On bidirectional enforcement ⋆

Luca Aceto[1,2], Ian Cassar[2,3], Adrian Francalanza[3], and Anna Ingólfsdóttir[2]

[1] Gran Sasso Science Institute, L'Aquila, Italy
[2] ICE-TCS, Department of Computer Science, Reykjavík University, Iceland
[3] Department of Computer Science, University of Malta, Malta

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. In particular, we explore bidirectional enforcement instrumentation to enforce properties about 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 and optimal monitors from sHML formulas, and show that this logic is enforceable via a specific type of bidirectional enforcement monitors called action disabling monitors.

1 Introduction

Our dependence on software systems is raising the demand for ensuring their correctness. Verifying systems is, however, becoming harder due to their ever increasing complexity. For instance, systems may now opt to collect data from multiple inputs before providing the required response, or they may supply multiple outputs in response to a single input. Moreover, most systems nowadays are made from concurrent interacting entities (such as processes, threads or actors) that may lead them to exhibit non-deterministic behaviour.

Automatic software verification techniques are gaining popularity as a way to ensure system correctness [21, 22, 28, 25]. In particular, *runtime enforcement* (RE) [31, 4, 5] is a dynamic verification technique that uses *monitors* to analyse the runtime behaviour of a system-under-scrutiny (SuS) and transform it to conform to some correctness *specification*. The seminal work in RE [31, 32, 10, 37, 12, 27] models the behaviour of the SuS as a *trace* of *arbitrary* actions, *e.g.*, $\alpha_1.\alpha_2, \ldots$, and assumes that the monitor can either *suppress* or *replace* any trace

⋆ This work was partly supported by the projects "TheoFoMon: Theoretical Foundations for Monitorability" (nr.163406-051) and "Developing Theoretical Foundations for Runtime Enforcement" (nr.184776-051) of the Icelandic Research Fund, by the Italian MIUR project PRIN 2017FTXR7S IT MATTERS "Methods and Tools for Trustworthy Smart Systems", by the EU H2020 RISE programme under the Marie Skłodowska-Curie grant agreement nr. 778233, and by the Endeavour Scholarship Scheme (Malta), part-financed by the European Social Fund (ESF) - Operational Programme II 2014-2020.

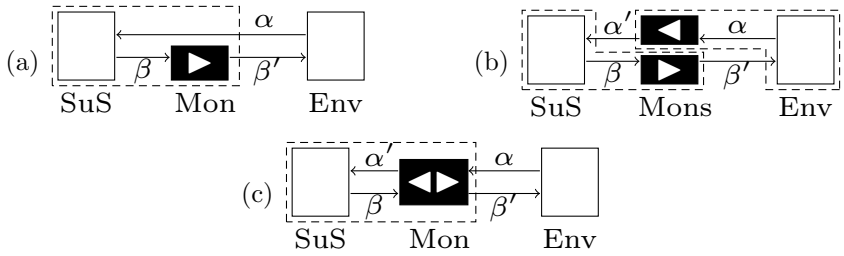

Fig. 1. Enforcement instrumentation setups.

action, and when possible *insert* additional actions into the trace. This work has been effectively used to implement *unidirectional* enforcement approaches [28, 16, 4, 8] that monitor the trace of *outputs* produced 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) and is tasked with transforming the output behaviour of the SuS to ensure its correctness. For instance, if the SuS executes an erroneous output β, this gets intercepted by the monitor and modified accordingly into β' to stop the error from propagating to the surrounding environment.

Despite its merits, unidirectional enforcement lacks the power to enforce properties that require modifying the *input* behaviour of the SuS. These properties are arguably harder to enforce as, unlike outputs, inputs are instigated by the environment and not the SuS itself, and hence the SuS possesses only partial control over them. Although the SuS can control when certain inputs can be supplied (*e.g.*, by opening and reading from a file, communication port, *etc.*), at runtime the environment determines what payload will be provided.

Instead of dealing with the complexities of directly transforming inputs, several work [14, 26, 15] has been conducted to circumvent this issue by using an extra monitor. As shown in Figure 1 (b), this monitor is attached to the environment to scrutinise its outputs before they are forwarded as inputs to the SuS. While this approach has been shown to be viable, it assumes that a monitor can actually be attached to the environment.

By contrast, Figure 1 (c) presents a less explored *bi-directional enforcement* setup. Unlike in (b), the monitor now scrutinises the entire behaviour of the SuS without instrumenting the environment (which is often inaccessible). Adopting this setup is, however, not an easy task, particularly since the SuS enjoys limited control over its inputs. In fact, since properties are violated when the system performs an invalid action, it may be too late for the monitor to prevent the violation if it allows the SuS to input a value that then turns out to be invalid. Hence, it is questionable whether the monitor can intercept and suppress (or replace) an invalid input that has already been provided by the environment. The monitor must therefore exploit the system's limited control over its inputs in the best way possible to ensure that the resulting composite system can perform

all the specified valid inputs, while preventing it from performing invalid ones. The development of a viable bidirectional setting, requires adopting a different enforcement approach than the ones conventionally used for enforcing output behaviour in a unidirectional one.

In this paper we thus explore how the existing monitor transformations can be repurposed to work with input actions. Since inputs and outputs must be handled differently by the monitor, we find it essential to distinguish between the monitor's transformations (*i.e.,* suppressions, insertions and replacements) and their resulting effect on the visible behaviour of the composite system. As a result, we develop a bidirectional enforcement instrumentation model that can enable and disable specific actions from the resulting composite system. This model permits us to study the enforceability of properties defined via the safety subset sHML of the well studied branching time logic μHML [36, 7, 30] (which is a reformulation of the modal μ-calculus [29]). A crucial aspect of our investigation is the *synthesis* function that maps the *declarative* sHML formulas to *algorithmic* monitors that enforce properties concerning both the input and output behaviour of the SuS.

Our results and contributions are:

- (*i*) A general instrumentation framework for bidirectional enforcement (Figure 4) that is parametrisable by any system behaviour that can be modelled as a labelled transition system. The novelty of this framework lies in how its effect on the visible behaviour of the resulting composite system differs according to whether the transformed action is an input or an output.
- (*ii*) A definition dictating what it means for a monitor to *adequately enforce* a logical formula in a bidirectional setting (Definitions 2 and 6), and a novel notion of monitor *optimality* (Definition 7). The latter assesses the level of intrusiveness of the monitor and guides in the search for the least intrusive one. 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 synthesis function (Definition 10) that maps sHML formulas into action disabling monitors. We evaluate the quality of this mapping by proving that the synthesised monitors are correct and optimal (Theorems 1 and 2).

Structure of the paper. The rest of the paper is structured as follows. Section 2 provides the necessary preliminary material including how we model systems as labelled transition systems and properties via the chosen logic. In Section 3 we then present the operational model for bidirectional enforcement, and in Section 4 we formalise the interdependent notions of correct and optimal enforcement. These act as a foundation for developing the synthesis function in Section 5. Section 6 overviews related work, and Section 7 concludes.

2 Preliminaries

The Model: We assume systems that are described as *labelled transition systems* (LTSs). An LTS consists of a triple $\langle \textsc{Sys}, \textsc{Act} \cup \{\tau\}, \rightarrow \rangle$ that defines: a set

Syntax

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

Semantics

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

Fig. 2. The syntax and semantics for sHML.

of *system states*, $s, r, q \in \text{Sys}$, a set of *visible actions*, $\alpha, \beta \in \text{Act}$, along with a distinguished invisible action $\tau \notin \text{Act}$ (where $\mu \in \text{Act} \cup \{\tau\}$), and a *transition* relation, $\longrightarrow \subseteq (\text{Sys} \times (\text{Act} \cup \{\tau\}) \times \text{Sys})$. We assume a countably infinite set of communication ports $\mathsf{a}, \mathsf{b}, \mathsf{c} \in \text{Port}$, a set of values $v, w \in \text{Val}$, and partition the set of actions into *inputs* $\mathsf{a}?v \in \text{iAct}$, and *outputs* $\mathsf{a}!v \in \text{oAct}$ where $\text{iAct} \cup \text{oAct} = \text{Act}$. 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 [23] to concisely describe LTSs and assume the classic notion of *strong bisimilarity* [23, 38], $s \sim r$, to denote system equivalence.

Traces $t, u \in \text{Act}^*$ range over (finite) sequences of visible actions. We write $s \stackrel{t}{\Rightarrow} r$ to denote a sequence of *weak* transitions $s \stackrel{\alpha_1}{\Longrightarrow} \ldots \stackrel{\alpha_n}{\Longrightarrow} r$ when $t = \alpha_1 \ldots \alpha_n$ for some $n \geq 0$; when $t = \varepsilon$, $s \stackrel{\varepsilon}{\Rightarrow} r$ means $s \xrightarrow{\tau}{}^* r$. Additionally, we represent system runs as *explicit traces* that include τ-actions, $t_\tau, u_\tau \in (\text{Act} \cup \{\tau\})^*$ and abuse notation by writing $s \xrightarrow{\mu_1 \ldots \mu_n} r$ to denote a sequence of strong transitions $s \xrightarrow{\mu_1} \ldots \xrightarrow{\mu_n} r$. Occasionally, we also use the function $\mathsf{sys}(t_\tau)$ to produce a system that executes the sequence of actions defined in the system run t_τ. For instance, $\mathsf{sys}(\mathsf{a}?3.\tau.\mathsf{a}!5)$ produces the process $\mathsf{a}?x.\tau.\mathsf{a}!5.\mathsf{nil}$.

The Logic: We consider the safety subset sHML based on a variation on value passing μHML [36, 24] that uses *symbolic actions* of the form $\{p, c\}$ defining an action pattern p and a condition c. Symbolic actions abstract over visible actions using *data variables* $x, y, z \in \text{DVar}$ that occur free in the condition c or as binders in the pattern p denoted as (x). We use function $\mathbf{bv}(p)$ to denote the set of binders in p, and $\mathbf{fv}(c)$ to represent the set of free variables referenced in condition c. 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. We assume a (partial) *matching function* for patterns $\mathsf{mtch}(p, \alpha)$ that (when successful) returns a substitution σ mapping variables in p to the corresponding values in α, so that if we instantiate every variable x in p with $\sigma(x)$ we obtain α. The *filtering condition*, c, may refer to variables bound in p and it is evaluated with respect to the substitutions returned by successful matches, written as $c\sigma \Downarrow b$ where $b \in \{\mathsf{true}, \mathsf{false}\}$.

A symbolic action $\{p, c\}$ is *closed* whenever the free variables in c only refer to the variables that are bound in p, *i.e.,* $\mathbf{fv}(c) \subseteq \mathbf{bv}(p)$; it denotes the *set* of actions $[\![\{p, c\}]\!] \stackrel{\text{def}}{=} \{\ \alpha \mid \exists \sigma \cdot \mathsf{mtch}(p, \alpha) = \sigma \ \textit{and}\ c\sigma \Downarrow \mathsf{true}\ \}$ and allows more adequate reasoning about LTSs with infinitely many actions (*e.g.,* inputs or outputs carrying data from infinite domains). As our systems have *no control* over the data values supplied in its inputs, the condition c of an input symbolic action, $\{(x)?(y), c\}$, may not restrict the values of binder y, *i.e.,* $y \notin \mathbf{fv}(c)$. Put differently, for a closed input symbolic action $\{(x)?(y), c\}$, if σ and σ' are substitutions that agree on x then $c\sigma \Downarrow \mathsf{true}$ iff $c\sigma' \Downarrow \mathsf{true}$. By contrast, as the system produces the output values itself, a closed output symbolic action, $\{(x)!(y), c\}$, may also describe *specific* output values by restricting the possible values of y in 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.,*$\{(x)!(y), x = \mathsf{a} \wedge y = 3\}$ and $\{(x)?(y), x = \mathsf{a}\}$ become $\{\mathsf{a}!3, \mathsf{true}\}$ and $\{\mathsf{a}?(y), \mathsf{true}\}$; we also elide the condition when it is true, and just write $\{\mathsf{a}!3\}$ and $\{\mathsf{a}?(y)\}$.

Figure 2 presents the sHML syntax that assumes a countably infinite set of logical variables $X, Y \in \text{LVar}$. It provides standard logical constructs such as truth, falsehood and conjunctions: where $\bigwedge_{i \in I} \varphi_i$ describes a *compound* conjunction, $\varphi_1 \wedge \ldots \wedge \varphi_n$, where $I = \{1, .., n\}$ is a finite set of indices. It allows for defining recursive properties using greatest fixpoints, $\mathsf{max}\, X.\varphi$, which bind free occurrences of X in φ. The logic also provides the symbolic *necessity* (universal) modal operator, $[\{p, c\}]\varphi$ where the binders $\mathbf{bv}(p)$ bind the free data variables in c and φ. To improve presentation, 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. A formula $[\{p, c\}]\varphi$ is satisfied by any system that either *cannot* perform an action α that matches p and satisfies condition c, or if it can perform such an α with a matching substitution σ, then its derivative state must satisfy the continuation $\varphi\sigma$. We finally assume that fixpoint variables, X, are guarded by a modal necessity (*e.g.,* $\mathsf{max}\, X.([\alpha]\mathsf{ff} \wedge X)$ is invalid, unlike $\mathsf{max}\, X.([\beta]\mathsf{ff} \wedge [\alpha]X)$ in which X is guarded by $[\alpha]$).

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, $\rho \in (\text{LVar} \to \mathcal{P}(\text{Sys}))$, 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 $Y \neq X$. 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 $[\![\varphi]\!] \neq \emptyset$.

Example 1. Consider a property stating that for *every* input request that is made on a specific port, the server should not input another request in succession. It may, however, output a *single* answer on the same port in response, and then log the serviced request by outputting a notification on a dedicated port b. Due to the special status of port b, this property does not apply to requests that are

input from this port. Using our logic we can formalise this property as φ_1.

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

This formula defines an invariant property $\mathsf{max}\, X.(..)$ and uses binder (x) to bind the port on which the request is input, and binders (y_1), (y_2) and (y_3) to bind the input and output payloads. The values bound to y_1 and y_2 are later referenced in condition $y_3 = (\mathsf{log}, y_1, y_2)$. The formula is violated by two consecutive inputs on the same port x, and when a request is serviced with multiple answers. An answer output followed by a log action on port b is normal, and thus the formula recurses.
Now consider systems $s_{\mathbf{a}}$, $s_{\mathbf{b}}$ and $s_{\mathbf{c}}$ (where $s_{\mathbf{cls}} \stackrel{\text{def}}{=} (\mathsf{b}?z.\mathsf{if}\ z{=}\mathsf{cls}\ \mathsf{then}\ \mathsf{nil}\ \mathsf{else}\ X)$).

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

$s_{\mathbf{a}}$ implements a request-response server that repeatedly inputs values (for some domain VAL) on port a, $\mathsf{a}?x$, for which it internally computes an answer and assigns it to the data variable y, $y := \mathsf{ans}(x)$. It then outputs the answer on port a in response to each request, $\mathsf{a}!y$, and finally logs the serviced request by outputting the triple (log, x, y) on port b, $\mathsf{b}!(\mathsf{log}, x, y)$. It terminates whenever it inputs 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. In fact, $s_{\mathbf{c}}$ is initialised in a suspended state that requires an extra (unused) input, $\mathsf{a}?y$, to start working as $s_{\mathbf{a}}$, whereas $s_{\mathbf{b}}$ may occasionally provide a redundant (underlined) answer prior to logging the serviced request. Using the semantics of Figure 2, one can check that $s_{\mathbf{a}} \in [\![\varphi_1]\!]$ whereas $s_{\mathbf{c}} \notin [\![\varphi_1]\!]$ since $s_{\mathbf{c}} \xRightarrow{\mathsf{a}?v_1.\mathsf{a}?v_2}$, and $s_{\mathbf{b}} \notin [\![\varphi_1]\!]$ since $s_{\mathbf{b}} \xRightarrow{\mathsf{a}?v_1.\mathsf{a}!\mathsf{ans}(v_1).\mathsf{a}!\mathsf{ans}(v_1)}$ (for some input values v_1 and v_2). □

As we aim to use our logic in conjunction to monitoring, it is sometimes useful to know the constraint that a system must satisfy after performing a number of steps. For instance, if $\mathsf{a}?v.\mathsf{a}!w.s$ satisfies formula $[\{\mathsf{a}?(\_)\}][\{\mathsf{a}!(\_)\}]\psi$ then its derivative s must also satisfy formula ψ after it performs actions $\mathsf{a}?v$ and $\mathsf{a}!w$. We thus define the function *after* to denote how sHML formulas evolve in reaction to an action μ.

Definition 1. *We define the function after*:(sHML×ACT∪{τ})→sHML *as:*

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

Remark 1. The function *after* is well-defined since, due to our assumption that formulas are guarded, $\varphi'\{^{\varphi}/_{X}\}$ has fewer top level occurrences of greatest fixpoint operators than $\mathsf{max}\, X.\varphi'$.

When applied to a fixpoint, the function unfolds the formula and reapplies itself to the unfolded equivalent. Similarly, in the case of conjunctions it recurses for each individual branch. It returns formula $\psi\sigma$ when α matches successfully the symbolic action of a modal operator preceding ψ, where σ is created by the successful match. However, when unsuccessful it returns tt to signify a trivial satisfaction. Truth and falsehood are definitive and thus do not change. Silent τ actions do not affect the formula φ as well. We justify our definition of the *after* function vis-a-vis the semantics of Figure 2 via Proposition 1.

Proposition 1. *For every system state s, formula φ and action α, if $s \in [\![\varphi]\!]$ and $s \stackrel{\alpha}{\Longrightarrow} s'$ then $s' \in [\![after(\varphi, \alpha)]\!]$.* □

A proof for this theorem is given in Appendix A. We abuse notation and lift the *after* function to (explicit) traces in the obvious way, *i.e.,* $after(\varphi, t_\tau)$ is equal to $after(after(\varphi, \mu), u_\tau)$ when $t_\tau = \mu u_\tau$ and to φ when $t_\tau = \varepsilon$.

