Synthesising Correct Concurrent Runtime Monitors (Extended Abstract)

Adrian Francalanza[1] and Aldrin Seychell[1]

Computer Science, ICT, University of Malta.
{afra1,asey0001}@um.edu.mt

Abstract. We study the correctness of automated synthesis for concurrent monitors. We adapt sHML, a subset of the Hennessy-Milner logic with recursion, to specify safety properties of Erlang programs, and define an automated translation from sHML formulas to Erlang monitors so as to detect formula violations at runtime. We then formalise monitor correctness for our concurrent setting and describe a technique that allows us to prove monitor correctness in stages; this technique is used to prove the correctness of our automated monitor synthesis.

Keywords: runtime verification, monitor synthesis, concurrency, actors

1 Introduction

Runtime Verification (RV) [3], is a lightweight verification technique for determining whether the *current system run* observes a correctness property. Two requirements are crucial for the adoption of this technique. First, runtime *monitor overheads* need to be kept to a minimum so as not to degrade system performance. Second, instrumented monitors need to form part of the trusted computing base of a system by adhering to an agreed notion of *monitor correctness*; amongst other things, this normally includes a guarantee that runtime checking corresponds (in some sense) to the property being checked for. Monitor overheads and correctness are occasionally conflicting concerns. For instance, in order to lower monitoring overheads, engineers increasingly use *concurrent monitors* [11, 22, 28] so as to exploit better the underlying parallel and distributed architectures pervasive to today's computers. However concurrent monitors are also more susceptible to elusive errors such as non-deterministic monitor behaviour, deadlocks or livelocks which may, in turn, affect their correctness.

Ensuring monitor correctness is non-trivial. One prominent obstacle is the fact that system properties are typically specified using one formalism, *e.g.*, a high-level logic, whereas the respective monitors checking these properties are described using another formalism, *e.g.*, a programming language—this makes it hard to ascertain the semantic correspondence between the two descriptions. *Automated monitor synthesis* can mitigate this problem by standardising the translation from the property logic to the monitor formalism. It also gives more scope for a formal treatment of monitor correctness.

In this work, we investigate the correctness of *synthesised* monitors in a *concurrent setting*, whereby (*i*) the system executes concurrently with the synthesised monitor (*ii*) the system and the monitor themselves consist of concurrent sub-systems and sub-monitors. Previous work on correct monitor synthesis[17, 27, 4] abstracts away from the internal working of a system, representing it as a string of events/states (execution trace). It also focusses on a logic that is readily amenable to runtime analysis, namely Linear Temporal Logic (LTL)[8]. Moreover, it expresses synthesis in terms of *abstract* or *single-threaded* monitors—using pseudocode or automata—executing *wrt.* such trace. By contrast, we strive towards a more intensional formal definition of online correctness for synthesised concurrent monitors whereby, for arbitrary property φ, the synthesised monitor M_φ running concurrently *wrt.* some system S (denoted as $S \parallel M_\varphi$) observes the following condition:

$$S \text{ violates } \varphi \text{ in the current execution} \qquad \textit{iff} \qquad S \parallel M_\varphi \text{ detects the violation} \qquad (1)$$

The setting described in (1) brings to the fore a number of additional issues:

(i) Apart from the formal semantics of the source logic (used to specify the property φ), we also require a formal semantics for the target languages of *both* the system and the monitor executing in parallel, *i.e.*, $S \parallel M_\varphi$. In most cases, the latter may not always be available.

(ii) A property logic semantics is often defined over *systems* rather than *traces*, which may not lend itself well to the formulation of correctness runtime analysis outlined in condition (1) above. In the case of concurrent systems, this aspect is accentuated by the fact that systems may behave non-deterministically and typically have multiple execution paths as a result of different thread interleavings scheduled at runtime.

(iii) Concurrent monitors may also have multiple execution paths. Condition (1) thus requires stronger guarantees than those for single-threaded monitors so as to ensure that *all* these paths correspond to an appropriate runtime check of system property being monitored. Stated otherwise, correct concurrent monitors must *always* detect violations, irrespective of their runtime interleaving.

(iv) Online monitor correctness needs to ensure that monitor execution cannot be interfered by the system, and viceversa. Whereas adequate monitor instrumentation typically prevents direct interferences, condition (1) must consider indirect interferences such as system divergences [25, 18], *i.e.*, infinite internal looping making the system unresponsive, which may prevent the monitors from progressing.

(v) Ensuring correctness along the lines of condition (1) can be quite onerous because *every* execution path of the monitor running concurrently with the monitored system, $S \parallel M_\varphi$, needs to be analysed so as to ensure consistent detections along every thread interleaving. Consequently, one needs to devise scalable techniques facilitating monitor correctness analysis.

We conduct our study in terms of actor-based [19] concurrent monitors written in Erlang [7, 2], an industry strength language for constructing fault-tolerant systems; we also restrict ourselves to the monitoring of systems written in the same language. We limit ourselves to *reactive properties* describing *system interactions with the environment* and focus on the synthesis of *asynchronous* monitors, performing runtime analysis through the Erlang Virtual Machine (EVM)'s tracing mechanism. Despite the typical

Actor Systems, Expressions, Values and Patterns

$$\begin{array}{rcl}
A, B, C \in \textsc{Actr} & ::= & i[e \triangleleft q]^m \mid A \parallel B \mid (\nu\, i)A \qquad\qquad q, r \in \textsc{MBox} \;::=\; \epsilon \mid v : q \\
e, d \in \textsc{Exp} & ::= & v \mid \mathsf{self} \mid e!d \mid \mathsf{rcv}\, g\, \mathsf{end} \mid e(d) \mid \mathsf{spw}\, e \mid \mathsf{case}\, e\, \mathsf{of}\, g\, \mathsf{end} \mid x = e, d \mid \ldots \\
v, u \in \textsc{Val} & ::= & x \mid i \mid a \mid \mu y.\lambda x.e \mid \{v, \ldots, v\} \mid l \mid \mathsf{exit} \mid \ldots \qquad l, k \in \textsc{Lst} \;::=\; \mathsf{nil} \mid v : l \\
p, o \in \textsc{Pat} & ::= & x \mid i \mid a \mid \{p, \ldots, p\} \mid \mathsf{nil} \mid p : x \mid \ldots \qquad g, f \in \textsc{PLst} \;::=\; \epsilon \mid p \rightarrow e; g
\end{array}$$

Fig. 1: Erlang Syntax

drawbacks associated with asynchrony, *e.g.*, late detections, our monitoring setup is in line with the asynchrony advocated by the actor concurrency model, which facilitates scalable coding techniques such as fail-fast design patterns [7]. Asynchronous monitoring has also been used in other RV tools, *e.g.*, [9, 12], and has proved to be less intrusive and easier to instrument than synchronous monitoring setups. It is also tends to be more feasible when monitoring distributed systems[14, 11]. More importantly, though, we expect most of the issues investigated to carry over in straightforward fashion to purely synchronous settings.

As an expository logic for describing reactive properties, we consider an adaptation of sHML [1]—a syntactic subset of the more expressive logic μ-calculus, describing safety, *i.e.*, monitorable[21], properties. Our choice for this logic was, in part, motivated by the fact that the full μ-calculus had already been adapted to describe Erlang program behaviour in [16], albeit for model-checking purposes. Given the usual drawbacks associated with full-blown model checking, our work contributes towards an investigation of lightweight verification techniques for μ-calculus properties of Erlang programs. More importantly, though, it illustrates well the correctness monitoring issues arising in actual implementations, as discussed earlier for (1).

The rest of the paper is structured as follows. Sec. 2 discusses the formal semantics of our systems and monitor target language. Sec. 3 discusses reformulations to the logic facilitating the formulation of monitor correctness, discussed later in Sec. 4. Sec. 5 describes a synthesis algorithm for the logic and a tool built using the algorithm. Subsequently, Sec. 6 proves the correctness of this monitor synthesis. Sec. 7 concludes.

2 The Language

We require a formal semantics for both our monitor-synthesis target language, and the systems we intend to monitor. We partially address this problem by expressing both in terms of the same language, *i.e.*, Erlang, thus only requiring one semantics. However, we still need to describe the Erlang tracing semantics we intend to use for our asynchronous monitoring. Although Erlang semantic formalisations exist, *e.g.*, [30, 16, 6], none describe this tracing mechanism. We therefore define a calculus—following [30, 16]—for modelling the *tracing semantics* of a (Turing-complete) subset of the Erlang language (we leave out distribution, process linking and fault-trapping mechanisms).

Figure 1 outlines the language syntax, assuming disjoint denumerable sets of process/actor identifiers $i, j, h \in \textsc{Pid}$, atoms $a, b \in \textsc{Atom}$, and variables $x, y, z \in \textsc{Var}$. An executing Erlang program is made up of a *system of actors*, Actr, composed in *parallel*, $A \parallel B$, where some identifiers are local (scoped) to subsystems of actors, and thus not known to the environment, *e.g.*, i in a system $A \parallel (\nu\, i)B$. Individual actors, denoted as $i[e \triangleleft q]^m$, are uniquely identified by an identifier, i, and consist of an expression, e, executing *wrt.* a local mailbox, q (denoted as a list of values). Actor *expressions* typically consist of a sequence of variable binding $x_i = e_i,$ terminated by an expression, e_{final}:

$$x_1 = e_1, \ \ldots, x_n = e_n, \ e_{\text{final}}$$

An expression e_i in a binding $x_i = e_i, e_{i+1}$ is expected to evaluate to a value, v, which is then bound to x_i in the continuation expression e_{i+1}. When instead e_i generates an exception, exit, it aborts subsequent computations[1] in e_{i+k}. Apart from bindings, expressions may also consist of self references (to the actor's own identifier), self, outputs to other actors, $e_1!e_2$, pattern-matching inputs from the mailbox, $\mathsf{rcv}\, g\, \mathsf{end}$, or pattern-matching for case-branchings, $\mathsf{case}\, e\, \mathsf{of}\, g\, \mathsf{end}$ (where g is a list of expressions guarded by patterns, $p_i \rightarrow e_i$), function applications, $e_1(e_2)$, and actor-spawning, $\mathsf{spw}\, e$, amongst others. Values consist of variables, x, process ids, i, recursive functions,[2] $\mu y.\lambda x.e$, tuples $\{v_1, \ldots, v_n\}$ and lists, l, amongst others.

Remark 1. We write $\lambda x.e$ and d, e for $\mu y.\lambda x.e$ and $y = d, e$ *resp.* when $y \notin \mathrm{fv}(e)$. In $p \rightarrow e$, we replace x in p with $\_$ whenever $x \notin \mathrm{fv}(e)$. We write $\mu y.\lambda(x_1, \ldots x_n).e$ for $\mu y.\lambda x_1. \ldots \lambda x_n.e$. We elide mailboxes, $i[e]$, when empty, $i[e \triangleleft \epsilon]$, or when they do not change in the transition rules that follow.

