mathbf{d}} + ((\_)?(\_)).\mathsf{id} + (\mathsf{b}!(y_3), y_3{=}(\mathsf{log}, y_1, y_2)).X$$
$$m_{\mathbf{det}}' \stackrel{\text{def}}{=} \mathsf{rec}\, Y_1.\underline{(\bullet, x?v_{\text{def}}).Y_1} + (x!(y_2)).m_{\mathbf{det}}'' + ((x_1)?(\_), x_1 \neq x).\mathsf{id}$$
$$m_{\mathbf{det}}'' \stackrel{\text{def}}{=} \mathsf{rec}\, Y_2.\big(\underline{(x!(\_), x \neq \mathsf{b}, \bullet).Y_2} + (\mathsf{b}!(y_3), y_3{=}(\mathsf{log}, y_1, y_2)).X + ((\_)?(\_)).\mathsf{id}\big)$$

When assessing their soundness in relation to the property φ_1, we have that:

- $m_{\mathbf{e}}$ is *unsound* for φ_1. When composed with $s_{\mathbf{b}}$, the resulting monitored system produces two consecutive output replies (namely those underlined in the trace t_{e}^1 below), thus meaning that the composite system violates the property in question, *i.e.*, $m_{\mathbf{e}}[s_{\mathbf{b}}] \notin [\![\varphi_1]\!]$. More concretely, we have

$$m_{\mathbf{e}}[s_{\mathbf{b}}] \xRightarrow{t_{\mathsf{e}}^1} \mathsf{id}[s_{\mathbf{b}}] \quad \text{ where } t_{\mathsf{e}}^1 \stackrel{\text{def}}{=} \mathsf{c}?v_1.\mathsf{c}!\mathsf{ans}(v_1).\mathsf{b}!(\mathsf{log}, v_1, \mathsf{ans}(v_1)).\mathsf{a}?v_2.\underline{\mathsf{a}!w_2.\mathsf{a}!w_2}.$$

  Similarly, the system $s_{\mathbf{c}}$ instrumented with the transducer $m_{\mathbf{e}}$ also violates property φ_1, *i.e.*, $m_{\mathbf{e}}[s_{\mathbf{c}}] \notin [\![\varphi_1]\!]$, since the $m_{\mathbf{e}}[s_{\mathbf{c}}]$ executes the erroneous trace with two consecutive inputs on port a (underlined), $\mathsf{c}?v_1.\mathsf{c}!\mathsf{ans}(v_1).\mathsf{b}!(\mathsf{log}, v_1, \mathsf{ans}(v_1)).\underline{\mathsf{a}?w_2.\mathsf{a}?w_3}$. This demonstrates that $m_{\mathbf{e}}[s_{\mathbf{c}}]$ can still input two consecutive requests on port a (underlined). Either one of these counterexamples *disproves* $\mathsf{senf}(m_{\mathbf{e}}, \varphi_1)$.
- Monitor $m_{\mathbf{a}}$ turns out to be *sound* for φ_1 because once instrumented, the resulting composite system is adapted to only interact on port b. In fact, we have $\{m_{\mathbf{a}}[s_{\mathbf{a}}], m_{\mathbf{a}}[s_{\mathbf{b}}], m_{\mathbf{a}}[s_{\mathbf{c}}]\} \subseteq [\![\varphi_1]\!]$. The other monitors $m_{\mathbf{d}}$, $m_{\mathbf{dt}}$ and $m_{\mathbf{det}}$ are also *sound* for φ_1. Whereas, monitor $m_{\mathbf{d}}$ prevents the violation of φ_1 by also blocking all input ports except b, the transducers $m_{\mathbf{dt}}$ and $m_{\mathbf{det}}$ achieve the same goal by disabling the invalid consecutive requests and answers that occur on any port except b. ■

By itself, sound enforcement is a weak criterion because it does not regulate the *extent* to which enforcement is applied. More specifically, although $m_{\mathbf{d}}$ from Example 3.2 is sound, it needlessly modifies the behaviour of $s_{\mathbf{a}}$ even though $s_{\mathbf{a}}$ satisfies φ_1: by blocking the initial input of $s_{\mathbf{a}}$, $m_{\mathbf{d}}$ causes it to block indefinitely. The requirement that a monitor should not modify the behaviour of a system that satisfies the property being enforced can be formalised using a transparency criterion.

Definition 4.4 (Transparent Enforcement [ACFI18])**.** A monitor m *transparently* enforces a formula φ, $\mathsf{tenf}(m, \varphi)$, iff for *all* $s \in \textsc{Sys}$, $s \in [\![\varphi]\!]$ implies $m[s] \sim s$. □

Example 4.5. As argued earlier, $s_{\mathbf{a}}$ suffices to disprove $\mathsf{tenf}(m_{\mathbf{d}}, \varphi_1)$. Monitor $m_{\mathbf{a}}$ from Example 3.3 also breaches Definition 4.4: although $s_{\mathbf{a}} \in [\![\varphi_1]\!]$, we have $m_{\mathbf{a}}[s_{\mathbf{a}}] \not\sim s_{\mathbf{a}}$ since for any value v and w, $s_{\mathbf{a}} \xrightarrow{\mathsf{a}?v}$ but for any value v we can *never* have $m_{\mathbf{a}}[s_{\mathbf{a}}] \xrightarrow{\mathsf{a}?v}$. By contrast, monitors $m_{\mathbf{dt}}$ and $m_{\mathbf{det}}$ turn out to satisfy Definition 4.4, since they only intervene when it becomes apparent that a violation will occur. For instance, they only disable inputs on a specific port, as a precaution, following an unanswered request on the same port, and they only disable the redundant responses that are produced after the first response to a request. ■

It turns out that, by some measures, Definition 4.4 is still a relatively weak requirement since it only limits transparency requirements to *well-behaved* systems, *i.e.,* those that satisfy the property in question, and disregards enforcement behaviour for systems that violate this property. For instance, consider monitor $m_{\mathbf{dt}}$ from Example 3.2 (restated in Example 4.3) and system $s_{\mathbf{b}}$ from Example 2.4 (restated in Example 3.2). At runtime, $s_{\mathbf{b}}$ can exhibit the following invalid behaviour:

$$s_{\mathbf{b}} \xRightarrow{t_1} \mathsf{b}!(\mathsf{log}, v, w).s_{\mathbf{a}} \quad \text{where } t_1 \stackrel{\text{def}}{=} \mathsf{a}?v.\mathsf{a}!w.\mathsf{a}!w \text{ for some appropriate pair of values } v, w.$$

In order to rectify this violating behaviour *wrt.* formula φ_1, it suffices to use a monitor that disables *one* of the responses in t_1, *i.e.,* $\mathsf{a}!w$. Following this disabling, no further modifications are required since the SuS reaches a state that does not violate the remainder of the formula φ_1, *i.e.,* $\mathsf{b}!(\mathsf{log}, v, w).s_{\mathbf{a}} \in [\![\mathit{after}(\varphi_1, t_1')]\!]$ where $t_1' \stackrel{\text{def}}{=} \mathsf{a}?v.\mathsf{a}!w$. However, when instrumented with $m_{\mathbf{dt}}$, this monitor does not only disable the invalid response, namely $m_{\mathbf{dt}}[s_{\mathbf{b}}] \xRightarrow{\mathsf{a}?v.\mathsf{a}!w.} m_{\mathbf{d}}[\mathsf{b}!(\mathsf{log}, v, w).s_{\mathbf{a}}]$, but subsequently disables every other action by reaching $m_{\mathbf{d}}$, $m_{\mathbf{d}}[\mathsf{b}!(\mathsf{log}, v, w).s_{\mathbf{a}}] \xrightarrow{\tau} m_{\mathbf{d}}[s_{\mathbf{a}}]$. To this end, we introduce the novel requirement of *eventual transparency.*

Definition 4.6 (Eventually Transparent Enforcement)**.** Monitor m enforces property φ in an eventually transparent way, $\mathsf{evtenf}(m, \varphi)$, iff for all systems s, s', traces t and monitors m', $m[s] \xRightarrow{t} m'[s']$ and $s' \in [\![\mathit{after}(\varphi, t)]\!]$ imply $m'[s'] \sim s'$. ■

Example 4.7. We have already argued why $m_{\mathbf{dt}}$ (restated in Example 4.3) does not adhere to eventual transparency via the counterexample $s_{\mathbf{b}}$. This is not the case for $m_{\mathbf{det}}$ (also restated in Example 4.3). Although the universal quantification over all systems and traces make it hard to prove this property, we get an intuition of why this is the case from the system $s_{\mathbf{b}}$. More concretely, when

$$m_{\mathbf{det}}[s_{\mathbf{b}}] \xRightarrow{\mathsf{a}?v_1.\mathsf{a}!w_1} \cdot \xrightarrow{\tau} m''_{\mathbf{det}}[\mathsf{b}!(\mathsf{log}, v_1, w_1).s_{\mathbf{a}}]$$

we have

$$\mathsf{b}!(\mathsf{log}, v_1, w_1).s_{\mathbf{a}} \in [\![\mathit{after}(\varphi_1, \mathsf{a}?v_1.\mathsf{a}!w_1)]\!]$$

since

$$\mathit{after}(\varphi_1, \mathsf{a}?v_1.\mathsf{a}!w_1) = ([((x_3)!(\_), x_3{=}\mathsf{a})]\mathsf{ff} \wedge [((x_4)!(y_3), x_4{=}\mathsf{b} \wedge y_3{=}(\mathsf{log}, v_1, w_1))]\varphi_1)$$

and, moreover, $m''_{\mathbf{det}}[\mathsf{b}!(\mathsf{log}, v_1, w_1).s_{\mathbf{a}}] \sim \mathsf{b}!(\mathsf{log}, v_1, w_1).s_{\mathbf{a}}$. ■

Corollary 4.8. *For all monitors $m \in \textsc{Trn}$ and properties $\varphi \in$ sHML, $\mathsf{evtenf}(m, \varphi)$ implies $\mathsf{tenf}(m, \varphi)$.* □

Along with Definition 4.2 (soundness), Definition 4.6 (eventual transparency) makes up our definition for *"m (adequately) enforces φ"*. From Corollary 4.8, it follows that is definition is stricter than the one given in [ACFI18].

Definition 4.9 (Adequate Enforcement)**.** A monitor m (adequately) *enforces* property φ, denoted as $\mathsf{enf}(n, \varphi)$, iff it adheres to (i) *soundness*, Definition 4.2, and (ii) *eventual transparency*, Definition 4.6. ■

5. Synthesising Action Disabling Monitors

Although Definition 4.1 (instantiated with Definition 4.9) enables us to rule out erroneous monitors that purport to enforce a property, the *universal quantifications* over all systems in Definitions 4.2 and 4.6 make it difficult to prove that a monitor does indeed enforce a property correctly in a bidirectional setting (disproving, however, is easier). Establishing that a formula is enforceable, Definition 4.9, involves a further existential quantification over a monitor that enforces it correctly; put differently, in order to show that a formula is *not* enforceable, amounts to another universal quantification, this time over all possible monitors. Moreover, establishing the enforceability of a logic entails yet another universal quantification, on all the formulas in the logic. In many cases (including ours), the sets of systems, monitors and formulas are infinite.

We address these problems through an *automated synthesis procedure* that produces an enforcement monitor from a safety μHML formula, expressed in a syntactic fragment of sHML. This fragment, called sHML$_{\mathbf{nf}}$, has already been used to establish enforceability results in a uni-directional setting [ACFI18] and is the source logic employed by the tool detectEr[1][AAA$^+$21, AAA$^+$22, AEF$^+$22] used to verify the correctness of concurrent systems written in Erlang [AAFI21] and Elixir [BAF21]; it also coincides with sHML in the regular setting [AAF$^+$20]. We show that the synthesised monitors are correct, according to Definition 4.9. For a unidirectional setting, it has been shown that monitors that only administer *omissions* are expressive enough to enforce *safety properties* [LBW05, FFM12, vHRF17, ACFI18]. Analogously, for our bidirectional case, we restrict ourselves to action disabling monitors and show that they can enforce *any* property expressed in terms of this sHML fragment.

Our synthesis procedure is compositional, meaning that the monitor synthesis of a composite formula is defined in terms of the enforcement monitors generated from its constituent sub-formulas. Compositionality simplifies substantially our correctness analysis of the generated monitors (*e.g.*, we can use standard inductive proof techniques). The choice

[1]`https://duncanatt.github.io/detecter/`

of the logical fragment, *i.e.,* sHML$_{\mathbf{nf}}$, facilitates this compositional definition. An automated procedure to translate an sHML formula with symbolic actions where the scope of the data binders is limited to the immediate symbolic action condition, into a corresponding sHML$_{\mathbf{nf}}$ one (with the same semantic meaning) is given in [ACFI18, AAF+20].

Definition 5.1 (sHML normal form)**.** The set of normalised sHML formulas is generated by the following grammar (where[2] $|I| \geq 1$):

$$\varphi, \psi \in \text{sHML}_{\mathbf{nf}} ::= \mathsf{tt} \quad | \quad \mathsf{ff} \quad | \quad \bigwedge_{i\in I} [p_i, c_i]\varphi_i \quad | \quad X \quad | \quad \mathsf{max}\, X.\varphi\ .$$

In addition, sHML$_{\mathbf{nf}}$ formulas are required to satisfy the following conditions:

(1) Every branch in $\bigwedge_{i\in I} [p_i, c_i]\varphi_i$, must be *disjoint*, *i.e.,* for every $i, j \in I$, $i \neq j$ implies $[\![(p_i, c_i)]\!] \cap [\![(p_j, c_j)]\!] = \emptyset$.
(2) For every $\mathsf{max}\, X.\varphi$ we have $X \in \mathbf{fv}(\varphi)$. ■

In a (closed) sHML$_{\mathbf{nf}}$ formula, the basic terms tt and ff can never appear unguarded unless they are at the top level (*e.g.,* we can never have $\varphi\wedge\mathsf{ff}$ or $\mathsf{max}\, X_0. \ldots \mathsf{max}\, X_n.\mathsf{ff}$). Modal operators are combined with conjunctions into one construct $\bigwedge_{i\in I} [p_i, c_i]\varphi_i$ that is written as $[p_0, c_0]\varphi_0\wedge \ldots \wedge[p_n, c_n]\varphi_n$ when $I = \{0, \ldots, n\}$ and simply as $[p_0, c_0]\varphi$ when $|\, I\,| = 1$. The conjunct modal guards must also be *disjoint* so that *at most one* necessity guard can satisfy any particular visible action. Along with these restrictions, we still assume that sHML$_{\mathbf{nf}}$ fixpoint variables are guarded, and that for every $((x)?(y), c)$, $y \notin \mathbf{fv}(c)$.

Example 5.2. The formula φ_3 defines a recursive property stating that an input on port a (carrying any value) cannot be followed by an output with value of 4 (on any port), and that this continues to hold if the subsequent output is made on port a with a value that is not equal to 3 (in which cases, the formula recurses)

$$\varphi_3 \stackrel{\text{def}}{=} \mathsf{max}\, X.[((x_1)?(y_1), x_1{=}\mathsf{a})] \begin{pmatrix} [((x_2)!(y_2), x_2{=}\mathsf{a} \wedge y_2{\neq}3)]X \\ \wedge\ [((x_3)!(y_3), y_3{=}4)]\mathsf{ff} \end{pmatrix}$$

φ_3 is not an sHML$_{\mathbf{nf}}$ formula since its conjunction is not disjoint (*e.g.,* the action $\mathsf{a}!4$ satisfies both branches). Still, we can reformulate φ_3 as $\varphi_3' \in \text{sHML}_{\mathbf{nf}}$:

$$\varphi_3' \stackrel{\text{def}}{=} \mathsf{max}\, X.[((x_1)?(y_1), x_1{=}\mathsf{a})] \begin{pmatrix} [((x_4)!(y_4), x_4{=}\mathsf{a} \wedge y_4{\neq}4 \wedge y_4{\neq}3)]X \\ \wedge\ [((x_4)!(y_4), x_4{=}\mathsf{a} \wedge y_4{=}4)]\mathsf{ff} \end{pmatrix}$$

where x_4 and y_4 are fresh variables. ■

Our monitor synthesis function in Definition 5.3 converts an sHML$_{\mathbf{nf}}$ formula φ into a transducer m. This conversion also requires information regarding the input ports employed by the SuS, as this is used to add the necessary insertion branches to silently unblock the SuS at runtime; this prevents the monitor from unnecessarily blocking the resulting composite system. . The synthesis function must therefore be supplied with this information in the form of a *finite* set of input ports $\Pi \subset \textsc{Port}$, which then relays this information to the resulting monitor. It also assumes a default value v_{def} for the payload data domain.

[2]Recall that from Figure 2, I always denotes a *finite* set of indices which is crucial for a synthesis process to terminate.