Example 2. When applied to φ_1 of Example 1 in relation to $t^1_\tau = \mathsf{a}?v_1.\tau.\mathsf{a}!\mathsf{ans}(v_1)$, formula $after(\varphi_1, t^1_\tau)$ denotes $([\{\mathsf{a}!(\_)\}]\mathsf{ff} \wedge [\{\mathsf{b}!(y_3), y_3{=}(\mathsf{log}, v_1, \mathsf{ans}(v_1))\}]\varphi_1)$ and when applied to trace $\mathsf{a}?v_1.\mathsf{a}?v_2$ it evolves into ff. □

3 A bi-directional enforcement model

In bidirectional enforcement we seek to transform the entire (input and output) behaviour of the SuS; this contrasts with unidirectional approaches that only modify its output traces. When changing the behaviour of a system (and not just a single trace) it makes sense to distinguish 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. Similarly, a visible action is *enabled* when the composite system can execute it unlike the SuS. Actions are *adapted* when either the payload of an action in the SuS differs from that of the composite system, or when the action is rerouted through a different port. However, since inputs and outputs are fundamentally different, the type of the action itself cannot be adapted, that is, an input cannot become an output, and vice versa.

Implementing action enabling, disabling and adaptation differs according to whether the action is an input or an output. In Figure 3 we illustrate our proposed instrumentation setup that implements them by using the monitor's existing transformations. As the instrumented monitor can only fully control the actions instigated by the SuS, enforcing its outputs is more intuitive. In fact, (a), (b) and (c) in Figure 3 respectively show that to disable an output it suffices to suppress it, to adapt it the monitor may replace the output data and forward it

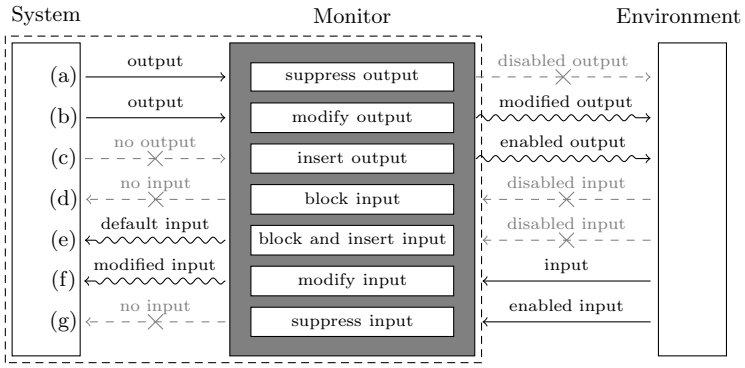

Fig. 3. Our bi-directional enforcement setup.

to the environment on a (potentially) different port, while to enable an output it suffices to produce the required data via an insertion transformation.

Working with inputs is less straightforward. In our setup, we propose that to *disable* an input, item (*d*) in Figure 3, it suffices that the monitor conceals the system's input port so to prevent the environment from forwarding a value as input to the system. As this technique may block the system's execution from progressing, the instrumented monitor may additionally *insert* a default input that unblocks the system[4], item (*e*) in Figure 3. Input *adaptation*, item (*f*) in Figure 3, is also attained via a *replacement* transformation, but unlike in the case of outputs, it is applied in the reverse direction. In fact, it modifies the data received by the monitor over some port, and forwards it to the SuS over the same (or a different) port. Inputs can also be *enabled*, item (*g*), by allowing the monitor to accept the required input on a desired port and then *suppress* it. To an external viewer, the input has been made, yet discarded internally.

More concretely, in Figure 4 we formalise the novel bidirectional instrumentation setup of Figure 3 in terms of the (symbolic) transducers $m, n \in \textsc{Trn}$ that were priorly introduced in [4]. Transducers are monitors that define *symbolic transformation triples*, $\{p, c, p'\}$, consisting of an action *pattern* p, *condition* c, and a *transformation pattern* p'. 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 pattern specifies the kind of transformation that should be applied. The symbolic transformation patterns p and p' are extended versions of those definable in symbolic actions. In addition to denoting inputs and outputs, these extended patterns may also specify $\bullet$ and use it as follows. When $p = \bullet$, the transformation pattern represents a point where the monitor can act independently from the system to insert the action specified by p', but when $p' = \bullet$, it represents the suppression of the action specified by p. The syntax of our transducers assumes a well-formedness constraint where, for every $\{p, c, p'\}.m$, either p or p' is $\bullet$ (but not both), or else both patterns are

[4] The added benefits of this mechanism are further discussed in the forthcoming sections.

Syntax

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

Dynamics

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

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

Instrumentation

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

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

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

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

where id is shorthand for $\mathsf{rec}\, Y.\{(x)!(y), \mathsf{true}, x!y\}.Y + \{(x)?(y), \mathsf{true}, x?y\}.Y$ and $m \xrightarrow{\ \gamma\ }\!\!\!\!\!\!\not\qquad$ means $\nexists \gamma', n \cdot m \xrightarrow{\ \gamma\blacktriangleright\gamma'\ } n$.

Fig. 4. A bi-directional instrumentation model for enforcement monitors.

of the *same* type *i.e.,* both are input or output patterns. For instance, symbolic transformations $\{\bullet, \mathsf{true}, \mathsf{a}?v\}$ and $\{(x)!(y), \mathsf{true}, \bullet\}$ are valid since only one of their patterns is $\bullet$, and so is $\{(x)!(y), \mathsf{true}, \mathsf{a}!v\}$ since both patterns are output patterns. This constraint ensures that input actions cannot be adapted into outputs and vice versa. It is crucial since inputs and outputs are instigated by different entities, namely, the environment and the SuS respectively.

The monitor transition rules in Figure 4 assume closed terms, *i.e.,* every *transformation-prefix transducer* of the form $\{p, c, p'\}.m$ must obey the constraint $\big(\mathbf{fv}(c) \cup \mathbf{fv}(p') \cup \mathbf{fv}(m)\big) \subseteq \mathbf{bv}(p) \cup \mathcal{V}$, where $\mathcal{V}$ is the set of variables that are bound by priorly defined transformation prefixes. 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 visible action γ into γ' and reduces to state n. In this sense, the transducer action $\alpha\blacktriangleright\beta$ represents the *replacing* of α by β, and $\alpha\blacktriangleright\alpha$ denotes the *identity* transformation. Cases $\alpha\blacktriangleright\bullet$ and $\bullet\blacktriangleright\alpha$ respectively encode the *suppression* and *insertion* transformations of action α. The key transition rule in Figure 4 is

eTrn. It states that the transformation-prefix transducer $\{p, c, p'\}.m$ transforms action γ into a (potentially) different action γ' and reduces to state $m\sigma$, whenever γ matches pattern p, *i.e.,* $\mathsf{mtch}(p, \gamma){=}\sigma$, and satisfies condition c, *i.e.,* $c\sigma \Downarrow \mathsf{true}$. Action γ' results from instantiating the free variables in p' with the corresponding values mapped by σ, *i.e.,* $\gamma'{=}p\sigma$. The remaining rules for recursion (eRec) and selection (eSel) are standard. We encode the identity monitor, id, as a recursive monitor defining identity transformations that match every action.

The primary contribution of this enforcement model lies in the new *instrumentation relation* of Figure 4. This relation links the behaviour of the SuS s with the transformations of a monitor m. The term $m[s]$ thus denotes the resulting *monitored system* whose behaviour is defined in terms of $\text{Act} \cup \{\tau\}$. Concretely, rule biTrnO states that if the SuS s transitions with an output $\mathsf{b}!w$ to s' and the transducer m can *replace* it with $\mathsf{a}!v$ and reduce 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 can be *suppressed* into $\bullet$, the instrumentation withholds this output and so the composite system transitions silently over τ thereby *disabling* it. Dually, rule biEnO *enables* and augments the composite system $m[s]$ with an output $\mathsf{a}!v$ whenever m is able to *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 [17, 19, 18, 1], and accounts for underspecification of transformations. We, however, restrict defaulting (termination) exclusively to output actions performed by the SuS. A monitor therefore 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.,* $m \xrightarrow{\mathsf{a}?v \blacktriangleright \gamma}\!\!\!\!\!\!/\;\;$, is by default considered as being invalid and blocked. This technique is thus used to implement item (d) of Figure 3. To avoid disabling valid inputs unnecessarily, the monitor must explicitly define symbolic transformations that specify *all* the valid inputs of the SuS. For instance, the symbolic transformation $\{\mathsf{a}?(x), \mathsf{true}, \mathsf{a}?x\}$ allows values to be input on port a only, while $\{(y)?(\_), y{\neq}\mathsf{b}, \bullet\}$ allows inputs on any port except b; any other input is invalid and thus blocked. By including such symbolic transformations, rules biTrnI and biEnI can be applied.

With rule biTrnI any value v that is input on some port a by the instrumented system is *adapted* into a (potentially) different value w and forwarded to the SuS over port b, provided the SuS is willing to accept that input. As far as the environment is concerned, the SuS accepted the input provided by the environment on port a. Similarly, rule biEnI *enables* an input on a port a by allowing the composite system to accept a value v which is then suppressed by the monitor and concealed from the SuS. Although unspecified inputs on a port a are implicitly *disabled* (since the monitor cannot react to them, *i.e.,* $m \xrightarrow{\mathsf{a}?v \blacktriangleright \gamma}\!\!\!\!\!\!/\;\;$), rule biDisI prevents the monitor from blocking systems that require the blocked input in order to progress. Specifically, this rule allows the monitor to generate a

default input value v and forward it to the SuS on a port a, thereby unblocking it by allowing the composite system to silently move on to the next state. Hence, rules BIDISI, BITRNI and BIENI respectively implement items (e), (f) and (g) of Figure 3. Finally, rule BIASY allows the SuS s to internally transition with a silent action τ to some state s' independent of m.

In our examples, we find it convenient to elide the transformation pattern p' in a transducer $\{p, c, p'\}.m$ and write $\{p, c\}.m$ when all the binding occurrences (x) of p are defined as free occurrences x in p', thus denoting an identity transformation. Similarly, we elide c whenever $c{=}\mathsf{true}$.

Example 3. Consider the following action disabling transducer $m_{\mathbf{d}}$:

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

It is a recursive transducer, $\mathsf{rec}\, Y.\_$, that repeatedly disables every output performed by the system via the branch $\{(x)!(y), \bullet\}.Y$. Moreover, by only defining the input branch $\{\mathsf{b}?(\_)\}.Y$ it also restricts the composite system by allowing it to only input values from port b. Concretely, inputs from other ports are disabled since none of the instrumentation rules in Figure 4 can be applied to allow the composite system to transition over these input actions. For instance, when instrumented with $s_{\mathbf{c}}$ from Example 1, $m_{\mathbf{d}}$ blocks its initial input so that for every action α, $m_{\mathbf{d}}[s_{\mathbf{c}}] \stackrel{\alpha}{\not\longrightarrow}$. For $s_{\mathbf{b}}$, the composite system $m_{\mathbf{d}}[s_{\mathbf{b}}]$ can only input termination requests on port b, *i.e.*, $m_{\mathbf{d}}[s_{\mathbf{b}}] \xrightarrow{\mathsf{b?cls}} m_{\mathbf{d}}[\mathsf{nil}]$.

Now consider the more elaborate transducer $m_{\mathbf{dt}}$.

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

On the one hand, by defining branch $\{\mathsf{b}?(\_)\}.\mathsf{id}$, monitor $m_{\mathbf{dt}}$ allows the SuS to immediately input a termination request on port b and defaults to id. On the other hand, the branch prefixed by $\{(x)?(y_1), x{\neq}\mathsf{b}\}$ permits the system to input the first request via any port $x \neq \mathsf{b}$. It then blocks subsequent inputs on the same port x (without deterring inputs on other ports) by defining the input branch $\{(x_1)?(\_), x_1 \neq x\}.\mathsf{id}$. In conjunction to this input branch, $m_{\mathbf{dt}}$ defines $\{x!(y_2)\}.m'_{\mathbf{dt}}$ to allow the SuS to perform an output 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 every *redundant* response that is output following the first one. However, as it also defines branches $\{\mathsf{b}!(y_3), y_3{=}(\mathsf{log}, y_1, y_2)\}.X$ and $\{(\_)?(\_)\}.\mathsf{id}$, it refrains from modifying log events and blocking further inputs that occur immediately after the first response.

When instrumented with $s_{\mathbf{c}}$ from Example 1, $m_{\mathbf{dt}}$ allows the composite system to perform the first input but then blocks the second one which means that it can only input termination requests, *i.e.*, $m_{\mathbf{dt}}[s_{\mathbf{c}}] \xrightarrow{\mathsf{a}?v} \cdot \xrightarrow{\mathsf{b?cls}} \mathsf{id}[\mathsf{nil}]$. It also disables the first redundant response of $s_{\mathbf{b}}$, and as a result, it reduces to $m_{\mathbf{d}}$ which carries on to suppress every subsequent output (even log actions) and blocks every port except b, *i.e.*, $m_{\mathbf{dt}}[s_{\mathbf{b}}] \xrightarrow{\mathsf{a}?v} \cdot \stackrel{\mathsf{a}!w}{\Longrightarrow} \cdot \xrightarrow{\tau} m_{\mathbf{d}}[\mathsf{b}!(\mathsf{log}, v, w).s_{\mathbf{b}}] \xrightarrow{\tau}$

$m_{\mathbf{d}}[s_{\mathbf{b}}] \xrightarrow{\mathsf{a}?v}\!\!\!\!\!\!/\;\;$ (for every port a where $\mathsf{a}{\neq}\mathsf{b}$ and value v). Moreover, it resorts to defaulting to handle unspecified outputs *e.g.,* for system $\mathsf{b}!(\mathsf{log},v,w).s_{\mathbf{a}}$ although $m_{\mathbf{dt}} \xrightarrow{\mathsf{b}!(\mathsf{log},v,w)}\!\!\!\!\!\!/\;\;$, using rule IDEF the composite system can still perform the output, *i.e.,* $m_{\mathbf{dt}}[\mathsf{b}!(\mathsf{log},v,w).s_{\mathbf{a}}] \xrightarrow{\mathsf{b}!(\mathsf{log},v,w)} \mathsf{id}[s_{\mathbf{a}}]$.

Monitor $m_{\mathbf{det}}$ (below) is similar to $m_{\mathbf{dt}}$ but instead of reducing to $m_{\mathbf{d}}$ after suppressing the first redundant response, it employs a loop of suppressions (underlined in $m''_{\mathbf{det}}$) that only disables further responses until a log or termination input is made.

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

Hence, contrary to $m_{\mathbf{dt}}$, after detecting and disabling the redundant response of $s_{\mathbf{b}}$, monitor $m_{\mathbf{det}}$ only attempts to disable further responses via the suppression loop of $m''_{\mathbf{det}}$, and thus allows the subsequent log action to go through, as follows:

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

It 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 silently, *i.e.,*

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

where $s'_{\mathbf{a}} \stackrel{\text{def}}{=} y := \mathsf{ans}(v_{\text{def}}).\mathsf{a}!y.\mathsf{b}!(\mathsf{log}, v_{\text{def}}, y).s_{\mathbf{a}}$. □

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

Example 4. Consider now transducers $m_{\mathbf{e}}$ and $m_{\mathbf{a}}$ below:

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

Once instrumented with a system, $m_{\mathbf{e}}$ first uses a suppression transformation to enable an input that may come from 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 $r \in \{s_{\mathbf{b}}, s_{\mathbf{c}}\}$ from Example 1, the execution of the composite system can only start as follows:

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

for some values v and $w = \mathsf{ans}(v)$. By contrast, $m_{\mathbf{a}}$ uses action adaptation to redirect the inputs and outputs of the SuS through port b. Specifically, the monitor allows the composite system to input values only from port b and forwards them to the SuS on its input port a. Similarly, outputs from the SuS on port a are rerouted to port b. As a result, the composite system is *only* able to interact on port b. For instance, $m_{\mathbf{a}}[s_{\mathbf{c}}] \xrightarrow{\mathsf{b}?v_1} m_{\mathbf{a}}[s_{\mathbf{a}}] \xRightarrow{\mathsf{b}?v_2.\mathsf{b}!w_2.\mathsf{b}!(\mathsf{log},v_2,w_2)} m_{\mathbf{a}}[s_{\mathbf{a}}]$ 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 and Optimality

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 (of Section 3 in our case) to enforce the specified behaviour on the other.

Definition 2 (Enforceability). *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. □

Definition 2 relies on the meaning of "*m adequately enforces φ*". Several meanings can be given, however, it is reasonable to expect that an adequate definition should be applicable to *any* system that can be instrumented with monitor m. In [4] we stipulated that one should at least expect soundness, *i.e.,* if the property of interest φ is *satisfiable*, *i.e.,* $[\![\varphi]\!] \neq \emptyset$, then the composite system, $m[s]$, should satisfy φ for *every* system state s.