Specific to our formalisation, we also subject each individual actor, $i[e \triangleleft q]^m$, to a *monitoring-modality*, $m, n \in \{\circ, \bullet, *\}$, where $\circ$, $\bullet$ and $*$ denote *monitored*, *unmonitored* and *tracing* actors *resp.* Modalities play a crucial role in our language semantics, defined as a labelled transition system over systems, $A \xrightarrow{\gamma} B$, where actions $\gamma \in \textsc{Act}_\tau$, include bound *output* labels, $(\tilde{h})i!v$, and *input* labels, $i?v$ and a distinguished *internal* label, τ. In line with the reactive properties we consider later, our formalisation only traces system interactions with the environment (send and receive messages) from *monitored* actors. Thus, whereas unmonitored, $\bullet$, and tracing, $*$, actors have standard input and output transition rules

$$\textsc{SndU}\,\frac{m \in \{\bullet, *\}}{j[i!v \triangleleft q]^m \xrightarrow{i!v} j[v \triangleleft q]^m} \qquad \textsc{RcvU}\,\frac{m \in \{\bullet, *\}}{i[e \triangleleft q]^m \xrightarrow{i?v} i[e \triangleleft q{:}v]^m}$$

actors with a monitored modality, $\circ$, *i.e.*, actors j and i in rules SndM and RcvM below, produce a residual message reporting the send and receive interactions ($\{\mathsf{sd}, i, v\}$ and $\{\mathsf{rv}, i, v\}$ *resp.*) at the tracer's mailbox *i.e.*, actor h with modality $*$ in the rules below; this models closely the tracing mechanism offered by the Erlang Virtual Machine (EVM) [7]. In our target language, the list of report messages at the tracer's mailbox constitutes

[1] Because of exit exceptions, variable bindings cannot be encoded as function applications.
[2] The preceding μy denotes the binder for function self-reference.

the system trace to be used for asynchronous monitoring.

$$\textsc{SndM}\ \frac{}{j[i!v \triangleleft q]^{\circ} \parallel h[d \triangleleft r]^{*} \xrightarrow{i!v} j[v \triangleleft q]^{\circ} \parallel h[d \triangleleft r{:}\{\mathsf{sd}, i, v\}]^{*}}$$

$$\textsc{RcvM}\ \frac{}{i[e \triangleleft q]^{\circ} \parallel h[d \triangleleft r]^{*} \xrightarrow{i?v} i[e \triangleleft q{:}v]^{\circ} \parallel h[d \triangleleft r{:}\{\mathsf{rv}, i, v\}]^{*}}$$

Our LTS semantics assumes *well-formed* actor systems, whereby every actor identifier is *unique*; it is termed to be a *tracing semantics* because a distinguished *tracer* actor, identified by the monitoring modality $*$, receives messages recording external communication events by *monitored* actors. Formally, we write $A \xrightarrow{\gamma} B$ in lieu of $\langle A, \gamma, B\rangle \in\longrightarrow$, the least ternary relation satisfying the rules in Fig. 2. These rules employ *evaluation contexts*, denoted as C (described below) specifying which subexpressions are active. For instance, an expression is only evaluated when at the top level variable binding, $x = C, e$ or when the expression denoting the destination of an output has evaluated to a value, $v!C$; the other cases are also fairly standard.[3] We denote the application of a context C to an expression e as $C[e]$.

$$C \quad ::= \quad [-] \mid C!e \mid v!C \mid C(e) \mid v(C) \mid \mathsf{case}\, C\, \mathsf{of}\, g\, \mathsf{end} \mid x = C, e \mid \ldots$$

Communication in actor systems happens in two stages: an actor receives messages, keeping them in order in its mailbox, and then selectively reads them at a later stage using pattern matching—rules Rd1 and Rd2 describe how mailbox messages are traversed in order to find the first one matching a pattern in the pattern list g, releasing the respective guarded expression e as a result (see Appendix A). We choose only to record *external* communication at tracer processes, *i.e.*, between the system and the environment, and do not trace internally communication between actors within the system, irrespective of their modality (see Com); structural equivalence rules, $A \equiv B$, are employed to simplify the presentation of our rules—see rule Str and the corresponding structural rules. In Par, the side-condition enforces the *single-receiver* property, inherent to actor systems; for instance, it prevents a transition with an action $j!v$ when actor j is part of the actor system B. Finally, spawned actors inherit monitorability when launched by a monitored actor, but are launched as unmonitored otherwise (see Spw). The rest of the transition rules are fairly standard; consult [15] for details.

Remark 2. Our tracing semantics sits at higher level of abstraction than that offered by the EVM [7] because trace entries typically contain more information. For instance, the EVM records internal communication between monitored actors, as an output trace entry *immediately followed by* the corresponding input trace entry; we here describe *sanitised* traces whereby consecutive matching trace entries are filtered out.

Example 1 (Non-deterministic behaviour). Our systems exhibit non-deterministic behaviour through either internal or external choices [23, 18]. Consider the actor system:

$$A \triangleq (\nu\, j_1, j_2, h)(\, i[\mathsf{rcv}\, x \rightarrow \mathsf{obs}!x\, \mathsf{end} \triangleleft \epsilon]^{\circ} \parallel j_1[i!v]^{\circ} \parallel j_2[i!u]^{\circ} \parallel h[e \triangleleft q]^{*}\,)$$

[3] In our formalisation, expressions are not allowed to evaluate under a spawn context, $\mathsf{spw}\,[-]$. This differs from standard Erlang semantics but allows a lightweight description of function application spawning; an adjustment in line with Erlang spawning would be straightforward.

$$\textsc{SndM}\;\frac{}{j[C[i!v] \triangleleft q]^{\circ} \parallel h[d \triangleleft r]^{*} \xrightarrow{i!v} j[C[v] \triangleleft q]^{\circ} \parallel h[d \triangleleft (r{:}\{\mathsf{sd}, i, v\})]^{*}}$$

$$\textsc{RcvM}\;\frac{\mathrm{fv}(v) = \emptyset}{i[e \triangleleft q]^{\circ} \parallel h[d \triangleleft r]^{*} \xrightarrow{i?v} i[e \triangleleft q{:}v]^{\circ} \parallel h[d \triangleleft (r{:}\{\mathsf{rv}, i, v\})]^{*}}$$

$$\textsc{SndU}\;\frac{m \in \{\bullet, *\}}{j[C[i!v] \triangleleft q]^{m} \xrightarrow{i!v} j[C[v] \triangleleft q]^{m}} \qquad \textsc{RcvU}\;\frac{m \in \{\bullet, *\} \qquad \mathrm{fv}(v) = \emptyset}{i[e \triangleleft q]^{m} \xrightarrow{i?v} i[e \triangleleft q{:}v]^{m}}$$

$$\textsc{Scp}\;\frac{A \xrightarrow{\gamma} B}{(\nu\, j)A \xrightarrow{\gamma} (\nu\, j)B}\; j \notin (\mathrm{obj}(\gamma) \cup \mathrm{subj}(\gamma))$$

$$\textsc{Opn}\;\frac{A \xrightarrow{(\tilde{h})i!v} B}{(\nu\, j)A \xrightarrow{(j,\tilde{h})i!v} B}\; i \neq j,\, j \in \mathrm{subj}((\tilde{h})i!v)$$

$$\textsc{Com}\;\frac{}{j[C[i!v] \triangleleft q]^{m} \parallel i[e \triangleleft q]^{n} \xrightarrow{\tau} j[C[v] \triangleleft q]^{m} \parallel i[e \triangleleft q{:}v]^{n}}$$

$$\textsc{Par}\;\frac{A \xrightarrow{\gamma} A'}{A \parallel B \xrightarrow{\gamma} A' \parallel B}\,\mathrm{obj}(\gamma) \cap \mathrm{fId}(B) = \emptyset \qquad \textsc{Rd1}\;\frac{\mathrm{mtch}(g, v) = e}{i[C[g] \triangleleft (v : q)]^{m} \xrightarrow{\tau} i[C[e] \triangleleft q]^{m}}$$

$$\textsc{Rd2}\;\frac{\mathrm{mtch}(g, v) = \bot \qquad i[C[\mathsf{rcv}\; g\; \mathsf{end}] \triangleleft q]^{m} \xrightarrow{\tau} i[C[e] \triangleleft r]^{m}}{i[C[\mathsf{rcv}\; g\; \mathsf{end}] \triangleleft (v : q)]^{m} \xrightarrow{\tau} i[C[e] \triangleleft (v : r)]^{m}}$$

$$\textsc{Cs1}\;\frac{\mathrm{mtch}(g, v) = e}{i[C[\mathsf{case}\; v\; \mathsf{of}\; g\; \mathsf{end}]]^{m} \xrightarrow{\tau} i[C[e]]^{m}} \qquad \textsc{Cs2}\;\frac{\mathrm{mtch}(g, v) = \bot}{i[C[\mathsf{case}\; v\; \mathsf{of}\; g\; \mathsf{end}]]^{m} \xrightarrow{\tau} i[C[\mathsf{exit}]]^{m}}$$

$$\textsc{Ass}\;\frac{v \neq \mathsf{exit}}{i[C[x = v, e]]^{m} \xrightarrow{\tau} i[C[e\{{}^{v}\!/\!{}_{x}\}]]^{m}} \qquad \textsc{Ext}\;\frac{}{i[C[x = \mathsf{exit}, e]]^{m} \xrightarrow{\tau} i[C[\mathsf{exit}]]^{m}}$$

$$\textsc{App}\;\frac{}{i[C[\mu y.\lambda x.e\,(v)]]^{m} \xrightarrow{\tau} i[C[e\{{}^{\mu y.\lambda x.e}\!/\!{}_{y}\}\{{}^{v}\!/\!{}_{x}\}]]^{m}} \qquad \textsc{Slf}\;\frac{}{i[C[\mathsf{self}]]^{m} \xrightarrow{\tau} i[C[i]]^{m}}$$

$$\textsc{Spw}\;\frac{(m = \circ = n) \text{ or } (n = \bullet)}{i[C[\mathsf{spw}\; e] \triangleleft q]^{m} \xrightarrow{\tau} (\nu\, j)(i[C[j] \triangleleft q]^{m} \parallel j[e \triangleleft \epsilon]^{n})} \qquad \textsc{Str}\;\frac{A \equiv A' \xrightarrow{\gamma} B' \equiv B}{A \xrightarrow{\gamma} B}$$

$$\textsc{sCom}\;\frac{}{A \parallel B \equiv B \parallel A} \qquad \textsc{sAss}\;\frac{}{(A \parallel B) \parallel C \equiv A \parallel (B \parallel C)} \qquad \textsc{sCtxP}\;\frac{A \equiv B}{A \parallel C \equiv B \parallel C}$$

$$\textsc{sExt}\;\frac{i \notin \mathrm{fId}(A)}{A \parallel (\nu\, i)B \equiv (\nu\, i)(B \parallel A)} \qquad \textsc{sSwp}\;\frac{}{(\nu\, i)(\nu j)A \equiv (\nu j)(\nu\, i)A} \qquad \textsc{sCtxS}\;\frac{A \equiv B}{(\nu i)A \equiv (\nu i)B}$$

Fig. 2: Erlang Semantics for Actor Systems

Actors j_1, j_2 and h are local, thus not visible to the environment. The monitored actor i may receive value v from actor j_1, read it from its mailbox, and then output it to some environment actor obs, while recording this *external* output at h's mailbox (the tracer).

$$A \xrightarrow{\tau} \cdot \xrightarrow{\tau} \cdot \xrightarrow{\mathsf{obs}!v} (\nu\, j_1, j_2, h)(\, i[v \triangleleft \epsilon]^{\circ} \parallel j_1[v] \parallel j_2[i!u] \parallel h[e \triangleleft q : \{\mathsf{sd}, \mathsf{obs}, v\}]^{*}\,)$$

But if actor j_2 sends its value to i before j_1, we observe a different external behaviour

$$A \xrightarrow{\tau} \cdot \xrightarrow{\tau} \cdot \xrightarrow{\mathsf{obs}!u} (\nu\, j_1, j_2, h)(\, i[u \triangleleft \epsilon]^{\circ} \parallel j_1[i!v] \parallel j_2[u] \parallel h[e \triangleleft q : \{\mathsf{sd}, \mathsf{obs}, u\}]^{*}\,)$$

i.e., A outputs u instead of v to obs (accordingly monitor h would hold the entry $\{\mathsf{sd}, \mathsf{obs}, u\}$ instead); these behaviours amounts to an *internal choice*.