Definition 5.3. The synthesis function $(\!|-|\!) : \mathrm{sHML}_{\mathbf{nf}} \times \mathcal{P}_{\mathrm{fin}}(\textsc{Port}) \to \textsc{Trn}$ is defined inductively as:

$$
\begin{aligned}
(\!| X, \Pi |\!) &\stackrel{\text{def}}{=} X \\
(\!| \mathsf{tt}, \Pi |\!) &\stackrel{\text{def}}{=} \mathsf{id} \\
(\!| \mathsf{ff}, \Pi |\!) &\stackrel{\text{def}}{=} \mathsf{id} \\
(\!| \max X.\varphi, \Pi |\!) &\stackrel{\text{def}}{=} \mathsf{rec}\, X.(\!| \varphi, \Pi |\!) \\
(\!| \varphi {=} \textstyle\bigwedge_{i \in I} [(p_i, c_i)]\varphi_i, \Pi |\!) &\stackrel{\text{def}}{=} \mathsf{rec}\, Y. \left(\sum_{i \in I} \begin{cases} \mathsf{dis}(p_i, c_i, Y, \Pi) & \text{if } \varphi_i {=} \mathsf{ff} \\ (p_i, c_i).(\!| \varphi_i, \Pi |\!) & \text{otherwise} \end{cases} \right) + \mathsf{def}(\varphi)
\end{aligned}
$$

where $\mathsf{dis}(p, c, m, \Pi) \stackrel{\text{def}}{=} \begin{cases} (p, c, \bullet).m & \text{if } p = (x)!(y) \\ \sum_{\mathsf{b} \in \Pi} (\bullet, c\{\mathsf{b}/x\}, \mathsf{b}?v_{\mathrm{def}}).m & \text{if } p = (x)?(y) \end{cases}$

and

$$
\mathsf{def}(\textstyle\bigwedge_{i \in I} [((x_i)?(y_i), c_i)]\varphi_i \wedge \psi) \stackrel{\text{def}}{=} \begin{cases} ((\_)?(\_)).\mathsf{id} & \textit{when } I {=} \emptyset \\ ((x)?(y), \bigwedge_{i \in I} (\neg c_i\{x/x_i, y/y_i\})).\mathsf{id} & \textit{otherwise} \end{cases}
$$

where ψ has no conjuncts starting with an input modality, variables x and y are fresh, and v_{def} is a default value. ■

The definition above assumes a bijective mapping between formula variables and monitor recursion variables. Normalised conjunctions, $\bigwedge_{i \in I} [p_i, c_i]\varphi_i$, are synthesised as a *recursive summation* of monitors, *i.e.*, $\mathsf{rec}\, Y. \sum_{i \in I} m_i$, where Y is fresh, and every branch m_i can be one of the following:

(*i*) when m_i is derived from a branch of the form $[p_i, c_i]\varphi_i$ where $\varphi_i \neq \mathsf{ff}$, the synthesis produces a monitor with the *identity transformation* prefix, (p_i, c_i), followed by the monitor synthesised from the continuation φ_i, *i.e.*, $(\!| \varphi_i, \Pi |\!)$;

(*ii*) when m_i is derived from a violating branch of the form $[p_i, c_i]\mathsf{ff}$, the synthesis produces an *action disabling transformation* via $\mathsf{dis}(p_i, c_i, Y, \Pi)$.

Specifically, in clause (*ii*), the dis function produces either a *suppression transformation*, $(p_i, c_i, \bullet)$, when p_i is an *output* pattern, $(x_i)!(y_i)$, or a *summation of insertions*, $\sum_{\mathsf{b} \in \Pi} (\bullet, c_i\{\mathsf{b}/x_i\}, \mathsf{b}?v_{\mathrm{def}}).m_i$, when p_i is an *input* pattern, $(x_i)?(y_i)$. The former signifies that the monitor must react to and suppress every matching (invalid) system output thus stopping it from reaching the environment. By not synthesising monitor branches that react to the erroneous input, the latter allows the monitor to hide the input synchronisations from the environment. At the same time, the synthesised insertion branches insert a default domain value v_{def} on every port $\mathsf{b} \in \Pi$ whenever the branch condition $c_i\{\mathsf{b}/x_i\}$ evaluates to true at runtime. This stops the monitor from blocking the runtime progression of the resulting composite system unnecessarily.

This blocking mechanism can, however, block *unspecified* inputs, *i.e.,* those that do not satisfy any modal necessity in the normalised conjunction. This is undesirable since the unspecified actions do not contribute towards a safety violation and, instead, lead to its trivial satisfaction. To prevent this, the *default monitor* $\mathsf{def}(\varphi)$ is also added to the resulting summation. Concretely, the def function produces a *catch-all* identity monitor that forwards an input to the SuS whenever it satisfies the negation of *all* the conditions associated with modal necessities for input patterns in the normalised conjunction. This condition is constructed for a normalised conjunction of the form $\bigwedge_{i \in I} [((x_i)?(y_i), c_i)]\varphi_i \wedge \psi$

(assuming that ψ does not include further input modalities). Otherwise, if none of the conjunct modalities define an input pattern, every input is allowed, *i.e.*, the default monitor becomes $((\_)?(\_)).\mathsf{id}$, which transitions to id after forwarding the input to the SuS.

Example 5.4. Recall (the full version of) formula φ_1 from Example 2.4.

$$\varphi_1 \stackrel{\text{def}}{=} \max X.[((x)?(y_1), x{\neq}\mathsf{b})]([((x_1)?(\_), x_1{=}x)]\mathsf{ff} \wedge [((x_2)!(y_2), x_2{=}x)]\varphi_1')$$
$$\varphi_1' \stackrel{\text{def}}{=} ([((x_3)!(\_), x_3{=}x)]\mathsf{ff} \wedge [((x_4)!(y_3), x_4{=}\mathsf{b} \wedge y_3{=}(\mathsf{log}, y_1, y_2))]X)$$

For any arbitrary set of ports Π, the synthesis of Definition 5.3 produces the following monitor.

$$m_{\varphi_1} \stackrel{\text{def}}{=} \mathsf{rec}\, X.\mathsf{rec}\, Z.(((x)?(y_1), x{\neq}\mathsf{b}).\mathsf{rec}\, Y_1.m'_{\varphi_1}) + ((x_{\text{def}})?(\_), x_{\text{def}} = \mathsf{b}).\mathsf{id}$$
$$m'_{\varphi_1} \stackrel{\text{def}}{=} \sum_{\mathsf{a}\in\Pi}(\bullet, \mathsf{a}{=}x, \mathsf{a}?v_{\text{def}}).Y_1 + ((x_2)!(y_2), x_2{=}x).\mathsf{rec}\, Y_2.m''_{\varphi_1} + ((x_{\text{def}})?(\_), x_{\text{def}}{\neq}x).\mathsf{id}$$
$$m''_{\varphi_1} \stackrel{\text{def}}{=} ((x_3)!(\_), x_3{=}x, \bullet).Y_2 + ((x_4)!(y_3), x_4{=}\mathsf{b} \wedge y_3{=}(\mathsf{log}, y_1, y_2)).X + ((\_)?(\_)).\mathsf{id}$$

Monitor m_{φ_1} can be optimised by removing redundant recursive constructs such as $\mathsf{rec}\, Z.\_$ that are introduced mechanically by our synthesis. ■

Monitor m_{φ_1} from Example 5.4 (with $(\!|\varphi_1, \Pi|\!) = m_{\varphi_1}$) is very similar to $m_{\mathbf{det}}$ of Example 3.2, differing only in how it defines its insertion branches for unblocking the SuS. For instance, if we consider $\Pi = \{\mathsf{b}, \mathsf{c}\}$, $(\!|\varphi_1, \Pi|\!)$ would synthesise two insertion branches, namely $(\bullet, \mathsf{b} = x, \mathsf{b}?v_{\text{def}})$ and $(\bullet, \mathsf{c} = x, \mathsf{c}?v_{\text{def}})$, but if Π also includes d, it would add another branch. By contrast, the manually defined $m_{\mathbf{det}}$ attains the same result more succinctly via the single insertion branch $(\bullet, x?v_{\text{def}})$. Importantly, our synthesis provides the witness monitors needed to show enforceability.

Theorem 5.5 (Enforceability). $\mathrm{sHML}_{\boldsymbol{nf}}$ *is bidirectionally enforceable using the monitors and instrumentation of Figure 4.*

Proof. By Definition 4.1 the result follows from showing that for every $\varphi \in \mathrm{sHML}_{\mathbf{nf}}$ and $\Pi \subseteq \textsc{Port}$, $(\!|\varphi, \Pi|\!)$ enforces φ (for every Π). By Definition 4.9, this follows from Propositions 5.8 and 5.12, stated and proved in Section 5.1. □

Theorem 5.5 entails that the synthesised monitors generated by the function described in Definition 5.3 do *enforce* their respective $\mathrm{sHML}_{\mathbf{nf}}$ formula and are correct by construction. By this we mean that, if the formula φ being enforced can be expressed in the syntactic fragment $\mathrm{sHML}_{\mathbf{nf}}$, and it is satisfiable (*i.e.*, $[\![\varphi]\!] \neq \emptyset$), then the resulting composite system, $m[s]$, consisting of the synthesises monitor, m, composed with the SuS, s, is guaranteed to satisfy φ and the changes to its original behaviour are only those that led to a violation. It is worth pointing out that should φ be unsatisfiable, there is very little that can be done by way of enforcement; the satisfiability caveats in Definitions 4.2, 4.4 and 4.6 are intentionally inserted so as not to require anything of the synthesised monitor in such cases. We argue that this way of dealing with unsatisfiable formulas is not a deficiency of the enforcement setup, but rather a flaw in the correctness specifications being imposed.

Note that the enforcement of formulas that use μHML constructs outside of the sHML is problematic. For instance, consider the disjunction formula $\varphi_1 \vee \varphi_2$ (recall that disjuctions are not part of the sHML syntax). In a branching-time setting, the subformulas φ_1 and φ_2 can, in principle, describe computation from different parts of the computation tree. This means that, although the current execution observed by a monitor might provide enough

$$
\begin{aligned}
(s,\mathsf{tt}) \in \mathcal{R} &\quad \textit{implies}\;\; \textit{true}\\
(s,\mathsf{ff}) \in \mathcal{R} &\quad \textit{implies}\;\; \textit{false}\\
(s,\textstyle\bigwedge_{i\in I}\varphi_i) \in \mathcal{R} &\quad \textit{implies}\; (s,\varphi_i)\in\mathcal{R} \textit{ for all } i\in I\\
(s,[(p,c)]\varphi) \in \mathcal{R} &\quad \textit{implies}\; (\forall\alpha, r\cdot s \xRightarrow{\alpha} r \textit{ and } (p,c)(\alpha)=\sigma)\;\; \textit{implies}\;\; (r,\varphi\sigma)\in\mathcal{R}\\
(s,\max X.\varphi) \in \mathcal{R} &\quad \textit{implies}\; (s,\varphi\{\max X.\varphi/X\})\in\mathcal{R}
\end{aligned}
$$

where $(p,c)(\alpha)=\sigma$ is short for $\mathsf{match}(p,\alpha)=\sigma$ and $c\sigma \Downarrow \mathsf{true}$.

FIGURE 5. A satisfaction relation for sHML formulas

information to determine that one subformula is about to be violated (say φ_1), there could never be an execution that allows the monitor to determine when to intervene whenever *both* subformulas become violated. More precisely, by intervening to prevent φ_1 from being violated might break transparency, Definition 4.6, in cases where φ_2 is still satisfied (and thus $\varphi_1 \vee \varphi_2$ still holds). Conversely, not intervening might affect soundness, Definition 4.2, in cases where φ_2 is also violated (and thus $\varphi_1 \vee \varphi_2$ is certainly violated). It has been well established that a number of μHML properties are not monitorable for a variety of settings [FAI17, AAFI18b, AAFI18a, AAF+19, AAF+21a, AAF+21b] and it is therefore reasonable to expect similar limits in the case of enforceability.

5.1. **Enforceability Proofs.** In what follows, we state and prove monitor soundness and transparency, Definitions 4.2 and 4.6 for the synthesis function presented in Definition 5.3. Upon first reading, the remainder of the section can be safely skipped without affecting the comprehension of the remaining material.

To facilitate the forthcoming proofs we occasionally use the satisfaction semantics for sHML from [AI99, HL95] which is defined in terms of the *satisfaction relation*, $\vDash$. When restricted to sHML, $\vDash$ is the *largest relation* $\mathcal{R}$ satisfying the implications defined in Figure 5. It is well known that this semantics agrees with the sHML semantics of Figure 2. As a result, we use $s \vDash \varphi$ in lieu of $s \in [\![\varphi]\!]$. At certain points in our proofs we also refer to the τ-closure property of sHML, Proposition 5.6, that was proven in [AI99].

Proposition 5.6. *if* $s \xrightarrow{\tau} s'$ *and* $s \vDash \varphi$ *then* $s' \vDash \varphi$. □

We start by stating and proving synthesis soundness, which relies on the following technical lemma relating recursive monitor unfolding and its behaviour.

Lemma 5.7. $\mathit{rec}\,X.m[s] \sim \big(m\{^{\mathit{rec}\,X.m}/_{X}\}\big)[s]$

Proof. Follows from the instrumentation relation of Figure 4 and, more importantly, the monitor rule eRec, also in Figure 4. □

Proposition 5.8 (Soundness)**.** *For every finite port set* Π*, system state* $s \in \mathrm{Sys}$ *whose port names are included in* Π*, and* $\varphi \in \mathrm{sHML}_{\boldsymbol{nf}}$*, if* $[\![\varphi]\!] \neq \emptyset$ *then* $(\!|\varphi,\Pi|\!)[s] \in [\![\varphi]\!]$*.*

Proof. To prove that for every system s, formula φ and finite set of ports Π

$$\textit{if } [\![\varphi]\!] \neq \emptyset \textit{ then } (\!|\varphi,\Pi|\!)[s] \vDash \varphi$$

we setup the relation $\mathcal{R}$ (below) and show that it is a *satisfaction relation* ($\vDash$) by demonstrating that that it abides by the rules in Figure 5.

$$\mathcal{R} \stackrel{\text{def}}{=} \left\{ (r,\varphi) \;\middle|\; \begin{array}{l} (i)\ [\![\varphi]\!] \neq \emptyset \textit{ and } r = (\!|\varphi,\Pi|\!)[s] \qquad \textit{or} \\ (ii)\ [\![\varphi]\!] \neq \emptyset \textit{ and } r = (\!|\mathsf{max}\,X_1.\ldots\mathsf{max}\,X_n.\psi,\Pi|\!)[s] \\ \quad \textit{and } \varphi = \psi\{\mathsf{max}\,X_1.\ldots\mathsf{max}\,X_n.\psi/X_1\}\ldots\{\mathsf{max}\,X_n.\psi/X_n\} \end{array} \right\}$$

The second case defining the tuples $(r,\varphi) \in \mathcal{R}$ (labeled as (ii) for clarity) maps monitored system $m[s]$ where m is obtain by synthesising a formula consisting of a prefix of maximal fixpoint binders of length n, *i.e.*, $m = (\!|\varphi,\Pi|\!)$ where $\varphi = \mathsf{max}\,X_1.\ldots\mathsf{max}\,X_n.\psi$, with the *resp. unfolded* formula $\psi\{\mathsf{max}\,X_1.\ldots\mathsf{max}\,X_n.\psi/X_1\}\ldots\{\mathsf{max}\,X_n.\psi/X_n\}$.

We prove the claim that $\mathcal{R} \subseteq \vDash$ by case analysis on the structure of φ. We here consider the two main cases:

Case $\varphi = \mathsf{max}\,X.\psi$**:** We consider two subcases for why $(r,\varphi) \in \mathcal{R}$, following either condition (i) or (ii):

Case (i)**:** We know that $[\![\mathsf{max}\,X.\psi]\!] \neq \emptyset$ and $r = (\!|\mathsf{max}\,X.\psi,\Pi|\!)[s]$ for some s. By the rules defining ($\vDash$) in Figure 5 we need to show that

$$((\!|\mathsf{max}\,X.\psi,\Pi|\!)[s], \psi\{\mathsf{max}\,X.\psi/X\}) \in \mathcal{R}$$

as well. This follows immediately from rule (ii) defining $\mathcal{R}$.

Case (ii)**:** We know that $[\![\mathsf{max}\,X.\psi]\!] \neq \emptyset$, that

$$\mathsf{max}\,X.\psi = \mathsf{max}\,X.(\phi\{\mathsf{max}\,Y_1.\ldots\mathsf{max}\,Y_k.\mathsf{max}\,X.\phi/Y_1\}\ldots\{\mathsf{max}\,Y_k.\mathsf{max}\,X.\phi/Y_k\})$$

for some ϕ and k, and that $r = (\!|\mathsf{max}\,Y_1.\ldots\mathsf{max}\,Y_k.\mathsf{max}\,X.\phi,\Pi|\!)[s]$ for some s. Again, by the rules defining ($\vDash$) in Figure 5 we need to show that

$$((\!|\mathsf{max}\,Y_1.\ldots\mathsf{max}\,Y_k.\mathsf{max}\,X.\phi,\Pi|\!)[s], \phi') \in \mathcal{R}$$

for $\phi' = \phi\{\mathsf{max}\,Y_1.\ldots\mathsf{max}\,Y_k.\mathsf{max}\,X.\phi/Y_1\}\ldots\{\mathsf{max}\,Y_k.\mathsf{max}\,X.\phi/Y_k\}\{\mathsf{max}\,X.\phi/X\}$. This follows again from rule (ii) defining $\mathcal{R}$ with an maximal fixpoint binder length set at $n = k+1$.