Definition 3 (Sound Enforcement). *Monitor m* soundly enforces *a satisfiable formula φ, denoted as $\mathsf{senf}(m, \varphi)$, iff $m[s] \in [\![\varphi]\!]$, for every state $s \in$ Sys.* □

Example 5. In general, showing that a monitor soundly enforces a formula requires showing this for *every* possible system. However, we give an intuition based on systems $s_{\mathbf{a}}$, $s_{\mathbf{b}}$, $s_{\mathbf{c}}$ and formula φ_1 (restated below) from Example 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 Example 3, and also that $s_{\mathbf{a}} \in [\![\varphi_1]\!]$ (hence $[\![\varphi_1]\!] \neq \emptyset$) and $s_{\mathbf{b}}, s_{\mathbf{c}} \notin [\![\varphi_1]\!]$. When assessing their soundness in relation to φ_1, we have that:

– $m_{\mathbf{e}}$ is *unsound* as it allows invalid behaviour such as $m_{\mathbf{e}}[s_{\mathbf{b}}] \stackrel{t^1_{\mathbf{e}}}{\Longrightarrow} \mathsf{id}[s_{\mathbf{b}}]$ where $t^1_{\mathbf{e}} \stackrel{\text{def}}{=} \mathsf{c}?v_1.\mathsf{c}!\mathsf{ans}(v_1).\mathsf{b}!(\mathsf{log}, v_1, \mathsf{ans}(v_1)).\mathsf{a}?v_2.\underline{\mathsf{a}!w_2.\mathsf{a}!w_2}$. This shows that the composite system $m_{\mathbf{e}}[s_{\mathbf{b}}]$ can still make two consecutive output replies (underlined), and so $m_{\mathbf{e}}[s_{\mathbf{b}}] \notin [\![\varphi_1]\!]$. Similarly, $m_{\mathbf{e}}[s_{\mathbf{c}}] \notin [\![\varphi_1]\!]$ since the $m_{\mathbf{e}}[s_{\mathbf{c}}]$ executes the erroneous trace $\mathsf{c}?v_1.\mathsf{c}!\mathsf{ans}(v_1).\mathsf{b}!(\mathsf{log}, v_1, \mathsf{ans}(v_1)).\underline{\mathsf{a}?v_2.\mathsf{a}?v_3}$. This demonstrates that $m_{\mathbf{e}}[s_{\mathbf{c}}]$ can still input two consecutive requests on port a (underlined). Either one of these counter examples suffice to prove that $\neg\mathsf{senf}(m_{\mathbf{e}}, \varphi_1)$.
– $m_{\mathbf{a}}$ is *sound* because once instrumented, the resulting composite system is adapted to only interact on port b, and so its actions are not of concern to φ_1. As $m_{\mathbf{a}}$ applies this enforcement strategy to any SuS, we can safely conclude that $\mathsf{senf}(m_{\mathbf{a}}, \varphi_1)$, in fact $m_{\mathbf{a}}[s_{\mathbf{a}}], m_{\mathbf{a}}[s_{\mathbf{b}}], m_{\mathbf{a}}[s_{\mathbf{c}}] \in [\![\varphi_1]\!]$. Monitors $m_{\mathbf{d}}$, $m_{\mathbf{dt}}$ and $m_{\mathbf{det}}$ are also *sound*. Intuitively, $m_{\mathbf{d}}$ prevents the violation of φ_1 by blocking all input ports except b, whereas $m_{\mathbf{dt}}$ and $m_{\mathbf{det}}$ achieve the same goal by disabling the invalid consecutive requests and answers that occur on a specific port (except b). □

Although sound enforcement is a fundamental aspect of enforceability, in [4] we had found it to be relatively weak to solely define adequate enforcement in Definition 2, namely, because it does not regulate the *extent* of the applied enforcement. More specifically, consider $m_{\mathbf{d}}$ from Example 3. Although it successfully prevents the violating behaviour of $s_{\mathbf{b}}$ and $s_{\mathbf{c}}$, 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. Hence, in addition to soundness, in [4] we posited that adequate enforcement must also require a degree of *transparency* that safeguards the integrity of well-behaved systems. Transparency thus dictates that, whenever a system s already satisfies the property φ, the assigned monitor m should not alter the behaviour of s.

Definition 4 (Transparent Enforcement). *A monitor m is* transparent *when enforcing a formula φ, written as $\mathsf{tenf}(m, \varphi)$, iff for* all *system states $s \in \text{SYS}$, if $s \in [\![\varphi]\!]$ then $m[s] \sim s$.* □

Example 6. As argued earlier, the counter-example given in relation to $s_{\mathbf{a}}$ suffices to prove that $\neg\mathsf{tenf}(m_{\mathbf{d}}, \varphi_1)$. Monitor $m_{\mathbf{a}}$ from Example 4 also fails to meet this requirement: although $s_{\mathbf{a}}$ satisfies φ_1, we have that $m_{\mathbf{a}}[s_{\mathbf{a}}] \not\sim s_{\mathbf{a}}$ since for any value v and w, $m_{\mathbf{a}}[s_{\mathbf{a}}] \xrightarrow{\mathsf{b}?v} \cdot \xrightarrow{\mathsf{b}!w}$ but $m_{\mathbf{a}}[s_{\mathbf{a}}] \xrightarrow{\mathsf{b}?v} \cdot \not\xrightarrow{\mathsf{b}!w}$. By contrast, monitors $m_{\mathbf{dt}}$ and $m_{\mathbf{det}}$ follow this criterion as 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. The universal quantification over all systems makes it difficult to show that $\mathsf{tenf}(m_{\mathbf{dt}}, \varphi_1)$ and $\mathsf{tenf}(m_{\mathbf{det}}, \varphi_1)$. However, since both monitors do not modify valid systems such as $s_{\mathbf{a}}$, *i.e.*, $s_{\mathbf{a}} \in [\![\varphi_1]\!]$ and $m_{\mathbf{dt}}[s_{\mathbf{a}}] \sim s_{\mathbf{a}} \sim m_{\mathbf{det}}[s_{\mathbf{a}}]$, and only modify invalid ones, such as $s_{\mathbf{b}}$ and $s_{\mathbf{c}}$ (see Example 5), we get a good intuition for why this is the case. □

It turns out that transparency is still a relatively weak constraint as it only restricts the enforcement applied to well-behaved systems, and disregards the extent of enforcement induced upon the erroneous ones. For instance, consider monitor $m_{\mathbf{dt}}$ from Example 3 and system $s_{\mathbf{b}}$ from Example 1. At runtime $s_{\mathbf{b}}$ can exhibit the following invalid behaviour: $s_{\mathbf{b}} \stackrel{t_1}{\Longrightarrow} \mathsf{b!}(\mathsf{log}, v, w).s_{\mathbf{a}}$ where $t_1 \stackrel{\text{def}}{=} \mathsf{a?}v.\mathsf{a!}w.\mathsf{a!}w$. In order to bring the invalid behaviour of $s_{\mathbf{b}}$ (show in t_1) in line with our specification φ_1, it suffices to use some monitor m that only disables *one* of its responses, $\mathsf{a!}w$. After correcting t_1 into $t_1' \stackrel{\text{def}}{=} \mathsf{a?}v.\mathsf{a!}w$, no further modifications are required by m since the SuS reaches a valid point, that is, it reduces into a state that does not violate the rest of the property. In this case, the SuS reduces into $\mathsf{b!}(\mathsf{log}, v, w).s_{\mathbf{a}}$ where $\mathsf{b!}(\mathsf{log}, v, w).s_{\mathbf{a}} \in [\![after(\varphi_1, t_1')]\!]$. However, when instrumented with $m_{\mathbf{dt}}$, this monitor does not only disable the invalid response, *i.e.,* $m_{\mathbf{dt}}[s_{\mathbf{b}}] \stackrel{\mathsf{a?}v.\mathsf{a!}w.}{\Longrightarrow} m_{\mathbf{d}}[\mathsf{b!}(\mathsf{log}, v, w).s_{\mathbf{a}}]$, but keeps on disabling every subsequent action as a result of reducing into $m_{\mathbf{d}}$, $m_{\mathbf{d}}[\mathsf{b!}(\mathsf{log}, v, w).s_{\mathbf{a}}] \stackrel{\tau}{\longrightarrow} m_{\mathbf{d}}[s_{\mathbf{a}}]$.

It thus makes sense that transparency should also start applying whenever an invalid SuS reaches a valid point while instrumented with the monitor. Put differently, if a composite system, $m[s]$ (where $s \notin [\![\varphi]\!]$), reduces to some state $m'[s']$ over a trace t, where s' is in agreement with φ after following t (*i.e.,* $s' \in [\![after(\varphi, t)]\!]$), then the behaviour of $m'[s']$ should be *equivalent* to that of s'. In this paper we therefore introduce the notion of *eventual transparency.*

Definition 5 (Eventual Transparent Enforcement). *A monitor m adheres to eventual transparency when enforcing φ, denoted as $\mathsf{evtenf}(m, \varphi)$, iff for all system states s, s', trace t and monitor m', $m[s] \stackrel{t}{\Rightarrow} m'[s']$ and $s' \in [\![after(\varphi, t)]\!]$ imply that $m'[s'] \sim s'$.* □

Example 7. We have already argued, via the counter example $s_{\mathbf{b}}$, why $m_{\mathbf{dt}}$ does not adhere to eventual transparency, *i.e.,* $\neg\mathsf{evtenf}(m_{\mathbf{dt}}, \varphi_1)$; this is not the case for $m_{\mathbf{det}}$ because $\mathsf{evtenf}(m_{\mathbf{det}}, \varphi_1)$. Although the universal quantification over all systems and traces make it hard to prove this property, we can get a good intuition of why this is the case from $s_{\mathbf{b}}$, as when $m_{\mathbf{det}}[s_{\mathbf{b}}] \stackrel{\mathsf{a?}v_1.\mathsf{a!}w_1}{\Longrightarrow} \cdot \stackrel{\tau}{\longrightarrow} m''_{\mathbf{det}}[\mathsf{b!}(\mathsf{log}, v_1, w_1).s_{\mathbf{a}}]$ we have that $\mathsf{b!}(\mathsf{log}, v_1, w_1).s_{\mathbf{a}} \in [\![after(\varphi_1, \mathsf{a?}v_1.\mathsf{a!}w_1)]\!]$ and that $m''_{\mathbf{det}}[\mathsf{b!}(\mathsf{log}, v_1, w_1).s_{\mathbf{a}}] \sim \mathsf{b!}(\mathsf{log}, v_1, w_1).s_{\mathbf{a}}$. □

Since Definition 4 (transparency) is just an instance of Definition 5 (eventual transparency) (*i.e.,* when t is the empty trace ε), the latter requirement along with Definition 3 (soundness) suffice to provide an adequate definition for "m enforces φ".

Definition 6 (Enforcement). *A monitor m* enforces *property φ whenever it adheres to* (*i*) soundness*, Definition 3, and* (*ii*) eventual transparency*, Definition 5.* □

Corollary 1. *Since Definition 6 is defined in terms of eventual transparency (Definition 5) that is stronger than transparency (Definition 4), our new definition for "m enforces φ" is* stricter *than the one given in [4].* □

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

Fig. 5. Modification Count (mc).

Definitions 2 and 6 define what it means for a monitor to adequately enforce a formula, but fail to assess whether a monitor is (to some extent) the "*best*" that one can find to enforce a property. 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 ideal measurement is to assess the monitor's level of *intrusiveness* when enforcing the property.

In Figure 5 we thus 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 $systr(t_\tau)$ 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.

Example 8. Recall the monitors of Example 3 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.,* $\forall\mu\cdot m_{\mathbf{d}}[\mathsf{sys}(t^0_\tau)] \xrightarrow{\mu}\!\!\!\!\!\!\rightarrow$ 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.,* $\forall\mu\cdot m_{\mathbf{dt}}[\mathsf{sys}(t^0_\tau)] \xrightarrow{\mathsf{a}?v_1}\cdot\xrightarrow{\mu}\!\!\!\!\!\!\rightarrow$. 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

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

Fig. 6. Enforcement Capabilities (ec).

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 6. 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 9. Recall the monitors of Example 3. 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, $ec(\mathsf{rec}\, X.\{(x)?(y), \bullet\}.X + \{(x)!(y), \bullet\}.X)$=$\{\mathsf{EN}, \mathsf{DIS}\}$. □

With these definitions we now define *optimal enforcement*.

Definition 7 (Optimal Enforcement). *A monitor m is* optimal *when enforcing φ, denoted as $\mathsf{oenf}(m, \varphi)$, iff it* enforces φ *(Definition 6) 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 7 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 10. Recall formula φ_1 of Example 1 and monitor $m_{\mathbf{det}}$ of Example 7. Although showing that $\mathsf{oenf}(m_{\mathbf{det}}, \varphi_1)$ is inherently difficult, from Example 8 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 and 4. We further reaffirm this intuition using systems $s_{\mathbf{b}}$ and $s_{\mathbf{c}}$ from Example 1. In fact, when considering the invalid runs $t_\tau^1 \stackrel{\text{def}}{=} \mathsf{a}?v_1.\tau.\mathsf{a}!w_1.\mathsf{a}!w_1.\mathsf{b}!(\mathsf{log}, v_1, w_1)$ of $s_{\mathbf{b}}$, and $t_\tau^2 \stackrel{\text{def}}{=} \mathsf{a}?v_1.\mathsf{a}?v_2.\tau.\mathsf{a}!w_2.\mathsf{b}!(\mathsf{log}, v_2, w_2)$ 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, $mc(m_{\mathbf{det}}, t_\tau^1) = mc(m_{\mathbf{det}}, t_\tau^2) = 1$. 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 10 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 11. 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 $\mathsf{enf}(m_1, \varphi_2)$.

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

When enforcing a system that generates the run $t^3_\tau \stackrel{\text{def}}{=} \mathsf{a}?v_1.\mathsf{b}?v_2.\mathsf{a}!w_1.u^3_\tau$, monitor m_1 modifies the trace only *once*. Although it disables the input $\mathsf{b}?v_2$, it subsequently unblocks the SuS by inserting $\mathsf{b}?v_{\text{def}}$ 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 $2+|\, u^3_\tau \,|$, as despite blocking 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 $m_2 \stackrel{\text{def}}{=} \{\mathsf{a}?(\_)\}.\mathsf{rec}\, Y.(\{\bullet, \mathsf{b}?v_{\text{def}}\}.Y + \underline{\{\bullet, \mathsf{c}?v_{\text{def}}\}.Y} + \{\mathsf{a}?(\_)\}.\mathsf{id})$, 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, $m_{\mathsf{opt}} \stackrel{\text{def}}{=} \{\mathsf{a}?(\_)\}.(\mathsf{rec}\, Y.\sum_{\mathsf{b} \in \text{Port } \textit{and } \mathsf{a} \neq \mathsf{b}} \{\bullet, \mathsf{b}?v_{\text{def}}\}.Y + \{\mathsf{a}?(\_)\}.\mathsf{id})$ or alternatively $\{\mathsf{a}?(\_)\}.(\mathsf{rec}\, Y.\sum_{\mathsf{b} \in \text{Port}} \{\bullet, \mathsf{a} \neq \mathsf{b}, \mathsf{b}?v_{\text{def}}\}.Y + \{\mathsf{a}?(\_)\}.\mathsf{id})$ where the condition $\mathsf{a} \neq \mathsf{b}$ is evaluated at runtime. □

Unlike Example 10, Example 11 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 7 is too strict. We thus mitigate this issue by weakening Definition 7 and redefine it in terms of the set of system states Sys_{Π}, *i.e.,* the set of states that can only perform inputs using the ports specified in a finite $\Pi \subset \text{Port}$. 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 8 (Weak Optimal Enforcement). *A monitor m is* weakly optimal *when enforcing φ, denoted as $\mathsf{oenf}(m, \varphi, \Pi)$, iff it* enforces φ *(Definition 6) and when for every state $\underline{s \in \text{Sys}_{\Pi}}$, 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 12. Monitor m_1 from Example 11 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 $\Pi = \{\mathsf{a}, \mathsf{b}, \mathsf{c}\}$. □

5 Synthesising Action Disabling Monitors

The universal quantifications in Definitions 3, 5 and 8 make it difficult to prove that a monitor enforces a property correctly and optimally in a bi-directional setting. Particularly, establishing that a formula is enforceable, in the sense of Definition 6, requires finding a monitor that enforces it correctly. This monitor must then be scrutinised vis-a-vis Definition 8 to determine whether it is optimal under certain assumptions about the system's ports. Establishing the enforceability of a logic is even harder, as it entails yet another universal quantification on all the formulas in the logic. We address these problems through an *automated synthesis procedure* that produces an enforcement monitor for every sHML formula and then prove that the synthesised monitors are correct and optimal as stipulated by Definitions 6 and 8.

In this paper, we restrict ourselves to action disabling when studying enforceability of sHML in a bidirectional setting. For a unidirectional setting, it has already been shown several times [4, 31, 16, 25] that the omission of actions from the resulting behaviour of a composite system can be used to enforce *safety properties*. Intuitively, safety is ensured when actions are omitted as soon as it becomes apparent that a violation is about to be committed by the SuS. Even though the works relating action omission to safety have mainly been explored in a unidirectional setting, we are confident that a similar result can be attained in the case of a bi-directional one.

We follow the standard way for achieving our aims by first defining a synthesis function from sHML formulas to action disabling monitors and then showing that for *every* $\varphi \in$ sHML, our synthesis can produce a monitor m_φ that adequately enforces φ as per Definition 6, and weak optimally in the sense of Definition 8. As learnt from our prior work [4, 5], it is imperative for the synthesis function to be *compositional*. This is desirable, as it simplifies our analysis of the produced monitors, and allows us to use standard inductive proof techniques to prove properties about the synthesis function, such as proving adherence to Definitions 6 and 8. However, we argued in the above-mentioned references that a naive approach to such a scheme is bound to fail, and that *normalisation* provides an effective way for preserving the compositionality and simplicity of our synthesis while circumventing the problems of a naive approach. We thus work with respect to $\text{sHML}_{\mathbf{nf}}$ defined in Definition 9, that is, a normalised syntactic subset of sHML that is known to be equally expressive to sHML [4, 3]. An automated procedure for translating from sHML to $\text{sHML}_{\mathbf{nf}}$ is also available in [4].

Definition 9 (sHML normal form). *The set of normalised* sHML *formulas by the following grammar:*

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

In addition, normalised sHML *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*,* $\#_{i\in I}\{p_i, c_i\}$*, which entails that 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* $X \in \boldsymbol{fv}(\varphi)$*.*

In a (closed) $\text{sHML}_{\mathbf{nf}}$ 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, $\text{sHML}_{\mathbf{nf}}$ also inherits the assumptions that fixpoint variables are guarded, and that for every $\{(x)?(y), c\}$, $y \notin \mathbf{fv}(c)$.

Example 13. Consider the following formula φ_3. It defines a recursive property that is violated when a system outputs a value of 4 on any port, after inputting a value from port a, but recurses if the output is made on port a with a value that is not equal to 3.

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

It turns out that $\varphi_3 \notin \text{sHML}_{\mathbf{nf}}$ since the conjunction it defines is not disjoint, *i.e.*, $[\![\{(x_2)!(y_2), x_2{=}\mathsf{a} \wedge y_2{\neq}3\}]\!] \cap [\![\{(x_3)!(y_3), y_3{=}4\}]\!] = \{\mathsf{a}!4\}$. Using the normalisation procedure of [4], we can reformulate φ_3 into $\varphi_3' \in \text{sHML}_{\mathbf{nf}}$:

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

where x_4 and y_4 are fresh variables. □

We now proceed to define the synthesis function in Definition 10. It defines a *compositional* mapping that converts an $\text{sHML}_{\mathbf{nf}}$ formula φ into a transducer m. This conversion also requires information regarding the input ports of the SuS, as this is used to add the necessary insertion branches that silently unblock the SuS at runtime. The synthesis function must therefore be supplied with this information in the form of a *finite* set of input ports $\Pi \subset \text{Port}$, which then relays this information to the resulting monitor.