External choice results when A receives different external inputs: we can derive $A \xrightarrow{i?v_1} B_1$, but also $A \xrightarrow{i?v_2} B_2$. Subsequently, B_1 can only produce the external output $B_1 \xrightarrow{\tau}^{*} \xrightarrow{\mathsf{obs}!v_1}$ whereas from B_2 can only produce $B_2 \xrightarrow{\tau}^{*} \xrightarrow{\mathsf{obs}!v_2}$. Note that, in the first case, h's mailbox is appended by entries $\{\mathsf{rv}, i, v_1\} : \{\mathsf{sd}, \mathsf{obs}, v_1\}$ whereas, in the second case, it is appended by $\{\mathsf{rv}, i, v_2\} : \{\mathsf{sd}, \mathsf{obs}, v_2\}$. ■

3 The Logic

To specify reactive properties of the systems we consider an adaptation of SafeHML[1] (sHML) , a sub-logic of the Hennessy-Milner Logic (HML) with recursion.[4] It assumes a denumerable set of formula variables, $X, Y \in \text{LVar}$, and is defined by the grammar:

$$\varphi, \psi \in \text{sHML} ::= \mathsf{ff} \mid \varphi \wedge \psi \mid [\alpha]\varphi \mid X \mid \mathsf{max}(X, \varphi)$$

The formulas for falsity, ff, conjunction, $\varphi \wedge \psi$, and action necessity, $[\alpha]\varphi$, are inherited from HML[18], whereas variables X and the recursion construct $\mathsf{max}(X, \varphi)$ are used to define *maximal* fixpoints; as expected, $\mathsf{max}(X, \varphi)$ is a binder for the free variables X in φ, inducing standard notions of open and closed formulas. We only depart from the logic of [1] by limiting formulas to *basic actions* $\alpha, \beta \in \text{BAct}$, including just input, $i?v$, and *unbound* outputs, $i!v$, so as to keep our technical development manageable.

Remark 3. The handling of bounded output actions, $(\tilde{h})i!v$, is well understood [24] and does not pose problems to monitoring, apart from making action pattern matching cumbersome; it also complicates instrumentation (see Sec. 4 and 5). Silent τ labels can also be monitored using minor adaptations; they however increase substantially the size of the traces recorded, unnecessarily cluttering the tracing semantics of Section 2.

The semantics of our logic is defined for closed formulas, using the operation $\varphi\{\psi/X\}$, which substitutes free occurrences of X in φ with ψ without introducing any variable capture. It is specified as the satisfaction relation of Def. 1 (adapted from [1]). In what follows, we write *weak* transitions $A \Longrightarrow B$ and $A \overset{\alpha}{\Longrightarrow} B$, for $A \xrightarrow{\tau}^{*} B$ and $A \xrightarrow{\tau}^{*} \cdot \xrightarrow{\alpha} \cdot \xrightarrow{\tau}^{*} B$ *resp.* We let $s, t \in (\text{BAct})^{*}$ range over *lists of basic actions* and write sequences of weak actions $A \overset{\alpha_1}{\Longrightarrow} \cdots \overset{\alpha_n}{\Longrightarrow} B$, where $s = \alpha_1, \ldots, \alpha_n$, as $A \overset{s}{\Longrightarrow} B$ (or as $A \overset{s}{\Longrightarrow}$ when B is unimportant).

[4] HML with recursion has been shown to be equally expressive to the μ-calculus[20].

Definition 1 (Satisfiability). *A relation $\mathcal{R} \in$ Actr $\times$ sHML is a satisfaction relation iff:*

$$\begin{array}{rl}
(A, \mathsf{ff}) \in \mathcal{R} & \textit{never} \\
(A, \varphi\wedge\psi) \in \mathcal{R} & \textit{implies } (A,\varphi) \in \mathcal{R} \textit{ and } (A,\psi) \in \mathcal{R} \\
(A, [\alpha]\varphi) \in \mathcal{R} & \textit{implies } (B,\varphi) \in \mathcal{R} \textit{ whenever } A \overset{\alpha}{\Longrightarrow} B \\
(A, \mathsf{max}(X,\varphi)) \in \mathcal{R} & \textit{implies } (A, \varphi\{\mathsf{max}(X,\varphi)/X\}) \in \mathcal{R}
\end{array}$$

Satisfiability, $\models_s$, is the largest *satisfaction relation; we write $A \models_s \varphi$ for $(A,\varphi) \in sat$.*[5]

Example 2 (Satisfiability). Consider the safety formula

$$\varphi_{\text{safe}} \triangleq \mathsf{max}(X,\ [\alpha][\alpha][\beta]\mathsf{ff} \wedge [\alpha]X\,) \tag{2}$$

stating that a satisfying actor system cannot perform a sequence of two external actions α followed by the action β (through the subformula $[\alpha][\alpha][\beta]\mathsf{ff}$), and that this needs to hold after every α action (through $[\alpha]X$); effectively the formula states that sequences of α-actions *greater than two* cannot be followed by a β-action.

A system A_1 exhibiting (just) the behaviour $A_1 \overset{\alpha\beta}{\Longrightarrow}$ satisfies φ_{safe}, as would a system A_2 with just the (infinite) behaviour $A_2 \overset{\alpha}{\Longrightarrow} A_2$. System A_3 with a trace $A_3 \overset{\alpha\alpha\beta}{\Longrightarrow}$ *does not* satisfy φ_{safe}. However, if at runtime, A_3 exhibits the alternate behaviour $A_3 \overset{\beta}{\Longrightarrow}$ (through an internal choice) we *would not be able to* detect the fact that $A_3 \not\models_s \varphi_{\text{safe}}$. ■

Since actors may violate a property along one execution but satisfy it along another, the inverse of $\models_s$, *i.e.*, $A \not\models_s \varphi$, is too coarse to be used for a definition of monitor correctness along the lines of (1) discussed earlier. We thus define a *violation relation*, Def. 2, characterising actors violating a property along a specific execution trace.

Definition 2 (Violation). *The violation relation, denoted as $\models_v$, is the* least *relation of the form* (Actr $\times$ BAct* $\times$ sHML) *satisfying the following rules:*[6]

$$\begin{array}{rl}
A, s \models_v \mathsf{ff} & \textit{always} \\
A, s \models_v \varphi\wedge\psi & \textit{if } A, s \models_v \varphi \textit{ or } A, s \models_v \psi \\
A, \alpha s \models_v [\alpha]\varphi & \textit{if } A \overset{\alpha}{\Longrightarrow} B \textit{ and } B, s \models_v \varphi \\
A, s \models_v \mathsf{max}(X,\varphi) & \textit{if } A, s \models_v \varphi\{\mathsf{max}(X,\varphi)/X\}
\end{array}$$

Example 3 (Violation). Recall the safety formula φ_{safe} defined in (2). Actor A_3, from Ex. 2, together with the witness violating trace $\alpha\alpha\beta$ violate φ_{safe}, *i.e.*, $(A_3, \alpha\alpha\beta) \models_v \varphi_{\text{safe}}$. However, A_3 together with trace β do not violate φ_{safe}, *i.e.*, $(A_3,\beta) \not\models_v \varphi_{\text{safe}}$. Def. 2 relates a violating trace with an actor *only when* that trace leads the actor to a violation: if A_3 cannot perform the trace $\alpha\alpha\alpha\beta$, by Def. 2, we have $(A_3, \alpha\alpha\alpha\beta) \not\models_v \varphi_{\text{safe}}$, even though the trace is prohibited by φ_{safe}. A violating trace may also lead a system to a violation before its end, *e.g.*, $(A_3, \alpha\alpha\beta\alpha) \models_v \varphi_{\text{safe}}$ according to Def. 2. ■

[5] It follows from standard fixed-point theory that the implications of satisfaction relation are bi-implications for Satisfiability.

[6] We write $A, s \models_v \varphi$ in lieu of $(A, s, \varphi) \in \models_v$. It also follows from standard fixed-point theory that the constraints of the violation relation are bi-implications.

Despite the technical discrepancies between the two definitions, *e.g.*, maximal versus minimal fixpoints, a different model semantics *etc.*, we show that Def. 2 corresponds, in some sense, to the dual of Def. 1.

Theorem 1 (Correspondence). $\exists s.(A, s) \models_v \varphi$ *iff* $A \not\models_s \varphi$

Proof. For the *if* case we prove the contrapositive, *i.e.*, that $\forall s.A, s \not\models_{\mathrm{v}} \varphi$ implies $A \models_{\mathrm{s}} \varphi$ by co-inductively showing that $\mathcal{R} = \{(A, \varphi) \mid \forall s.A, s \not\models_{\mathrm{v}} \varphi\}$ is a satisfaction relation. For the *only-if* case we prove $\exists s.A, s \models_{\mathrm{v}} \varphi$ implies $A \not\models_{\mathrm{s}} \varphi$ by rule induction on $A, s \models_{\mathrm{v}} \varphi$. See [15] for details.

4 Correctness

Specifying online monitor correctness is complicated by the fact that, in general, we have limited control over the behaviour of the systems being monitored. For starters, a system that does not satisfy a property may still exhibit runtime behaviour that does not violate it, as discussed earlier in the case of system A_3 of Ex. 2 and Ex. 3. We deal with system non-determinism by only requiring monitor detection when the system performs a violating execution: this can be expressed through the violation relation of Def. 2.

At runtime, a system may also interfere with the execution of monitors. Appropriate *instrumentation* can limit system effects on the monitors. In our asynchronous actor setting, *direct* interferences from the system to the monitors can be precluded by (*i*) locating the monitors at process identifiers *not* known to the system (*ii*) preventing the monitors from communicating these identifiers to the system. These measures inhibit the system's ability to send messages to the monitors.

A system may also interfere with monitor executions indirectly by *diverging*, *i.e.*, infinite internal computation (τ-transitions) without external actions. This can prevent the monitors from executing and thus postpone indefinitely violation detections [25]. In our case, divergence is handled, in part, by the EVM itself, which guarantees fair executions for concurrent actors [7]. In settings where fair executions may be assumed, it suffices to require a weaker property from monitors, reminiscent of the condition in fair/should-testing[26]. Def. 3 states that, for an arbitrary basic action α, an actor system A satisfies the predicate *should-*α if, for *any* sequence of internal actions, there always *exists* an execution that can produce the action α; in the case of monitors, the external should-action is set to a reserved violation-detection action, *e.g.*, $\mathsf{fail!}$.

Definition 3 (Should-α). $A \Downarrow_\alpha \overset{def}{=} A \Longrightarrow B \quad implies \quad B \overset{\alpha}{\Longrightarrow}$

We limit monitoring to *monitorable* systems, where all actors are subject to a monitorable modality.

$$A \equiv (\nu\, \tilde{h})(i[e \triangleleft q]^m \parallel B) \quad \text{implies} \quad m = \circ$$

This guarantees that (*i*) they can be composed with a tracer actor (*ii*) all the basic actions produced by the system are recorded as trace entries at the tracer's mailbox.[7] Monitor

[7] Due to asynchronous communication, even scoped actors can produce visible actions by sending messages to environment actors.

correctness is defined for (unmonitored) *basic* systems, satisfying the condition:

$$A \equiv (\nu\,\tilde{h})(i[e \triangleleft q]^m \parallel B) \quad \text{implies} \quad m = \bullet$$

which are instrumented to execute in parallel with the monitor. Our instrumentation is defined through the operation $\lceil - \rceil$, Def. 4, converting basic systems to monitorable ones using `trace/2` and `set_on_spawn` Erlang commands [7]; see Lemma 1. Importantly, instrumentation does not affect the visible behaviour of a basic system; see Lemma 2.