Case $\varphi = \bigwedge_{i\in I}[(p_i,c_i)]\varphi_i$ **and** $\#_{h\in I}(p_h,c_h)$**:** Again we have two subcases to consider for why we have the inclusion $(r,\varphi) \in \mathcal{R}$:

Case (i)**:** We know that

$$[\![\bigwedge_{i\in I}[(p_i,c_i)]\varphi_i]\!] \neq \emptyset \tag{5.1}$$

and that $r = (\!|\bigwedge_{i\in I}[(p_i,c_i)]\varphi_i,\Pi|\!)[s]$ for some s. Recall that

$$(\!|\bigwedge_{i\in I}[(p_i,c_i)]\varphi_i,\Pi|\!) = \mathsf{rec}\,Y.\Big(\sum_{i\in I}\begin{cases}\mathsf{dis}(p_i,c_i,Y,\Pi) & \text{if } \varphi_i{=}\mathsf{ff}\\ (p_i,c_i).(\!|\varphi_i,\Pi|\!) & \text{otherwise}\end{cases}\Big) + \mathsf{def}(\bigwedge_{i\in I}[(p_i,c_i)]\varphi_i) \tag{5.2}$$

By the rules defining ($\vDash$) in Figure 5 (for the case involving $\bigwedge_{i\in I}\varphi_i$ and $[(p,c)]\varphi$ combined) we need to show that

$$\forall i \in I, \alpha, q \text{ if } (\!|\bigwedge_{i\in I}[(p_i,c_i)]\varphi_i,\Pi|\!)[s] \stackrel{\alpha}{\Longrightarrow} q \text{ and } (p_i,c_i)(\alpha) = \sigma) \text{ then } (q,\varphi_i\sigma) \in \mathcal{R} \tag{5.3}$$

Pick any $[(p_i,c_i)]\varphi_i$ and proceed by case analysis:

Case $[(p_i,c_i)]\varphi_i = [((x)!(y),c_i)]\mathsf{ff}$**:** For any output action $\mathsf{a}!v$ that the system s can produce, *i.e.*, $s \stackrel{\mathsf{a}!v}{\Longrightarrow} s'$, that matches the pattern of the necessity formula considered, *i.e.*, $((x)!(y), c_i)(\mathsf{a}!v) = \sigma$, the monitor synthesised in Equation (5.2) transitions as

$$(\!|\bigwedge_{i\in I}[(p_i,c_i)]\varphi_i,\Pi|\!) \xrightarrow{(\mathsf{a}!v)\blacktriangleright\bullet} (\!|\bigwedge_{i\in I}[(p_i,c_i)]\varphi_i,\Pi|\!)$$

Thus, by the instrumentation in Figure 4 (particularly rule biDisO), we conclude that it could *never* be the case that

$$(\!|\bigwedge_{i\in I}[(p_i,c_i)]\varphi_i,\Pi|\!)[s] \xRightarrow{\mathsf{a}!v} q \text{ for any } q$$

meaning that condition (5.3) is satisfied.

Case $[(p_i,c_i)]\varphi_i = [((x)?(y),c_i)]\mathsf{ff}$**:** The reasoning is analogous to the previous case. For any input action $\mathsf{a}?v$ that $s \xRightarrow{\mathsf{a}?v} s'$, that matches the pattern of the necessity formula, $((x)?(y), c_i)(\mathsf{a}?v) = \sigma$, the monitor synthesised in Equation (5.2) transitions as

$$(\!|\bigwedge_{i\in I}[(p_i,c_i)]\varphi_i,\Pi|\!) \xrightarrow{\bullet\blacktriangleright(\mathsf{a}?v)} (\!|\bigwedge_{i\in I}[(p_i,c_i)]\varphi_i,\Pi|\!)$$

Thus, by the instrumentation in Figure 4 (particularly rule biDisI), we conclude that it could *never* be the case that

$$(\!|\bigwedge_{i\in I}[(p_i,c_i)]\varphi_i,\Pi|\!)[s] \xRightarrow{\mathsf{a}?v} q \text{ for any } q$$

meaning that condition (5.3) is satisfied.

Case $\varphi_i \neq \mathsf{ff}$**:** From Equation (5.1) we know that for any α such that $(p_i, c_i)(\alpha) = \sigma$ it holds that

$$[\![\varphi_i\sigma]\!] \neq \emptyset. \tag{5.4}$$

Now if $s \xRightarrow{\alpha} s'$, from the form of $(\!|\bigwedge_{i\in I}[(p_i,c_i)]\varphi_i,\Pi|\!)$ in Equation (5.2) and $\#_{h\in I}(p_h, c_h)$ we conclude that

$$(\!|\bigwedge_{i\in I}[(p_i,c_i)]\varphi_i,\Pi|\!) \xrightarrow{\alpha\blacktriangleright\alpha} (\!|\varphi_i,\Pi|\!)\sigma = (\!|\varphi_i\sigma,\Pi|\!)$$

Thus, by the instrumentation in Figure 4 (particularly rules biTrnI and biTrnO) we conclude

$$(\!|\bigwedge_{i\in I}[(p_i,c_i)]\varphi_i,\Pi|\!)[s] \xRightarrow{\alpha} (\!|\varphi_i\sigma,\Pi|\!)[s']$$

and from Equation (5.4) and the definition of $\mathcal{R}$ (i) we conclude that $(eBI(\!|\varphi_i\sigma,\Pi|\!)s', \varphi_i\sigma) \in \mathcal{R}$, thus satisfying Equation (5.3) as required.

Case (ii)**:** We know that $[\![\bigwedge_{i\in I}[(p_i,c_i)]\varphi_i]\!] \neq \emptyset$, that for all $i \in I$

$$\varphi_i = \psi_i\{\max Y_1.\ldots\max Y_k.\psi_i/Y_1\}\ldots\{\max Y_k.\psi_i/Y_k\} \text{ for some } \psi_i$$

and that $r = (\!|\max Y_1.\ldots\max Y_k.\bigwedge_{i\in I}[(p_i,c_i)]\varphi_i,\Pi|\!)[s]$ for some s. Similar to the previous case, we need to show that r satisfies a requirement akin to Equation (5.3). This follows using a similar reasoning employed in the previous case, Lemma 5.7 and the transitivity of (strong) bisimulation. ∎

We next state and prove the fact that the synthesis function of Definition 5.3 is eventually transparent, according to Definition 4.6. This proof for eventual transparency refers to the auxiliary Lemma 5.10 and another transparency result Proposition 5.11 (Transparency) for Definition 4.4, defined and proved below. The proof of Lemma 5.10, in turn, relies on the following technical lemma which states that any sequence τ transitions from a composite system enforced by a monitor synthesised from a conjuncted modal guard formula according to Definition 5.3 can be decomposed such that the monitored system is allowed to produce external actions by the monitor remains in the same state. which states that any sequence of τ transitions from a composite system enforced by a monitor synthesised from a conjunctive formula, according to Definition 5.3, stems from a corresponding sequence of external actions of the monitored system while the monitor remains in the same state.

Lemma 5.9. *For every formula of the form $\bigwedge_{i\in I}[(p_i,c_i)]\varphi_i$ and system states s and r, if $(\!|\bigwedge_{i\in I}[(p_i,c_i)]\varphi_i,\Pi|\!)[s]\xrightarrow{\tau}{}^{*}r$ then there exists some state s' and trace u such that $s\overset{u}{\Longrightarrow}s'$ and $r=(\!|\bigwedge_{i\in I}[(p_i,c_i)]\varphi_i,\Pi|\!)[s']$.*

Proof. We proceed by mathematical induction on the number of τ transitions.

Case 0 *transitions.* This case holds trivially given that $s\overset{\varepsilon}{\Longrightarrow}s$ and so that $r=(\!|\bigwedge_{i\in I}[(p_i,c_i)]\varphi_i,\Pi|\!)[s]$.

Case $k+1$ *transitions.* Assume that $(\!|\bigwedge_{i\in I}[(p_i,c_i)]\varphi_i,\Pi|\!)[s]\xrightarrow{\tau}{}^{k+1}r$ and so we can infer that

$$(\!|\textstyle\bigwedge_{i\in I}[(p_i,c_i)]\varphi_i,\Pi|\!)[s]\xrightarrow{\tau}r'\quad\text{(for some } r'\text{)}\tag{5.5}$$

$$r'\xrightarrow{\tau}{}^{k}r.\tag{5.6}$$

By the definition of $(\!|-|\!)$ we know that $(\!|\bigwedge_{i\in I}[(p_i,c_i)]\varphi_i,\Pi|\!)$ synthesises the monitor

$$\mathsf{rec}\,Y.\sum_{i\in I}\begin{cases}\mathsf{dis}(p_i,c_i,Y,\Pi) & \textit{if } \varphi{=}\mathsf{ff}\\ (p_i,c_i).(\!|\varphi_i,\Pi|\!) & \textit{otherwise}\end{cases}$$

which can be unfolded into

$$(\!|\bigwedge_{i\in I}[(p_i,c_i)]\varphi_i,\Pi|\!)=\sum_{i\in I}\begin{cases}\mathsf{dis}(p_i,c_i,m,\Pi) & \textit{if } \varphi_i{=}\mathsf{ff}\\ (p_i,c_i).(\!|\varphi_i,\Pi|\!) & \textit{otherwise}\end{cases}\tag{5.7}$$

and so from (5.7) we know that the τ-reduction in (5.5) can be the result of rules iAsy, iDisO or iDisI. We therefore inspect each case.

- iAsy: By rule iAsy, from (5.5) we can deduce that

$$\exists s''\cdot s\xrightarrow{\tau}s''\tag{5.8}$$

$$r'=(\!|\textstyle\bigwedge_{i\in I}[(p_i,c_i)]\varphi_i|\!)[s'']\tag{5.9}$$

  and so by (5.6), (5.9) and the *inductive hypothesis* we know that

$$\exists s',u\cdot s''\overset{u}{\Longrightarrow}s'\textit{ and } r=(\!|\textstyle\bigwedge_{i\in I}[(p_i,c_i)]\varphi_i|\!)[s'].\tag{5.10}$$

  Finally, by (5.8) and (5.10) we can thus conclude that $\exists s',u\cdot s\overset{u}{\Longrightarrow}s'$ and also that $r=(\!|\bigwedge_{i\in I}[(p_i,c_i)]\varphi_i|\!)[s']$.

- iDisI: By rule iDisI and from (5.5) we infer that

$$\exists s'' \cdot s \xrightarrow{(\mathsf{a}?v)} s'' \tag{5.11}$$

$$(\!|\, \bigwedge_{i\in I} [(p_i, c_i)]\varphi_i, \Pi \,|\!) \xrightarrow{\bullet\blacktriangleright(\mathsf{a}?v)} m' \tag{5.12}$$

$$r' = m'[s''] \tag{5.13}$$

and from (5.7) and by the definition of dis we can infer that the reduction in (5.12) occurs when the synthesised monitor inserts action $\mathsf{a}?v$ and then reduces back to $(\!|\, \bigwedge_{i\in I} [(p_i, c_i)]\varphi_i, \Pi \,|\!)$ allowing us to infer that

$$m' = (\!|\, \bigwedge_{i\in I} [(p_i, c_i)]\varphi_i, \Pi \,|\!). \tag{5.14}$$

Hence, by (5.6), (5.13) and (5.14) we can apply the *inductive hypothesis* and deduce that

$$\exists s', u \cdot s'' \stackrel{u}{\Longrightarrow} s' \;\textit{and}\; r = (\!|\, \bigwedge_{i\in I} [(p_i, c_i)]\varphi_i, \Pi \,|\!)[s'] \tag{5.15}$$

so that by (5.11) and (5.15) we finally conclude that $\exists s', u \cdot s \xRightarrow{(\mathsf{a}?v)u} s'$ and that $r = (\!|\, \bigwedge_{i\in I} [(p_i, c_i)]\varphi_i, \Pi \,|\!)[s']$ as required, and so we are done.
- iDisO: We omit the proof for this case as it is very similar to that of case iDisI. □

The following lemma builds on Lemma 5.9, and states that the monitor obtained from a sequence of transitions t and a synthesised monitor $(\!|\, \psi, \Pi \,|\!)$ can be calculated using the function $\mathit{after}(\varphi, t)$ and the synthesis function given in Definition 5.3.

Lemma 5.10. *For every set of names Π, formula $\varphi \in \mathrm{sHML}_{\mathbf{nf}}$, state s and trace t, if $(\!|\, \varphi, \Pi \,|\!)[s] \stackrel{t}{\Longrightarrow} m'[s']$ then $\exists \psi \in \mathrm{sHML}_{\mathbf{nf}} \cdot \psi = \mathit{after}(\varphi, t)$ and $(\!|\, \psi, \Pi \,|\!) = m'$.*

Proof. We need to prove that for every formula $\varphi \in \mathrm{sHML}_{\mathbf{nf}}$, if we assume that $(\!|\, \varphi, \Pi \,|\!)[s] \stackrel{t}{\Longrightarrow} m'[s']$ then there must exist some formula ψ, such that $\psi = \mathit{after}(\varphi, t)$ and $(\!|\, \psi, \Pi \,|\!) = m'$. We proceed by induction on the length of t.

Case $t = \varepsilon$. This case holds vacuously since when $t = \varepsilon$ then $m' = (\!|\, \varphi, \Pi \,|\!)$ and $\varphi = \mathit{after}(\varphi, \varepsilon)$.

Case $t = \alpha u$. Assume that $(\!|\, \varphi, \Pi \,|\!)[s] \stackrel{\alpha u}{\Longrightarrow} m'[s']$ from which by the definition $\stackrel{t}{\Longrightarrow}$ we can infer that there are r and r' such that

$$(\!|\, \varphi, \Pi \,|\!)[s] \xrightarrow{\tau}{}^{*} r \tag{5.16}$$

$$r \xrightarrow{\alpha} r' \tag{5.17}$$

$$r' \stackrel{u}{\Longrightarrow} m'[s']. \tag{5.18}$$

We now proceed by case analysis on φ.

- $\varphi = X$: This case does not apply since $(\!|\, \mathsf{ff}, \Pi \,|\!)$ and $(\!|\, X, \Pi \,|\!)$ do not yield a valid monitor.
- $\varphi \in \{\mathsf{ff}, \mathsf{tt}\}$: Since $(\!|\, \mathsf{tt}, \Pi \,|\!) = \mathsf{id}$ we know that the τ-reductions in (5.16) are only possible via rule iAsy which means that $s \xrightarrow{\tau}{}^{*} s''$ and $r = (\!|\, \mathsf{tt}, \Pi \,|\!)[s'']$. The latter allows us to deduce that the reduction in (5.17) is only possible via rule iTrn and so we also know that

$s'' \xrightarrow{\alpha}{}^{*} s'''$ and $r' = (\!|\, \mathsf{tt}, \Pi \,|\!)[s''']$. Hence, by (5.18) and the *inductive hypothesis* we conclude that

$$\exists \psi \in \mathrm{sHML}_{\mathbf{nf}} \cdot \psi = \mathit{after}(\mathsf{tt}, u) \tag{5.19}$$

$$(\!|\, \psi, \Pi \,|\!) = m'. \tag{5.20}$$

Since from the definition of *after* we know that $\mathit{after}(\mathsf{tt}, \alpha u)$ equates to $\mathit{after}(\mathit{after}(\mathsf{tt}, \alpha), u)$ and $\mathit{after}(\mathsf{tt}, \alpha) = \mathsf{tt}$, from (5.19) we can conclude that $\psi = \mathit{after}(\mathsf{tt}, \alpha u)$ and so this case holdssince we also know (5.20). The case for ff is analogous.

- $\varphi = \bigwedge_{i\in I} [(p_i, c_i)]\varphi_i$ and $\#_{i\in I}(p_i, c_i)$: Since $\varphi = \bigwedge_{i\in I} [(p_i, c_i)]\varphi_i$, by the definition of $(\!|\, - \,|\!)$ we know that $\mathsf{rec}\, Y. \sum_{i\in I} \begin{cases} \mathsf{dis}(p_i, c_i, Y, \Pi) & \textit{if } \varphi_i = \mathsf{ff} \\ (p_i, c_i).(\!|\, \varphi_i, \Pi \,|\!) & \textit{otherwise} \end{cases}$ which can be unfolded into

$$(\!|\, \bigwedge_{i\in I} [(p_i, c_i)]\varphi_i, \Pi \,|\!) = \sum_{i\in I} \begin{cases} \mathsf{dis}(p_i, c_i, m, \Pi) & \textit{if } \varphi_i = \mathsf{ff} \\ (p_i, c_i).(\!|\, \varphi_i, \Pi \,|\!) & \textit{otherwise} \end{cases} \tag{5.21}$$

and so by (5.16), (5.21) and Lemma 5.9 we conclude that $\exists s'' \cdot s \xRightarrow{u} s''$ and that

$$r = (\!|\, \textstyle\bigwedge_{i\in I} [(p_i, c_i)]\varphi_i, \Pi \,|\!)[s'']. \tag{5.22}$$

Hence, by (5.21) and (5.22) we know that the reduction in (5.17) can only happen if $\exists s''' \cdot s'' \xrightarrow{\alpha} s'''$ and α matches an identity transformation $(p_j, c_j).(\!|\, \varphi_j, \Pi \,|\!)$ (for some $j \in I$) which was derived from $[(p_j, c_j)]\varphi_j$ (where $\varphi_j \neq \mathsf{ff}$). We can thus deduce that

$$r' = (\!|\, \varphi_j\sigma, \Pi \,|\!)[s'''] \tag{5.23}$$

$$\mathsf{match}(p_j, \alpha) = \sigma \ \textit{and}\ c_j\sigma \Downarrow \mathsf{true} \tag{5.24}$$

and so by (5.18), (5.23) and the *inductive hypothesis* we deduce that

$$\exists \psi \in \mathrm{sHML}_{\mathbf{nf}} \cdot \psi = \mathit{after}(\varphi_j\sigma, u) \tag{5.25}$$

$$(\!|\, \psi, \Pi \,|\!) = m'. \tag{5.26}$$

Now since we know (5.24), by the definition of *after* we infer that

$$\begin{aligned} \mathit{after}(\textstyle\bigwedge_{i\in I} [(p_i, c_i)]\varphi_i, \alpha u) &= \mathit{after}(\mathit{after}(\textstyle\bigwedge_{i\in I} [(p_i, c_i)]\varphi_i, \alpha), u) \\ &= \mathit{after}(\varphi_j\sigma, u) \end{aligned} \tag{5.27}$$

and so from (5.25) and (5.27) we conclude that

$$\psi = \mathit{after}(\textstyle\bigwedge_{i\in I} [(p_i, c_i)]\varphi_i, \alpha u). \tag{5.28}$$

Hence, this case is done by (5.26) and (5.28).