Example 14. Recall formula φ_2 and monitors m_1 and m_2 from Example 11. Synthesising m_1 from φ_2 the SuS can be unblocked when a value is inserted on port b. This information must be supplied to the synthesis as $\Pi = \{\mathsf{b}\}$ which in turn uses this information to add the insertion branch $\{\bullet, \mathsf{b}?v_{\text{def}}\}.Y$ in m_1. However, if $\Pi = \{\mathsf{b}, \mathsf{c}\}$ monitor m_2 is synthesised instead. □

Definition 10. *The synthesis function* $(\!|-|\!) : \text{sHML}_{\boldsymbol{nf}} \times \mathcal{P}_{fin}(\text{Port}) \mapsto \text{Trn}$ *is defined inductively as:*

$$(\!|X, \Pi|\!) \stackrel{\text{def}}{=} X \qquad (\!|\mathsf{tt}, \Pi|\!) \stackrel{\text{def}}{=} (\!|\mathsf{ff}, \Pi|\!) \stackrel{\text{def}}{=} \mathit{id} \qquad (\!|\max X.\varphi, \Pi|\!) \stackrel{\text{def}}{=} \mathit{rec}\, X.(\!|\varphi, \Pi|\!)$$

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

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

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

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 and converts logical variables X accordingly, whereas maximal fixpoints, $\max X.\varphi$, are converted into the corresponding recursive monitor. The synthesis also converts truth, tt, and falsehood, ff, formulas into the identity monitor id. Normalized conjunctions, $\bigwedge_{i\in I}[\{p_i, c_i\}]\varphi_i$, are synthesised into 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.,* $[\{p_i, c_i\}]\varphi_i$ is synthesised as $\{p_i, c_i\}.(\!|\varphi_i, \Pi|\!)$;

(*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 (*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*, $\sum_{\mathsf{b}\in\Pi}\{\bullet, c_i\{\mathsf{b}/x_i\}, \mathsf{b}?v_{\text{def}}\}.m_i$, 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 handshake from the environment. However, the synthesised insertion branches allow the resulting monitor to insert a default domain value v_{def} on every port $\mathsf{a} \in \Pi$ 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.

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 unspecified actions do not contribute towards a safety violation and, on the contrary, lead to its trivial satisfaction. To prevent this, the *default*

monitor $\mathsf{def}(\varphi)$ is also added to the resulting summation. 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 to modal necessities defining an input pattern in the normalized conjunction. Put differently, the default monitor allows inputs to reach the system whenever they satisfy the condition $\bigwedge_{i\in I} \neg c_i$. This condition is constructed when the normalised conjunction is $\bigwedge_{i\in I} [\{(x_i)?(y_i), c_i\}]\varphi_i \wedge \psi$ (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 $\{(\_)?(\_)\}.\mathsf{id}$. Upon matching a system action, the default monitor transitions to id after forwarding the input to the SuS.

Example 15. Consider the following unshortened version of formula φ_1 from Example 1.

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

For any set of ports Π, the synthesis function of Definition 10 produces the following monitor.

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

The synthesised monitor m_{φ_1} can be further optimized by removing redundant recursive constructs such as $\mathsf{rec}\, Z.\_$. □

Notice that monitor m_{φ_1} (synthesised via $(\!|\, \varphi_1, \Pi \,|\!)$ in Example 15) has essentially the same structure as $m_{\mathbf{det}}$ of Example 3 but mainly varies in how it defines its insertion branches for unblocking the SuS. For instance if $\Pi = \{\mathsf{b}, \mathsf{c}\}$, $(\!|\, \varphi_1, \Pi \,|\!)$ would synthesise two insertion branches, namely, $\{\bullet, \mathsf{b} = x, \mathsf{b}?v_{\text{def}}\}$ and $\{\bullet, \mathsf{c} = x, \mathsf{c}?v_{\text{def}}\}$, whereas if Π also includes d, it would add another branch. By contrast, the manually defined $m_{\mathbf{det}}$ attains the same result more elegantly via the single insertion branch $\{\bullet, x?v_{\text{def}}\}$. As argued in Example 11, it is not always possible to define a monitor like $m_{\mathbf{det}}$, especially when the formula defines complex conditions in its violating modal necessities, such as φ_2 of Example 11. Despite this, the following results, namely, Theorems 1 and 2, show that our synthesis still guarantees enforceability and optimality.

Theorem 1 (Enforceability). *The logic* sHML *is enforceable in a bi-directional setting.* □

Proof. Since sHML is logically equivalent to $\text{sHML}_{\mathbf{nf}}$, by Definition 2 the result follows from showing that for every $\varphi \in \text{sHML}_{\mathbf{nf}}$ and $\Pi \subseteq \textsc{Port}$, $(\!|\, \varphi, \Pi \,|\!)$ enforces φ (for every Π). By Definition 6, this claim comes as a result of the below stated Propositions 2 and 3 which entail that the synthesised monitors *enforce* their respective $\text{sHML}_{\mathbf{nf}}$ formula and are correct by construction. □

Proposition 2 (Soundness). *For every input port set Π, system state $s \in \text{SYS}$ and $\varphi \in \text{sHML}_{nf}$, if $[\![\varphi]\!] \neq \emptyset$ then $(\!|\varphi, \Pi|\!)[s] \in [\![\varphi]\!]$.* □

Proposition 3 (Eventual Transparency). *For every input port set Π, sHML formula φ, system states $s, s' \in \text{SYS}$, action disabling monitor m' and trace t, if $(\!|\varphi, \Pi|\!)[s] \stackrel{t}{\Rightarrow} m'[s']$ and $s' \in [\![\mathsf{after}(\varphi, t)]\!]$ then $m'[s'] \sim s'$.* □

The proofs of these propositions are given in Appendix A.

We now proceed to show that the synthesised monitor $(\!|\varphi, \Pi|\!)$ is also guaranteed to be optimal (as stated by Definition 8) when enforcing φ on a SuS s whose input ports are specified by Π, *i.e.*, $s \in \text{SYS}_{\Pi}$.

Theorem 2 (Weak Optimal Enforcement). *For every system state $s \in \text{SYS}_{\Pi}$, explicit trace t_τ and monitor m, if $ec(m) \subseteq ec((\!|\varphi, \Pi|\!))$, $\mathsf{enf}(m, \varphi)$ and $s \xrightarrow{t_\tau}$ then $mc((\!|\varphi, \Pi|\!), t_\tau) \leq mc(m, t_\tau)$.* □

The proof for this theorem is also given in Appendix A.

6 Related Work

In his seminal work [39], 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 [31] 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 [39] and Ligatti *et al.* [31] 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 [4], 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 [11, 13] 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 [28] 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 [35, 34], 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 [35, 34] 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 [35, 34] 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.

7 Conclusion

In this paper we conduct a preliminary study of the enforceability of a branching time logic in a bi-directional enforcement setting. To enable this study, we conceptualised a novel distinction between the enforcement transformations (suppression, insertions and replacements) performed by the enforcement monitor, and the way the instrumentation interprets these transformations to enable, disable or adapt actions of the SuS at runtime. Based on this distinction, we have developed an instrumentation model for bi-directional enforcement. We then focused on identifying the properties that can be both expressed via the process logic sHML and also enforced using our model.

We thus defined a strict notion of enforceability that builds upon the conventional definitions that are typically based on soundness and transparency.

Particularly, we included the criterion of eventual transparency that not only forbids modifications to valid systems (as imposed by transparency), but also requires the monitor to stop applying transformations whenever the SuS reaches a valid point. In addition, we introduced the notion of optimality to assess the level of intrusiveness of a monitor, and determine whether it is the least intrusive monitor that one can find. Based on these notions, we devised a synthesis function that produces action disabling monitors that are correct and optimal by construction. This allowed us to conclude that sHML is enforceable via action disabling in a bidirectional setting.

Future Work. We plan to expand our exploration into bi-directional enforcement by studying the enforceability of more expressive logics such as the full μHML. This requires investigating how the capabilities of the other enforcement instrumentation mechanisms (action enabling and adaptation) can be fully harnessed to enforce properties that cannot be expressed via sHML. We also intend to study the maximality results for action disabling enforcement, along the lines of [2, 19].

Another interesting avenue for future research would be to explore the implementability and feasibility of our instrumentation model for bi-directional enforcement. Languages closer to an actual implementation (*e.g.,* actor or channel-based languages along the lines of [20, 9]) can be used to implement our monitor descriptions and refinement analysis techniques can then be used to refine our abstract enforcement monitor descriptions into more concrete ones. Our synthesis function in conjunction with the refinement technique can then be used to guide and facilitate tool construction.

References

1. Aceto, L., Achilleos, A., Francalanza, A., Ingólfsdóttir, A.: A framework for parameterized monitorability. In: Foundations of Software Science and Computation Structures. pp. 203–220. Springer International Publishing, Cham (2018)
2. Aceto, L., Achilleos, A., Francalanza, A., Ingólfsdóttir, A.: Monitoring for silent actions. In: Lokam, S., Ramanujam, R. (eds.) FSTTCS 2017: Foundations of Software Technology and Theoretical Computer Science. LIPIcs, vol. 93, pp. 7:1–7:14. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany (2018)
3. Aceto, L., Achilleos, A., Francalanza, A., Ingólfsdóttir, A., Kjartansson, S.Ö.: Determinizing monitors for HML with recursion. arXiv preprint (2016)
4. Aceto, L., Cassar, I., Francalanza, A., Ingólfsdóttir, A.: On runtime enforcement via suppressions. In: 29th International Conference on Concurrency Theory, CONCUR 2018, September 4-7, 2018, Beijing, China. pp. 34:1–34:17 (2018). https://doi.org/10.4230/LIPIcs.CONCUR.2018.34
5. Aceto, L., Cassar, I., Francalanza, A., Ingólfsdóttir, A.: Comparing controlled system synthesis and suppression enforcement. In: Runtime Verification. Springer International Publishing, Cham (2019), (to appear)
6. Aceto, L., Ingólfsdóttir, A.: Testing hennessy-milner logic with recursion. In: Thomas, W. (ed.) Foundations of Software Science and Computation Structures. pp. 41–55. Springer Berlin Heidelberg, Berlin, Heidelberg (1999)

7. Aceto, L., Ingólfsdóttir, A., Larsen, K.G., Srba, J.: Reactive Systems: Modelling, Specification and Verification. Cambridge University Press, New York, NY, USA (2007)
8. Alur, R., Černý, P.: Streaming transducers for algorithmic verification of single-pass list-processing programs. In: Proceedings of the 38th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. pp. 599–610. ACM (2011)
9. Attard, D.P., Francalanza, A.: A monitoring tool for a branching-time logic. In: Runtime Verification. pp. 473–481. Springer International Publishing, Cham (2016)
10. Berstel, J., Boasson, L.: Transductions and context-free languages. Ed. Teubner pp. 1–278 (1979)
11. Bielova, N.: A theory of constructive and predictable runtime enforcement mechanisms. Ph.D. thesis, University of Trento (2011)
12. Bielova, N., Massacci, F.: Do you really mean what you actually enforced?-edited automata revisited. International Journal of Information Security **10**(4), 239–254 (2011)
13. Bielova, N., Massacci, F.: Predictability of enforcement. In: International Symposium on Engineering Secure Software and Systems. pp. 73–86. Springer (2011)
14. Bocchi, L., Chen, T.C., Demangeon, R., Honda, K., Yoshida, N.: Monitoring networks through multiparty session types. Theoretical Computer Science **669**, 33 – 58 (2017)
15. Chen, T.C., Bocchi, L., Deniélou, P.M., Honda, K., Yoshida, N.: Asynchronous distributed monitoring for multiparty session enforcement. In: Bruni, R., Sassone, V. (eds.) Trustworthy Global Computing. pp. 25–45. Springer Berlin Heidelberg, Berlin, Heidelberg (2012)
16. Falcone, Y., Fernandez, J.C., Mounier, L.: What can you verify and enforce at runtime? International Journal on Software Tools for Technology Transfer **14**(3), 349 (Jun 2012)
17. Francalanza, A.: A Theory of Monitors. In: International Conference on Foundations of Software Science and Computation Structures. pp. 145–161. Springer (2016)
18. Francalanza, A.: Consistently-Detecting Monitors. In: 28th International Conference on Concurrency Theory (CONCUR 2017). Leibniz International Proceedings in Informatics (LIPIcs), vol. 85, pp. 8:1–8:19. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany (2017)
19. Francalanza, A., Aceto, L., Ingólfsdóttir, A.: Monitorability for the Hennessy-Milner logic with recursion. Formal Methods in System Design **51**(1), 87–116 (2017)
20. Francalanza, A., Seychell, A.: Synthesising correct concurrent runtime monitors. Formal Methods in System Design **46**(3), 226–261 (2015)
21. Goldberg, A., Havelund, K., McGann, C.: Runtime verification for autonomous spacecraft software. In: 2005 IEEE Aerospace Conference. pp. 507–516. IEEE (2005)
22. Havelund, K., Roşu, G.: An overview of the runtime verification tool java pathexplorer. Formal methods in system design **24**(2), 189–215 (2004)
23. Hennessy, M., Lin, H.: Proof systems for message-passing process algebras. Formal Aspects of Computing **8**(4), 379–407 (Jul 1996). https://doi.org/10.1007/BF01213531
24. Hennessy, M., Liu, X.: A modal logic for message passing processes. Acta Informatica **32**(4), 375–393 (Apr 1995). https://doi.org/10.1007/BF01178384, https://doi.org/10.1007/BF01178384
25. van Hulst, A.C., Reniers, M.A., Fokkink, W.J.: Maximally permissive controlled system synthesis for non-determinism and modal logic. Discrete Event Dynamic Systems **27**(1), 109–142 (Mar 2017)

26. Jia, L., Gommerstadt, H., Pfenning, F.: Monitors and blame assignment for higher-order session types. In: Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. pp. 582–594. ACM, New York, NY, USA (2016)
27. Khoury, R., Tawbi, N.: Which security policies are enforceable by runtime monitors? a survey. Computer Science Review **6**(1), 27–45 (2012)
28. Könighofer, B., Alshiekh, M., Bloem, R., Humphrey, L., Könighofer, R., Topcu, U., Wang, C.: Shield synthesis. Formal Methods in System Design **51**(2), 332–361 (Nov 2017)
29. Kozen, D.C.: Results on the propositional μ-calculus. Theoretical Computer Science **27**, 333–354 (1983)
30. Larsen, K.G.: Proof systems for satisfiability in Hennessy-Milner logic with recursion. Theoretical Computer Science **72**(2), 265–288 (1990)
31. Ligatti, J., Bauer, L., Walker, D.: Edit automata: enforcement mechanisms for run-time security policies. International Journal of Information Security **4**(1), 2–16 (Feb 2005)
32. Ligatti, J., Bauer, L., Walker, D.: Run-time enforcement of nonsafety policies. ACM Trans. Inf. Syst. Secur. **12**(3), 19:1–19:41 (Jan 2009)
33. Milner, R., Parrow, J., Walker, D.: A calculus of mobile processes, I. Information and computation **100**(1), 1–40 (1992)
34. Pinisetty, S., Roop, P.S., Smyth, S., Allen, N., Tripakis, S., Hanxleden, R.V.: Runtime enforcement of cyber-physical systems. ACM Trans. Embed. Comput. Syst. **16**(5s), 178:1–178:25 (Sep 2017)
35. Pinisetty, S., Roop, P.S., Smyth, S., Tripakis, S., von Hanxleden, R.: Runtime enforcement of reactive systems using synchronous enforcers. CoRR **abs/1612.05030** (2016)
36. Rathke, J., Hennessy, M.: Local model checking for value-passing processes (extended abstract). In: Abadi, M., Ito, T. (eds.) Theoretical Aspects of Computer Software. pp. 250–266. Springer Berlin Heidelberg, Berlin, Heidelberg (1997)
37. Sakarovitch, J.: Elements of Automata Theory. Cambridge University Press, New York, NY, USA (2009)
38. Sangiorgi, D.: Introduction to Bisimulation and Coinduction. Cambridge University Press, New York, NY, USA (2011)
39. Schneider, F.B.: Enforceable security policies. ACM Transactions on Information and System Security (TISSEC) **3**(1), 30–50 (2000)

A Main Proofs

To facilitate the forthcoming proofs we occasionally use the satisfaction semantics for sHML from [6, 24] which is defined in terms of the *satisfaction relation*, $\models$. When restricted to sHML, $\models$ is the *largest relation* $\mathcal{R}$ satisfying the implications defined in Figure 7. As these semantics are well known to agree with the sHML semantics of Figure 2, we use $s \models \varphi$ in lieu of $s \in [\![\varphi]\!]$. At certain points we also refer to the τ-closure property of sHML, Proposition 4, that was proven in [6].

Proposition 4. *if* $s \xrightarrow{\tau} s'$ *and* $s \models \varphi$ *then* $s' \models \varphi$.

We also assume the classic notion of *strong similarity*, $s \sqsubseteq r$ as our touchstone system preorder for LTSs [33, 38].

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

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

Fig. 7. A satisfaction relation for sHML formulas

Definition 11 (Strong Similarity). *A relation $\mathcal{R}$ over a set of system states is a* strong simulation *iff whenever $(s, r) \in \mathcal{R}$ for every action μ:*

- *$s \xrightarrow{\mu} s'$ implies there exists a transition $r \xrightarrow{\mu} r'$ such that $(s', r') \in \mathcal{R}$*

States s and r are similar*, $s \precsim r$, iff they are related by a* strong simulation. □

We now present the proof for Proposition 1 followed by the proofs for the theorems and propositions omitted from Section 5.

A.1 Proving Proposition 1

We need to prove that for every system transition $s \stackrel{\alpha}{\Longrightarrow} s'$ and sHML formula φ, if $s \in [\![\varphi]\!]$ then $s' \in [\![\mathit{after}(\varphi, \alpha)]\!]$. We prove the contrapositive, *i.e.*, if $s \stackrel{\alpha}{\Longrightarrow} s'$ and $s' \notin [\![\mathit{after}(\varphi, \alpha)]\!]$ then $s \notin [\![\varphi]\!]$.