Definition 4 (Instrumentation). $\lceil - \rceil :: \textsc{Actr} \to \textsc{Actr}$ *is defined inductively as:*

$$\lceil i[e \triangleleft q]^m \rceil \overset{\text{def}}{=} i[e \triangleleft q]^\circ \qquad \lceil B \parallel C \rceil \overset{\text{def}}{=} \lceil B \rceil \parallel \lceil C \rceil \qquad \lceil (\nu\, i)B \rceil \overset{\text{def}}{=} (\nu\, i)\lceil B \rceil$$

Lemma 1. *If A is a basic system then $\lceil A \rceil$ is monitorable.*

Lemma 2. *For all basic actors A:*

$$A \xrightarrow{\gamma} B \text{ iff } \begin{cases} (\nu\, i)(\lceil A \rceil \parallel i[e \triangleleft q]^*) \xrightarrow{j!v} (\nu\, i)(\lceil B \rceil \parallel i[e \triangleleft q : \{\mathsf{sd}, j, v\}]^*) & \text{if } \gamma = j!v \\ (\nu\, i)(\lceil A \rceil \parallel i[e \triangleleft q]^*) \xrightarrow{j?v} (\nu\, i)(\lceil B \rceil \parallel i[e \triangleleft q : \{\mathsf{rv}, j, v\}]^*) & \text{if } \gamma = j?v \\ (\nu\, i)(\lceil A \rceil \parallel i[e \triangleleft q]^*) \xrightarrow{\tau} (\nu\, i)(\lceil B \rceil \parallel i[e \triangleleft q]^*) & \text{if } \gamma = \tau \end{cases}$$

We are now in a position to state monitor correctness, for some predefined violation-detection monitor action $\mathsf{fail!}$, Def. 5. We restrict our definition to expressions e located at a fresh scoped location i (not used by the system, *i.e.*, $i \notin \mathrm{fId}(A)$) with an empty mailbox, ϵ; expression e may then spawn concurrent submonitors while executing. The definition can be extended to generic concurrent monitors, *i.e.*, multiple expressions, in straightforward fashion.

Definition 5 (Correctness). $e \in \textsc{Exp}$ *is a correct monitor for* $\varphi \in$ sHML *iff for any basic actors $A \in \textsc{Actr}$, $i \notin \mathrm{fId}(A)$, and execution traces $s \in (\textsc{Act}^* \setminus \{\mathsf{fail!}\})$:*

$$(\nu\, i)(\lceil A \rceil \parallel i[e \triangleleft \epsilon]^*) \overset{s}{\Longrightarrow} B \quad \textit{implies} \quad (A, s \models_{\nu} \varphi \quad \textit{iff} \quad B \Downarrow_{\mathsf{fail!}})$$

Def. 5 states that e correctly monitors property φ whenever, for any trace of environment interactions, s, of a monitored system, $(\nu\, i)(\lceil A \rceil \parallel i[e \triangleleft \epsilon]^*)$, yielding system B, if s leads A to a violation of φ, then system B should always detect it, and viceversa.

5 Automated Monitor Synthesis

We define a translation from sHML formulas to Erlang *monitors* that asynchronously analyse a system and flag an alert whenever they detect violations by the current system execution (for the respective sHML formula). This translation describes the core algorithm for a tool automating monitor synthesis from sHML formulas [29].

Despite its relative simplicity, the sHML provides opportunities to perform concurrent monitoring. The most obvious case is the translation of conjunction formulas, $\varphi_1 \wedge \varphi_2$, whereby the resulting code needs to check *both* sub-formulas φ_1 and φ_2 so as to ensure that *neither* is violated.[8] A translation in terms of two concurrent (sub)monitors, each analysing different parts of the trace so as to ensure the observation of its respective sub-formula, constitutes a natural synthesis of the conjunction formula in our target language: it adheres to recommended Erlang practices advocating for concurrency wherever possible [7], but also allows us to benefit from the advantages of concurrent monitors discussed in the Introduction.

Example 4 (Conjunction Formulas). Consider the two sHML formulas

$$\begin{aligned}\varphi_{\text{no\_dup\_ans}} &\triangleq [\alpha_{\text{call}}]\,(\,\mathsf{max}(X,\ [\beta_{\text{ans}}]\,[\beta_{\text{ans}}]\,\mathsf{ff} \wedge [\beta_{\text{ans}}]\,[\alpha_{\text{call}}]\,X)\,)\\ \varphi_{\text{react\_ans}} &\triangleq \mathsf{max}(Y,\ [\beta_{\text{ans}}]\,\mathsf{ff} \wedge [\alpha_{\text{call}}]\,[\beta_{\text{ans}}]\,Y\,)\end{aligned}$$

Formula $\varphi_{\text{no\_dup\_ans}}$ requires that call actions α_{call} are at most serviced by a *single* answer action β_{ans}, whereas formula $\varphi_{\text{react\_ans}}$ requires that answer actions are only produced *in response to* call actions. Although one can rephrase the conjunction of the two formulas as a formula without a top-level conjunction, it is more straightforward to use two concurrent monitors executing in parallel (one for each sub-formula in $\varphi_{\text{no\_dup\_ans}} \wedge \varphi_{\text{react\_ans}}$). There are also other reasons why it would be beneficial to keep the sub-formulas separate: for instance, keeping the formulas disentangled improves maintainability and separation of concerns when subformulas originate from distinct specifying parties. ■

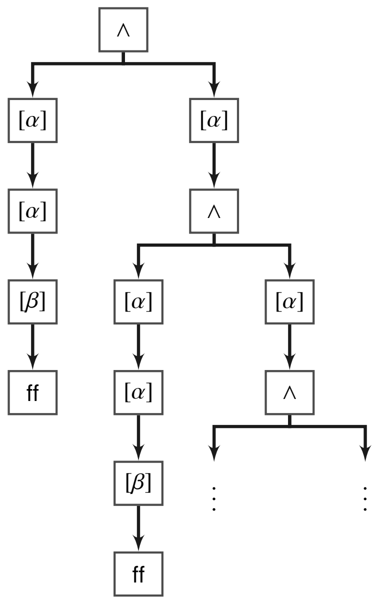
Multiple conjunctions also arise indirectly when used under fix-point operators. When synthesising concurrent monitors analysing separate branches of such recursive properties, it is important to generate monitors that can dynamic spawn further submonitors themselves as required at runtime, so as to keep the monitoring overheads to a minimum.

Example 5 (Conjunctions and Fixpoints). Recall φ_{safe}, from (2) in Ex. 2. Semantically, the formula represents the infinite-depth tree with an infinite number of conjunctions, depicted in Fig. 3(a). Although in practice, we cannot generate an infinite number of concurrent monitors, φ_{safe} will translate into possibly more than two concurrent monitors executing in parallel. ■

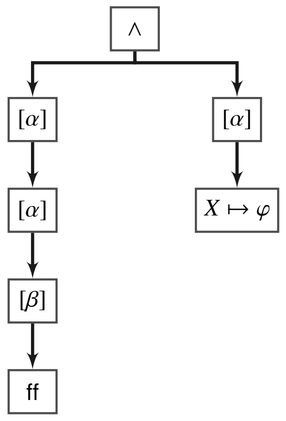
Our monitor synthesis, $[\![-]\!]^{\mathbf{m}} :: \text{sHML} \rightarrow \textsc{Exp}$, takes a *closed, guarded*[9] sHML formula and returns an Erlang function that is then parameterised by a map (encoded as a list of tuples) from formula variables to other synthesised monitors of the same form. The map encodes the variable bindings introduced by the construct $\mathsf{max}(X, \varphi)$; it is used for *lazy* recursive unrolling of formulas so as to minimize monitoring overhead. For instance, when synthesising formula φ_{safe} from Ex. 2, the algorithm initially spawns only two concurrent submonitors, one checking for the subformula $[\alpha][\alpha][\beta]\mathsf{ff}$, and another one checking for the formula $[\alpha]X$, as is depicted in Fig. 3(b). Whenever the rightmost submonitor in Fig. 3(b) observes the action α and reaches X, it unfolds X and spawns an additonal submonitor as depicted in Fig. 3(c), thereby increasing the monitor overheads incrementally.

[8] Since conjunctions are found in many monitoring logics, the concepts discussed here extend directly to other RV settings.

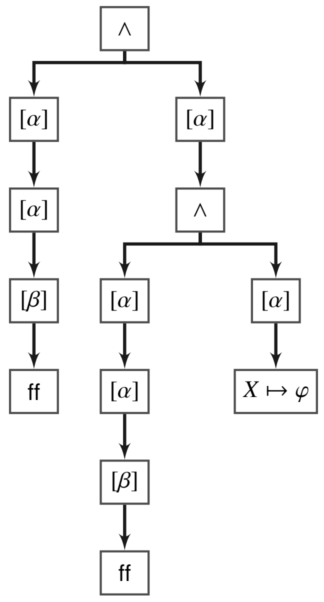
[9] In guarded sHML formulas, variables appear only as a sub-formula of a necessity formula.



(a) Denotation of φ_{safe} defined in (2)



(b) Constructed concurrent monitors for φ_{safe} where $\varphi = [\alpha][\alpha][\beta]\mathsf{ff} \wedge [\alpha]X$



(c) First expansion of the constructed monitor for φ_{safe}

Fig. 3: Monitor Combinator generation for φ_{safe} of Ex. 2

Definition 6 (Synthesis). $[\![-]\!]^{\mathbf{m}}$ *is defined on the structure of the* sHML *formula:*

$$[\![\mathsf{ff}]\!]^{\mathbf{m}} \stackrel{\mathit{def}}{=} \lambda x_{env}.\mathit{fail}!$$

$$[\![\varphi\wedge\psi]\!]^{\mathbf{m}} \stackrel{\mathit{def}}{=} \begin{cases}\lambda x_{env}.\ y_{pid1} = \mathsf{spw}\,([\![\varphi]\!]^{\mathbf{m}}(x_{env})),\ y_{pid2} = \mathsf{spw}\,([\![\psi]\!]^{\mathbf{m}}(x_{env})),\\ \qquad \mathit{fork}(y_{pid1}, y_{pid2})\end{cases}$$

$$[\![[\alpha]\varphi]\!]^{\mathbf{m}} \stackrel{\mathit{def}}{=} \lambda x_{env}.\mathsf{rcv}\,(\,\mathrm{tr}(\alpha) \to [\![\varphi]\!]^{\mathbf{m}}(x_{env});\ \ \_ \to \mathit{stop}\,)\,\mathsf{end}$$

$$[\![\mathsf{max}(X,\varphi)]\!]^{\mathbf{m}} \stackrel{\mathit{def}}{=} \lambda x_{env}.\ y_{mon} = [\![\varphi]\!]^{\mathbf{m}}, y_{mon}(\{'X', y_{mon}\} : x_{env})$$

$$[\![X]\!]^{\mathbf{m}} \stackrel{\mathit{def}}{=} \lambda x_{env}.\ y_{mon} = \mathit{lookUp}('X', x_{env}),\ y_{mon}(x_{env})$$

Auxiliary Function definitions and meta-operators:

$$\mathit{fork} \stackrel{\mathit{def}}{=} \mu y_{rec}.\lambda(x_{pid1}, x_{pid2}).\mathsf{rcv}\ z \to (x_{pid1}!z,\ x_{pid2}!z)\,\mathsf{end},\ y_{rec}(x_{pid1}, x_{pid2})$$

$$\mathit{lookUp} \stackrel{\mathit{def}}{=} \begin{cases}\mu y_{rec}.\lambda(x_{var}, x_{map}).\mathsf{case}\ x_{map}\ \mathsf{of}\ (\{x_{var}, z_{mon}\} : \_) \to z_{mon}\\ \qquad\qquad\qquad\qquad\qquad\qquad \_ : z_{tl} \to y_{rec}(x_{var}, z_{tl})\\ \qquad\qquad\qquad\qquad\qquad\qquad \mathit{nil} \to \mathit{exit}\\ \qquad\qquad\qquad \mathsf{end}\end{cases}$$