- $\varphi = \mathsf{max}\, X.\psi$ and $X \in \mathbf{fv}(\psi)$: Since $\varphi = \mathsf{max}\, X.\psi$, by the syntactic rules of $\mathrm{sHML}_{\mathbf{nf}}$ we know that $\psi \notin \{\mathsf{ff}, \mathsf{tt}\}$ since $X \notin \mathbf{fv}(\psi)$, and that $\psi \neq X$ since logical variables must be guarded, hence we know that ψ can only be of the form

$$\psi = \mathsf{max}\, Y_1. \ldots \mathsf{max}\, Y_n. \textstyle\bigwedge_{i\in I} [(p_i, c_i)]\varphi_i. \tag{5.29}$$

where $\mathsf{max}\, Y_1. \ldots \mathsf{max}\, Y_n.$ denotes an arbitrary number of fixpoint declarations, possibly none. Hence, knowing (5.29), by unfolding every fixpoint in $\mathsf{max}\, X.\psi$ we reduce the formula to

$$\varphi = \bigwedge_{i\in I}[(p_i,c_i)]\varphi_i\{^{\mathsf{max}\,X.\mathsf{max}\,Y_1\ldots\mathsf{max}\,Y_n.\bigwedge_{i\in I}[(p_i,c_i)]\varphi_i}/_{X},\ldots\}$$

and so from this point onwards the proof proceeds as per that of case $\varphi = \bigwedge_{i\in I}[(p_i,c_i)]\varphi_i$ which allows us to deduce that

$$\exists\psi' \in \mathrm{sHML}_{\mathbf{nf}} \cdot \psi' = \mathit{after}(\textstyle\bigwedge_{i\in I}[(p_i,c_i)]\varphi_i\{\ldots\},\alpha u) \tag{5.30}$$

$$(\!|\, \psi',\Pi \,|\!) = m'. \tag{5.31}$$

From (5.29), (5.30) and the definition of *after* we can therefore conclude that

$$\exists\psi' \in \mathrm{sHML}_{\mathbf{nf}} \cdot \psi' = \mathit{after}(\mathsf{max}\,X.\psi,\alpha u) \tag{5.32}$$

and so this case holds by (5.31) and (5.32).

Hence, the above cases suffice to show that the case for when $t = \alpha u$ holds. ☐

The transparency proof following Definition 4.4 is given below.

Proposition 5.11 (Transparency). *For every state $s \in \mathrm{SYS}$ and $\varphi \in \mathrm{sHML}_{\boldsymbol{nf}}$, if $s \in [\![\varphi]\!]$ then $(\!|\, \varphi,\Pi \,|\!)[s] \sim s$.*

Proof. Since $s \in [\![\varphi]\!]$ is analogous to $s \vDash \varphi$ we prove that relation $\mathcal{R} \overset{\text{def}}{=} \{\, (s, (\!|\, \varphi,\Pi \,|\!)[s]) \mid s \vDash \varphi \,\}$ is a *strong bisimulation relation* that satisfies the following transfer properties:

(a) if $s \xrightarrow{\mu} s'$ then $(\!|\, \varphi,\Pi \,|\!)[s] \xrightarrow{\mu} r'$ and $(s',r') \in \mathcal{R}$

(b) if $(\!|\, \varphi,\Pi \,|\!)[s] \xrightarrow{\mu} r'$ then $s \xrightarrow{\mu} s'$ and $(s',r') \in \mathcal{R}$

We prove (a) and (b) separately by assuming that $s \vDash \varphi$ in both cases as defined by relation $\mathcal{R}$ and conduct these proofs by case analysis on φ. We now proceed to prove (a) by case analysis on φ.

Cases $\varphi \in \{\mathsf{ff}, X\}$. Both cases do not apply since $\nexists s \cdot s \vDash \mathsf{ff}$ and similarly since X is an open-formula and so $\nexists s \cdot s \vDash X$.

Case $\varphi = \mathsf{tt}$. We now assume that

$$s \vDash \mathsf{tt} \tag{5.33}$$

$$s \xrightarrow{\mu} s' \tag{5.34}$$

and since $\mu \in \{\tau,\alpha\}$, we must consider both cases.

- $\mu = \tau$*:* Since $\mu = \tau$, we can apply rule iAsy on (5.34) and get that

$$(\!|\, \mathsf{tt},\Pi \,|\!)[s] \xrightarrow{\tau} (\!|\, \mathsf{tt},\Pi \,|\!)[s'] \tag{5.35}$$

  as required. Also, since we know that every process satisfies tt, we know that $s' \vDash \mathsf{tt}$, and so by the definition of $\mathcal{R}$ we conclude that

$$(s', (\!|\, \mathsf{tt},\Pi \,|\!)[s']) \in \mathcal{R} \tag{5.36}$$

  as required. This means that this case is done by (5.35) and (5.36).
- $\mu = \alpha$*:* Since $(\!|\, \mathsf{tt},\Pi \,|\!) = \mathsf{id}$ encodes the 'catch-all' monitor, $\mathsf{rec}\,Y.((x)!(y),\mathsf{true},x!y).Y + ((x)?(y),\mathsf{true},x?y).Y$, by rules eRec and eTrn we can apply rule iTrnI/O and deduce

that $\mathsf{id} \xrightarrow{\alpha \blacktriangleright \alpha} \mathsf{id}$, which we can further refine as

$$(\!|\, \mathsf{tt}, \Pi \,|\!)[s] \xrightarrow{\alpha} (\!|\, \mathsf{tt}, \Pi \,|\!)[s'] \tag{5.37}$$

as required. Once again since $s' \vDash \mathsf{tt}$, by the definition of $\mathcal{R}$ we can infer that

$$(s', (\!|\, \mathsf{tt}, \Pi \,|\!)[s']) \in \mathcal{R} \tag{5.38}$$

as required, and so this case is done by (5.37) and (5.38).

Case $\varphi = \bigwedge_{i \in I} [(p_i, c_i)]\varphi_i$. We assume that

$$s \vDash \bigwedge_{i \in I} [(p_i, c_i)]\varphi_i \tag{5.39}$$

$$s \xrightarrow{\mu} s' \tag{5.40}$$

and by the definition of $\vDash$ and (5.39) we have that for every index $i \in I$ and action $\beta \in \textsc{Act}$,

$$\textit{if } s \xRightarrow{\beta} s' \textit{ and } (p_i, c_i)(\beta) = \sigma \textit{ then } s' \vDash \varphi_i\sigma. \tag{5.41}$$

Since $\mu \in \{\tau, \alpha\}$, we must consider both possibilities.

- $\mu = \tau$*:* Since $\mu = \tau$, we can apply rule iAsy on (5.40) and obtain

$$(\!|\, \bigwedge_{i \in I} [(p_i, c_i)]\varphi_i, \Pi \,|\!)[s] \xrightarrow{\tau} (\!|\, \bigwedge_{i \in I} [(p_i, c_i)]\varphi_i, \Pi \,|\!)[s''] \tag{5.42}$$

  as required. Since $\mu = \tau$, and since we know that sHML is τ-closed, from (5.39), (5.40) and Proposition 5.6, we can deduce that $s' \vDash \bigwedge_{i \in I} [(p_i, c_i)]\varphi_i$, so that by the definition of $\mathcal{R}$ we conclude that

$$(s'', (\!|\, \bigwedge_{i \in I} [(p_i, c_i)]\varphi_i, \Pi \,|\!)[s'']) \in \mathcal{R} \tag{5.43}$$

  as required. This subcase is therefore done by (5.42) and (5.43).
- $\mu = \alpha$*:* Since $\mu = \alpha$, from (5.40) we know that

$$s \xrightarrow{\alpha} s' \tag{5.44}$$

  and by the definition of $(\!|\, - \,|\!)$ we can immediately deduce that

$$(\!|\, \varphi_\wedge, \Pi \,|\!) = \mathsf{rec}\, Y. \Big(\sum_{i \in I} \begin{cases} \mathsf{dis}(p_i, c_i, Y, \Pi) & \text{if } \varphi_i = \mathsf{ff} \\ (p_i, c_i).(\!|\, \varphi_i, \Pi \,|\!) & \text{otherwise} \end{cases} \Big) + \mathsf{def}(\varphi_\wedge) \tag{5.45}$$

  where $\varphi_\wedge \stackrel{\text{def}}{=} \bigwedge_{i \in I} [(p_i, c_i)]\varphi_i$. Since the branches in the conjunction are all disjoint, $\#_{i \in I}(p_i, c_i)$, we know that *at most one* of the branches can match the same (input or output) action α. Hence, we consider two cases, namely:
  - *No matching branches (i.e.,* $\forall i \in I \cdot (p_i, c_i)(\alpha) = \textit{undef}$*):* Since none of the symbolic actions in (5.45) can match action α, we can infer that if α is an *input, i.e.,* $\alpha = \mathsf{a}?v$, then it will match the default monitor $\mathsf{def}(\varphi_\wedge)$ and transition via rule iTrnI, while if it is an *output, i.e.,* $\alpha = \mathsf{a}!v$, rule iDef handles the underspecification. In both cases, the monitor reduces to id. Also, notice that rules iDisO and iDisI cannot be applied since if they do, it would mean that s can also perform an erroneous action, which is not the

case since we assume (5.39). Hence, we infer that

$$\llparenthesis \bigwedge_{i\in I} [(p_i,c_i)]\varphi_i, \Pi \rrparenthesis[s] \xrightarrow{\alpha} \llparenthesis \mathsf{tt}, \Pi \rrparenthesis[s'] \quad (\text{since } \mathsf{id} = \llparenthesis \mathsf{tt}, \Pi \rrparenthesis) \tag{5.46}$$

as required. Also, since any process satisfies tt, we know that $s' \vDash \mathsf{tt}$, and so by the definition of $\mathcal{R}$ we conclude that

$$(s', \llparenthesis \mathsf{tt}, \Pi \rrparenthesis[s']) \in \mathcal{R} \tag{5.47}$$

as required. This case is therefore done by (5.46) and (5.47).

– *One matching branch (i.e., $\exists j \in I \cdot (p_j, c_j)(\alpha) = \sigma$):* From (5.45) we can infer that the synthesised monitor can only disable the (input or output) actions that are defined by violating modal necessities. However, from (5.41) we also deduce that s is *incapable* of executing such an action as that would contradict assumption (5.39). Hence, since we now assume that $\exists j \in I \cdot (p_j, c_j)(\alpha) = \sigma$, from (5.45) we deduce that this action can only be transformed by an identity transformation and so by rule eTrn we have that

$$(p_j, c_j).\llparenthesis \varphi_j, \Pi \rrparenthesis \xrightarrow{\alpha \blacktriangleright \alpha} \llparenthesis \varphi_j\sigma, \Pi \rrparenthesis. \tag{5.48}$$

By applying rules eSel, eRec on (5.48) and by (5.44), (5.45) and iTrnI/O (depending on whether α is an input or output action) we get that

$$\llparenthesis \bigwedge_{i\in I} [(p_i,c_i)]\varphi_i, \Pi \rrparenthesis[s] \xrightarrow{\alpha} \llparenthesis \varphi_j\sigma, \Pi \rrparenthesis[s'] \tag{5.49}$$

as required. By (5.41), (5.44) and since we assume that $\exists j \in I \cdot (p_j, c_j)(\alpha) = \sigma$ we have that $s' \vDash \varphi_j\sigma$, and so by the definition of $\mathcal{R}$ we conclude that

$$(s', \llparenthesis \varphi_j\sigma, \Pi \rrparenthesis[s']) \in \mathcal{R} \tag{5.50}$$

as required. Hence, this subcase holds by (5.49) and (5.50).

Case $\varphi = \mathsf{max}\, X.\varphi$ and $X \in \boldsymbol{fv}(\varphi)$. Now, lets assume that

$$s \xrightarrow{\mu} s' \tag{5.51}$$

and that $s \vDash \mathsf{max}\, X.\varphi$ from which by the definition of $\vDash$ we have that

$$s \vDash \varphi\{\mathsf{max}\, X.\varphi/X\}. \tag{5.52}$$

Since $\varphi\{\mathsf{max}\, X.\varphi/X\} \in \mathrm{sHML}_{\mathbf{nf}}$, by the restrictions imposed by $\mathrm{sHML}_{\mathbf{nf}}$ we know that: φ cannot be X because (bound) logical variables are required to be *guarded*, and it also cannot be tt or ff since X is required to be defined in φ, *i.e.*, $X \in \mathbf{fv}(\varphi)$. Hence, we know that φ can only have the following form, that is

$$\varphi = \mathsf{max}\, Y_0. \dots \mathsf{max}\, Y_n. \bigwedge_{i\in I} [(p_i,c_i)]\varphi_i \tag{5.53}$$

and so by (5.52), (5.53) and the definition of $\vDash$ we have that

$$s \vDash (\bigwedge_{i\in I} [(p_i,c_i)]\varphi_i)\{\cdot\cdot\} \quad \textit{where} \tag{5.54}$$
$$\{\cdot\cdot\} = \{\mathsf{max}\, X.\varphi/X, (\mathsf{max}\, Y_0. \dots \mathsf{max}\, Y_n. \bigwedge_{i\in I} [(p_i,c_i)]\varphi_i)/Y_0, \dots\}.$$

Since we know (5.51) and (5.54), from this point onwards the proof proceeds as in the previous case. We thus omit the details.

These cases thus allow us to conclude that (a) holds. We now proceed to prove (b) using a similar case analysis approach.

Cases $\varphi \in \{\mathsf{ff}, X\}$. Both cases do not apply since $\nexists s \cdot s \models \mathsf{ff}$ and similarly since X is an open-formula and $\nexists s \cdot s \models X$.

Case $\varphi = \mathsf{tt}$. Assume that

$$s \models \mathsf{tt} \tag{5.55}$$

$$(\!|\,\mathsf{tt}, \Pi\,|\!)[s] \xrightarrow{\mu} r' \tag{5.56}$$

Since $\mu \in \{\tau, \mathsf{a}?v, \mathsf{a}!v\}$, we must consider each case.

- $\mu = \tau$*:* Since $\mu = \tau$, the transition in (5.56) can be performed via iDisI, iDisO or iAsy. We must therefore consider these cases.
  - iAsy*:* From rule iAsy and (5.56) we thus know that $r' = (\!|\,\mathsf{tt}, \Pi\,|\!)[s']$ and that $s \xrightarrow{\tau} s'$ as required. Also, since every process satisfies tt, we know that $s' \models \mathsf{tt}$ as well, and so we are done since by the definition of $\mathcal{R}$ we know that $(s', (\!|\,\mathsf{tt}, \Pi\,|\!)[s']) \in \mathcal{R}$.
  - iDisI*:* From rule iDisI and (5.56) we know that: $r' = m'[s']$, $s \xrightarrow{\mathsf{a}?v} s'$ and that

$$(\!|\,\mathsf{tt}, \Pi\,|\!) \xrightarrow{\bullet \blacktriangleright (\mathsf{a}?v)} m'. \tag{5.57}$$

    Since $(\!|\,\mathsf{tt}, \Pi\,|\!) = \mathsf{id}$ we can deduce that (5.57) is *false* and hence this case does not apply.
  - iDisO*:* The proof for this case is analogous as to that of case iDisI.
- $\mu = \mathsf{a}?v$*:* Since $\mu = \mathsf{a}?v$, the transition in (5.56) can be performed either via iTrnI or iEnI. We consider both cases.
  - iEnI*:* This case also does not apply since if the transition in (5.56) is caused by rule iEnI we would have that $(\!|\,\mathsf{tt}, \Pi\,|\!) \xrightarrow{\mathsf{a}?v \blacktriangleright \bullet} m$ which is *false* since $(\!|\,\mathsf{tt}, \Pi\,|\!) = \mathsf{id} = \mathsf{rec}\, Y.((x)!(y), \mathsf{true}, x!y).Y + ((x)?(y), \mathsf{true}, x?y).Y$ and rules eRec, eSel and eTrn state that for every $\mathsf{a}?v$, $\mathsf{id} \xrightarrow{\mathsf{a}?v \blacktriangleright \mathsf{a}?v} \mathsf{id}$, thus leading to a contradiction.
  - iTrnI*:* By applying rule iTrnI on (5.56) we know that $r' = m'[s']$ such that

$$(\!|\,\mathsf{tt}, \Pi\,|\!) \xrightarrow{\mathsf{a}?v \blacktriangleright \mathsf{b}?w} m'. \tag{5.58}$$

$$s \xrightarrow{\mathsf{b}?w} s' \tag{5.59}$$

    Since $(\!|\,\mathsf{tt}, \Pi\,|\!) = \mathsf{id} = \mathsf{rec}\, Y.((x)!(y), \mathsf{true}, x!y).Y + ((x)?(y), \mathsf{true}, x?y).Y$, by applying rules eRec, eSel and eTrn to (5.58) we know that $\mathsf{a}?v = \mathsf{b}?w$, $m' = \mathsf{id} = (\!|\,\mathsf{tt}, \Pi\,|\!)$, meaning that $r' = (\!|\,\mathsf{tt}, \Pi\,|\!)[s']$. Hence, since every process satisfies tt we know that $s' \models \mathsf{tt}$, so that by the definition of $\mathcal{R}$ we conclude

$$(s', (\!|\,\mathsf{tt}, \Pi\,|\!)[s']) \in \mathcal{R}. \tag{5.60}$$

    Hence, we are done by (5.59) and (5.60) since we know that $\mathsf{a}?v = \mathsf{b}?w$.
- $\mu = \mathsf{a}!v$*:* When $\mu = \mathsf{a}!v$, the transition in (5.56) can be performed via iDef, iTrnO or iEnO. We omit this proof as it is very similar to that of case $\mu = \mathsf{a}?v$.