Proof. As we assume that logical variables are guarded (see Remark 1 on 7), we can proceed by rule induction on *after*.

Case $\mathit{after}(\mathsf{ff}, \alpha)$. This case holds trivially since $s \notin [\![\mathsf{ff}]\!]$.

Case $\mathit{after}(\mathsf{tt}, \alpha)$. This case does not apply since $\mathit{after}(\mathsf{tt}, \alpha) = \mathsf{tt}$ and so the assumption that $s' \notin [\![\mathit{after}(\mathsf{tt}, \alpha)]\!]$ is invalid.

Case $\mathit{after}(\bigwedge_{i \in I} \varphi_i, \alpha)$. Assume that

$$s \stackrel{\alpha}{\Longrightarrow} s' \tag{1}$$

and that $s' \notin [\![\mathit{after}(\bigwedge_{i \in I} \varphi_i, \alpha)]\!]$ from which by the definition of *after* we have that

$$s' \notin [\![\bigwedge_{i \in I} \mathit{after}(\varphi_i, \alpha)]\!] \;\equiv\; \exists j \in I \cdot s' \notin [\![\mathit{after}(\varphi_j, \alpha)]\!]. \tag{2}$$

Hence, by (1) and (2) we can apply the inductive hypothesis and deduce that there exists a $j \in I$ such that $s \notin [\![\varphi_j]\!]$ which means that $s \notin \bigcap_{i \in I} [\![\varphi_i]\!] = [\![\bigwedge_{i \in I} \varphi_i]\!]$ as required.

Case after$(\mathsf{max}\, X.\varphi, \alpha)$. Assume that

$$s \stackrel{\alpha}{\Longrightarrow} s' \tag{3}$$

and that $s' \notin [\![after(\mathsf{max}\, X.\varphi, \alpha)]\!]$ from which, by the definition of *after*, we have that

$$s' \notin [\![after(\varphi\{\mathsf{max}\, X.\varphi/X\}, \alpha)]\!]. \tag{4}$$

By (3), (4) and the inductive hypothesis we have that $s \notin [\![\varphi\{\mathsf{max}\, X.\varphi/X\}]\!]$. Since $[\![\varphi\{\mathsf{max}\, X.\varphi/X\}]\!] = [\![\mathsf{max}\, X.\varphi]\!]$, we can conclude that $s \notin [\![\mathsf{max}\, X.\varphi]\!]$ as required.

Case after$([\{p, c\}]\varphi, \alpha)$. Assume that

$$s \stackrel{\alpha}{\Longrightarrow} s' \quad \textit{and} \tag{5}$$

$$s' \notin [\![after([\{p, c\}]\varphi, \alpha)]\!]. \tag{6}$$

Now consider the following two cases:

- $\mathsf{mtch}(p, \alpha) = \sigma$ and $c\sigma \Downarrow \mathsf{true}$ (for some σ): By (6) and the definition of *after* we know that

  $$s' \notin [\![\varphi\sigma]\!] \tag{7}$$

  and so from (5), (7) and by the definition of $[\![-]\!]$ we can infer that $s \notin [\![[\{p, c\}]\varphi]\!]$ since there exists a transition, *i.e.,* (5), that leads to a violation, *i.e.,* (7).
- Otherwise: This case does not apply since $after([\{p, c\}]\varphi, \alpha) = \mathsf{tt}$ which contradicts assumption (6).

□

A.2 Proving Proposition 2 (Sound Enforcement)

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

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

we show a stronger result stating that for every system r simulated by $(\!|\, \varphi, \Pi \,|\!)[s]$,

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

We prove this by showing that relation $\mathcal{R}$ (below) is a *satisfaction relation* ($\vDash$) and so that it abides by the rules in Figure 7.

$$\mathcal{R} \stackrel{\text{def}}{=} \left\{ (r, \varphi) \;\middle|\; [\![\varphi]\!] \neq \emptyset \textit{ and } r \sqsubseteq_{\approx} (\!|\, \varphi, \Pi \,|\!)[s] \right\}.$$

We prove this claim by case analysis on the structure of φ.

Cases $\varphi \in \{X, \mathsf{ff}\}$. These cases do not apply since $[\![\mathsf{ff}]\!] = \emptyset$ and $(\!|\, X, \Pi \,|\!)$ does not yield a valid monitor.

Case $\varphi = \mathsf{tt}$. This case holds trivially as for *every process* $r \sqsubseteq_{\approx} (\!|\, \mathsf{tt}, \Pi \,|\!)[s]$ the pair (r, tt) is in $\mathcal{R}$ since we know that $[\![\mathsf{tt}]\!] \neq \emptyset$.

Case $\varphi = \mathsf{max}\, X.\varphi$ *and* $X \in \boldsymbol{fv}(\varphi)$. Lets assume that $(r, \mathsf{max}\, X.\varphi) \in \mathcal{R}$ and so we have that

$$[\![\mathsf{max}\, X.\varphi]\!] \neq \emptyset \tag{1}$$

$$r \sqsubseteq_{\approx} (\!|\, \mathsf{max}\, X.\varphi, \Pi \,|\!)[s]. \tag{2}$$

To prove that $\mathcal{R}$ is a satisfaction relation we show that $(r, \varphi\{\mathsf{max}\, X.\varphi/X\}) \in \mathcal{R}$ as well. Hence, since $(\!|\, \varphi\{\mathsf{max}\, X.\varphi/X\}, \Pi \,|\!)$ produces a monitor that is the *unfolded equivalent* of $(\!|\, \mathsf{max}\, X.\varphi, \Pi \,|\!)$ we can conclude that $(\!|\, \mathsf{max}\, X.\varphi, \Pi \,|\!) \sim (\!|\, \varphi\{\mathsf{max}\, X.\varphi/X\}, \Pi \,|\!)$ and so from (2) we have that

$$r \sqsubseteq_{\approx} (\!|\, \varphi\{\mathsf{max}\, X.\varphi/X\}, \Pi \,|\!)[s]. \tag{3}$$

Finally, since from (1) and $[\![\mathsf{max}\, X.\varphi]\!] = [\![\varphi\{\mathsf{max}\, X.\varphi/X\}]\!]$ we can also deduce that $[\![\varphi\{\mathsf{max}\, X.\varphi/X\}]\!] \neq \emptyset$, by (3) and the definition of $\mathcal{R}$ we can conclude that $(r, \varphi\{\mathsf{max}\, X.\varphi/X\}) \in \mathcal{R}$ as required.

Case $\varphi = \bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i$ *and* $\#_{h \in I}\{p_h, c_h\}$. Assume that $(r, \bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i) \in \mathcal{R}$ and so we have that

$$[\![\bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i]\!] \neq \emptyset \tag{4}$$

$$r \sqsubseteq_{\approx} (\!|\, \bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i, \Pi \,|\!)[s]. \tag{5}$$

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

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

which can be further unfolded as

$$(\!|\, \bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i, \Pi \,|\!) = \Big(\sum_{i \in I} \begin{cases} \mathsf{dis}(p_i, c_i, m, \Pi) & \text{if } \varphi_i{=}\mathsf{ff} \\ \{p_i, c_i\}.(\!|\, \varphi_i, \Pi \,|\!) & \text{otherwise} \end{cases} \Big) + \mathsf{def}(\bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i). \tag{6}$$

In order to prove that $\mathcal{R}$ is a satisfaction relation, for this case we must show that for every $j \in I$, $(r, [\{p_j, c_j\}]\varphi_j) \in \mathcal{R}$ as well. We thus inspect the different types of branches that are definable in $\mathrm{sHML}_{\mathbf{nf}}$ and hence we consider the following cases:

(i) *A violating output branch,* $[\{(x)!(y), c_j\}]\mathsf{ff}$:

To prove that $(r, [\{(x)!(y), c_j\}]\mathsf{ff}) \in \mathcal{R}$ we show that (*a*) $[\![[\{(x)!(y), c_j\}]\mathsf{ff}]\!] \neq \emptyset$, (*b*) $r \sqsubseteq (\!|\, [\{(x)!(y), c_j\}]\mathsf{ff}, \Pi \,|\!)[s]$, and (*c*) that for every output action $\mathsf{a}!v$, when $\{(x)!(y), c_j\}(\mathsf{a}!v) = \sigma$, then there does not exist a system r' such that

$r \stackrel{\mathsf{a}!v}{\Longrightarrow} r'$. From (4) and the definition of $[\![-]\!]$ we can immediately infer that (a) holds, and so we have that

$$[\![[\{(x)!(y), c_j\}]\mathsf{ff}]\!] \neq \emptyset. \tag{7}$$

We now note that since from (6) we know that branch $[\{(x)!(y), c_j\}]\mathsf{ff}$ is synthesised into $\mathsf{dis}((x)!(y), c_j, m, \Pi)$ in m, by the definition of dis we can infer that one of the summed monitors in (6) is a *suppression monitor* of the form

$$\mathsf{dis}((x)!(y), c_j, m, \Pi) = \{(x)!(y), c_j, \bullet\}.m. \tag{8}$$

From (8) we infer that since $(\!|[\{(x)!(y), c_j\}]\mathsf{ff}, \Pi|\!) \stackrel{\text{def}}{=} \mathsf{rec}\, Y.\{(x)!(y), c_j, \bullet\}.Y + \mathsf{def}([\{(x)!(y), c_j\}]\mathsf{ff})$ (where $\mathsf{def}([\{(x)!(y), c_j\}]\mathsf{ff}) \stackrel{\text{def}}{=} \{(x)?(y), \mathsf{true}\}.\mathsf{id}$) we know that this monitor can only disable actions matching $\{(x)!(y), c_j\}$, while $m = (\!|\bigwedge_{i\in I}[\{p_i, c_i\}]\varphi_i, \Pi|\!)$ can possibly disable other actions as well. Hence, the composite system $m[s]$ (for any s) can perform *at most* the same actions as $(\!|[\{(x)!(y), c_j\}]\mathsf{ff}, \Pi|\!)[s]$ and so from (5) we can deduce that (b) holds since

$$r \sqsubseteq_{\approx} (\!|\bigwedge_{i\in I}[\{p_i, c_i\}]\varphi_i, \Pi|\!)[s] \sqsubseteq_{\approx} (\!|[\{(x)!(y), c_j\}]\mathsf{ff}, \Pi|\!)[s] \tag{9}$$

as required. Finally, from (6) we know that monitor m was synthesised from a normalized conjunction which is *disjoint* (since $\#_{h\in I}\{p_h, c_h\}$) and that the synthesised monitors $\mathsf{def}(\bigwedge_{i\in I}[\{p_i, c_i\}]\varphi_i)$ can only transform actions that do not satisfy the conditions of the other monitors in m. This enables us to conclude that whenever the system performs action $\mathsf{a}!v$ such that $\{(x)!(y), c_j\}(\mathsf{a}!v) = \sigma$, only the suppression branch presented in (8) (which is a single branch of m in (6)) can be selected via rule eSel. Once this branch is selected, the action is suppressed via rule eTrn and as a result disabled via rule iDisO which causes the composite system $m[s]$ to transition over a silent τ action to its recursive derivative m. This means that $m[s] \stackrel{\mathsf{a}!v}{\Longrightarrow\!\!\!\!\!/}$ and so from (5) we can also deduce that (c) also holds since

$$\nexists r' \cdot r \stackrel{\mathsf{a}!v}{\Longrightarrow} r' \tag{10}$$

which means that any output modal necessity that precedes ff can never be satisfied by r as required. This case thus holds by (7), (9) and (10).

(ii) *A violating input branch,* $[\{(x)?(y), c_j\}]\mathsf{ff}$ where $Y \notin \mathbf{fv}(c_j)$:
To prove that $(r, [\{(x)?(y), c_j\}]\mathsf{ff}) \in \mathcal{R}$ we show that: (a) $[\![[\{(x)?(y), c_j\}]\mathsf{ff}]\!] \neq \emptyset$, (b) $r \sqsubseteq (\!|[\{(x)?(y), c_j\}]\mathsf{ff}, \Pi|\!)[s]$ and then that (c) for every input action $\mathsf{a}?v$, when $\{(x)?(y), c_j[]\}(\mathsf{a}?v) = \sigma$, then there does not exist a system r' such that $r \stackrel{\mathsf{a}?v}{\Longrightarrow} r'$.

By (4) and the definition of $[\![-]\!]$ we can infer that (a) holds, and so that

$$[\![[\{(x)?(y), c_j\}]\mathsf{ff}]\!] \neq \emptyset. \tag{11}$$

Now, from (6) we can infer that the summation of monitors in m includes a *summation of insertion monitors* and so by the definition of dis we have that

$$\mathsf{dis}((x)?(y), c_j, m, \Pi) = \sum_{\mathsf{b}\in\Pi}\{\bullet, c_j\{\mathsf{b}/x\}, \mathsf{b}?w\}.m. \tag{12}$$

Therefore from (12) we can deduce that monitor

$$\begin{aligned}(\!|\, [\{(x)?(y), c_j\}]\mathsf{ff}, \Pi \,|\!) = \mathsf{rec}\, Y.\sum_{\mathsf{b}\in\Pi}\{\bullet, c_j\{\mathsf{b}/x\}, \mathsf{b}?w\}.m + \mathsf{def}([\{(x)?(y), c_j\}]\mathsf{ff})\\ \textit{where } \mathsf{def}([\{(x)?(y), c_j\}]\mathsf{ff}) \stackrel{\text{def}}{=} \{(x)?(y), \neg c_j[]\}.\mathsf{id}\end{aligned} \tag{13}$$

can only block and disable erroneous input actions satisfying $[\{(x)?(y), c_j\}]\mathsf{ff}$, while monitor m in (6) can possibly disable additional actions. Hence the monitored system $m[s]$ can only perform either a subset or exactly the same action as per $(\!|\, [\{(x)?(y), c_j\}]\mathsf{ff}, \Pi \,|\!)[s]$ and so from (5) we can deduce that (b) holds, and so that

$$r \sqsubseteq \approx (\!|\, \bigwedge_{i\in I} [\{p_i, c_i\}]\varphi_i, \Pi \,|\!)[s] \sqsubseteq \approx (\!|\, [\{(x)?(y), c_j\}]\mathsf{ff}, \Pi \,|\!)[s] \tag{14}$$

as required. Finally, to show that (c) holds, recall that every system s *is unable perform an input unless the environment provides it*, and that the monitor *cannot allow an input to go through* unless it has an *identity* (or replacement) branch that forwards the environment's input to the system. From (12) and (13) we thus know that the synthesised monitor m does not include such an identity (or replacement) branch for $[\{(x)?(y), c_j\}]\mathsf{ff}$, and unless $\Pi = \emptyset$, it instead provides a summation of insertion transformations that allow the monitor to *insert a default domain value* w on every port b in Π whenever $c_j\{^{\mathsf{b}}/_x\}$ evaluates to true at runtime. In this way, whenever the system is expecting to erroneously procure an input from the environment, the monitor *blocks and disables* the input and unless $\Pi{=}\emptyset$ it also (non-deterministically) selects one of the synthesised branches in (12) via rule eSel and performs the insertion via rule eTrn which subsequently allows the instrumentation to proceed via rule iDisI that forwards the generated input to the system. This causes the composite system $m[s]$ to transition over a silent τ action to the recursive derivative m. Since the erroneous input is *blocked* regardless of whether the monitor inserts a value or not, we can infer that $m[s] \stackrel{\mathsf{a}?v}{\Longrightarrow\!\!\!\!\!/}$ and so from (5) we can also deduce that

$$\nexists r' \cdot r \stackrel{\mathsf{a}?v}{\Longrightarrow} r' \tag{15}$$

and so the violating input modalities cannot ever be satisfied by r as required. Therefore, this case holds by (11), (14) and (15).

(iii) *A non-violating branch,* $[\{p_j, c_j\}]\varphi_j$ *(where* $\varphi_j \neq \mathsf{ff}$*)*:
To prove that this branch is in $\mathcal{R}$, $(r, [\{p_j, c_j\}]\varphi_j){\in}\mathcal{R}$, we show that (a)

$[\![[\{p_j, c_j\}]\varphi_j]\!] \neq \emptyset$, (b) $r \sqsubseteq_{\approx} (\!| [\{p_j, c_j\}]\varphi_j, \Pi |\!)[s]$ and then that (c) for every action α and derivative r', when $\{p_j, c_j\}(\alpha){=}\sigma$ and $r \stackrel{\alpha}{\Longrightarrow} r'$ then $(r', \varphi_j\sigma){\in}\mathcal{R}$.

From (4) and by the definition of $[\![-]\!]$ we can immediately determine that (a) holds, and so that

$$[\![[\{p_j, c_j\}]\varphi_j]\!] \neq \emptyset \tag{16}$$

and since by the definition of $(\!|-|\!)$ we know that monitor

$$(\!| [\{p_j, c_j\}]\varphi_j, \Pi |\!) = \mathsf{rec}\, Y.\{p_j, c_j\}.(\!| \varphi, \Pi |\!) + \mathsf{def}([\{p_j, c_j\}]\varphi_j)$$

from (6) we can infer that both monitors m and $(\!| [\{p_j, c_j\}]\varphi_j, \Pi |\!)$ refrain from modifying actions matching $\{p_j, c_j\}$ but m may disable more actions. Hence we can infer that for all s, $m[s] \sqsubseteq_{\approx} (\!| [\{p_j, c_j\}]\varphi_j, \Pi |\!)[s]$ and so from (5) we can deduce that (b) holds since

$$r \sqsubseteq_{\approx} m[s] \sqsubseteq_{\approx} (\!| [\{p_j, c_j\}]\varphi_j, \Pi |\!)[s] \tag{17}$$

as required. We now prove that (c) holds by assuming that

$$\{p_j, c_j\}(\alpha) = \sigma \tag{18}$$

$$r \stackrel{\alpha}{\Longrightarrow} r' \tag{19}$$

and so from (5) and (19) we can deduce that

$$m[s] \stackrel{\alpha}{\Longrightarrow} q \text{ (where } r' \sqsubseteq_{\approx} q). \tag{20}$$

Hence, by the definition of $\stackrel{\alpha}{\Longrightarrow}$ we know that the weak transition in (20) is composed of zero or more τ-transitions followed by the α-transition, *i.e.*,

$$m[s] \xrightarrow{\tau} {}^* q' \xrightarrow{\alpha} q. \tag{21}$$

By the rules in our model we know that the τ-reductions in (21) could have been the result of either one of these instrumentation rules, namely iDisI, iDisO or iAsy. From (6) we however know that whenever an action is disabled (via rules iDisO/I) the synthesised monitor m always recurses back to its original form m and in this case only s changes its state to some s'; the same effect occurs if rule iAsy is applied instead. Hence we know that $q' = m[s']$ (for some derivative s' of s), and so from (21) we thus have that

$$m[s'] \xrightarrow{\alpha} q. \tag{22}$$

From (18) we also know that the reduction in (22) can be the result of either iTrnI or iTrnO, and so we consider both cases.