In Def. 6, the monitor for the formula ff immediately reports a violation to some supervisor actor identified as fail, handling the violation. Conjunction, $\varphi_1\wedge\varphi_2$, translates into spawning the respective monitors for φ_1 and φ_2 and subsequently forwarding any trace messages to these spawned monitors through the auxiliary function fork. The translated monitor for $[\alpha]\varphi$ behaves as the monitor translation for φ once it receives a trace message encoding the occurrence of action α, using the function:

$$\mathrm{tr}(i?v) \stackrel{\mathbf{def}}{=} \{\mathsf{rv}, i, v\} \qquad\qquad \mathrm{tr}(i!v) \stackrel{\mathbf{def}}{=} \{\mathsf{sd}, i, v\}$$

Importantly, the monitor for $[\alpha]\varphi$ terminates if the trace message does not correspond to α. The translations of $\mathsf{max}(X,\varphi)$ and X are best understood together. The monitor for $\mathsf{max}(X,\varphi)$ behaves like that for φ, under the extended map where X is mapped to the monitor for φ, effectively modelling the formula unrolling $\varphi\{^{\mathsf{max}(X,\varphi)}/_X\}$ from Def. 2. The monitor for X retrieves the respective monitor translation bound to X in the map using function lookUp, and behaves like this monitor. *Closed* formulas guarantee that map entries are always found by lookUp, whereas *guarded* formulas guarantee that formula variables, X, are guarded by necessity conditions, $[\alpha]\varphi$ — this implements the *lazy* recursive unrolling of formulas and prevents infinite bound-variable expansions.

$$\mathsf{Mon} \stackrel{\mathbf{def}}{=} \lambda x_{\mathrm{frm}}.z_{\mathrm{pid}} = \mathsf{spw}\,([\![x_{\mathrm{frm}}]\!]^{\mathbf{m}}(\mathsf{nil})), \mathsf{mLoop}(z_{\mathrm{pid}})$$

$$\mathsf{mLoop} \stackrel{\mathbf{def}}{=} \mu y_{\mathrm{rec}}.\lambda x_{\mathrm{pid}}.\mathsf{rcv}\ z_{\mathrm{msg}} \to (x_{\mathrm{pid}}!z_{\mathrm{msg}})\,\mathsf{end}, y_{\mathrm{rec}}(x_{\mathrm{pid}})$$

Monitor instrumentation, performed through the function Mon (defined above), spawns the synthesised function initialised to the empty map, nil, and then acts as a message forwarder to the spawned process, through the function mLoop (defined above), for any trace messages it receives through the tracing semantics discussed in Sec. 2.

We have constructed a tool [29] that implements the monitor synthesis of Def. 6: given an sHML formula it generates a monitor that can be instrumented with minimal changes to the system code, as discussed earlier in Sec. 4. The performance of our synthesised monitor was evaluated through a simulated server that launches individual workers to handle a series of requests from individual clients; we also injected faults making certain workers non-deterministically behave erratically. We synthesised monitors to check that each worker observes the *no-duplicate-reply* property from Ex. 4:

$$\varphi_{wrkr} \triangleq [wrk?req]\,(\,\mathsf{max}(X,\;[clnt!rply]\,[clnt!rply]\,\mathsf{ff} \wedge [clnt!rply]\,[wrk?req]\,X)\,)$$

and calculated the overheads incurred for varying number of client requests (*i.e.*, concurrent workers); we also compared this with the performance a monitor that checks for property violations in sequential fashion. Tests were carried out on an Intel Core i7 processor with 8GB of RAM, running Microsoft Windows 8 and EVM version R15B02. The result, summarised in the table below, show that our synthesised concurrent monitoring yields acceptable overheads that are consistently lower than those of a sequential monitor. We conjecture that this discrepancy can be increased further when monitoring for recursive properties with longer chains of necessity formulas.

	Unmonitored	**Sequential**		**Concurrent**		
No of. Reqs.	**Time**(μs)	**Time** (μs)	**Ovhd**(%)	**Time**(μs)	**Ovhd.**(%)	**Improv.**(%)
250	117.813	121.667	3.27	118.293	0.40	2.86
350	185.232	202.500	9.32	194.793	5.16	4.16
450	237.606	248.333	4.51	242.380	2.01	2.51
550	286.461	319.167	11.42	308.853	7.82	3.60
650	345.543	372.232	7.72	354.333	2.54	5.18

6 Proving Correctness

The preliminary results obtained in Sec. 5 advocate for the feasibility of using concurrent monitors. We however still need to show that the monitors synthesised are correct. Def. 5 allows us to state one of the main results of the paper, Theorem 2.

Theorem 2 (Correctness). *For all $\varphi \in$ sHML, $\mathsf{Mon}(\varphi)$ is a correct monitor for φ.*

Proving Theorem 2 directly can be an arduous task: for *any* sHML formula, it requires reasoning about *all* the possible execution paths of *any* monitored system in parallel with the instrumented monitor. We propose a formal technique for alleviating the task of ascertaining the monitor correctness of Def. 5 by teasing apart *three* separate (weaker) monitor-conditions: they are referred to as *Violation Detectability*, *Detection Preservation* and *Monitor Separability*. These conditions are important properties in their own right— for instance, Detection Preservation requires the monitor to behave *deterministically wrt.* violation detections. Moreover, the three conditions pose advantages to the checking of monitor correctness: since these conditions are independent to one another, they can be checked in parallel by distinct analysing entities; alternatively, the conditions that are inexpensive to check may be carried out before the more expensive ones, thus acting as vetting phases that abort early and keep the analysis cost to a minimum. More importantly though, the three conditions together imply our original monitor-correctness criteria.

The first sub-property is *Violation Detectability*, Lemma 3, guaranteeing that every violating trace s of formula φ is detectable by the respective synthesised monitor,[10] (the *only-if* case) and that there are no false detections (the *if* case). This property is easier to verify than Theorem 2 since it requires us to consider the execution of the monitor in isolation and, more importantly, requires us to verify the *existence of an execution path* that detects the violation; concurrent monitors typically have multiple execution paths.

Lemma 3 (Violation Detectability). *For basic* $A \in \text{Actr}$ *and* $i \notin \text{fId}(A)$, $A \overset{s}{\Longrightarrow}$ *implies:*

$$A, s \models_v \varphi \quad \textit{iff} \quad i[\mathsf{Mon}(\varphi) \triangleleft \text{tr}(s)]^* \overset{\mathsf{fail!}}{\Longrightarrow}$$

Detection Preservation (Lemma 4), the second sub-property, is not concerned with relating detections to actual violations. Instead it guarantees that if a monitor can potentially detect a violation, further reductions do not exclude the possibility of this detection. In the case where monitors always have a finite reduction *wrt.* their mailbox contents (as it turns out to be the case for monitors synthesised by Def. 6) this condition guarantees that the monitor will deterministically detect violations.

Lemma 4 (Detection Preservation). *For all* $\varphi \in \text{sHML}$, $q \in \text{Val}^*$

$$\Big(i[\mathsf{Mon}(\varphi) \triangleleft q]^* \overset{\mathsf{fail!}}{\Longrightarrow} \ \textit{and} \ \ i[\mathsf{Mon}(\varphi) \triangleleft q]^* \Longrightarrow B\Big) \quad \textit{implies} \quad B \overset{\mathsf{fail!}}{\Longrightarrow}$$

The third sub-property is Separability, Lemma 5, which implies that the behaviour of a (monitored) system *is independent of* the monitor and, dually, the behaviour of the monitor depends, *at most*, on the trace generated by the system.

Lemma 5 (Monitor Separability). *For all basic* $A \in \text{Actr}, i \notin \text{fId}(A), \varphi \in \text{sHML}$, *and* $s \in \text{Act}^* \setminus \{\mathsf{fail!}\}$,

$(\nu\, i)(\lceil A \rceil \parallel i[\mathsf{Mon}(\varphi)]^*) \overset{s}{\Longrightarrow} B$ *implies* $\exists B', B''$ *s.t.*

$$B \equiv (\nu\, i)(B' \parallel B'') \quad \textit{and} \quad A \overset{s}{\Longrightarrow} A' \ \textit{s.t.}\ B' = \lceil A' \rceil \quad \textit{and} \quad i[\mathsf{Mon}(\varphi) \triangleleft \text{tr}(s)]^* \Longrightarrow B''$$

These three properties suffice to show monitor correctness.

Theorem 2 (Correctness). *For all* $\varphi \in \text{sHML}$, $\mathsf{Mon}(\varphi)$ *is a correct monitor for* φ.

Proof. According to Def. 5 we have to show:

$$(\nu\, i)(\lceil A \rceil \parallel i[\mathsf{Mon}(\varphi)]^*) \overset{s}{\Longrightarrow} B \quad \text{implies} \quad (A, s \models_v \varphi \quad \text{iff} \quad B \Downarrow_{\mathsf{fail!}})$$

For the *only-if* case, we assume

$$(\nu\, i)(A \parallel i[\mathsf{Mon}(\varphi)]^*) \overset{s}{\Longrightarrow} B \qquad (3)$$

$$A, s \models_v \varphi \qquad (4)$$

[10] We elevate tr to basic action sequences s in pointwise fashion, $\text{tr}(s)$, where $\text{tr}(\epsilon) = \epsilon$.

To show $B \Downarrow_{\mathsf{fail!}}$, by Def. 3 we also assume $B \Longrightarrow B'$, for arbitrary B', and then be required to prove that $B' \overset{\mathsf{fail!}}{\Longrightarrow}$. From (3), $B \Longrightarrow B'$ and Lemma 5 we know

$$\exists B'', B''' s.t.\ B' \equiv (\nu\, i)(B'' \parallel B''') \tag{5}$$

$$A \overset{s}{\Longrightarrow} A' \text{ for some } A' \text{ where } \lceil A' \rceil = B'' \tag{6}$$

$$i[\mathsf{Mon}(\varphi) \triangleleft \mathrm{tr}(s)]^* \Longrightarrow B''' \tag{7}$$

From (6), (4) and Lemma 3 we obtain $i[\mathsf{Mon}(\varphi) \triangleleft \mathrm{tr}(s)]^* \overset{\mathsf{fail!}}{\Longrightarrow}$, and from (7) and Lemma 4 we get $B''' \overset{\mathsf{fail!}}{\Longrightarrow}$. Hence, by (5), and standard transition rules for parallel composition and scoping (see Sec. A) we can reconstruct $B' \overset{\mathsf{fail!}}{\Longrightarrow}$, as required.
For the *if* case we assume:

$$(\nu\, i)(\lceil A \rceil \parallel i[\mathsf{Mon}(\varphi)]^*) \overset{s}{\Longrightarrow} B \tag{8}$$

$$B \Downarrow_{\mathsf{fail!}} \tag{9}$$

and have to prove $A, s \models_{\mathrm{v}} \varphi$. From (9) we know $B \overset{\mathsf{fail!}}{\Longrightarrow}$. Together with (8) this implies

$$\exists B'\ s.t.\ (\nu\, i)(\lceil A \rceil \parallel i[\mathsf{Mon}(\varphi)]^*) \overset{s}{\Longrightarrow} B' \xrightarrow{\mathsf{fail!}} \tag{10}$$

From Lemma 5 and (10) we obtain

$$\exists B'', B''' s.t.\ B' = (\nu\, i)(B'' \parallel B''') \tag{11}$$

$$A \overset{s}{\Longrightarrow} A' \text{ for some } A' \text{ where } \lceil A' \rceil = B'' \tag{12}$$

$$i[\mathsf{Mon}(\varphi) \triangleleft \mathrm{tr}(s)]^* \Longrightarrow B''' \tag{13}$$

From (10), (11) and the freshness of $\mathsf{fail!}$ to A we deduce that $B''' \xrightarrow{\mathsf{fail!}}$, and subsequently, by (13), we get $i_{\mathrm{mtr}}[\mathsf{Mon}(\varphi) \triangleleft \mathrm{tr}(s)] \overset{\mathsf{fail!}}{\Longrightarrow}$. Therefore, by (12) and Lemma 3 we obtain $A, s \models_{\mathrm{v}} \varphi$, as required. □

7 Conclusion

We have studied a more intensional notion of correctness for monitor synthesis in a concurrent online setting; we worked close to the actual implementation level of abstraction so as to enhance our confidence in the correctness of our instrumented monitors. More precisely, we have identified a number of additional issues raised when proving monitor correctness in this concurrent setting, illustrating them through a case study that builds a tool [29] automating monitor synthesis from a reactive property logic (sHML) to asynchronous monitors in a concurrent language (Erlang). The specific contributions of the paper, in order of importance, are:

1. A novel formal *definition of monitor correctness*, Def. 5, dealing with issues such as system non-determinism and system interference.