Case $\varphi = \bigwedge_{i\in I}[(p_i, c_i)]\varphi_i$. We now assume that

$$s \vDash \bigwedge_{i\in I}[(p_i, c_i)]\varphi_i \tag{5.61}$$

$$(\!| \bigwedge_{i\in I}[(p_i, c_i)]\varphi_i, \Pi |\!)[s] \xrightarrow{\mu} r'. \tag{5.62}$$

From (5.61) and by the definition of $\vDash$ we can deduce that

$$\forall i \in I, \beta \in \textsc{Act} \cdot \ \textit{if } s \overset{\beta}{\Longrightarrow} s' \textit{ and } (p_i, c_i)(\beta) = \sigma \textit{ then } s' \vDash \varphi_i\sigma \tag{5.63}$$

and from (5.62) and the definition of $(\!| - |\!)$ we have that

$$\Big(\mathsf{rec}\, Y.\Big(\sum_{i\in I}\begin{cases}\mathsf{dis}(p_i, c_i, Y, \Pi) & \text{if } \varphi_i = \mathsf{ff}\\ (p_i, c_i).(\!| \varphi_i, \Pi |\!) & \text{otherwise}\end{cases}\Big) + \mathsf{def}(\bigwedge_{i\in I}[(p_i, c_i)]\varphi_i)\Big)[s'] \xrightarrow{\mu} r'. \tag{5.64}$$

From (5.64) we can deduce that the synthesised monitor can only disable an (input or output) action β when its occurrence would violate a conjunct of the form $[(p_i, c_i)]\mathsf{ff}$ for some $i \in I$. However, we also know that s is *unable* to perform such an action as otherwise it would contradict assumption (5.63). Hence, we can safely conclude that the synthesised monitor in (5.64) does *not* disable any (input or output) actions of s, and so by the definition of dis we conclude that

$$\forall \mathsf{a}?v, \mathsf{a}!v \in \textsc{Act}, s' \in \textsc{Sys}\cdot \begin{pmatrix} s \xrightarrow{\mathsf{a}?v} s' \ \textit{implies} \ (\!| \bigwedge_{i\in I}[(p_i, c_i)]\varphi_i, \Pi |\!) \overset{\bullet\blacktriangleright\mathsf{a}?w}{\not\longrightarrow} \text{ (for all } w) \ \textit{and} \\ s \xrightarrow{\mathsf{a}!v} s' \ \textit{implies} \ (\!| \bigwedge_{i\in I}[(p_i, c_i)]\varphi_i, \Pi |\!) \overset{\mathsf{a}?v\blacktriangleright\bullet}{\not\longrightarrow} \end{pmatrix}. \tag{5.65}$$

Since $\mu \in \{\tau, \mathsf{a}?v, \mathsf{a}!v\}$, we must consider each case.

- $\mu = \tau$*:* Since $\mu = \tau$, from (5.62) we know that

$$(\!| \bigwedge_{i\in I}[(p_i, c_i)]\varphi_i, \Pi |\!)[s] \xrightarrow{\tau} r' \tag{5.66}$$

The τ-transition in (5.66) can be the result of rules iAsy, iDisI or iDisO; we thus consider each eventuality.

  - iAsy*:* As we assume that the reduction in (5.66) is the result of rule iAsy, we know that $r' = (\!| \bigwedge_{i\in I}[(p_i, c_i)]\varphi_i, \Pi |\!)[s']$ and that

$$s \xrightarrow{\tau} s' \tag{5.67}$$

as required. Also, since sHML is τ-closed, by (5.61), (5.67) and Proposition 5.6 we deduce that $s' \vDash \bigwedge_{i\in I}[(p_i, c_i)]\varphi_i$ as well, so that by the definition of $\mathcal{R}$ we conclude that

$$(s', (\!| \bigwedge_{i\in I}[(p_i, c_i)]\varphi_i, \Pi |\!)[s']) \in \mathcal{R} \tag{5.68}$$

and so we are done by (5.67) and (5.68).

  - iDisI*:* By assuming that reduction (5.66) results from iDisI, we have that $r' = m'[s']$ and that

$$(\!| \bigwedge_{i\in I}[(p_i, c_i)]\varphi_i, \Pi |\!) \xrightarrow{\bullet\blacktriangleright\mathsf{a}?v} m' \tag{5.69}$$

$$s \xrightarrow{\mathsf{a}?v} s' \tag{5.70}$$

By (5.65) and (5.70) we can, however, deduce that for every value w, we have that $(\!|\bigwedge_{i\in I}[(p_i,c_i)]\varphi_i,\Pi|\!) \xrightarrow{\bullet\blacktriangleright\mathsf{a}?w}\!\!\!\!\!\!\!/\;\;\;$. This contradicts (5.69) and so this case does not apply.

– iDisO: As we now assume that the reduction in (5.66) results from iDisO, we have that $r' = m'[s']$ and that

$$s \xrightarrow{\mathsf{a}!v} s' \tag{5.71}$$

$$(\!|\bigwedge_{i\in I}[(p_i,c_i)]\varphi_i,\Pi|\!) \xrightarrow{\mathsf{a}!v\blacktriangleright\bullet} m'. \tag{5.72}$$

Again, this case does not apply since from (5.65) and (5.71) we can deduce that $(\!|\bigwedge_{i\in I}[(p_i,c_i)]\varphi_i,\Pi|\!) \xrightarrow{\mathsf{a}!v\blacktriangleright\bullet}\!\!\!\!\!\!\!/\;\;\;$ which contradicts (5.72).

- $\mu = \mathsf{a}?v$: When $\mu = \mathsf{a}?v$, the transition in (5.64) can be performed via rules iEnI or iTrnI, we consider both possibilities.
  – iEnI: This case does not apply since from (5.64) and by the definition of $(\!|-|\!)$ we know that the synthesised monitor does not include action enabling transformations.
  – iTrnI: By assuming that (5.64) is obtained from rule iTrnI we know that

$$\mathsf{rec}\,Y.\Big(\sum_{i\in I}\begin{cases}\mathsf{dis}(p_i,c_i,Y,\Pi) & \text{if } \varphi_i=\mathsf{ff}\\ (p_i,c_i).(\!|\varphi_i,\Pi|\!) & \text{otherwise}\end{cases}\Big) + \mathsf{def}(\bigwedge_{i\in I}[(p_i,c_i)]\varphi_i) \xrightarrow{\mathsf{a}?v\blacktriangleright\mathsf{b}?w} m' \tag{5.73}$$

$$s \xrightarrow{\mathsf{b}?w} s' \tag{5.74}$$

$$r' = m'[s']. \tag{5.75}$$

Since from (5.65) we know that the synthesised monitor in (5.73) does not disable any action performable by s, and since from the definition of $(\!|-|\!)$ we know that the synthesis is incapable of producing action replacing monitors, we can deduce that

$$\mathsf{a}?v = \mathsf{b}?w. \tag{5.76}$$

With the knowledge of (5.76), from (5.74) we can thus deduce that

$$s \xrightarrow{\mathsf{a}?v} s' \tag{5.77}$$

as required. Knowing (5.76) we can also deduce that in (5.73) the monitor transforms an action $\mathsf{a}?v$ either (i) via an identity transformation that was synthesised from one of the *disjoint* conjunction branches, *i.e.*, from a branch $(p_j,c_j).(\!|\varphi_j,\Pi|\!)$ for some $j\in I$, or else (ii) via the default monitor synthesised by $\mathsf{def}(\bigwedge_{i\in I}[(p_i,c_i)]\varphi_i)$. We consider both eventualities.

(i) In this case we apply rules eRec, eSel and eTrn on (5.73) and deduce that

$$\exists j\in I\cdot(p_j,c_j)(\mathsf{a}?v)=\sigma \tag{5.78}$$

$$m' = (\!|\varphi_j\sigma,\Pi|\!). \tag{5.79}$$

and so from (5.77), (5.78) and (5.63) we infer that $s' \vDash \varphi_j\sigma$ from which by the definition of $\mathcal{R}$ we have that $(s',(\!|\varphi_j\sigma,\Pi|\!)[s'])\in\mathcal{R}$, and so from (5.75) and (5.79) we can conclude that

$$(s',r')\in\mathcal{R} \tag{5.80}$$

as required, and so this case is done by (5.77) and (5.80).

(ii) When we apply rules eRec, eSel and eTrn we deduce that $m' = \mathsf{id}$ and so by the definition of $(\!|-|\!)$ we have that

$$m' = (\!|\mathsf{tt}, \Pi|\!). \tag{5.81}$$

Consequently, as every process satisfies tt, we know that $s' \vDash \mathsf{tt}$ and so by the definition of $\mathcal{R}$ we have that $(s', (\!|\mathsf{tt}, \Pi|\!)[s']) \in \mathcal{R}$, so that from (5.75) and (5.81) we can conclude that

$$(s', r') \in \mathcal{R} \tag{5.82}$$

as required. Hence this case is done by (5.77) and (5.82).

- *$\mu = \mathsf{a}!v$:* When $\mu = \mathsf{a}!v$, the transition in (5.64) can be performed via iDef, iTrnO or iEnO. We omit the proof for this case due to its strong resemblance to that of case $\mu = \mathsf{a}?v$.

Case $\varphi = \mathsf{max}\, X.\varphi$ and $X \in \boldsymbol{fv}(\varphi)$. Now, lets assume that

$$(\!|\mathsf{max}\, X.\varphi, \Pi|\!)[s] \xrightarrow{\mu} r' \tag{5.83}$$

and that $s \vDash \mathsf{max}\, X.\varphi$ from which by the definition of $\vDash$ we have that

$$s \vDash \varphi\{\mathsf{max}\, X.\varphi/X\}. \tag{5.84}$$

Since $\varphi\{\mathsf{max}\, X.\varphi/X\} \in \text{sHML}_{\mathbf{nf}}$, by the restrictions imposed by $\text{sHML}_{\mathbf{nf}}$ we know that: φ cannot be X because (bound) logical variables are required to be *guarded*, and it also cannot be tt or ff since X is required to be defined in φ, *i.e.*, $X \in \mathbf{fv}(\varphi)$. Hence, we know that φ can only have the following form, that is

$$\varphi = \mathsf{max}\, Y_0. \ldots \mathsf{max}\, Y_n. \bigwedge_{i \in I} [(p_i, c_i)]\varphi_i \tag{5.85}$$

and so by (5.84), (5.85) and the definition of $\vDash$ we have that

$$\begin{gathered} s \vDash \textstyle\bigwedge_{i \in I} [(p_i, c_i)]\varphi_i\{\cdot\cdot\} \qquad \textit{where} \\ \{\cdot\cdot\} = \{\mathsf{max}\, X.\varphi/X, (\mathsf{max}\, Y_0. \ldots \mathsf{max}\, Y_n. \textstyle\bigwedge_{i \in I} [(p_i, c_i)]\varphi_i)/Y_0, \ldots\}. \end{gathered} \tag{5.86}$$

Since $(\!|\bigwedge_{i \in I} [(p_i, c_i)]\varphi_i\{\cdot\cdot\}, \Pi|\!)$ synthesises the *unfolded equivalent* of $(\!|\mathsf{max}\, X.\varphi, \Pi|\!)$, from (5.83) we know that

$$(\!|\textstyle\bigwedge_{i \in I} [(p_i, c_i)]\varphi_i\{\cdot\cdot\}, \Pi|\!)[s] \xrightarrow{\mu} r'. \tag{5.87}$$

Hence, since we know (5.86) and (5.87), from this point onwards the proof proceeds as per the previous case. We thus omit showing the remainder of this proof.

From the above cases we can therefore conclude that (b) holds as well. □

We are finally in a position to state and prove our eventual transparency results following Definition 4.6.

$$mc(m,t_\tau) \stackrel{\text{def}}{=} \begin{cases} 1+mc(m',t'_\tau) & \textit{if } t_\tau = \mu t'_\tau \textit{ and } m[\mathsf{sys}(\mu t'_\tau)] \xrightarrow{\mu'} m'[\mathsf{sys}(t'_\tau)] \textit{ and } \mu \neq \mu' \\ 1+mc(m',t_\tau) & \textit{if } t_\tau \in \{\mu t'_\tau, \varepsilon\} \textit{ and } m[\mathsf{sys}(t_\tau)] \xrightarrow{\mu'} m'[\mathsf{sys}(t_\tau)] \\ mc(m',t'_\tau) & \textit{if } t_\tau = \mu t'_\tau \textit{ and } m[\mathsf{sys}(\mu t'_\tau)] \xrightarrow{\mu} m'[\mathsf{sys}(t'_\tau)] \\ |\, t_\tau \,| & \textit{if } t_\tau \in \{\mu t'_\tau, \varepsilon\} \textit{ and } \forall \mu' \cdot m[\mathsf{sys}(t_\tau)] \stackrel{\mu'}{\not\longrightarrow} \end{cases}$$

FIGURE 6. Modification Count (mc).

Proposition 5.12 (Eventual Transparency)**.** *For every input port set* Π*,* sHML$_{\boldsymbol{nf}}$ *formula* φ*, system states* $s, s' \in \textsc{Sys}$*, action disabling monitor* m' *and trace* t*, if* $(\!|\varphi,\Pi|\!)[s] \stackrel{t}{\Longrightarrow} m'[s']$ *and* $s' \in [\![\mathit{after}(\varphi,t)]\!]$ *then* $m'[s'] \sim s'$*.* □

Proof. We must prove that for every formula $\varphi \in$ sHML$_{\mathbf{nf}}$ if $(\!|\varphi,\Pi|\!) = m$ then $\mathsf{evtenf}(m,\varphi)$. We prove that for every $\varphi \in$ sHML$_{\mathbf{nf}}$, if $(\!|\varphi,\Pi|\!)[s] \stackrel{t}{\Longrightarrow} m'[s']$ and $s' \vDash \mathit{after}(\varphi,t)$ then $m'[s'] \sim s'$.
Now, assume that

$$(\!|\varphi,\Pi|\!)[s] \stackrel{t}{\Longrightarrow} m'[s'] \tag{5.88}$$

$$s' \vDash \mathit{after}(\varphi,t) \tag{5.89}$$

and so from (5.88) and Lemma 5.10 we have that

$$\exists \psi \in \text{sHML}_{\mathbf{nf}} \cdot \psi = \mathit{after}(\varphi,t) \tag{5.90}$$

$$(\!|\mathit{after}(\varphi,t),\Pi|\!) = m' = (\!|\psi,\Pi|\!). \tag{5.91}$$

Hence, knowing (5.89) and (5.90), by Proposition 5.11 (Transparency) we conclude that $(\!|\mathit{after}(\varphi,t),\Pi|\!)[s'] \sim s'$ as required, and so we are done. □

6. Transducer Optimality

Recall Definition 4.9 from Section 4. Through criteria such as Definitions 4.2 and 4.6, it defined what it means for a monitor to adequately enforce a formula. However, it did *not* assess whether a monitor is (to some extent) the "*best*" that one can find to enforce a property. In order to define such a notion we must first be able to compare monitors to one another via some kind of *distance measurement* that tells them apart. One potential measurement is to assess the monitor's level of *intrusiveness* when enforcing the property.

In Figure 6 we define function mc that inductively analyses a system run, represented as an explicit trace t_τ, and counts the number of modifications applied by the monitor. In each case the function reconstructs a trace system $\mathsf{sys}(t_\tau)$ and instruments it with the monitor m in order to assess the type of transformation applied. Specifically, in the first two cases, mc increments the counter whenever the monitor adapts, disables or enables an action, and then it recurses to keep on inspecting the run (*i.e.*, the suffix t'_τ in the first, and the same trace t_τ in the second) vis-a-vis the subsequent monitor state, m'. The third case, specifies that the counter stays unmodified when the monitor applies an identity transformation, while the last case returns the length of t_τ when $m[\mathsf{sys}(t_\tau)]$ is unable to execute further.

$$ec(m) \stackrel{\text{def}}{=} \begin{cases} \emptyset & \textit{if } m = X \\ \bigcup_{i\in I} ec(m_i) & \textit{if } m = \sum_{i\in I} m_i \\ ec(m') & \textit{if } m = \mathsf{rec}\, X.m' \textit{ or } m = (p, c, \underline{p}).m' \\ \{\mathsf{DIS}\} \cup ec(m') & \textit{if } m = ((x)!(y), c, \bullet).m' \textit{ or } m = (\bullet, c, \mathsf{a}?v).m' \\ \{\mathsf{EN}\} \cup ec(m') & \textit{if } m = ((x)?(y), c, \bullet).m' \textit{ or } m = (\bullet, c, \mathsf{a}!v).m' \\ \{\mathsf{ADPT}\} \cup ec(m') & \textit{if } m = (p, c, p').m' \textit{ and } p' \neq \underline{p} \neq \bullet \end{cases}$$

FIGURE 7. Enforcement Capabilities (ec).