(i) iTrnI: By assuming that (22) is the result of rule iTrnI we infer that $\alpha = \mathsf{a}?v$ and that

$$m \xrightarrow{\mathsf{a}?v \blacktriangleright \mathsf{b}?w} m' \tag{23}$$

$$s' \xrightarrow{\mathsf{b}?w} s'' \tag{24}$$

$$q = m'[s'']. \tag{25}$$

Since we know that $[\{p_j, c_j\}]\varphi_j$ and $\varphi_j \neq \mathsf{ff}$, from (6) we know that m defines an *identity branch* of the form $\{p_j, c_j\}.(\!|\, \varphi_j, \Pi \,|\!)$ which is *completely disjoint* from the rest of the monitors. This is true since m is derived from a normalized conjunction in which $\#_{i \in I}\{p_i, c_i\}$, and the default monitors, $\mathsf{def}(\bigwedge_{i \in I}[\{p_i, c_i\}]\varphi_i)$, can only match actions that do not match with any other monitor. Hence from (18) and (23) we can deduce that

$$m' = (\!|\, \varphi_j\sigma, \Pi \,|\!). \tag{26}$$

Since from (16) and by the definition of $[\![-]\!]$ we know that $[\![\varphi_j\sigma]\!] \neq \emptyset$ and from (20), (25) and (26) we have that $r' \sqsubseteq_{\approx} (\!|\, \varphi_j\sigma, \Pi \,|\!)[s'']$, by the definition of $\mathcal{R}$ we can conclude that $(r', \varphi_j\sigma) \in \mathcal{R}$ as required.

(ii) iTrnO: We omit this proof due to its strong resemblance to that of case iTrnI.

Therefore from (i) and (ii) we can conclude that (c) holds as well, which means that

$$\forall \alpha, r' \cdot \ \textit{if}\ \{p_j, c_j\}(\alpha) = \sigma \ \textit{and}\ r \overset{\alpha}{\Longrightarrow} r' \text{ then } (r', \varphi_j\sigma) \in \mathcal{R}. \tag{27}$$

Hence this case is done by (16), (17) and (27). □

A.3 Proving Proposition 3 (Eventual Transparent Enforcement)

Proof for Proposition 3. We must prove that for every formula $\varphi \in \text{sHML}_{\mathbf{nf}}$ if $(\!|\, \varphi, \Pi \,|\!) = m$ then $\mathsf{evtenf}(m, \varphi)$. Since $\text{sHML}_{\mathbf{nf}}$ is equally expressive as sHML we prove that for every $\varphi \in \text{sHML}_{\mathbf{nf}}$, if $(\!|\, \varphi, \Pi \,|\!)[s] \overset{t}{\Longrightarrow} m'[s']$ and $s' \vDash \mathit{after}(\varphi, t)$ then $m'[s'] \sim s'$. We also refer to Proposition 5 (Transparency) and Lemma 1, defined below, and whose proofs are provided in Appendix B.

Proposition 5 (Transparency). *For every state $s \in \text{Sys}$ and $\varphi \in \text{sHML}_{\boldsymbol{nf}}$, if $s \in [\![\varphi]\!]$ then $(\!|\, \varphi, \Pi \,|\!)[s] \sim s$.* □

Lemma 1. *For every formula $\varphi \in \text{sHML}_{\boldsymbol{nf}}$, state s and trace t, if $(\!|\, \varphi, \Pi \,|\!)[s] \overset{t}{\Longrightarrow} m'[s']$ then $\exists \psi \in \text{sHML}_{\boldsymbol{nf}} \cdot \psi = \mathit{after}(\varphi, t)$ and $(\!|\, \psi, \Pi \,|\!) = m'$.* □

Now, assume that

$$(\!|\, \varphi, \Pi \,|\!)[s] \overset{t}{\Longrightarrow} m'[s'] \tag{28}$$

$$s' \vDash \mathit{after}(\varphi, t) \tag{29}$$

and so from (28) and Lemma 1 we have that

$$\exists \psi \in \text{sHML}_{\mathbf{nf}} \cdot \psi = \mathit{after}(\varphi, t) \tag{30}$$

$$(\!|\, \mathit{after}(\varphi, t), \Pi \,|\!) = m' = (\!|\, \psi, \Pi \,|\!). \tag{31}$$

Hence, knowing (29) and (30), by Proposition 5 (Transparency) we conclude that $(\!|\, \mathit{after}(\varphi, t), \Pi \,|\!)[s'] \sim s'$ as required, and so we are done. □

A.4 Proving Theorem 2 (Optimal Enforcement)

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 $\textsc{DisTrn} \stackrel{\text{def}}{=} \{n \mid \text{if } ec(n) \subseteq \{\mathsf{DIS}\}\}$. To further simplify the proof for Theorem 2 we refer to the following lemmas.

Lemma 2. *For every $m \in \textsc{DisTrn}$ and explicit trace t_τ, $mc(m, t_\tau) = N$.* □

Lemma 3. *For every action α and monitor $m \in \textsc{DisTrn}$, if $m \xrightarrow{\alpha \blacktriangleright \alpha} m'$, $\mathit{enf}(m, \bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i)$ and $\{p_j, c_j\}(\alpha) = \sigma$ (for some $j \in I$) then $\mathit{enf}(m', \varphi_j \sigma)$.* □

Lemma 4. *For every monitor $m \in \textsc{DisTrn}$, whenever $\mathit{enf}(m, \bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i)$ and $m \xrightarrow{(\mathsf{a}!v) \blacktriangleright \bullet} m'$ then $\mathit{enf}(m', \bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i)$.* □

Lemma 5. *For every monitor $m \in \textsc{DisTrn}$, whenever $\mathit{enf}(m, \bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i)$ and $m \xrightarrow{\bullet \blacktriangleright (\mathsf{a}?v)} m'$ then $\mathit{enf}(m', \bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i)$.* □

Proof. As from Lemma 2 we know that for every $m \in \textsc{DisTrn}$, $mc(m, t_\tau) = N$, 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 $(\!|\, \varphi, \Pi \,|\!)[\mathit{sys}(\mu t'_\tau)] \xrightarrow{\mu} m'_\varphi[\mathit{sys}(t'_\tau)]$. Assume that

$$mc((\!|\, \varphi, \Pi \,|\!), \mu t'_\tau) = mc(m'_\varphi, t'_\tau) = N \tag{32}$$

which implies that

$$(\!|\, \varphi, \Pi \,|\!)[\mathsf{sys}(\mu t'_\tau)] \xrightarrow{\mu} m'_\varphi[\mathsf{sys}(t'_\tau)] \tag{33}$$

and also assume that

$$\mathsf{enf}(m, \varphi) \tag{34}$$

and that $s \xrightarrow{\mu t'_\tau}$. By the rules in our model we can infer that the reduction in (33) 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 (33) we know that $\mu = \tau$ and that

$$m'_\varphi = (\!|\, \varphi, \Pi \,|\!). \tag{35}$$

Since from (34) 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,

$$m[\mathsf{sys}(\tau t'_\tau)] \xrightarrow{\tau} m[\mathsf{sys}(t'_\tau)]. \tag{36}$$

Hence, by (32), (34) 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 (36) and the definition of mc, we conclude that $N \leq mc(m, \tau t'_\tau)$ as required.

– iDef: From (33) and rule iDef we know that $\mu = \mathsf{a}!v$, $(\!|\, \varphi, \Pi \,|\!) \not\xrightarrow{\mathsf{a}!v}$ 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 (33) and rule iTrnI we know that $\mu = \mathsf{a}?v$ and that

$$(\!|\, \varphi, \Pi \,|\!) \xrightarrow{(\mathsf{a}?v) \blacktriangleright (\mathsf{a}?v)} m'_\varphi. \tag{37}$$

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

$$\begin{aligned} &(\!|\, \varphi_\wedge = \textstyle\bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i, \Pi \,|\!) \\ &= \mathsf{rec}\, Y. \left(\sum_{i \in I} \begin{cases} \mathsf{dis}(p_i, c_i, Y, \Pi) & \textit{if } \varphi_i = \mathsf{ff} \\ \{p_i, c_i\}.(\!|\, \varphi_i, \Pi \,|\!) & \textit{otherwise} \end{cases} \right) + \mathsf{def}(\bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i) \\ &= \left(\sum_{i \in I} \begin{cases} \mathsf{dis}(p_i, c_i, (\!|\, \varphi_\wedge, \Pi \,|\!), \Pi) & \textit{if } \varphi_i = \mathsf{ff} \\ \{p_i, c_i\}.(\!|\, \varphi_i, \Pi \,|\!) & \textit{otherwise} \end{cases} \right) + \mathsf{def}(\bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i) \end{aligned} \tag{38}$$

Since normalized conjunctions are disjoint, *i.e.*, $\#_{i \in I} \{p_i, c_i\}$, from (38) we can infer that the identity reduction in (37) 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

$$\{p_j, c_j\}(\mathsf{a}?v) = \sigma. \tag{39}$$

Hence, knowing (37) and (39), by rule eTrn we know that $m'_\varphi = (\!|\, \varphi_j \sigma, \Pi \,|\!)$ and so by (32) we can infer that

$$mc(m'_\varphi, t'_\tau) = N \quad \textit{where } m'_\varphi = (\!|\, \varphi_j \sigma, \Pi \,|\!). \tag{40}$$

Since from (38) 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 (34). Therefore, we can deduce that

$$m \xrightarrow{(\mathsf{a}?v)\blacktriangleright(\mathsf{a}?v)} m' \quad \text{(for some } m') \tag{41}$$

and subsequently, knowing (41) 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

$$mc(m, (\mathsf{a}?v)t'_\tau) = mc(m', t'_\tau). \tag{42}$$

As by (34), (37), (39) and Lemma 3 we know that $\mathsf{enf}(m', \varphi_j\sigma)$, by (40) 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 (42) 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

$$mc(\!|\, \varphi, \Pi \,|\!), \mu t'_\tau) = 1 + M \tag{43}$$

$$\textit{where } M = mc(m'_\varphi, t'_\tau) \tag{44}$$

which implies that

$$(\!|\, \varphi, \Pi \,|\!)[\mathsf{sys}(\mu t'_\tau)] \xrightarrow{\mu'} m'_\varphi[\mathsf{sys}(t'_\tau)] \;\; \textit{where } \mu' \neq \mu \tag{45}$$

and also assume that

$$\mathsf{enf}(m, \varphi) \tag{46}$$

and that $s \xrightarrow{\mu t'_\tau}$. Since we only consider action disabling monitors, the μ' reduction of (45) can only be achieved via rules iDisO or iDisI. We thus explore both cases.

– iDisI: From (45) and by rule iDisI we have that $\mu = \mathsf{a}?v$ and $\mu' = \tau$ and that

$$(\!|\, \varphi, \Pi \,|\!) \xrightarrow{\bullet\blacktriangleright \mathsf{a}?v} m'_{\varphi}. \tag{47}$$

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 (47).
- $\varphi = \bigwedge_{i\in I} [\{p_i, c_i\}]\varphi_i$ 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

$$\begin{aligned}
&(\!|\, \varphi_\wedge = \textstyle\bigwedge_{i\in I} [\{p_i, c_i\}]\varphi_i, \Pi \,|\!) \\
&= \mathsf{rec}\, Y.\left(\sum_{i\in I} \begin{cases} \mathsf{dis}(p_i, c_i, Y, \Pi) & \text{if } \varphi_i = \mathsf{ff} \\ \{p_i, c_i\}.(\!|\, \varphi_i, \Pi \,|\!) & \text{otherwise} \end{cases}\right) + \mathsf{def}(\textstyle\bigwedge_{i\in I} [\{p_i, c_i\}]\varphi_i) \\
&= \left(\sum_{i\in I} \begin{cases} \mathsf{dis}(p_i, c_i, (\!|\, \varphi_\wedge, \Pi \,|\!), \Pi) & \text{if } \varphi_i = \mathsf{ff} \\ \{p_i, c_i\}.(\!|\, \varphi_i, \Pi \,|\!) & \text{otherwise} \end{cases}\right) + \mathsf{def}(\textstyle\bigwedge_{i\in I} [\{p_i, c_i\}]\varphi_i)
\end{aligned} \tag{48}$$

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 (48) we can deduce that the reduction in (47) 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

$$m'_{\varphi} = (\!|\, \textstyle\bigwedge_{i\in I} [\{p_i, c_i\}]\varphi_i, \Pi \,|\!) \tag{49}$$

$$p_j = (x)?(y) \text{ and } c_j\{\mathsf{a}/x\} \Downarrow \mathsf{true}. \tag{50}$$

Knowing (50) 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 (46) 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.,* $m \xrightarrow{\mathsf{a}?v}\!\!\!\!\!/\;\;$, or by additionally inserting a default value v, *i.e.,* $m \xrightarrow{\bullet\blacktriangleright(\mathsf{a}?v)} m'$. We explore both cases.

  * $m \xrightarrow{\mathsf{a}?v}\!\!\!\!\!/\;\;$: Since $\mathsf{sys}((\mathsf{a}?v)t'_\tau) \xrightarrow{\mathsf{a}?v} \mathsf{sys}(t'_\tau)$ and since $m \xrightarrow{\mathsf{a}?v}\!\!\!\!\!/\;\;$, by the rules in our model we know that for every action μ', $m[\mathsf{sys}((\mathsf{a}?v)t'_\tau)] \xrightarrow{\mu'}\!\!\!\!\!/\;\;$ 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 (43) 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

$$mc(m, (\mathsf{a}?v)t'_\tau) = 1 + mc(m', t'_\tau). \tag{51}$$

As by (46), (47) and Lemma 5 we infer that $\mathsf{enf}(m', \bigwedge_{i\in I}[\{p_i, c_i\}]\varphi_i)$, by (44), (51) 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 $M \leq mc(m', t'_\tau)$ and so from (43), (44) and (51) we conclude that $1+M \leq mc(m, (\mathsf{a}?v)t'_\tau)$ 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 4 is used instead of Lemma 5.

Case $mc(\!|\, \varphi, \Pi \,|\!), t_\tau)$ when $t_\tau \in \{\mu t'_\tau, \varepsilon\}$ and $(\!|\, \varphi, \Pi \,|\!)[\mathit{sys}(\mu t'_\tau)] \xrightarrow{\mu'}\!\!\!\!\!/\;\;$. Assume that

$$mc(\!|\, \varphi, \Pi \,|\!), t_\tau) = |\, t_\tau \,| \quad (\text{where } t_\tau \in \{\mu t'_\tau, \varepsilon\}) \tag{52}$$

$$(\!|\, \varphi, \Pi \,|\!)[\mathsf{sys}(\mu t'_\tau)] \xrightarrow{\mu'}\!\!\!\!\!/\;\; \tag{53}$$

$$\mathsf{enf}(m, \varphi) \tag{54}$$

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

– $t_\tau = \varepsilon$: This case holds trivially since by (52), (53) 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 (53) 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 (53) 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 (53).

- $\varphi = \bigwedge_{i\in I}[\{p_i,c_i\}]\varphi_i$ 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

$$\begin{aligned}
&(\!|\varphi_\wedge = \textstyle\bigwedge_{i\in I}[\{p_i,c_i\}]\varphi_i, \Pi|\!) \\
&= \mathsf{rec}\, Y.\left(\sum_{i\in I}\left\{\begin{array}{ll}\mathsf{dis}(p_i,c_i,Y,\Pi) & \textit{if } \varphi_i = \mathsf{ff}\\ \{p_i,c_i\}.(\!|\varphi_i,\Pi|\!) & \textit{otherwise}\end{array}\right.\right) + \mathsf{def}(\bigwedge_{i\in I}[\{p_i,c_i\}]\varphi_i) \qquad (55)\\
&= \left(\sum_{i\in I}\left\{\begin{array}{ll}\mathsf{dis}(p_i,c_i,(\!|\varphi_\wedge,\Pi|\!),\Pi) & \textit{if } \varphi_i = \mathsf{ff}\\ \{p_i,c_i\}.(\!|\varphi_i,\Pi|\!) & \textit{otherwise}\end{array}\right.\right) + \mathsf{def}(\bigwedge_{i\in I}[\{p_i,c_i\}]\varphi_i)
\end{aligned}$$

  Since $\mu = \mathsf{a}?v$, from (55) and by the definitions of dis and def we can infer that the only case when (53) 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 $v\in \textsc{Val}$, $\mathsf{mtch}(p_j,\mathsf{a}?v) = \sigma$ 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 $\textsc{Sys}_\Pi$, *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 subcase 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 $(\!|\varphi,\Pi|\!)[\mathsf{sys}(t_\tau)] \xrightarrow{\mu'} m'_\varphi[\mathsf{sys}(t_\tau)]$ can only be achieved via action enabling and rules iEnO and iEnI. □

B Auxiliary Proofs

In this section we give the omitted proofs for auxiliary lemmas defined in Appendix A.

B.1 Proving Auxiliary Lemmas for Proposition 3 (Eventual Transparency)

Proof for Lemma 1. We need to prove that for every formula $\varphi\in \mathrm{sHML}_{\mathbf{nf}}$, if we assume that $(\!|\varphi,\Pi|\!)[s] \stackrel{t}{\Rightarrow} m'[s']$ then there must exist some formula ψ, such that $\psi = \mathit{after}(\varphi,t)$ and $(\!|\psi,\Pi|\!) = m'$. This proof relies on the following lemma whose proof is given following the end of the current one.