2. A *proof technique* teasing apart aspects of the monitor correctness definition, Lem. 3, Lem. 4 and Lem. 5, allowing us to prove correctness in stages. We subsequently apply this technique to prove the *correctness of our tool*, Thm. 2.
3. An alternative *violation* characterisation of the logic, sHML, that is more amenable to runtime analysis and reasoning about monitor correctness, together with a proof of correspondence for this reformulation, Thm. 1.
4. An extension of a formalisation for Erlang describing its tracing semantics, Sec. 2.
5. A formal monitor synthesis definition from sHML formulas to Erlang code, Def. 6.

Related Work: The aforementioned work, [17, 27, 4], discusses monitor synthesis from a different logic, namely LTL, to either pseudocode, automata or Büchi automata; none of this work considers online concurrent monitoring, circumventing issues associated with concurrency and system interference. There is considerable work on runtime monitoring of web services, *e.g.,* [13, 5] verifying the correctness of reactive (communication) properties, similar to those expressed through sHML; to the best of our knowledge, none of this work tackles correct monitor synthesis from a specified logic. In [9], Colombo *et al.* develop an Erlang RV tool using the EVM tracing mechanism but do not consider the issue of correct monitor generation. Fredlund [16] adapted a variant of HML to specify correctness properties in Erlang, albeit for model checking purposes. There is also work relating HML formulas with tests, namely [1]. Our monitors differ from tests, as in [1], in a number of ways: (*i*) they are defined in terms of *concurrent* actors, as opposed to *sequential* CCS processes; (*ii*) they analyse systems *asynchronously*, acting on traces, whereas tests interact with the system *directly*, forcing certain system behaviour; (*iii*) they are expected to *always* detect violations when they occur whereas tests are only required to have *one* possible execution that detects violations.

Future Work: The monitoring semantics of Section 2 can be used as a basis to formally prove existing Erlang monitoring tools such as [9, 10]. sHML can also be extended to handle limited, monitorable forms of liveness properties (often termed co-safety properties [21]). It is also worth exploring mechanisms for synchronous monitoring, as opposed to asynchronous variant studied in this paper. Erlang also facilitates monitor *distribution* which can be used to lower monitoring overheads [11]. Distributed monitoring can also be used to increase the expressivity of our tool so as to handle correctness properties for distributed programs. The latter extension, however, poses a departure from our setting because the unique trace described by our framework would be replaced by separate independent traces at each location, where the lack of a total ordering of events may prohibit the detection of certain violations [14].

References

1. L. Aceto and A. Ingólfsdóttir. Testing Hennessy-Milner Logic with Recursion. In *FoSSaCS'99*, pages 41–55. Springer, 1999.
2. J. Armstrong. *Programming Erlang*. The Pragmatic Bookshelf, 2007.
3. H. Barringer, Y. Falcone, B. Finkbeiner, K. Havelund, I. Lee, G. J. Pace, G. Rosu, O. Sokolsky, and N. Tillmann, editors. *RV 2010*, volume 6418 of *LNCS*. Springer, 2010.

4. A. Bauer, M. Leucker, and C. Schallhart. Runtime verification for ltl and tltl. *ACM Trans. Softw. Eng. Methodol.*, 20:14:1–14:64, September 2011.
5. T.-D. Cao, T.-T. Phan-Quang, P. Felix, and R. Castanet. Automated runtime verification for web services. In *ICWS*, pages 76–82. IEEE, 2010.
6. R. Carlsson. An introduction to core erlang. In *PLI01 (Erlang Workshop)*, 2001.
7. F. Cesarini and S. Thompson. *ERLANG Programming*. O'Reilly, 2009.
8. E. Clarke, Jr., O. Grumberg, and D. Peled. *Model checking*. MIT Press, 1999.
9. C. Colombo, A. Francalanza, and R. Gatt. Elarva: A monitoring tool for erlang. In *RV*, volume 7186 of *LNCS*, pages 370–374. Springer, 2011.
10. C. Colombo, A. Francalanza, and I. Grima. Simplifying contract-violating traces. In *FLACOS*, volume 94 of *EPTCS*, pages 11–20, 2012.
11. C. Colombo, A. Francalanza, R. Mizzi, and G. J. Pace. polylarva: Runtime verification with configurable resource-aware monitoring boundaries. In *SEFM*, pages 218–232, 2012.
12. B. D'Angelo, S. Sankaranarayanan, C. Sánchez, W. Robinson, B. Finkbeiner, H. B. Sipma, S. Mehrotra, and Z. Manna. Lola: Runtime monitoring of synchronous systems. In *TIME*. IEEE, 2005.
13. Y. Falcone, M. Jaber, T.-H. Nguyen, M. Bozga, and S. Bensalem. Runtime verification of component-based systems. In *SEFM*, volume 7041 of *LNCS*, pages 204–220. Springer, 2011.
14. A. Francalanza, A. Gauci, and G. Pace. Distributed system contract monitoring. *JLAP*, 2013. *(to appear)*.
15. A. Francalanza and A. Seychell. Synthesising correct concurrent runtime monitors in erlang. Technical Report CS2013-01, University of Malta, Jan 2013. Accessible at `www.cs.um.edu.mt/svrg/papers.html`.
16. L.-Å. Fredlund. *A Framework for Reasoning about Erlang Code*. PhD thesis, Royal Institute of Technology, Stockholm, Sweden, 2001.
17. M. Geilen. On the construction of monitors for temporal logic properties. *ENTCS*, 55(2):181–199, 2001.
18. M. Hennessy and R. Milner. Algebraic laws for nondeterminism and concurrency. *J. ACM*, 32(1):137–161, Jan. 1985.
19. C. Hewitt, P. Bishop, and R. Steiger. A universal modular actor formalism for artificial intelligence. In *IJCAI*, pages 235–245. Morgan Kaufmann, 1973.
20. D. Kozen. Results on the propositional μ-calculus. *TCS*, 27:333–354, 1983.
21. Z. Manna and A. Pnueli. A hierarchy of temporal properties (invited paper, 1989). In *PODC*, pages 377–410. ACM, 1990.
22. P. O. Meredith, D. Jin, D. Griffith, F. Chen, and G. Rosu. An overview of the MOP runtime verification framework. *STTT*, 14(3):249–289, 2012.
23. R. Milner. *Communication and concurrency*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1989.
24. R. Milner, J. Parrow, and D. Walker. Modal logics for mobile processes. *Theoretical Computer Science*, 114:149–171, 1993.
25. R. D. Nicola and M. C. B. Hennessy. Testing equivalences for processes. *TCS*, pages 83–133, 1984.
26. A. Rensink and W. Vogler. Fair testing. *Inf. Comput.*, 205(2):125–198, Feb. 2007.
27. K. Sen, G. Rosu, and G. Agha. Generating optimal linear temporal logic monitors by coinduction. In *ASIAN*, volume 2896 of *LNCS*, pages 260–275. Springer-Verlag, 2004.
28. K. Sen, A. Vardhan, G. Agha, and G. Roşu. Efficient decentralized monitoring of safety in distributed systems. *ICSE*, pages 418–427, 2004.
29. A. Seychell. DetectEr. Accessible at `www.cs.um.edu.mt/svrg/Tools/detectEr`.
30. H. Svensson, L.-Å. Fredlund, and C. Benac Earle. A unified semantics for future erlang. In *Erlang Workshop*, pages 23–32. ACM, 2010.

A The Language Semantics

Mailbox reading—defined by the rules Rd1 and Rd2 in Fig. 2—includes pattern-matching functionality, allowing the actor to selectively choose which messages to read first from its mailbox whenever the first pattern $p \rightarrow e$ from the pattern list g is matched, returning $e\sigma$, where σ substitutes free variables in e for value binding resulting from the pattern match; when no pattern is matched, mailbox reading *blocks* - see Definition 7.

Definition 7 (Pattern Matching). *We define* mtch : PLst × Val → Exp$_\bot$ *and* vmtch : Pat × Val → Sub$_\bot$ *as follows:*

$$\mathrm{mtch}(g,v) \overset{\mathit{def}}{=} \begin{cases} e\sigma & \textit{if } g = p \rightarrow e : f, \mathrm{vmtch}(p,v) = \sigma \\ d & \textit{if } g = p \rightarrow e : f, \mathrm{vmtch}(p,v) = \bot, \mathrm{mtch}(f,v) = d \\ \bot & \textit{otherwise} \end{cases}$$

$$\mathrm{vmtch}(p,v) \overset{\mathit{def}}{=} \begin{cases} \emptyset & \textit{if } p = v \textit{ (whenever } p \textit{ is } a, i \textit{ or } \mathsf{nil}) \\ \{^v\!/x\} & \textit{if } p = x \\ \biguplus_{i=1}^{n} \sigma_i & \textit{if } p = \{p_1, \dots, p_n\}, v = \{v_1, \dots, v_n\} \textit{ where } \mathrm{vmtch}(p_i, v_i) = \sigma_i \\ \sigma \uplus \{^l\!/x\} & \textit{if } p = o : x, v = u : l \textit{ where } \mathrm{vmtch}(o,u) = \sigma \\ \bot & \textit{otherwise} \end{cases}$$

$$\sigma_1 \uplus \sigma_2 \overset{\mathit{def}}{=} \begin{cases} \sigma_1 \cup \sigma_2 & \textit{if } \mathrm{dom}(\sigma_1) \cap \mathrm{dom}(\sigma_2) = \emptyset \\ \sigma_1 \cup \sigma_2 & \textit{if } \forall v \in \mathrm{dom}(\sigma_1) \cap \mathrm{dom}(\sigma_2). \sigma_1(v) = \sigma_2(v) \\ \bot & \textit{if } \sigma_1 = \bot \textit{ or } \sigma_2 = \bot \\ \bot & \textit{otherwise} \end{cases}$$

This differs from pattern matching in case branching, described by the rules Cs1 and Cs2 in Fig. 2: similar to the mailbox read construct, it matches a values to the first appropriate pattern in the pattern list, launching the respective guarded expression with the appropriate variable bindings resulting from the pattern match; if, however, no match is found it generates an exception, exit, which aborts subsequent computation, Ext.