Example 6.1. Recall the monitors of Example 3.2 and consider the following system run t^0_τ=$\mathsf{a}?v_1.\mathsf{a}?v_2.\tau.\mathsf{a}!w_2.\mathsf{a}!w_2.\mathsf{b}!(\mathsf{log}, v_2, w_2)$. For $m_\mathbf{e}$ and $m_\mathbf{a}$, function mc respectively counts three enabled actions, *i.e.,* $mc(m_\mathbf{e}, t^0_\tau)$=3, and four adapted actions, *i.e.,* $mc(m_\mathbf{a}, t^0_\tau)$=4 (since $\mathsf{b}!(\mathsf{log}, v_2, w_2)$ remains unmodified). The maximum count of 5 is attained by $m_\mathbf{d}$ as it immediately blocks the first input $\mathsf{a}?v_1$, and so none of the actions in t^0_τ can be executed by the composite system *i.e.,* $\forall\mu\cdot m_\mathbf{d}[\mathsf{sys}(t^0_\tau)] \stackrel{\mu}{\not\rightarrow}$ and so $mc(m_\mathbf{d}, t^0_\tau)$=5. Similarly, $mc(m_\mathbf{dt}, t^0_\tau)$=4 since $m_\mathbf{dt}$ allows the first request to be made, but blocks the second erroneous one, and as a result it also forbids the execution of the subsequent actions, *i.e.,* $\forall\mu\cdot m_\mathbf{dt}[\mathsf{sys}(t^0_\tau)] \xrightarrow{\mathsf{a}?v_1} \cdot \stackrel{\mu}{\not\rightarrow}$. Finally, $m_\mathbf{det}$ performs the least number of modifications, namely $mc(m_\mathbf{det}, t^0_\tau)$=2. The first modification is caused when the monitor blocks the second erroneous input and internally *inserts* a default input value that allows the composite system to proceed over a τ-action. This contrasts with $m_\mathbf{d}$ and $m_\mathbf{dt}$ which fail to perform this insertion step thereby contributing to their high intrusiveness score. The second modification is attained when $m_\mathbf{det}$ suppresses the redundant response. ■

We can now use function mc to compare monitors to each other in order to identify the least intrusive one, *i.e.,* the monitor that applies the least amount of transformations when enforcing a specific property. However, for this comparison to be fair, we must also compare like with like. This means that if a monitor enforces a formula by only disabling actions, it is only fair to compare it to other monitors of the same kind. It is reasonable to expect that monitors with more enforcement capabilities are likely to be "better" than those with fewer capabilities. We determine the enforcement capabilities of a monitor via function ec of Figure 7. It inductively analyses the structure of the monitor and deduces whether it can enable, disable and adapt actions based on the type of transformation triples it defines. For instance, if the monitor defines an output suppression triple, $((x)!(y), c, \bullet).m'$, or an input insertion branch, $(\bullet, c, \mathsf{a}?v).m'$, then ec determines that the monitor can disable actions DIS, while if it defines an input suppression, $((x)?(y), c, \bullet).m'$, or an output insertion branch, $(\bullet, c, \mathsf{a}!v).m'$, then it concludes that the monitor can enable actions, EN. Similarly, if a monitor defines a replacement transformation, it infers that the monitor can adapt actions, ADPT.

Example 6.2. Recall the monitors of Example 3.2. With function ec we determine that $ec(m_\mathbf{e})$=$\{\mathsf{EN}\}$, $ec(m_\mathbf{a})$=$\{\mathsf{ADPT}\}$, $ec(m_\mathbf{d})$=$ec(m_\mathbf{dt})$=$ec(m_\mathbf{det})$=$\{\mathsf{DIS}\}$. Monitors may also have multiple types of enforcement capabilities. For instance,

$$ec(\mathsf{rec}\, X.((x)?(y), \bullet).X + ((x)!(y), \bullet).X) = \{\mathsf{EN}, \mathsf{DIS}\}. \qquad ■$$

With these definitions we now define *optimal enforcement.*

Definition 6.3 (Optimal Enforcement)**.** A monitor m is *optimal* when enforcing φ, denoted as $\mathsf{oenf}(m,\varphi)$, iff it *enforces* φ (Definition 4.9) and when for every state s, explicit trace t_τ and monitor n, if $ec(n)\subseteq ec(m)$, $\mathsf{enf}(n,\varphi)$ and $s \xrightarrow{t_\tau}$ then $mc(m,t_\tau)\leq mc(n,t_\tau)$. □

Definition 6.3 states that an adequate (sound and eventually transparent) monitor m is *optimal* for φ, if one cannot find another adequate monitor n, with the same (or fewer) enforcement capabilities, that performs *fewer modifications* than m and is thus *less intrusive*.

Example 6.4. Recall formula φ_1 of Example 2.4 and monitor $m_{\mathbf{det}}$ of Example 4.7. Although showing that $\mathsf{oenf}(m_{\mathbf{det}},\varphi_1)$ is inherently difficult, from Example 6.1 we already get the intuition that it holds since $m_{\mathbf{det}}$ imposes the least amount of modifications compared to the other monitors of Examples 3.2 and 3.3. We further reaffirm this intuition using systems $s_{\mathbf{b}}$ and $s_{\mathbf{c}}$ from Example 2.4. In fact, when considering the invalid runs $t^1_\tau\overset{\text{def}}{=}\mathsf{a}?v_1.\tau.\mathsf{a}!w_1.\mathsf{a}!w_1.\mathsf{b}!(\mathsf{log},v_1,w_1)$ of $s_{\mathbf{b}}$, and $t^2_\tau\overset{\text{def}}{=}\mathsf{a}?v_1.\mathsf{a}?v_2.\tau.\mathsf{a}!w_2.\mathsf{b}!(\mathsf{log},v_2,w_2)$ of $s_{\mathbf{c}}$, one can easily deduce that no other adequate action disabling monitor can enforce φ_1 with fewer modifications than those imposed by $m_{\mathbf{det}}$, namely, $mc(m_{\mathbf{det}},t^1_\tau)=mc(m_{\mathbf{det}},t^2_\tau)=1$. Furthermore, consider the invalid traces $t^1_\tau\{{}^{\mathsf{c}}/_{\mathsf{a}}\}$ and $t^2_\tau\{{}^{\mathsf{c}}/_{\mathsf{a}}\}$ that are respectively produced by versions of $s_{\mathbf{b}}$ and $s_{\mathbf{c}}$ that interact on some port c instead of a (for any port $\mathsf{c}\neq\mathsf{a}$). Since $m_{\mathbf{det}}$ binds the port c to its data binder x and uses this information in its insertion branch, $(\bullet,(x?(\_))).Y$, the *same* modification count is achieved for these traces, as well *i.e.*, $mc(m_{\mathbf{det}},t^1_\tau\{{}^{\mathsf{c}}/_{\mathsf{a}}\})=mc(m_{\mathbf{det}},t^2_\tau\{{}^{\mathsf{c}}/_{\mathsf{a}}\})=1$. ■

Example 6.4 describes the case where formula φ is optimally enforced by a *finite-state* and *finitely-branching* monitor, $m_{\mathbf{det}}$. In the general case, this is not always possible.

Example 6.5. Consider formula φ_2 stating that an initial input on port a followed by another input from some other port $x_2{\neq}\mathsf{a}$ constitutes invalid system behaviour. Also consider monitor m_1 where $\mathsf{enf}(m_1,\varphi_2)$.

$$\begin{aligned}\varphi_2 &\overset{\text{def}}{=} [(\mathsf{a}?(\_))][((x_2)?(\_),x_2{\neq}\mathsf{a})]\mathsf{ff}\\ m_1 &\overset{\text{def}}{=} (\mathsf{a}?(\_)).\mathsf{rec}\,Y.((\bullet,\mathsf{b}?v_{\mathrm{def}}).Y+(\mathsf{a}?(\_)).\mathsf{id})\end{aligned}$$

When enforcing a system that generates the run $t^3_\tau \overset{\text{def}}{=} \mathsf{a}?v_1.\mathsf{b}?v_2.\mathsf{a}!w_1.u^3_\tau$, monitor m_1 modifies the trace only *once*. Although it disables the input $\mathsf{b}?v_2$, it subsequently unblocks the SuS by inserting $\mathsf{b}?v_{\mathrm{def}}$ and so trace t^3_τ is transformed into $\mathsf{a}?v_1.\tau.\mathsf{a}!w_1.u^3_\tau$. However, for a slightly modified version of t^3_τ, *e.g.*, $t^3_\tau\{{}^{\mathsf{c}}/_{\mathsf{b}}\}$, m_1 scores a modification count of $2+|\,u^3_\tau\,|$. This is the case because, although it blocks the invalid input on port c, it fails to insert the default value that unblocks the SuS. A more expressive version of m_1, such as

$$m_2 \overset{\text{def}}{=} (\mathsf{a}?(\_)).\mathsf{rec}\,Y.((\bullet,\mathsf{b}?v_{\mathrm{def}}).Y+\underline{(\bullet,\mathsf{c}?v_{\mathrm{def}}).Y}+(\mathsf{a}?(\_)).\mathsf{id}),$$

circumvents this problem by defining an extra insertion branch (underlined), but still fails to be optimal in the case of $t^3_\tau\{{}^{\mathsf{d}}/_{\mathsf{b}}\}$. In this case, there does not exist a way to finitely define a monitor that can insert a default value on every possible input port $x_2\neq\mathsf{a}$. Hence, it means that the optimal monitor m_{opt} for φ_1 would be an *infinite branching* one, *i.e.*, it requires a countably infinite summation that is not expressible in TRN,

$$m_{\mathsf{opt}}\overset{\text{def}}{=}(\mathsf{a}?(\_)).(\mathsf{rec}\,Y.\sum_{\mathsf{b}\in\textsc{Port}\ \textit{and}\ \mathsf{a}\neq\mathsf{b}}(\bullet,\mathsf{b}?v_{\mathrm{def}}).Y+(\mathsf{a}?(\_)).\mathsf{id})$$

or alternatively

$$(\mathsf{a}?(\_)).(\mathsf{rec}\,Y.\sum_{\mathsf{b}\in\textsc{Port}}(\bullet,\mathsf{a}{\neq}\mathsf{b},\mathsf{b}?v_{\mathrm{def}}).Y+(\mathsf{a}?(\_)).\mathsf{id})$$

where the condition $\mathsf{a}{\neq}\mathsf{b}$ is evaluated at runtime. ■

Unlike Example 6.4, Example 6.5 presents a case where optimality can only be attained by a monitor that defines an *infinite number of branches*; this is problematic since monitors are required to be *finitely* described. As it is not always possible to find a finite monitor that enforces a formula using the least amount of transformation for every possible system, this indicates that Definition 6.3 is too strict. We thus mitigate this issue by weakening Definition 6.3 and redefine it in terms of the set of system states $\textsc{Sys}_\Pi$, *i.e.*, the set of states that can only perform inputs using the ports specified in a finite $\Pi \subset \textsc{Port}$. Although this weaker version does *not* guarantee that the monitor m optimally enforces φ on *all* possible systems, it, however, ensures optimal enforcement for all the systems that input values via the ports specified in Π.

Definition 6.6 (Weak Optimal Enforcement)**.** A monitor m is *weakly optimal* when enforcing φ, denoted as $\mathsf{oenf}(m,\varphi,\Pi)$, iff it *enforces* φ (Definition 4.9) and when for every state $\underline{s \in \textsc{Sys}_\Pi}$, explicit trace t_τ and monitor n, if $ec(n) \subseteq ec(m)$, $\mathsf{enf}(n,\varphi)$ and $s \xrightarrow{t_\tau}$ then $mc(m,t_\tau) \leq mc(n,t_\tau)$. □

Example 6.7. Monitor m_1 from Example 6.5 ensures that φ_2 is optimally enforced on systems that interact on ports a and b, *i.e.*, when $\Pi = \{\mathsf{a},\mathsf{b}\}$, while monitor m_2 guarantees it when $\Pi = \{\mathsf{a},\mathsf{b},\mathsf{c}\}$. ■

We can show that a synthesised monitor $(\!|\, \varphi,\Pi \,|\!)$ obtained using the synthesis function Definition 5.3 from Section 5 is also guaranteed to be weakly optimal (as stated by Definition 6.6) when enforcing φ on a SuS s whose input ports are specified by Π, *i.e.*, $s \in \textsc{Sys}_\Pi$. Since our synthesis produces only action disabling monitors, *i.e.*, $ec((\!|\, \varphi,\Pi \,|\!)) = \{\mathsf{DIS}\}$ for all φ and Π, we can limit ourselves to monitors pertaining to the set $\textsc{DisTrn} \stackrel{\text{def}}{=} \{n \mid \text{if } ec(n) \subseteq \{\mathsf{DIS}\}\}$. The proof for Theorem 6.12 below relies on the following lemmas.

Lemma 6.8. *For every $m \in \textsc{DisTrn}$ and explicit trace t_τ, there exists some N such that $mc(m,t_\tau) = N$.* □

Lemma 6.9. *For every action α and monitor $m \in \textsc{DisTrn}$, if it is the case that $m \xrightarrow{\alpha \blacktriangleright \alpha} m'$, $\mathsf{enf}(m, \bigwedge_{i\in I}[(p_i,c_i)]\varphi_i)$ and $(p_j,c_j)(\alpha) = \sigma$ (for some $j{\in}I$) then $\mathsf{enf}(m',\varphi_j\sigma)$.* □

Lemma 6.10. *For every monitor $m \in \textsc{DisTrn}$, whenever $\mathsf{enf}(m, \bigwedge_{i\in I}[(p_i,c_i)]\varphi_i)$ and, for some m', $m \xrightarrow{(\mathsf{a}!v)\blacktriangleright\bullet} m'$ then $\mathsf{enf}(m', \bigwedge_{i\in I}[(p_i,c_i)]\varphi_i)$.* □

Lemma 6.11. *For every monitor $m \in \textsc{DisTrn}$, whenever $\mathsf{enf}(m, \bigwedge_{i\in I}[(p_i,c_i)]\varphi_i)$ and, for some m', $m \xrightarrow{\bullet\blacktriangleright(a?v)} m'$ then $\mathsf{enf}(m', \bigwedge_{i\in I}[(p_i,c_i)]\varphi_i)$.* □

Theorem 6.12 (Weak Optimal Enforcement)**.** *For every system $s \in \textsc{Sys}_\Pi$, explicit trace t_τ and monitor m, if $ec(m) \subseteq ec((\!|\, \varphi,\Pi \,|\!))$, $\mathsf{enf}(m,\varphi)$ and $s \xrightarrow{t_\tau}$ implies $mc((\!|\, \varphi,\Pi \,|\!),t_\tau) \leq mc(m,t_\tau)$.*

Proof. Since, from Lemma 6.8, we know that for every $m \in \textsc{DisTrn}$, there exists some N such that $mc(m,t_\tau) = N$, we can prove that if $\mathsf{enf}(m,\varphi)$, $s \xrightarrow{t_\tau}$ and $mc((\!|\, \varphi,\Pi \,|\!),t_\tau) = N$ then $N \leq mc(m,t_\tau)$. We proceed by rule induction on $mc((\!|\, \varphi,\Pi \,|\!),t_\tau)$.

Case $mc((\!|\, \varphi,\Pi \,|\!),t_\tau)$ when $t_\tau = \mu t'_\tau$ and $(\!|\, \varphi,\Pi \,|\!)[\mathsf{sys}(\mu t'_\tau)] \xrightarrow{\mu} m'_\varphi[\mathsf{sys}(t'_\tau)]$. Assume that

$$mc((\!|\, \varphi,\Pi \,|\!),\mu t'_\tau) = mc(m'_\varphi,t'_\tau) = N \tag{6.1}$$

which implies that

$$(\!|\, \varphi, \Pi \,|\!)[\mathsf{sys}(\mu t'_\tau)] \xrightarrow{\mu} m'_\varphi[\mathsf{sys}(t'_\tau)] \tag{6.2}$$

and also assume that

$$\mathsf{enf}(m, \varphi) \tag{6.3}$$

and that $s \xrightarrow{\mu t'_\tau}$. By the rules in our model we can infer that the reduction in (6.2) can result from rule iAsy when $\mu = \tau$, iDef and iTrnO when $\mu = \mathsf{a}!v$, or iTrnI when $\mu = \mathsf{a}?v$. We consider each case individually.