Lemma 6. *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* $s \stackrel{u}{\Rightarrow} s'$ *and* $r = (\!|\bigwedge_{i\in I}[\{p_i,c_i\}]\varphi_i,\Pi|\!)[s']$.

We now 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

$$(\!|\, \varphi, \Pi \,|\!)[s] \xrightarrow{\tau} {}^* r \tag{56}$$

$$r \xrightarrow{\alpha} r' \tag{57}$$

$$r' \stackrel{u}{\Longrightarrow} m'[s']. \tag{58}$$

We now proceed by case analysis on φ.

- $\varphi \in \{\mathsf{ff}, X\}$: These cases do not apply since $(\!|\, \mathsf{ff}, \Pi \,|\!)$ and $(\!|\, X, \Pi \,|\!)$ do not yield a valid monitor.
- $\varphi = \mathsf{tt}$: Since $(\!|\, \mathsf{tt}, \Pi \,|\!) = \mathsf{id}$ we know that the τ-reductions in (56) 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 (57) is only possible via rule iTrn and so we also know that $s'' \xrightarrow{\alpha} {}^* s'''$ and $r' = (\!|\, \mathsf{tt}, \Pi \,|\!)[s''']$. Hence, by (58) and the *inductive hypothesis* we conclude that

  $$\exists \psi \in \mathrm{sHML}_{\mathbf{nf}} \cdot \psi = \mathit{after}(\mathsf{tt}, u) \tag{59}$$

  $$(\!|\, \psi, \Pi \,|\!) = m'. \tag{60}$$

  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 (59) we can conclude that $\psi = \mathit{after}(\mathsf{tt}, \alpha u)$ and so this case holdssince we also know (60).
- $\varphi = \bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i$ 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 $\mathsf{rec}\, Y. \sum_{i \in I} \begin{cases} \mathsf{dis}(p_i, c_i, Y, \Pi) & \mathit{if}\ \varphi = \mathsf{ff} \\ \{p_i, c_i\}.(\!|\, \varphi_i, \Pi \,|\!) & \mathit{otherwise} \end{cases}$ which can be unfolded into

  $$(\!|\, \bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i, \Pi \,|\!) = \sum_{i \in I} \begin{cases} \mathsf{dis}(p_i, c_i, m, \Pi) & \mathit{if}\ \varphi = \mathsf{ff} \\ \{p_i, c_i\}.(\!|\, \varphi_i, \Pi \,|\!) & \mathit{otherwise} \end{cases} \tag{61}$$

  and so by (56), (61) and Lemma 6 we conclude that $\exists s'' \cdot s \stackrel{u}{\Longrightarrow} s''$ and that

  $$r = (\!|\, \textstyle\bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i, \Pi \,|\!)[s''] . \tag{62}$$

  Hence, by (61) and (62) we know that the reduction in (57) 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

  $$r' = (\!|\, \varphi_j \sigma, \Pi \,|\!)[s'''] \tag{63}$$

  $$\mathsf{mtch}(p_j, \alpha) = \sigma \ \mathit{and}\ c_j \sigma \Downarrow \mathsf{true} \tag{64}$$

and so by (58), (63) and the *inductive hypothesis* we deduce that

$$\exists \psi \in \text{sHML}_{\mathbf{nf}} \cdot \psi = \mathit{after}(\varphi_j \sigma, u) \tag{65}$$
$$(\!| \psi, \Pi |\!) = m'. \tag{66}$$

Now since we know (64), by the definition of *after* we infer that

$$\begin{aligned} \mathit{after}(\textstyle\bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i, \alpha u) &= \mathit{after}(\mathit{after}(\textstyle\bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i, \alpha), u) \\ &= \mathit{after}(\varphi_j \sigma, u) \end{aligned} \tag{67}$$

and so from (65) and (67) we conclude that

$$\psi = \mathit{after}(\textstyle\bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i, \alpha u). \tag{68}$$

Hence, this case is done by (66) and (68).

– $\varphi = \mathsf{max}\, X.\psi$ and $X \in \mathbf{fv}(\psi)$: Since $\varphi = \mathsf{max}\, X.\psi$, by the syntactic rules of $\text{sHML}_{\mathbf{nf}}$ 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

$$\psi = \mathsf{max}\, Y_1 \ldots . \mathsf{max}\, Y_n . \textstyle\bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i. \tag{69}$$

where $\mathsf{max}\, Y_1 \ldots . \mathsf{max}\, Y_n .$ denotes an arbitrary number of fixpoint declarations, possibly none. Hence, knowing (69), by unfolding every fixpoint in $\mathsf{max}\, X.\psi$ we reduce the formula to

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

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

$$\exists \psi' \in \text{sHML}_{\mathbf{nf}} \cdot \psi' = \mathit{after}(\textstyle\bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i \{\ldots\}, \alpha u) \tag{70}$$
$$(\!| \psi', \Pi |\!) = m'. \tag{71}$$

From (69), (70) and the definition of *after* we can therefore conclude that

$$\exists \psi' \in \text{sHML}_{\mathbf{nf}} \cdot \psi' = \mathit{after}(\mathsf{max}\, X.\psi, \alpha u) \tag{72}$$

and so this case holds by (71) and (72).

□

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

Proof for Lemma 6. We must now prove that for every formula of the form $\bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i$ and 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 $r = (\!| \bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i, \Pi |\!)[s']$. We proceed by mathematical induction on the number of τ transitions.

Case 0 *transitions.* This case holds vacuously 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

$$(\!|\bigwedge_{i\in I}[\{p_i,c_i\}]\varphi_i, \Pi|\!)[s] \xrightarrow{\tau} r' \quad \text{(for some } r') \tag{73}$$

$$r' \xrightarrow{\tau}{}^{k} r. \tag{74}$$

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

$$(\!|\bigwedge_{i\in I}[\{p_i,c_i\}]\varphi_i, \Pi|\!) = \sum_{i\in I}\begin{cases}\mathsf{dis}(p_i,c_i,m,\Pi) & \textit{if } \varphi{=}\mathsf{ff}\\ \{p_i,c_i\}.(\!|\varphi_i,\Pi|\!) & \textit{otherwise}\end{cases} \tag{75}$$

and so from (75) we know that the τ-reduction in (73) can be the result of rules iAsy, iDisO or iDisI. We therefore inspect each case.

- iAsy: By rule iAsy, from (73) we can deduce that

$$\exists s''\cdot s \xrightarrow{\tau} s'' \tag{76}$$

$$r' = (\!|\bigwedge_{i\in I}[\{p_i,c_i\}]\varphi_i|\!)[s''] \tag{77}$$

  and so by (74), (77) and the *inductive hypothesis* we know that

$$\exists s', u\cdot s'' \stackrel{u}{\Longrightarrow} s' \textit{ and } r = (\!|\bigwedge_{i\in I}[\{p_i,c_i\}]\varphi_i|\!)[s']. \tag{78}$$

  Finally, by (76) and (78) we can thus conclude that $\exists s', u\cdot s \stackrel{u}{\Longrightarrow} s'$ and $r = (\!|\bigwedge_{i\in I}[\{p_i,c_i\}]\varphi_i|\!)[s']$.
- iDisI: By rule iDisI and from (73) we infer that

$$\exists s''\cdot s \xrightarrow{(\mathsf{a}?v)} s'' \tag{79}$$

$$(\!|\bigwedge_{i\in I}[\{p_i,c_i\}]\varphi_i, \Pi|\!) \xrightarrow{\bullet\blacktriangleright(\mathsf{a}?v)} m' \tag{80}$$

$$r' = m'[s''] \tag{81}$$

  and from (75) and by the definition of dis we can infer that the reduction in (80) 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

$$m' = (\!|\bigwedge_{i\in I}[\{p_i,c_i\}]\varphi_i, \Pi|\!). \tag{82}$$

  Hence, by (74), (81) and (82) we can apply the *inductive hypothesis* and deduce that

$$\exists s', u\cdot s'' \stackrel{u}{\Longrightarrow} s' \textit{ and } r = (\!|\bigwedge_{i\in I}[\{p_i,c_i\}]\varphi_i, \Pi|\!)[s'] \tag{83}$$

  so that by (79) and (83) 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 showing the proof for this case as it is very similar to that of case iDisI. □

Proof for Proposition 5 (Transparency). We need to prove that for every system s, if $s \in [\![\varphi]\!]$ then $m[s] \sim s$. Since $s \in [\![\varphi]\!]$ is analogous to $s \vDash \varphi$ we prove that relation $\mathcal{R} \stackrel{\text{def}}{=} \{ (s, (\!|\varphi, \Pi|\!)[s]) \mid s \vDash \varphi \}$ 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

$$s \vDash \mathsf{tt} \tag{84}$$

$$s \xrightarrow{\mu} s' \tag{85}$$

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

– $\mu = \tau$*:* Since $\mu = \tau$, we can apply rule iAsy on (85) and get that

$$(\!|\mathsf{tt}, \Pi|\!)[s] \xrightarrow{\tau} (\!|\mathsf{tt}, \Pi|\!)[s'] \tag{86}$$

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

$$(s', (\!|\mathsf{tt}, \Pi|\!)[s']) \in \mathcal{R} \tag{87}$$

as required. This means that this case is done by (86) and (87).

– $\mu = \alpha$*:* 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

$$(\!|\mathsf{tt}, \Pi|\!)[s] \xrightarrow{\alpha} (\!|\mathsf{tt}, \Pi|\!)[s'] \tag{88}$$

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

$$(s', (\!|\mathsf{tt}, \Pi|\!)[s']) \in \mathcal{R} \tag{89}$$

as required, and so this case is done by (88) and (89).

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

$$s \vDash \bigwedge_{i\in I} [\{p_i, c_i\}]\varphi_i \tag{90}$$

$$s \xrightarrow{\mu} s' \tag{91}$$

and by the definition of $\vDash$ and (90) we have that for every index $i \in I$ and action $\beta \in \text{Act}$,

$$\textit{if } s \xRightarrow{\beta} s' \textit{ and } \{p_i, c_i\}(\beta) = \sigma \textit{ then } s \vDash \bigwedge_{i\in I} [\{p_i, c_i\}]\varphi_i. \tag{92}$$

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

- $\mu = \tau$*:* Since $\mu = \tau$, we can apply rule iAsy on (91) and obtain

$$(\!| \bigwedge_{i\in I} [\{p_i, c_i\}]\varphi_i, \Pi |\!)[s] \xrightarrow{\tau} (\!| \bigwedge_{i\in I} [\{p_i, c_i\}]\varphi_i, \Pi |\!)[s'] \tag{93}$$

  as required. Since $\mu = \tau$, and since we know that sHML is τ-closed, from (90), (91) and Proposition 4, 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

$$(s', (\!| \bigwedge_{i\in I} [\{p_i, c_i\}]\varphi_i, \Pi |\!)[s']) \in \mathcal{R} \tag{94}$$

  as required. This subcase is therefore done by (93) and (94).
- $\mu = \alpha$*:* Since $\mu = \alpha$, from (91) we know that

$$s \xrightarrow{\alpha} s' \tag{95}$$

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

$$(\!| \varphi_\wedge, \Pi |\!) = \mathsf{rec}\, Y.\Big(\sum_{i\in I} \begin{cases} \mathsf{dis}(p_i, c_i, Y, \Pi) & \text{if } \varphi_i = \mathsf{ff} \\ \{p_i, c_i\}.(\!| \varphi_i, \Pi |\!) & \text{otherwise} \end{cases} \Big) + \mathsf{def}(\varphi_\wedge) \tag{96}$$

  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.,* $\forall i \in I \cdot \{p_i, c_i\}(\alpha) = \mathit{undef}$*):* Since none of the symbolic actions in (96) 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 (90). Hence, we infer that

$$(\!| \bigwedge_{i\in I} [\{p_i, c_i\}]\varphi_i, \Pi |\!)[s] \xrightarrow{\alpha} (\!| \mathsf{tt}, \Pi |\!)[s'] \quad (\text{since } \mathsf{id} = (\!| \mathsf{tt}, \Pi |\!)) \tag{97}$$

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

$$(s', (\!| \mathsf{tt}, \Pi |\!)[s']) \in \mathcal{R} \tag{98}$$

as required. This case is therefore done by (97) and (98).

- *One matching branch (i.e., $\exists j \in I \cdot \{p_j, c_j\}(\alpha) = \sigma$):* From (96) we can infer that the synthesised monitor can only disable the (input or output) actions that are defined by violating modal necessities. However, from (92) we also deduce that s is *incapable* of executing such an action as otherwise would contradict assumption (90). Hence, since we now assume that $\exists j \in I \cdot \{p_j, c_j\}(\alpha) = \sigma$, from (96) we deduce that this action can only be transformed by an identity transformation and so by rule eTrn we have that

$$\{p_j, c_j\}.(\!| \varphi_j, \Pi |\!) \xrightarrow{\alpha \blacktriangleright \alpha} (\!| \varphi_j \sigma, \Pi |\!). \tag{99}$$

By applying rules eSel, eRec on (99) and by (95), (96) and iTrnI/O (depending on whether α is an input or output action) we get that

$$(\!| \textstyle\bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i, \Pi |\!)[s] \xrightarrow{\alpha} (\!| \varphi_j \sigma, \Pi |\!)[s'] \tag{100}$$

as required. By (92), (95) 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

$$(s', (\!| \varphi_j \sigma, \Pi |\!)[s']) \in \mathcal{R} \tag{101}$$

as required. Hence, this subcase holds by (100) and (101).

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

$$s \xrightarrow{\mu} s' \tag{102}$$

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

$$s \vDash \varphi\{\mathsf{max}\, X.\varphi / X\}. \tag{103}$$

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

$$\varphi = \mathsf{max}\, Y_0 \ldots \mathsf{max}\, Y_n . \bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i \tag{104}$$

and so by (103), (104) and the definition of $\vDash$ we have that

$$s \vDash (\textstyle\bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i)\{\cdot\cdot\} \quad \textit{where} \tag{105}$$
$$\{\cdot\cdot\} = \{\mathsf{max}\, X.\varphi/X, (\mathsf{max}\, Y_0 \ldots \mathsf{max}\, Y_n .\, \textstyle\bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i)/Y_0, \ldots\}.$$

Since we know (102) and (105), from this point onwards the proof proceeds as per the previous case. We thus omit this part of the proof and immediately deduce that

$$\exists m' \cdot (\!|\, (\textstyle\bigwedge_{i\in I} [\{p_i, c_i\}]\varphi_i)\{\cdot\cdot\}, \Pi \,|\!)[s] \xrightarrow{\mu} (\!|\, m', \Pi \,|\!)[s'] \tag{106}$$

$$(s', (\!|\, m', \Pi \,|\!)[s']) \in \mathcal{R} \tag{107}$$

and so since $(\!|\, (\bigwedge_{i\in I} [\{p_i, c_i\}]\varphi_i)\{\cdot\cdot\}, \Pi \,|\!)$ synthesises the *unfolded equivalent* as per $(\!|\, \varphi\{\mathsf{max}\, X.\varphi/X\}, \Pi \,|\!)$, from (106) we can conclude that

$$\exists m' \cdot (\!|\, \varphi\{\mathsf{max}\, X.\varphi/X\}, \Pi \,|\!)[s] \xrightarrow{\mu} (\!|\, m', \Pi \,|\!)[s'] \tag{108}$$

as required, and so this case holds by (107) and (108).

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

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 $\nexists s \cdot s \vDash X$.

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

$$s \vDash \mathsf{tt} \tag{109}$$

$$(\!|\, \mathsf{tt}, \Pi \,|\!)[s] \xrightarrow{\mu} r' \tag{110}$$

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

- $\mu = \tau$*:* Since $\mu = \tau$, the transition in (110) can be performed via iDisI, iDisO or iAsy. We must therefore consider these cases.
  - iAsy*:* From rule iAsy and (110) 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 $s' \vDash \mathsf{tt}$ 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 (110) we know that: $r' = m'[s']$, $s \xrightarrow{\mathsf{a}?v} s'$ and that

    $$(\!|\, \mathsf{tt}, \Pi \,|\!) \xrightarrow{\bullet \blacktriangleright (\mathsf{a}?v)} m'. \tag{111}$$

    Since $(\!|\, \mathsf{tt}, \Pi \,|\!) = \mathsf{id}$ we can deduce that (111) 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 (110) can be performed either via iTrnI or iEnI. We consider both cases.
  - iEnI*:* This case also does not apply since if the transition in (110) 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 (110) we know that $r' = m'[s']$ such that

$$(\!|\, \mathsf{tt}, \Pi \,|\!) \xrightarrow{\mathsf{a}?v \blacktriangleright \mathsf{b}?w} m'. \tag{112}$$

$$s \xrightarrow{\mathsf{b}?w} s' \tag{113}$$

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 (112) 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 $s' \vDash \mathsf{tt}$, so that by the definition of $\mathcal{R}$ we conclude

$$(s', (\!|\, \mathsf{tt}, \Pi \,|\!)[s']) \in \mathcal{R}. \tag{114}$$

Hence, we are done by (113) and (114) since we know that $\mathsf{a}?v = \mathsf{b}?w$.