B Monitor Properties

In this section we detail the proofs of the individual sub-properties identified in Sec. 6. To simplify the technical development, we assume the following convention. The tracer, which is also where the synthesised monitor is places in the instrumentation of Def. 5 is located the process identifier i_{mtr}. Also sequences of process identifiers are denoted as $\tilde{h}$: for instance, the system $(\nu\, j_1)\ldots(\nu\, j_n)A$ is denoted as $(\nu\,\tilde{j})A$ when the scoped identifiers are not important.

Violation Detection The first sub-property we consider is Lemma 3, Violation Detection. One of the main Lemmas used in the proof, namely Lemma 9, relies on an encoding of formula substitutions, $\theta :: \text{LV}_{\text{AR}} \rightharpoonup \text{sHML}$, partial maps from formula variables to (possibly open) formulas, to lists of tuples containing a string representation of the variable and the respective monitor translation of the formula as defined in Def. 6. Formula substitutions are denoted as lists of individual substitutions, $\{\varphi_1/X_1\}\ldots\{\varphi_n/X_n\}$ where every X_i is distinct, and empty substitutions are denoted as ϵ.

Definition 8 (Formula Substitution Encoding).

$$\text{enc}(\theta) \overset{\text{def}}{=} \begin{cases} \mathsf{nil} & \textit{when } \theta = \epsilon \\ \{'X', [\![\varphi]\!]^{\mathbf{m}}\} : \text{enc}(\theta') & \textit{if } \theta = \{\mathsf{max}(X,\varphi)/X\}\theta' \end{cases}$$

We can show that our monitor lookup function of Def. 6 models variable substitution, Lemma 6. We can also show that different representations of the same formula substitution do not affect the outcome of the execution of lookUp on the respective encoding, which justifies the abuse of notation in subsequent proofs that assume a single possible representation of a formula substitution.

Lemma 6. *If* $\theta(X) = \varphi$ *then* $i[\mathsf{lookUp}('X', \text{enc}(\theta)) \triangleleft q]^m \Longrightarrow i[[\![\varphi]\!]^{\mathbf{m}} \triangleleft q]^m$

Proof. By induction on the number of mappings $\{\varphi_1/X_1\}\ldots\{\varphi_n/X_n\}$ in θ. □

Lemma 7. *If* $\theta(X) = \varphi$ *then* $i[\mathsf{lookUp}('X', \text{enc}(\theta')) \triangleleft q]^m \Longrightarrow i[[\![\varphi]\!]^{\mathbf{m}} \triangleleft q]^m$ *whenever* θ *and* θ' *denote the same substitution.*

Proof. By induction on the number of mappings $\{\varphi_1/X_1\}\ldots\{\varphi_n/X_n\}$ in θ. □

In one direction, Lemma 3 relies on Lemma 9 in order to establish the correspondence between violations and the possibility of detections; this lemma, in turn, uses Lemma 8 which relates possible detections by monitors synthesised from subformulas to possible detections by monitors synthesised from conjunctions using these subformulas.

Lemma 8. *For an arbitrary* θ, $(\nu\, i)(i_{mtr}[\mathsf{mLoop}(j_1) \triangleleft \text{tr}(s)]^* \parallel i[[\![\varphi_1]\!]^{\mathbf{m}}(\text{enc}(\theta))]^\bullet) \overset{\mathsf{fail!}}{\Longrightarrow}$ *implies* $(\nu\, i)(i_{mtr}[\mathsf{mLoop}(i) \triangleleft \text{tr}(s)]^* \parallel i[[\![\varphi_1 \wedge \varphi_2]\!]^{\mathbf{m}}(\text{enc}(\theta))]^\bullet) \overset{\mathsf{fail!}}{\Longrightarrow}$ *for any* $\varphi_2 \in \text{sHML}$.

Proof. By Def. 6 we know that we can derive the sequence of reductions

$$(\nu\, i)(i_{\mathrm{mtr}}[\mathsf{mLoop}(i) \triangleleft \mathrm{tr}(s)]^{*} \parallel i[[\![\varphi_1 \wedge \varphi_2]\!]^{\mathbf{m}}(\mathrm{enc}(\theta))]^{\bullet}) \Longrightarrow$$
$$(\nu\, i)(i_{\mathrm{mtr}}[\mathsf{mLoop}(i) \triangleleft \mathrm{tr}(s)]^{*} \parallel (\nu\, j,h)(i[\mathsf{fork}(j,h)]^{\bullet} \parallel j[[\![\varphi_1]\!]^{\mathbf{m}}(\mathrm{enc}(\theta))]^{\bullet} \parallel h[[\![\varphi_2]\!]^{\mathbf{m}}(\mathrm{enc}(\theta))]^{\bullet}))$$

We then prove by induction on the structure of s that

$$(\nu\, i)(i_{\mathrm{mtr}}[\mathsf{mLoop}(i) \triangleleft \mathrm{tr}(s)]^{\bullet} \parallel i[[\![\varphi_1]\!]^{\mathbf{m}}(\mathrm{enc}(\theta)) \triangleleft q]^{\bullet}) \overset{\mathsf{fail!}}{\Longrightarrow} \quad \text{implies}$$
$$(\nu\, i)\begin{pmatrix} i_{\mathrm{mtr}}[\mathsf{mLoop}(i) \triangleleft \mathrm{tr}(s)]^{*} \parallel \\ (\nu\, j,h)(i[\mathsf{fork}(j,h)]^{\bullet} \parallel j[[\![\varphi_1]\!]^{\mathbf{m}}(\mathrm{enc}(\theta)) \triangleleft q]^{\bullet} \parallel h[[\![\varphi_2]\!]^{\mathbf{m}}(\mathrm{enc}(\theta)) \triangleleft q]^{\bullet}) \end{pmatrix} \overset{\mathsf{fail!}}{\Longrightarrow}$$

See [15] for details.

Lemma 9. *If $A, s \models_v \varphi\theta$ and $l_{env} = \mathrm{enc}(\theta)$ then*

$$(\nu\, i)(i_{mtr}[\mathit{mLoop}(i) \triangleleft \mathrm{tr}(s)]^{*} \parallel i[[\![\varphi]\!]^{\mathbf{m}}(l_{env})]^{\bullet}) \overset{\mathit{fail!}}{\Longrightarrow} .$$

Proof. Proof by rule induction on $A, s \models_{\mathrm{v}} \varphi\theta$:

$A, s \models_{\mathbf{v}} \mathsf{ff}\theta$**:** Using Def. 6 for the definition of $[\![\mathsf{ff}]\!]^{\mathbf{m}}$ and the rule App (and Par and Scp), we have

$$(\nu\, i)(i_{\mathrm{mtr}}[\mathsf{mLoop}(i) \triangleleft \mathrm{tr}(s)]^{*} \parallel i[[\![\mathsf{ff}]\!]^{\mathbf{m}}(l_{\mathrm{env}})]^{\bullet}) \Longrightarrow (\nu\, i)(i_{\mathrm{mtr}}[\mathsf{mLoop}(i) \triangleleft \mathrm{tr}(s)]^{*} \parallel i[\mathsf{fail!}]^{\bullet})$$

The result follows trivially, since the process i can transition with a fail! action in a single step using the rule SndU.

$A, s \models_{\mathbf{v}} (\varphi_1 \wedge \varphi_2)\theta$ **because** $A, s \models_{\mathbf{v}} \varphi_1\theta$**:** By $A, s \models_{\mathrm{v}} \varphi_1\theta$ and I.H. we have

$$(\nu\, i)(i_{\mathrm{mtr}}[\mathsf{mLoop}(i) \triangleleft \mathrm{tr}(s)]^{*} \parallel i[[\![\varphi_1]\!]^{\mathbf{m}}(l_{\mathrm{env}})]^{\bullet}) \overset{\mathsf{fail!}}{\Longrightarrow}$$

The result thus follows from Lem. 8, which allows us to conclude that

$$(\nu\, i)(i_{\mathrm{mtr}}[\mathsf{mLoop}(i) \triangleleft \mathrm{tr}(s)]^{*} \parallel i[[\![\varphi_1 \wedge \varphi_2]\!]^{\mathbf{m}}(l_{\mathrm{env}})]^{\bullet}) \overset{\mathsf{fail!}}{\Longrightarrow}$$

$A, s \models_{\mathbf{v}} (\varphi_1 \wedge \varphi_2)\theta$ **because** $A, s \models_{\mathbf{v}} \varphi_2\theta$**:** Analogous.

$A, s \models_{\mathbf{v}} ([\alpha]\varphi)\theta$ **because** $s = \alpha t, A \overset{\alpha}{\Longrightarrow} B$ **and** $B, t \models_{\mathbf{v}} \varphi\theta$**:** Using the rule App Scp and Def. 6 for the property $[\alpha]\varphi$ we derive (14), by executing mLoop— see Def. 6 — we obtain (15), and then by rule Rd1 we derive (16) below.

$$(\nu\, i)(i_{\mathrm{mtr}}[\mathsf{mLoop}(i) \triangleleft \mathrm{tr}(\alpha t)]^{*} \parallel i[[\![\varphi]\!]^{\mathbf{m}}(l_{\mathrm{env}})]^{\bullet}) \xrightarrow{\tau} \tag{14}$$
$$(\nu\, i)(i_{\mathrm{mtr}}[\mathsf{mLoop}(i) \triangleleft \mathrm{tr}(\alpha t)]^{*} \parallel i[\mathsf{rcv}\,(\mathrm{tr}(\alpha) \rightarrow [\![\varphi]\!]^{\mathbf{m}}(l_{\mathrm{env}})\,;\, \_ \rightarrow \mathsf{ok})\,\mathsf{end}]^{\bullet}) \Longrightarrow \tag{15}$$
$$(\nu\, i)(i_{\mathrm{mtr}}[\mathsf{mLoop}(i) \triangleleft \mathrm{tr}(t)]^{*} \parallel i[\mathsf{rcv}\,(\mathrm{tr}(\alpha) \rightarrow [\![\varphi]\!]^{\mathbf{m}}(l_{\mathrm{env}})\,;\, \_ \rightarrow \mathsf{ok})\,\mathsf{end} \triangleleft \mathrm{tr}(\alpha)]^{\bullet}) \xrightarrow{\tau} \tag{16}$$
$$(\nu\, i)(i_{\mathrm{mtr}}[\mathsf{mLoop}(i) \triangleleft \mathrm{tr}(t)]^{*} \parallel i[[\![\varphi]\!]^{\mathbf{m}}(l_{\mathrm{env}})]^{\bullet})$$

By $B, t \models_{\mathrm{v}} \varphi\theta$ and I.H. we obtain

$$(\nu\, i)(i_{\mathrm{mtr}}[\mathsf{mLoop}(i) \triangleleft \mathrm{tr}(t)]^{*} \parallel i[[\![\varphi]\!]^{\mathbf{m}}(l_{\mathrm{env}})]^{\bullet}) \overset{\mathsf{fail!}}{\Longrightarrow}$$

and, thus, the result follows by (14), (15) and (16).