- iAsy: By rule iAsy from (6.2) we know that $\mu = \tau$ and that

$$m'_\varphi = (\!|\, \varphi, \Pi \,|\!). \tag{6.4}$$

  Since from (6.3) we know that m is sound and eventual transparent, we can thus deduce that m does not hinder internal τ-actions from occurring and so the composite system $(\!|\, \varphi, \Pi \,|\!)[\mathsf{sys}(\tau t'_\tau)]$ can always transition over τ via rule iAsy, that is,

$$m[\mathsf{sys}(\tau t'_\tau)] \xrightarrow{\tau} m[\mathsf{sys}(t'_\tau)]. \tag{6.5}$$

  Hence, by (6.1), (6.3) and since $s \xrightarrow{\tau t'_\tau}$ entails $s \xrightarrow{\tau} s'$ and $s' \xrightarrow{t'_\tau}$ we can apply the *inductive hypothesis* and deduce that $N \leq mc(m, t'_\tau)$ so that by (6.5) and the definition of mc, we conclude that $N \leq mc(m, \tau t'_\tau)$ as required.
- iDef: From (6.2) and rule iDef we know that $\mu = \mathsf{a}!v$, $(\!|\, \varphi, \Pi \,|\!) \stackrel{\mathsf{a}!v}{\not\longrightarrow}$ and that $m'_\varphi = \mathsf{id}$. Since id does not modify actions, we can deduce that $mc(m'_\varphi, t'_\tau) = 0$ and so by the definition of mc we know that $mc((\!|\, \varphi, \Pi \,|\!), (\mathsf{a}!v)t'_\tau) = 0$ as well. This means that we cannot find a monitor that performs fewer transformations, and so we conclude that $0 \leq mc(m, (\mathsf{a}!v)t'_\tau)$ as required.
- iTrnI: From (6.2) and rule iTrnI we know that $\mu = \mathsf{a}?v$ and that

$$(\!|\, \varphi, \Pi \,|\!) \xrightarrow{(\mathsf{a}?v)\blacktriangleright(\mathsf{a}?v)} m'_\varphi. \tag{6.6}$$

  We now inspect the cases for φ.
  - $\varphi \in \{\mathsf{ff}, \mathsf{tt}, X\}$: The cases for ff and X do not apply since $(\!|\, \mathsf{ff}, \Pi \,|\!)$ and $(\!|\, X, \Pi \,|\!)$ do not yield a valid monitor, while the case when $\varphi = \mathsf{tt}$ gets trivially satisfied since $(\!|\, \mathsf{tt}, \Pi \,|\!) = \mathsf{id}$ and $mc(\mathsf{id}, (\mathsf{a}?v)t'_\tau) = 0$.
  - $\varphi = \bigwedge_{i\in I} [(p_i, c_i)]\varphi_i$ where $\#_{i\in I}(p_i, c_i)$: Since $\varphi = \bigwedge_{i\in I} [(p_i, c_i)]\varphi_i$, by the definition of $(\!|\, - \,|\!)$ we have that

$$\begin{aligned} &(\!|\, \varphi_\wedge = \textstyle\bigwedge_{i\in I} [(p_i, c_i)]\varphi_i, \Pi \,|\!) \\ &\quad = \mathsf{rec}\, Y. \left(\sum_{i\in I} \begin{cases} \mathsf{dis}(p_i, c_i, Y, \Pi) & \textit{if } \varphi_i = \mathsf{ff} \\ (p_i, c_i).(\!|\, \varphi_i, \Pi \,|\!) & \textit{otherwise} \end{cases} \right) + \mathsf{def}(\bigwedge_{i\in I} [(p_i, c_i)]\varphi_i) \\ &\quad = \left(\sum_{i\in I} \begin{cases} \mathsf{dis}(p_i, c_i, (\!|\, \varphi_\wedge, \Pi \,|\!), \Pi) & \textit{if } \varphi_i = \mathsf{ff} \\ (p_i, c_i).(\!|\, \varphi_i, \Pi \,|\!) & \textit{otherwise} \end{cases} \right) + \mathsf{def}(\bigwedge_{i\in I} [(p_i, c_i)]\varphi_i) \end{aligned} \tag{6.7}$$

    Since normalized conjunctions are disjoint, *i.e.*, $\#_{i\in I}(p_i, c_i)$, from (6.7) we can infer that the identity reduction in (6.6) can only happen when $\mathsf{a}?v$ matches an identity branch,

$(p_j, c_j).(\!|\varphi_j, \Pi|\!)$ (for some $j \in I$), and so we have that

$$(p_j, c_j)(\mathsf{a}?v) = \sigma. \tag{6.8}$$

Hence, knowing (6.6) and (6.8), by rule eTrn we know that $m'_\varphi = (\!|\varphi_j\sigma, \Pi|\!)$ and so by (6.1) we can infer that

$$mc(m'_\varphi, t'_\tau) = N \quad \textit{where } m'_\varphi = (\!|\varphi_j\sigma, \Pi|\!). \tag{6.9}$$

Since from (6.7) we also know that the monitor branch $(p_j, c_j).(\!|\varphi_j, \Pi|\!)$ is derived from a non-violating modal necessity, *i.e.*, $[(p_j, c_j)]\varphi_j$ where $\varphi_j \neq \mathsf{ff}$, we can infer that $\mathsf{a}?v$ is not a violating action and so it should not be modified by any other monitor m, as otherwise it would infringe the eventual transparency constraint of assumption (6.3). Therefore, we can deduce that

$$m \xrightarrow{(\mathsf{a}?v)\blacktriangleright(\mathsf{a}?v)} m' \quad \text{(for some } m'\text{)} \tag{6.10}$$

and subsequently, knowing (6.10) and that $t_\tau = (\mathsf{a}?v)t'_\tau$ and also that $\mathsf{sys}((\mathsf{a}?v)t'_\tau) \xrightarrow{\mathsf{a}?v} \mathsf{sys}(t'_\tau)$, by rule iTrnI and the definition of mc we infer that

$$mc(m, (\mathsf{a}?v)t'_\tau) = mc(m', t'_\tau). \tag{6.11}$$

As by (6.3), (6.6), (6.8) and Lemma 6.9 we know that $\mathsf{enf}(m', \varphi_j\sigma)$, by (6.9) and since $s \xrightarrow{(\mathsf{a}?v)t'_\tau}$ entails that $s \xrightarrow{\mathsf{a}?v} s'$ and $s' \xrightarrow{t'_\tau}$, we can apply the *inductive hypothesis* and deduce that $N \leq mc(m', t'_\tau)$ and so from (6.11) we conclude that $N \leq mc(m, (\mathsf{a}?v)t'_\tau)$ as required.

– $\varphi = \mathsf{max}\, X.\varphi'$ and $X \in \mathbf{fv}(\varphi')$: Since $\varphi = \mathsf{max}\, X.\varphi'$, by the syntactic restrictions of sHML$_{\mathbf{nf}}$ we infer that φ' cannot be ff or tt since $X \notin \mathbf{fv}(\varphi')$ otherwise, and it cannot be X since every logical variable must be guarded. Hence, φ' must be of a specific form, *i.e.*, $\mathsf{max}\, Y_1 \ldots Y_n. \bigwedge_{i\in I} [(p_i, c_i)]\varphi_i$, and so by unfolding every fixpoint in $\mathsf{max}\, X.\varphi'$ we reduce our formula to $\varphi \stackrel{\text{def}}{=} \bigwedge_{i\in I} [(p_i, c_i)]\varphi_i\{^{\mathsf{max}\, X.\varphi'}/_X, \ldots\}$. We thus omit the remainder of this proof as it becomes identical to that of the subcase when $\varphi = \bigwedge_{i\in I} [(p_i, c_i)]\varphi_i$.

• iTrnO: We elide the proof for this case as it is very similar to that of iTrnI.

Case $mc((\!|\varphi, \Pi|\!), t_\tau)$ when $t_\tau = \mu t'_\tau$ and $(\!|\varphi, \Pi|\!)[\mathsf{sys}(\mu t'_\tau)] \xrightarrow{\mu'} m'_\varphi[\mathsf{sys}(t'_\tau)]$ and $\mu' \neq \mu$. Assume that

$$mc((\!|\varphi, \Pi|\!), \mu t'_\tau) = 1 + M \tag{6.12}$$
$$\textit{where } M = mc(m'_\varphi, t'_\tau) \tag{6.13}$$

which implies that

$$(\!|\varphi, \Pi|\!)[\mathsf{sys}(\mu t'_\tau)] \xrightarrow{\mu'} m'_\varphi[\mathsf{sys}(t'_\tau)] \quad \textit{where } \mu' \neq \mu \tag{6.14}$$

and also assume that

$$\mathsf{enf}(m, \varphi) \tag{6.15}$$

and that $s \xrightarrow{\mu t'_\tau}$. Since we only consider action disabling monitors, the μ' reduction of (6.14) can only be achieved via rules iDisO or iDisI. We thus explore both cases.

- iDisI: From (6.14) and by rule iDisI we have that $\mu = \mathsf{a}?v$ and $\mu' = \tau$ and that

$$(\!|\, \varphi, \Pi \,|\!) \xrightarrow{\bullet\blacktriangleright \mathsf{a}?v} m'_\varphi. \tag{6.16}$$

We now inspect the cases for φ.
 – $\varphi \in \{\mathsf{ff}, \mathsf{tt}, X\}$: These cases do not apply since $(\!|\, \mathsf{ff}, \Pi \,|\!)$ and $(\!|\, X, \Pi \,|\!)$ do not yield a valid monitor, while $(\!|\, \mathsf{tt}, \Pi \,|\!) = \mathsf{id}$ does not perform the reduction in (6.16).
 – $\varphi = \bigwedge_{i \in I} [(p_i, c_i)]\varphi_i$ where $\#_{i \in I}(p_i, c_i)$: Since $\varphi = \bigwedge_{i \in I} [(p_i, c_i)]\varphi_i$, by the definition of $(\!|\, - \,|\!)$ we have that

$$\begin{aligned} &(\!|\, \varphi_\wedge = \textstyle\bigwedge_{i \in I} [(p_i, c_i)]\varphi_i, \Pi \,|\!) \\ &\quad = \; \mathsf{rec}\, Y. \left(\sum_{i \in I} \begin{cases} \mathsf{dis}(p_i, c_i, Y, \Pi) & \textit{if } \varphi_i = \mathsf{ff} \\ (p_i, c_i).(\!|\, \varphi_i, \Pi \,|\!) & \textit{otherwise} \end{cases} \right) + \mathsf{def}(\bigwedge_{i \in I} [(p_i, c_i)]\varphi_i) \\ &\quad = \; \left(\sum_{i \in I} \begin{cases} \mathsf{dis}(p_i, c_i, (\!|\, \varphi_\wedge, \Pi \,|\!), \Pi) & \textit{if } \varphi_i = \mathsf{ff} \\ (p_i, c_i).(\!|\, \varphi_i, \Pi \,|\!) & \textit{otherwise} \end{cases} \right) + \mathsf{def}(\bigwedge_{i \in I} [(p_i, c_i)]\varphi_i) \end{aligned} \tag{6.17}$$

Since normalized conjunctions are disjoint *i.e.*, $\#_{i \in I}(p_i, c_i)$, and since $s \xrightarrow{\mu t'_\tau}$ where $\mu = (\mathsf{a}?v)$, by the definition of dis, from (6.17) we deduce that the reduction in (6.16) can only be performed by an insertion branch of the form, $(\bullet, c_j\{\mathsf{a}/x\}, \mathsf{a}?v).(\!|\, \bigwedge_{i \in I} [(p_i, c_i)]\varphi_i, \Pi \,|\!)$ that can only be derived from a violating modal necessity $[(p_j, c_j)]\mathsf{ff}$ (for some $j \in I$). Hence, we can infer that

$$m'_\varphi = (\!|\, \textstyle\bigwedge_{i \in I} [(p_i, c_i)]\varphi_i, \Pi \,|\!) \tag{6.18}$$

$$p_j = (x)?(y) \;\textit{and}\; c_j\{\mathsf{a}/x\} \Downarrow \mathsf{true}. \tag{6.19}$$

Knowing (6.19) and that $[(p_j, c_j)]\mathsf{ff}$ we can deduce that any input on port a is erroneous and so for the soundness constraint of assumption (6.15) to hold, any other monitor m is obliged to somehow *block* this input port. As we consider action disabling monitors, *i.e.*, $m \in \textsc{DisTrn}$, we can infer that monitor m may block this input in two ways, namely, either by not reacting to the input action, *i.e.*, $m \not\xrightarrow{\mathsf{a}?v}$, or by additionally inserting a default value v, *i.e.*, $m \xrightarrow{\bullet\blacktriangleright(\mathsf{a}?v)} m'$. We explore both cases.
 * $m \not\xrightarrow{\mathsf{a}?v}$: Since $\mathsf{sys}((\mathsf{a}?v)t'_\tau) \xrightarrow{\mathsf{a}?v} \mathsf{sys}(t'_\tau)$ and since $m \not\xrightarrow{\mathsf{a}?v}$, by the rules in our model we know that for every action μ', $m[\mathsf{sys}((\mathsf{a}?v)t'_\tau)] \not\xrightarrow{\mu'}$ and so by the definition of mc we have that $mc(m, (\mathsf{a}?v)t'_\tau) = |\,(\mathsf{a}?v)t'_\tau\,|$ meaning that by blocking inputs on a, m also blocks (and thus modifies) every subsequent action of trace t'_τ. Hence, this suffices to deduce that *at worst* $1 + M$ is equal to $|\,(\mathsf{a}?v)t'_\tau\,|$, that is $1 + M \leq |\,(\mathsf{a}?v)t'_\tau\,|$, and so from (6.12) we can deduce that $1 + M \leq mc((\!|\, \varphi, \Pi \,|\!), \mu t'_\tau)$ as required.
 * $m \xrightarrow{\bullet\blacktriangleright(\mathsf{a}?v)} m'$: Since $\mathsf{sys}((\mathsf{a}?v)t'_\tau) \xrightarrow{\mathsf{a}?v} \mathsf{sys}(t'_\tau)$ and since $m \xrightarrow{\bullet\blacktriangleright(\mathsf{a}?v)} m'$, by rule iDisI we know that $m[\mathsf{sys}((\mathsf{a}?v)t'_\tau)] \xrightarrow{\tau} m[\mathsf{sys}(t'_\tau)]$ and so by the definition of mc we have that

$$mc(m, (\mathsf{a}?v)t'_\tau) = 1 + mc(m', t'_\tau). \tag{6.20}$$

As by (6.15), (6.16) and Lemma 6.11 we infer that $\mathsf{enf}(m', \bigwedge_{i\in I}[(p_i,c_i)]\varphi_i)$, by (6.13), (6.20) and since $s \xrightarrow{(\mathsf{a}?v)t'_\tau}$ entails that $s \xrightarrow{(\mathsf{a}?v)} s'$ and $s' \xrightarrow{t'_\tau}$, we can apply the *inductive hypothesis* and deduce that $M \le mc(m', t'_\tau)$ and so from (6.12), (6.13) and (6.20) we conclude that $1 + M \le mc(m, (\mathsf{a}?v)t'_\tau)$ as required.

– $\varphi = \mathsf{max}\, X.\varphi'$ and $X \in \mathbf{fv}(\varphi')$: We omit showing this proof as it is a special case of when $\varphi = \bigwedge_{i\in I}[(p_i,c_i)]\varphi_i$.

- iDisO: We omit showing the proof for this subcase as it is very similar to that of case iDisI. Apart from the obvious differences (*e.g.*, $\mathsf{a}!v$ instead of $\mathsf{a}?v$), Lemma 6.10 is used instead of Lemma 6.11.

Case $mc((\!|\varphi,\Pi|\!), t_\tau)$ when $t_\tau \in \{\mu t'_\tau, \varepsilon\}$ and $(\!|\varphi,\Pi|\!)[\mathsf{sys}(\mu t'_\tau)] \stackrel{\mu'}{\not\longrightarrow}$. Assume that

$$mc((\!|\varphi,\Pi|\!), t_\tau) = |\, t_\tau \,| \quad (\text{where } t_\tau \in \{\mu t'_\tau, \varepsilon\}) \tag{6.21}$$

$$(\!|\varphi,\Pi|\!)[\mathsf{sys}(\mu t'_\tau)] \stackrel{\mu'}{\not\longrightarrow} \tag{6.22}$$

$$\mathsf{enf}(m,\varphi) \tag{6.23}$$

Since $t_\tau \in \{\mu t'_\tau, \varepsilon\}$ we consider both cases individually.

- $t_\tau = \varepsilon$: This case holds trivially since by (6.21), (6.22) and the definition of mc, $mc((\!|\varphi,\Pi|\!), \varepsilon) = |\,\varepsilon\,| = 0$.
- $t_\tau = \mu t'_\tau$: Since $t_\tau = \mu t'_\tau$ we can immediately exclude the cases when $\mu \in \{\tau, \mathsf{a}!v\}$ since rules iAsy and iDef make it impossible for (6.22) to be attained in such cases. Particularly, rule iAsy always permits the SuS to independently perform an internal τ-move, while rule iDef allows the monitor to default to id whenever the system performs an unspecified output $\mathsf{a}!v$. However, in the case of inputs, $\mathsf{a}?v$, the monitor may completely block inputs on a port a and as a consequence cause the entire composite system $(\!|\varphi,\Pi|\!)[\mathsf{sys}(\mu t'_\tau)]$ to block, thereby making (6.22) a possible scenario. We thus inspect the cases for φ vis-a-vis $\mu = \mathsf{a}?v$.

– $\varphi \in \{\mathsf{ff}, \mathsf{tt}, X\}$: These cases do not apply since $(\!|\mathsf{ff},\Pi|\!)$ and $(\!|X,\Pi|\!)$ do not yield a valid monitor and since $(\!|\mathsf{tt},\Pi|\!) = \mathsf{id}$ is incapable of attaining (6.22).

– $\varphi = \bigwedge_{i\in I}[(p_i,c_i)]\varphi_i$ where $\#_{i\in I}(p_i,c_i)$: Since $\varphi = \bigwedge_{i\in I}[(p_i,c_i)]\varphi_i$, by the definition of $(\!|-|\!)$ we have that

$$\begin{aligned}
&(\!|\varphi_\wedge = \textstyle\bigwedge_{i\in I}[(p_i,c_i)]\varphi_i, \Pi|\!)\\
&\quad = \mathsf{rec}\, Y.\left(\sum_{i\in I}\begin{cases}\mathsf{dis}(p_i,c_i,Y,\Pi) & \textit{if } \varphi_i = \mathsf{ff}\\ (p_i,c_i).(\!|\varphi_i,\Pi|\!) & \textit{otherwise}\end{cases}\right) + \mathsf{def}(\bigwedge_{i\in I}[(p_i,c_i)]\varphi_i)\\
&\quad = \left(\sum_{i\in I}\begin{cases}\mathsf{dis}(p_i,c_i,(\!|\varphi_\wedge,\Pi|\!),\Pi) & \textit{if } \varphi_i = \mathsf{ff}\\ (p_i,c_i).(\!|\varphi_i,\Pi|\!) & \textit{otherwise}\end{cases}\right) + \mathsf{def}(\bigwedge_{i\in I}[(p_i,c_i)]\varphi_i)
\end{aligned}\tag{6.24}$$

Since $\mu = \mathsf{a}?v$, from (6.24) and by the definitions of dis and def we can infer that the only case when (6.22) is possible is when the inputs on port a satisfy a violating modal necessity, that is, there exists some $j \in I$ such that $[(p_j,c_j)]\mathsf{ff}$ and for every $v \in \mathrm{VAL}$, $\mathsf{match}(p_j, \mathsf{a}?v) = \sigma$ and $c_j\sigma \Downarrow \mathsf{true}$. At the same time, the monitor is also *unaware* of the port on which the erroneous input can be made, *i.e.*, $\mathsf{a} \notin \Pi$. Hence, this case does not apply since we limit ourselves to SYS_Π, *i.e.*, states of system that can only input values via the ports specified in Π.

– $\varphi = \mathsf{max}\, X.\varphi'$: As argued in previous cases, this is a special case of $\varphi = \bigwedge_{i\in I} [(p_i, c_i)]\varphi_i$ and so we omit this part of the proof.