– $\mu = \mathsf{a}!v$*:* When $\mu = \mathsf{a}!v$, the transition in (110) 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

$$s \vDash \textstyle\bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i \tag{115}$$

$$(\!|\, \textstyle\bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i, \Pi \,|\!)[s] \xrightarrow{\mu} r'. \tag{116}$$

From (115) and by the definition of $\vDash$ we can deduce that

$$\forall i \in I, \beta \in \textsc{Act} \cdot \ \textit{if } s \stackrel{\beta}{\Longrightarrow} s' \textit{ and } \{p_i, c_i\}(\beta) = \sigma \textit{ then } s' \vDash \varphi_i\sigma \tag{117}$$

and from (116) and the definition of $(\!|-|\!)$ we have that

$$\Big(\mathsf{rec}\, Y.\Big(\sum_{i \in I} \begin{cases} \mathsf{dis}(p_i, c_i, Y, \Pi) & \text{if } \varphi_i = \mathsf{ff} \\ \{p_i, c_i\}.(\!|\, \varphi_i, \Pi \,|\!) & \text{otherwise} \end{cases}\Big) + \mathsf{def}(\bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i)\Big)[s'] \xrightarrow{\mu} r'. \tag{118}$$

From (118) we can deduce that the synthesised monitor can only disable an (input or output) action β when this satisfies a violating modal necessity. However, we also know that s is *unable* to perform such an action as otherwise it would contradict assumption (117). Hence, we can safely conclude that the synthesised monitor in (118) does *not* disable any (input or output) actions of s, and so by the definition of dis we conclude that

$$\forall \mathsf{a}?v, \mathsf{a}!v \in \textsc{Act}, s' \in \textsc{Sys} \cdot \begin{pmatrix} s \xrightarrow{\mathsf{a}?v} s' \ \textit{ implies } \ (\!|\, \textstyle\bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i, \Pi \,|\!) \xrightarrow{\bullet \blacktriangleright \mathsf{a}?w} \not\longrightarrow \text{ (for all } w) \ \textit{ and} \\ s \xrightarrow{\mathsf{a}!v} s' \ \textit{ implies } \ (\!|\, \textstyle\bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i, \Pi \,|\!) \xrightarrow{\mathsf{a}?v \blacktriangleright \bullet} \not\longrightarrow \end{pmatrix}. \tag{119}$$

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

– $\mu = \tau$*:* Since $\mu = \tau$, from (116) we know that

$$(\!| \bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i, \Pi |\!)[s] \xrightarrow{\tau} r' \tag{120}$$

The τ-transition in (120) can be the result of rules iAsy, iDisI or iDisO; we thus consider each eventuality.

- iAsy*:* As we assume that the reduction in (120) is the result of rule iAsy, we know that $r' = (\!| \bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i, \Pi |\!)[s']$ and that

$$s \xrightarrow{\tau} s' \tag{121}$$

as required. Also, since sHML is τ-closed, by (115), (121) and Proposition 4 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

$$(s', (\!| \bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i, \Pi |\!)[s']) \in \mathcal{R} \tag{122}$$

and so we are done by (121) and (122).

- iDisI*:* By assuming that reduction (120) results from iDisI, we have that $r' = m'[s']$ and that

$$(\!| \bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i, \Pi |\!) \xrightarrow{\bullet \blacktriangleright \mathsf{a}?v} m' \tag{123}$$

$$s \xrightarrow{\mathsf{a}?v} s' \tag{124}$$

By (119) and (124) we can, however, deduce that for every value w, $(\!| \bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i, \Pi |\!) \xrightarrow{\bullet \blacktriangleright \mathsf{a}?w} \not\rightarrow$. This contradicts with (123) and so this case does not apply.

- iDisO*:* As we now assume that the reduction in (120) results from iDisO, we have that $r' = m'[s']$ and that

$$s \xrightarrow{\mathsf{a}!v} s' \tag{125}$$

$$(\!| \bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i, \Pi |\!) \xrightarrow{\mathsf{a}!v \blacktriangleright \bullet} m'. \tag{126}$$

Again, this case does not apply since from (119) and (125) we can deduce that $(\!| \bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i, \Pi |\!) \xrightarrow{\mathsf{a}!v \blacktriangleright \bullet} \not\rightarrow$ which contradicts with (126).

– $\mu = \mathsf{a}?v$*:* When $\mu = \mathsf{a}?v$, the transition in (118) can be performed via rules iEnI or iTrnI, we consider both possibilities.

- iEnI*:* This case does not apply since from (118) and by the definition of $(\!| - |\!)$ we know that the synthesised monitor does not include action enabling transformations.

- iTrnI*:* By assuming that (118) is obtained from rule iTrnI we know that

$$\mathsf{rec}\,Y.\Big(\sum_{i\in I}\begin{cases}\mathsf{dis}(p_i,c_i,Y,\Pi) & \text{if } \varphi_i=\mathsf{ff}\\ \{p_i,c_i\}.(\!|\,\varphi_i,\Pi\,|\!) & \text{otherwise}\end{cases}\Big)+\mathsf{def}(\bigwedge_{i\in I}[\{p_i,c_i\}]\varphi_i)\xrightarrow{\mathsf{a}?v\blacktriangleright\mathsf{b}?w}m' \tag{127}$$

$$s\xrightarrow{\mathsf{b}?w}s' \tag{128}$$

$$r'=m'[s']. \tag{129}$$

Since from (119) we know that the synthesised monitor in (127) 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

$$\mathsf{a}?v=\mathsf{b}?w. \tag{130}$$

With the knowledge of (130), from (128) we can thus deduce that

$$s\xrightarrow{\mathsf{a}?v}s' \tag{131}$$

as required. Knowing (130) we can also deduce that in (127) 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 (127) and deduce that

$$\exists j\in I\cdot\{p_j,c_j\}(\mathsf{a}?v)=\sigma \tag{132}$$

$$m'=(\!|\,\varphi_j\sigma,\Pi\,|\!). \tag{133}$$

and so from (131), (132) and (117) 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 (129) and (133) we can conclude that

$$(s',r')\in\mathcal{R} \tag{134}$$

as required, and so this case is done by (131) and (134).

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

$$m'=(\!|\,\mathsf{tt},\Pi\,|\!). \tag{135}$$

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 (129) and (135) we can conclude that

$$(s', r') \in \mathcal{R} \tag{136}$$

as required. Hence this case is done by (131) and (136).

– $\mu = \mathsf{a}!v$*:* When $\mu = \mathsf{a}!v$, the transition in (118) 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

$$(\!|\, \mathsf{max}\, X.\varphi, \Pi \,|\!)[s] \xrightarrow{\mu} r' \tag{137}$$

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

$$s \vDash \varphi\{\mathsf{max}\, X.\varphi/X\}. \tag{138}$$

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

$$\varphi = \mathsf{max}\, Y_0. \ldots \mathsf{max}\, Y_n. \bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i \tag{139}$$

and so by (138), (139) and the definition of $\vDash$ we have that

$$\begin{gathered} s \vDash \textstyle\bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i\{\cdot\cdot\} \quad \textit{where} \\ \{\cdot\cdot\} = \{\mathsf{max}\, X.\varphi/X, (\mathsf{max}\, Y_0. \ldots \mathsf{max}\, Y_n. \textstyle\bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i)/Y_0, \ldots\}. \end{gathered} \tag{140}$$

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 (137) we know that

$$(\!|\, \textstyle\bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i\{\cdot\cdot\}, \Pi \,|\!)[s] \xrightarrow{\mu} r'. \tag{141}$$

Hence, since we know (140) and (141), 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. □

B.2 Proving Auxiliary Lemmas for Theorem 2 (Optimal Enforcement)

Proof for Lemma 2. We must prove that for every monitor $m \in \mathrm{DisTrn}$ and explicit trace t_τ, $mc(m, t_\tau) = N$. We proceed by induction on the length of t_τ.

Case $t_\tau = \varepsilon$. As we assume that $t_\tau = \varepsilon$, we must consider the following two cases:

– $\forall \mu \cdot m[\mathsf{sys}(\varepsilon)] \xrightarrow{\mu}$: This case holds trivially since by the definition of mc we have that $mc(m, \varepsilon) = |\, \varepsilon \,| = 0$.
– $\exists \mu, m', s \cdot m[\mathsf{sys}(\varepsilon)] \xrightarrow{\mu} m'[s]$: Since $\mathsf{sys}(\varepsilon) = \mathsf{nil} \xrightarrow{\mu}$, by the rules in our model we can infer that such a transition is only possible when the monitor *enables* an action β via rules iEnO and iEnI, and so this case does not apply since $m \notin \text{DisTrn}$.

Case $t_\tau = \mu t'_\tau$. Since we assume that $t_\tau = \mu t'_\tau$, we consider the following two cases:

– $\forall \mu \cdot m[\mathsf{sys}(\mu t'_\tau)] \xrightarrow{\mu}$: Since $\mu \in \{\tau, \mathsf{a}?v, \mathsf{a}!v\}$, we start by immediately excluding the cases when $\mu \in \{\tau, \mathsf{a}!v\}$ since rules iAsy and iDef prevent the monitor from blocking the composite system. However, in the case of inputs, $\mu = \mathsf{a}?v$, the monitor may block the input port by not reacting to the input, *i.e.*, $m \xrightarrow{\mathsf{a}?v}$. In this case, however, by the definition of mc we can still deduce that $mc(m, (\mathsf{a}?v)t_\tau) = |\, (\mathsf{a}?v)t_\tau \,|$ as required.
– $\exists \mu', m', s \cdot m[\mathsf{sys}(\mu t'_\tau)] \xrightarrow{\mu'} m'[s]$: When considering only the action disabling monitors defined in DisTrn, by the rules in our model we can infer that this instrumented reduction over action μ' can be attained via rules iDef, iAsy, iDisI, iDisO, iTrnI and iTrnO. We thus consider each case.
  - iDisO: Since by rule iDisO we know that $\mu = \mathsf{a}!v$, $\mu' = \tau$ and $s = \mathsf{sys}(t'_\tau)$, by the definition of mc we deduce that $mc(m, (\mathsf{a}!v)t'_\tau) = mc(m', t'_\tau) + 1$ and since by the *inductive hypothesis* we know that $mc(m', t'_\tau) = N$, then we conclude that $mc(m, (\mathsf{a}!v)t'_\tau) = N + 1$ as required.
  - iDisI: We elicit this proof as it is identical to that of iDisO.
  - iDef: Since by rule iDef we know that $\mu = \mu' = \mathsf{a}!v$, $m' = \mathsf{id}$ and $s = \mathsf{sys}(t'_\tau)$, by the definition of mc we deduce that $mc(m, (\mathsf{a}!v)t'_\tau) = mc(\mathsf{id}, t'_\tau)$ and since by the *inductive hypothesis* we know that $mc(\mathsf{id}, t'_\tau) = N$, then we can conclude that $mc(m, (\mathsf{a}!v)t'_\tau) = N$.
  - iAsy, iTrnO and iTrnI: We omit the proofs for these cases as they are very similar to that of case iDef, and so we are done. □

Proof for Lemma 3. The aim of this proof is to show that for every action α and monitors $m, m' \in \text{DisTrn}$, if $\mathsf{enf}(m, \bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i)$, $m \xrightarrow{\alpha \blacktriangleright \alpha} m'$ and $\{p_i, c_i\}(\alpha) = \sigma$ (for some $j \in I$) then we have that $\mathsf{senf}(m', \varphi_j \sigma)$ and $\mathsf{evtenf}(m', \varphi_j \sigma)$. We therefore start this proof by assuming that

$$m \xrightarrow{\alpha \blacktriangleright \alpha} m' \tag{142}$$

$$\exists j \in I \cdot \{p_i, c_i\}(\alpha) = \sigma. \tag{143}$$

and that $\mathsf{enf}(m, \bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i)$ which means that

$$\mathsf{senf}(m, \bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i) \stackrel{\text{def}}{=} \forall s \cdot m[s] \vDash \bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i \tag{144}$$

$$\begin{aligned} \mathsf{evtenf}(m, \textstyle\bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i) \stackrel{\text{def}}{=} \forall s, s'', t \cdot \mathit{if}\ m[s] \stackrel{t}{\Rightarrow} m''[s'']\ \mathit{and} \\ s'' \vDash \mathit{after}(\textstyle\bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i, t)\ \mathit{then}\ m''[s''] \sim s''. \end{aligned} \tag{145}$$

Since both (144) and (145) quantify on every s, we must consider the following two cases, namely, when $m[s]$ transitions over α and reach m', *i.e.,* $m[s] \xrightarrow{\alpha} m'[s']$ (for some system state s'), and when $m[s]$ does not reach m' via action α, *i.e.,* $m[s] \xrightarrow{\alpha} \not\rightarrow m'[s']$.

- $m[s] \xrightarrow{\alpha} \not\rightarrow m'[s']$: This case does not apply since, as stated by assumption (142), we only consider the cases where the instrumented system causes the monitor to perform the identity transformation of (142) via rules iTrnI when $\alpha{=}\mathsf{a}?v$ and iTrnO when $\alpha{=}\mathsf{a}!v$.
- $m[s] \xrightarrow{\alpha} m'[s']$: Since $m[s] \xrightarrow{\alpha} m'[s']$, from (144), (143) and by the definition of $\vDash$ we get that

$$\mathsf{senf}(m', \varphi_j\sigma) \stackrel{\text{def}}{=} \forall s' \cdot m'[s'] \vDash \varphi_j\sigma \tag{146}$$

as required. Now, lets assume that

$$\forall s''', u \cdot m'[s'] \stackrel{u}{\Longrightarrow} m'''[s'''] \tag{147}$$

$$s''' \vDash \mathit{after}(\varphi_j\sigma, u) \tag{148}$$

and since $m[s] \xrightarrow{\alpha} m'[s']$ when combined with (147) we know that $m[s] \stackrel{\alpha u}{\Longrightarrow} m'''[s''']$ and so from (145) and (148) we can deduce that

$$m'''[s'''] \sim s'''. \tag{149}$$

Hence, from assumptions (147), (148) and conclusion (149) we can introduce the implication and conclude that

$$\mathsf{evtenf}(m', \varphi_j\sigma) \stackrel{\text{def}}{=} \forall s', s'', u \cdot \ \mathit{if}\ m'[s'] \stackrel{u}{\Longrightarrow} m'''[s'''] \ \mathit{and}\ s''' \vDash \mathit{after}(\varphi_j\sigma, u) \ \mathit{then}\ m'''[s'''] \sim s''' \tag{150}$$

and so we are done by (146), (150) and the definition of enf. □

Proof for Lemma 4. In this proof we show that for every action α and monitors $m, m' \in \textsc{DisTrn}$, if $\mathsf{senf}(m, \bigwedge_{i\in I}[\{p_i, c_i\}]\varphi_i)$, $\mathsf{evtenf}(m, \bigwedge_{i\in I}[\{p_i, c_i\}]\varphi_i)$ and $m \xrightarrow{(\mathsf{a}!v)\blacktriangleright\bullet} m'$ then $\mathsf{senf}(m', \bigwedge_{i\in I}[\{p_i, c_i\}]\varphi_i)$ and $\mathsf{evtenf}(m', \bigwedge_{i\in I}[\{p_i, c_i\}]\varphi_i)$. We therefore start this proof by assuming that

$$m \xrightarrow{(\mathsf{a}!v)\blacktriangleright\bullet} m' \tag{151}$$

and that $\mathsf{enf}(m, \bigwedge_{i\in I}[\{p_i, c_i\}]\varphi_i)$, which means that

$$\mathsf{senf}(m, \textstyle\bigwedge_{i\in I}[\{p_i, c_i\}]\varphi_i) \stackrel{\text{def}}{=} \forall s \cdot m[s] \vDash \bigwedge_{i\in I}[\{p_i, c_i\}]\varphi_i \tag{152}$$

$$\mathsf{evtenf}(m, \textstyle\bigwedge_{i\in I}[\{p_i, c_i\}]\varphi_i) \stackrel{\text{def}}{=} \forall s, s'', t \cdot \mathit{if}\ m[s] \stackrel{t}{\Longrightarrow} m''[s''] \ \mathit{and}\ s'' \vDash \mathit{after}(\textstyle\bigwedge_{i\in I}[\{p_i, c_i\}]\varphi_i, t)\ \mathit{then}\ m''[s''] \sim s'' \tag{153}$$

We now consider the following two cases, namely, when $m[s]$ transitions over τ and reaches m', *i.e.,* $m[s] \xrightarrow{\tau} m'[s']$ (for some arbitrary state s'), and when $m[s]$ does not reach m' via action τ, *i.e.,* $m[s] \xrightarrow{\tau} \not\rightarrow m'[s']$.

– $m[s] \stackrel{\tau}{\not\longrightarrow} m'[s']$: This case does not apply since, as stated by assumption (151), we only consider the cases where the instrumented system causes the monitor to perform the suppression transformation of (151) via rule iDisO.
– $m[s] \stackrel{\tau}{\longrightarrow} m'[s']$: Since $m[s] \stackrel{\tau}{\longrightarrow} m'[s']$, from (152) and by Proposition 4 we deduce that

$$\mathsf{senf}(m', \bigwedge_{i\in I} [\{p_i, c_i\}]\varphi_i) \stackrel{\text{def}}{=} \forall s' \cdot m'[s'] \vDash \bigwedge_{i\in I} [\{p_i, c_i\}]\varphi_i \tag{154}$$

as required. We now assume that

$$\forall s''', u \cdot m'[s'] \stackrel{u}{\Longrightarrow} m'''[s'''] \tag{155}$$
$$s''' \vDash \mathit{after}(\bigwedge_{i\in I} [\{p_i, c_i\}]\varphi_i, u) \tag{156}$$

and since $m[s] \stackrel{\tau}{\longrightarrow} m'[s']$, by (155) and the definition of $\stackrel{u}{\Longrightarrow}$ we have that $m[s] \stackrel{u}{\longrightarrow} m'''[s''']$ and so from (153) and (156) we can deduce that

$$m'''[s'''] \sim s'''. \tag{157}$$

Hence, from assumptions (155), (156) and conclusion (157) we can introduce the implication and conclude that

$$\begin{aligned}\mathsf{evtenf}(m', \bigwedge_{i\in I} [\{p_i, c_i\}]\varphi_i) \stackrel{\text{def}}{=} \forall s', s'', u \cdot\ & \mathit{if}\ m'[s'] \stackrel{u}{\Longrightarrow} m'''[s'''] \ \mathit{and} \\ & s''' \vDash \mathit{after}(\bigwedge_{i\in I} [\{p_i, c_i\}]\varphi_i, u)\ \mathit{then}\ m'''[s'''] \sim s'''\end{aligned} \tag{158}$$

and so we are done by (154) and (158). □

Proof for Lemma 5. We elide the proof for this lemma as it is very similar to Lemma 4. In fact, it can be easily derived by replacing the references to rule iDisO and the assumption that $m \xrightarrow{(\mathsf{a}!v)\blacktriangleright\bullet} m'$ from Lemma 4, by rule iDisI and assumption $m \xrightarrow{\bullet\blacktriangleright(\mathsf{a}?v)} m'$ respectively. □