Case $mc((\!|\varphi,\Pi|\!), t_\tau)$ when $t_\tau \in \{\mu t'_\tau, \varepsilon\}$ and $(\!|\varphi,\Pi|\!)[\mathsf{sys}(t_\tau)] \xrightarrow{\mu'} m'_\varphi[\mathsf{sys}(t_\tau)]$. As we only consider action disabling monitors, this case does not apply since the transition $(\!|\varphi,\Pi|\!)[\mathsf{sys}(t_\tau)] \xrightarrow{\mu'} m'_\varphi[\mathsf{sys}(t_\tau)]$ can only be achieved via action enabling and rules iEnO and iEnI. □

7. Conclusions and Related Work

This work extends the framework presented in the precursor to this work [ACFI18] to the setting of bidirectional enforcement where observable actions such as inputs and outputs require different treatment. We achieve this by:

(1) augmenting substantially our instrumentation relation (Figure 4);
(2) refining our definition of enforcement to incorporate transparency over violating systems (Definition 4.9);
(3) providing a more extensive synthesis function (Definition 5.3) that is proven correct (Theorem 5.5); and
(4) exploring notions of transducer optimality in terms of limited levels of intrusiveness (Definitions 6.3 and 6.6 and Theorem 6.12).

Future work. There are a number of possible avenues for extending our work. One immediate step would be the implementation of the monitor operational model presented in Section 3 together with the synthesis function described in Section 5. This effort should be integrated it within the detectEr tool suite [CFS15, AF16, CFAI17, CFA$^+$17, AAA$^+$21]. This would allow us to assess the overhead induced by our proposed bidirectional monitoring [AAFI21]. Another possible direction would be the development of behavioural theories for the transducer operational model presented in Section 3, along the lines of the refinement preorders studied in earlier work on sequence recognisers [Fra21, Fra17, AAF$^+$21a]. Finally, applications of this theory on transducers, along the lines of [LMM20] are also worth exploring.

Related work. In his seminal work [Sch00], Schneider introduced the concept of runtime enforcement and enforceability in a linear-time setting. Particularly, in his setting a property is deemed enforceable if its *violation* can be *detected* by a *truncation automaton*, and prevented via system termination. By preventing misbehaviour, these automata can only enforce safety properties. Ligatti *et al.* extended this work in [LBW05] via *edit automata*—an enforcement mechanism capable of *suppressing* and *inserting* system actions. A property is thus enforceable if it can be expressed as an edit automaton that *transforms* invalid executions into valid ones via suppressions and insertions. As a means to assess the correctness of these automata, the authors introduced *soundness* and *transparency*.

Both settings by Schneider [Sch00] and Ligatti *et al.* [LBW05] assume a trace based view of the SuS and that every action can be freely manipulated by the monitor. They also do not distinguish between the specification and the enforcement mechanism, as properties are encoded in terms of the enforcement model itself, *i.e.*, as edit/truncation automata. In our prior work [ACFI18], we addressed this issue by separating the specification and verification aspects of the logic and explored the enforceability of μHML in a unidirectional context and in relation to a definition of adequate enforcement defined in terms of soundness and transparency. In this paper we adopt a stricter notion of enforceability that requires adherence to eventual transparency and investigate the enforceability of sHML formulas in the context of bidirectional enforcement.

Bielova and Massacci [Bie11, BM11b] remark that, on their own, soundness and transparency fail to specify the extent in which a transducer should modify invalid runtime behaviour and thus introduce a *predictability* criterion. A transducer is *predictable* if one can predict the edit-distance between an invalid execution and a valid one. With this criterion, adequate monitors are further restricted by setting an upper bound on the number of transformations that a monitor can apply to correct invalid traces. Although this is similar to our notion of optimality, we however use it to compare an adequate (sound and eventual transparent) monitor to *all* the other adequate monitors and determine whether it is the least intrusive monitor that can enforce the property of interest.

In [KAB+17] Könighofer *et al.* present a synthesis algorithm similar to our own that produces action replacement monitors called *shields* from safety properties encoded as automata-based specifications. Although their shields can analyse both the inputs and outputs of a reactive system, they still perform unidirectional enforcement since they only modify the data associated with the system's output actions. By definition, shields should adhere to correctness and minimum deviation which are, in some sense, analogous to soundness and transparency respectively.

In [PRS+16, PRS+17], Pinisetty *et al.* conduct a preliminary investigation of RE in a bidirectional setting. They, however, model the behaviour of the SuS as a trace of input and output pairs, *a.k.a. reactions*, and focus on enforcing properties by modifying the payloads exchanged by these reactions. This way of modelling system behaviour is, however, quite restrictive as it only applies to synchronous reactive systems that output a value in reaction to an input. This differs substantially from the way we model systems as LTSs, particularly since we can model more complex systems that may opt to collect data from multiple inputs, or supply multiple outputs in response to an input. The enforcement abilities studied in [PRS+16, PRS+17] are also confined to action replacement that only allows the monitor to modify the data exchanged by the system in its reactions, and so the monitors in [PRS+16, PRS+17] are unable to disable and enable actions. Due to their trace based view of the system, their correctness specifications do not allow for defining correct system behaviour in view of its different execution branches. This is particularly useful when considering systems whose inputs may lead them into taking erroneous computation branches that produce invalid outputs. Moreover, since their systems do not model communication ports, their monitors cannot influence directly the control structure of the SuS, *e.g.*, by opening, closing or rerouting data through different ports.

Finally, Lanotte *et al.* [LMM20] employ similar synthesis techniques and correctness criteria to ours (Definitions 4.2 and 4.4) to generate enforcement monitors for a timed setting. They apply their process-based approach to build tools that enforce data-oriented security properties. Although their implementations handle the enforcement of first-order properties, the theory on which it is based does not, nor does it investigate logic enforceability.

References

[AAA+21] Duncan Paul Attard, Luca Aceto, Antonis Achilleos, Adrian Francalanza, Anna Ingólfsdóttir, and Karoliina Lehtinen. Better late than never or: Verifying asynchronous components at runtime. In Kirstin Peters and Tim A. C. Willemse, editors, *Formal Techniques for Distributed Objects,* *Components, and Systems - 41st IFIP WG 6.1 International Conference, FORTE 2021, Held as Part of the 16th International Federated Conference on Distributed Computing Techniques, DisCoTec 2021, Valletta, Malta, June 14-18, 2021, Proceedings*, volume 12719 of *Lecture Notes in Computer Science*, pages 207–225. Springer, 2021.

[AAA+22] Luca Aceto, Antonis Achilleos, Duncan Paul Attard, Léo Exibard, Adrian Francalanza, and Anna Ingólfsdóttir. A monitoring tool for linear-time μhml. In Maurice H. ter Beek and Marjan Sirjani, editors, *Coordination Models and Languages - 24th IFIP WG 6.1 International Conference, COORDINATION 2022, Held as Part of the 17th International Federated Conference on Distributed Computing Techniques, DisCoTec 2022, Lucca, Italy, June 13-17, 2022, Proceedings*, volume 13271 of *Lecture Notes in Computer Science*, pages 200–219. Springer, 2022.

[AAF+19] Luca Aceto, Antonis Achilleos, Adrian Francalanza, Anna Ingólfsdóttir, and Karoliina Lehtinen. Adventures in monitorability: From branching to linear time and back again. *Proc. ACM Program. Lang.*, 3(POPL):52:1–52:29, January 2019.

[AAF+20] Luca Aceto, Antonis Achilleos, Adrian Francalanza, Anna Ingólfsdóttir, and Sævar Örn Kjartansson. Determinizing monitors for HML with recursion. *J. Log. Algebraic Methods Program.*, 111:100515, 2020.

[AAF+21a] Luca Aceto, Antonis Achilleos, Adrian Francalanza, Anna Ingólfsdóttir, and Karoliina Lehtinen. The Best a Monitor Can Do. In *CSL*, volume 183 of *LIPIcs*, pages 7:1–7:23. Schloss Dagstuhl, 2021.

[AAF+21b] Luca Aceto, Antonis Achilleos, Adrian Francalanza, Anna Ingólfsdóttir, and Karoliina Lehtinen. An operational guide to monitorability with applications to regular properties. *Softw. Syst. Model.*, 20(2):335–361, 2021.

[AAFI18a] Luca Aceto, Antonis Achilleos, Adrian Francalanza, and Anna Ingólfsdóttir. A Framework for Parameterized Monitorability. In *FoSSaCS*, pages 203–220. Springer, 2018.

[AAFI18b] Luca Aceto, Antonis Achilleos, Adrian Francalanza, and Anna Ingólfsdóttir. Monitoring for silent actions. In Satya Lokam and R. Ramanujam, editors, *FSTTCS 2017: Foundations of Software Technology and Theoretical Computer Science*, volume 93 of *LIPIcs*, pages 7:1–7:14, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[AAFI21] Luca Aceto, Duncan Paul Attard, Adrian Francalanza, and Anna Ingólfsdóttir. On benchmarking for concurrent runtime verification. In *FASE*, volume 12649 of *LNCS*, pages 3–23. Springer, 2021.

[ACFI18] Luca Aceto, Ian Cassar, Adrian Francalanza, and Anna Ingólfsdóttir. On Runtime Enforcement via Suppressions. In *CONCUR*, volume 118, pages 34:1–34:17. Schloss Dagstuhl, 2018.

[ACFI21] Luca Aceto, Ian Cassar, Adrian Francalanza, and Anna Ingólfsdóttir. On bidirectional runtime enforcement. In Kirstin Peters and Tim A. C. Willemse, editors, *Formal Techniques for Distributed Objects, Components, and Systems - 41st IFIP WG 6.1 International Conference, FORTE 2021, Held as Part of the 16th International Federated Conference on Distributed Computing Techniques, DisCoTec 2021, Valletta, Malta, June 14-18, 2021, Proceedings*, volume 12719 of *Lecture Notes in Computer Science*, pages 3–21. Springer, 2021.

[AEF+22] Antonis Achilleos, Léo Exibard, Adrian Francalanza, Karoliina Lehtinen, and Jasmine Xuereb. A synthesis tool for optimal monitors in a branching-time setting. In Maurice H. ter Beek and Marjan Sirjani, editors, *Coordination Models and Languages - 24th IFIP WG 6.1 International Conference, COORDINATION 2022, Held as Part of the 17th International Federated Conference on Distributed Computing Techniques, DisCoTec 2022, Lucca, Italy, June 13-17, 2022, Proceedings*, volume 13271 of *Lecture Notes in Computer Science*, pages 181–199. Springer, 2022.

[AF16] Duncan Paul Attard and Adrian Francalanza. A Monitoring Tool for a Branching-Time Logic. In *RV*, pages 473–481. Springer, 2016.

[AI99] Luca Aceto and Anna Ingólfsdóttir. Testing Hennessy-Milner Logic with Recursion. In Wolfgang Thomas, editor, *FoSSaCS*, pages 41–55. Springer, 1999.

[AILS07] Luca Aceto, Anna Ingólfsdóttir, Kim Guldstrand Larsen, and Jiri Srba. *Reactive Systems: Modelling, Specification and Verification.* Cambridge Univ. Press, NY, USA, 2007.

[Av11] Rajeev Alur and Pavol Černý. Streaming Transducers for Algorithmic Verification of Single-pass List-processing Programs. In *POPL*, pages 599–610. ACM, 2011.

[BAF21] Matthew Alan Le Brun, Duncan Paul Attard, and Adrian Francalanza. Graft: general purpose raft consensus in elixir. In Stavros Aronis and Annette Bieniusa, editors, *Proceedings of the 20th*

ACM SIGPLAN International Workshop on Erlang, Erlang@ICFP 2021, Virtual Event, Korea, August 26, 2021, pages 2–14. ACM, 2021.

[BCD+17] Laura Bocchi, Tzu-Chun Chen, Romain Demangeon, Kohei Honda, and Nobuko Yoshida. Monitoring networks through multiparty session types. *TCS*, 669:33 – 58, 2017.

[Bie11] Nataliia Bielova. *A theory of constructive and predictable runtime enforcement mechanisms*. PhD thesis, University of Trento, 2011.

[BM11a] Nataliia Bielova and Fabio Massacci. Do you really mean what you actually enforced?-edited automata revisited. *Journal of Information Security*, 10(4):239–254, 2011.

[BM11b] Nataliia Bielova and Fabio Massacci. Predictability of enforcement. In *International Symposium on Engineering Secure Software and Systems*, pages 73–86. Springer, 2011.

[CBD+12] Tzu-Chun Chen, Laura Bocchi, Pierre-Malo Deniélou, Kohei Honda, and Nobuko Yoshida. Asynchronous Distributed Monitoring for Multiparty Session Enforcement. In *TGC*, pages 25–45. Springer, 2012.

[CFA+17] Ian Cassar, Adrian Francalanza, Duncan Paul Attard, Luca Aceto, and Anna Ingólfsdóttir. A Suite of Monitoring Tools for Erlang. In *RV-CuBES*, volume 3 of *Kalpa Publications in Computing*, pages 41–47. EasyChair, 2017.

[CFAI17] Ian Cassar, Adrian Francalanza, Luca Aceto, and Anna Ingólfsdóttir. eAOP: An Aspect Oriented Programming Framework for Erlang. In *Erlang*, ACM SIGPLAN, 2017.

[CFS15] Ian Cassar, Adrian Francalanza, and Simon Said. Improving Runtime Overheads for detectEr. In *FESCA*, volume 178 of *EPTCS*, pages 1–8, 2015.

[FAI17] Adrian Francalanza, Luca Aceto, and Anna Ingólfsdóttir. Monitorability for the Hennessy-Milner logic with recursion. *Formal Methods in System Design*, 51(1):87–116, 2017.

[FFM08] Yliès Falcone, Jean-Claude Fernandez, and Laurent Mounier. Synthesizing enforcement monitors wrt. the safety-progress classification of properties. In *Information Systems Security*. Springer, 2008.

[FFM12] Yliès Falcone, Jean-Claude Fernandez, and Laurent Mounier. What can you verify and enforce at runtime? *Journal on Software Tools for Technology Transfer*, 14(3):349, 2012.

[Fra17] Adrian Francalanza. Consistently-Detecting Monitors. In *CONCUR*, volume 85 of *LIPIcs*, pages 8:1–8:19, Dagstuhl, Germany, 2017.

[Fra21] Adrian Francalanza. A Theory of Monitors. *Information and Computation*, page 104704, 2021.

[HL95] M. Hennessy and X. Liu. A Modal Logic for Message Passing Processes. *Acta Informatica*, 32(4):375–393, Apr 1995.

[HL96] M. Hennessy and H. Lin. Proof systems for message-passing process algebras. *Formal Aspects of Computing*, 8(4):379–407, 1996.

[HRTZ18] Klaus Havelund, Giles Reger, Daniel Thoma, and Eugen Zălinescu. *Monitoring Events that Carry Data*, pages 61–102. Springer International Publishing, Cham, 2018.

[JGP16] Limin Jia, Hannah Gommerstadt, and Frank Pfenning. Monitors and blame assignment for higher-order session types. In *POPL*, pages 582–594, NY, USA, 2016. ACM.

[KAB+17] Bettina Könighofer, Mohammed Alshiekh, Roderick Bloem, Laura Humphrey, Robert Könighofer, Ufuk Topcu, and Chao Wang. Shield synthesis. *Formal Methods in System Design*, 51(2):332–361, Nov 2017.

[Koz83] Dexter C. Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27:333–354, 1983.

[KT12] Raphaël Khoury and Nadia Tawbi. Which security policies are enforceable by runtime monitors? a survey. *Computer Science Review*, 6(1):27–45, 2012.

[Lar90] Kim G Larsen. Proof systems for satisfiability in Hennessy-Milner logic with recursion. *Theoretical Computer Science*, 72(2):265–288, 1990.

[LBW05] Jay Ligatti, Lujo Bauer, and David Walker. Edit automata: enforcement mechanisms for run-time security policies. *Journal of Information Security*, 4(1):2–16, 2005.

[LBW09] Jay Ligatti, Lujo Bauer, and David Walker. Run-time enforcement of nonsafety policies. *ACM Trans. Inf. Syst. Secur.*, 12(3):19:1–19:41, January 2009.

[LMM20] Ruggero Lanotte, Massimo Merro, and Andrei Munteanu. Runtime enforcement for control system security. In *CSF*, pages 246–261. IEEE, 2020.

[MPW92] Robin Milner, Joachim Parrow, and David Walker. A Calculus of Mobile Processes, I. *Information and Computation*, 100(1):1–40, 1992.

[PRS+16] Srinivas Pinisetty, Partha S. Roop, Steven Smyth, Stavros Tripakis, and Reinhard von Hanxleden. Runtime enforcement of reactive systems using synchronous enforcers. *CoRR*, abs/1612.05030, 2016.

[PRS+17] Srinivas Pinisetty, Partha S. Roop, Steven Smyth, Nathan Allen, Stavros Tripakis, and Reinhard Von Hanxleden. Runtime enforcement of cyber-physical systems. *ACM Trans. Embed. Comput. Syst.*, 16(5s):178:1–178:25, September 2017.

[RH97] J. Rathke and M. Hennessy. Local model checking for value-passing processes (extended abstract). In *TACS*, pages 250–266. Springer, 1997.

[Sak09] Jacques Sakarovitch. *Elements of Automata Theory*. Cambridge University Press, New York, NY, USA, 2009.

[San11] Davide Sangiorgi. *Introduction to Bisimulation and Coinduction*. Cambridge University Press, New York, NY, USA, 2011.

[Sch00] Fred B Schneider. Enforceable security policies. *ACM Transactions on Information and System Security (TISSEC)*, 3(1):30–50, 2000.

[vHRF17] A. C. van Hulst, M. A. Reniers, and W. J. Fokkink. Maximally permissive controlled system synthesis for non-determinism and modal logic. *Discrete Event Dynamic Systems*, 27(1):109–142, 2017.