$A, s \models_{\mathsf{v}} (\mathsf{max}(X,\varphi))\theta$ **because** $A, s \models_{\mathsf{v}} \varphi\{^{\mathsf{max}(X,\varphi)}/_{X}\}\theta$**:** By Def. 6 and App for process i, we derive

$$(\nu\, i)(i_{\mathrm{mtr}}[\mathsf{mLoop}(i) \triangleleft \mathrm{tr}(s)]^{*} \parallel i[[\![\mathsf{max}(X,\varphi)]\!]^{\mathbf{m}}(l_{\mathrm{env}})]^{\bullet}) \Longrightarrow$$
$$(\nu\, i)(i_{\mathrm{mtr}}[\mathsf{mLoop}(i) \triangleleft \mathrm{tr}(s)]^{*} \parallel i[[\![\varphi]\!]^{\mathbf{m}}(\{'X', [\![\varphi]\!]^{\mathbf{m}}\} : l_{\mathrm{env}})]^{\bullet}) \quad (17)$$

Assuming the appropriate α-conversion for X in $\mathsf{max}(X,\varphi)$, we note that from $l_{\mathrm{env}} = \mathrm{enc}(\theta)$ and Def. 8 we obtain

$$\mathrm{enc}(\{^{\mathsf{max}(X,\varphi)}/_{X}\}\theta) = \{'X', [\![\varphi]\!]^{\mathbf{m}}\} : l_{\mathrm{env}} \quad (18)$$

By $A, s \models_{\mathrm{v}} \varphi\{^{\mathsf{max}(X,\varphi)}/_{X}\}\rho$, (18) and I.H. we obtain

$$(\nu\, i)(i_{\mathrm{mtr}}[\mathsf{mLoop}(i) \triangleleft \mathrm{tr}(s)]^{*} \parallel i[[\![\varphi]\!]^{\mathbf{m}}(\{'X', [\![\varphi]\!]^{\mathbf{m}}\} : l_{\mathrm{env}})]^{\bullet}) \overset{\mathsf{fail!}}{\Longrightarrow} \quad (19)$$

The result follows from (17) and (19). □

In the other direction, Lemma 3 relies on Lemma 13, which establishes a correspondence between violation detections and actual violations, as formalised in Def. 2.

Lemma 13 relies on a technical result, Lemma 12 which allows us to recover a violating reduction sequence for a subformula φ_1 or φ_2 from that of the synthesised monitor of a conjuction formula $\varphi_1 \wedge \varphi_2$. Lemma 12 relies on Lemma 10.

Lemma 10. *For some $l \leq n$:*

$$(\nu\, j, h)\Big(i\,\big[\mathsf{fork}(j,h) \triangleleft q_{frk}\big]^{\bullet} \parallel j[[\![\varphi_1]\!]^{\mathbf{m}}(l_{env}) \triangleleft q]^{\bullet} \parallel h[[\![\varphi_2]\!]^{\mathbf{m}}(l_{env}) \triangleleft r]^{\bullet}\Big)(\xrightarrow{\tau})^{n} \xrightarrow{\mathsf{fail!}}$$
$$\textit{implies} \quad (\nu\, j)(i_{mtr}[\mathsf{mLoop}(j) \triangleleft q_{frk}]^{*} \parallel j[[\![\varphi_1]\!]^{\mathbf{m}}(l_{env}) \triangleleft q]^{\bullet})(\xrightarrow{\tau})^{l} \xrightarrow{\mathsf{fail!}}$$
$$\textit{or} \quad (\nu\, h)(i_{mtr}[\mathsf{mLoop}(h) \triangleleft q_{frk}]^{*} \parallel h[[\![\varphi_2]\!]^{\mathbf{m}}(l_{env}) \triangleleft r]^{\bullet})(\xrightarrow{\tau})^{l} \xrightarrow{\mathsf{fail!}}$$

Proof. By induction on the structure of the mailbox q_{frk} at actor i. See [15] for details. □

Lemma 12 uses another technical result, Lemma 11, stating that silent actions are, in some sense, preserved when actor-mailbox contents of a free actor are increased; note that the lemma only applies for cases where the mailbox at this free actor decreases in size or remains unaffected by the τ-action, specified through the sublist condition $q' \leq q$.

Lemma 11 (Mailbox Increase). $(\nu\, \tilde{h})(i[e \triangleleft q]^{m} \parallel A) \xrightarrow{\tau} (\nu\, \tilde{j})(i[e' \triangleleft q']^{m} \parallel B)$ *where $i \notin \tilde{h}$ and $q' \leq q$ implies* $(\nu\, \tilde{h})(i[e \triangleleft q : v]^{m} \parallel A) \xrightarrow{\tau} (\nu\, \tilde{j})(i[e' \triangleleft q' : v]^{m} \parallel B)$

Proof. By rule induction on $(\nu\, \tilde{h})(i[e \triangleleft q]^{m} \parallel A) \xrightarrow{\tau} (\nu\, \tilde{j})(i[e' \triangleleft q']^{m} \parallel B)$. □

Equipped with Lemma 10 and Lemma 11, we are in a position to prove Lemma 12.

Lemma 12. *For some $l \leq n$*

$$(\nu\, i)\left(i_{mtr}[\mathit{mLoop}(i) \triangleleft \mathrm{tr}(s)]^* \parallel (\nu\, j,h)\begin{pmatrix} i\,[\mathit{fork}(j,h) \triangleleft \mathrm{tr}(t)]^\bullet \\ \parallel j[[\![\varphi_1]\!]^{\mathbf{m}}(l_{env})]^\bullet \parallel h[[\![\varphi_2]\!]^{\mathbf{m}}(l_{env})]^\bullet \end{pmatrix}\right)(\xrightarrow{\tau})^k \xrightarrow{\mathit{fail!}}$$

$$\textit{implies}\quad (\nu\, i)(i_{mtr}[\mathit{mLoop}(i) \triangleleft \mathrm{tr}(ts)]^* \parallel i[[\![\varphi_1]\!]^{\mathbf{m}}(l_{env})]^\bullet)(\xrightarrow{\tau})^l \xrightarrow{\mathit{fail!}}$$

$$\textit{or}\quad (\nu\, i)(i_{mtr}[\mathit{mLoop}(i) \triangleleft \mathrm{tr}(ts)]^* \parallel i[[\![\varphi_2]\!]^{\mathbf{m}}(l_{env})]^\bullet)(\xrightarrow{\tau})^l \xrightarrow{\mathit{fail!}}$$

Proof. Proof by induction on the structure of s.

$s = \epsilon$: From the structure of mLoop, we know that after the function application, the actor $i_{\mathrm{mtr}}[\mathsf{mLoop}(i)]^*$ is stuck. Thus we conclude that it must be the case that

$$(\nu\, j,h)\begin{pmatrix} i\,[\mathsf{fork}(j,h) \triangleleft \mathrm{tr}(t)]^\bullet \\ \parallel j[[\![\varphi_1]\!]^{\mathbf{m}}(l_{\mathrm{env}})]^\bullet \parallel h[[\![\varphi_2]\!]^{\mathbf{m}}(l_{\mathrm{env}})]^\bullet \end{pmatrix}(\xrightarrow{\tau})^k \xrightarrow{\mathsf{fail!}}$$

where $k = n$ or $k = n-1$. In either case, the required result follows from Lemma 10.

$s = \alpha s'$: We have two subcases:

- If

$$(\nu\, j,h)\begin{pmatrix} i\,[\mathsf{fork}(j,h) \triangleleft \mathrm{tr}(t)]^\bullet \\ \parallel j[[\![\varphi_1]\!]^{\mathbf{m}}(l_{\mathrm{env}})]^\bullet \parallel h[[\![\varphi_2]\!]^{\mathbf{m}}(l_{\mathrm{env}})]^\bullet \end{pmatrix}(\xrightarrow{\tau})^k \xrightarrow{\mathsf{fail!}}$$

  for some $k \leq n$ then, by Lemma 10 we obtain

$$(\nu\, j)(i_{\mathrm{mtr}}[\mathsf{mLoop}(j) \triangleleft \mathrm{tr}(t)]^* \parallel j[[\![\varphi_1]\!]^{\mathbf{m}}(l_{\mathrm{env}})]^\bullet)(\xrightarrow{\tau})^l \xrightarrow{\mathsf{fail!}}$$

$$\text{or}\quad (\nu\, h)(i_{\mathrm{mtr}}[\mathsf{mLoop}(h) \triangleleft \mathrm{tr}(t)]^* \parallel h[[\![\varphi_2]\!]^{\mathbf{m}}(l_{\mathrm{env}})]^\bullet)(\xrightarrow{\tau})^l \xrightarrow{\mathsf{fail!}}$$

  for some $l \leq k$. By Lemma 11 we thus obtain

$$(\nu\, j)(i_{\mathrm{mtr}}[\mathsf{mLoop}(j) \triangleleft \mathrm{tr}(ts)]^* \parallel j[[\![\varphi_1]\!]^{\mathbf{m}}(l_{\mathrm{env}})]^\bullet)(\xrightarrow{\tau})^l \xrightarrow{\mathsf{fail!}}$$

$$\text{or}\quad (\nu\, h)(i_{\mathrm{mtr}}[\mathsf{mLoop}(h) \triangleleft \mathrm{tr}(ts)]^* \parallel h[[\![\varphi_2]\!]^{\mathbf{m}}(l_{\mathrm{env}})]^\bullet)(\xrightarrow{\tau})^l \xrightarrow{\mathsf{fail!}}$$

  as required.
- Otherwise, it must be the case that

$$(\nu\, i)\left(\begin{array}{l} i_{\mathrm{mtr}}[\mathsf{mLoop}(i) \triangleleft \mathrm{tr}(s)]^* \\ \parallel (\nu\, j,h)\begin{pmatrix} i\,[\mathsf{fork}(j,h) \triangleleft \mathrm{tr}(t)]^\bullet \\ \parallel j[[\![\varphi_1]\!]^{\mathbf{m}}(l_{\mathrm{env}})]^\bullet \parallel h[[\![\varphi_2]\!]^{\mathbf{m}}(l_{\mathrm{env}})]^\bullet \end{pmatrix}\end{array}\right)(\xrightarrow{\tau})^k \tag{20}$$

$$(\nu\, i)\begin{pmatrix} i_{\mathrm{mtr}}[\mathsf{mLoop}(i) \triangleleft \mathrm{tr}(s')]^* \\ \parallel (\nu\, j,h)\,(i\,[e_{\mathrm{fork}} \triangleleft q : \mathrm{tr}(\alpha)]^\bullet \parallel A) \end{pmatrix}(\xrightarrow{\tau})^{n-k} \xrightarrow{\mathsf{fail!}} \tag{21}$$

  For some $k = 3 + k_1$ where

$$(\nu\, j,h)\begin{pmatrix} i\,[\mathsf{fork}(j,h) \triangleleft \mathrm{tr}(t)]^\bullet \\ \parallel j[[\![\varphi_1]\!]^{\mathbf{m}}(l_{\mathrm{env}})]^\bullet \parallel h[[\![\varphi_2]\!]^{\mathbf{m}}(l_{\mathrm{env}})]^\bullet \end{pmatrix}(\xrightarrow{\tau})^{k_1} \\ (\nu\, j,h)\,(i\,[e_{\mathrm{fork}} \triangleleft q]^\bullet \parallel A) \tag{22}$$

By (22) and Lemma 11 we obtain

$$(\nu\, j,h)\begin{pmatrix} i\,[\mathsf{fork}(j,h) \triangleleft \mathrm{tr}(t):\mathrm{tr}(\alpha)]^{\bullet} \\ \parallel j[[\![\varphi_1]\!]^{\mathbf{m}}(l_{\mathrm{env}})]^{\bullet} \parallel h[[\![\varphi_2]\!]^{\mathbf{m}}(l_{\mathrm{env}})]^{\bullet}\end{pmatrix}(\xrightarrow{\tau})^{k_1}$$
$$(\nu\, j,h)\,(i\,[e_{\mathrm{fork}} \triangleleft q:\mathrm{tr}(\alpha)]^{\bullet} \parallel A)$$

and by (21) we can construct the sequence of transitions:

$$(\nu\, i)\begin{pmatrix} i_{\mathrm{mtr}}[\mathsf{mLoop}(i) \triangleleft \mathrm{tr}(s')]^{*} \\ \parallel (\nu\, j,h)\begin{pmatrix} i\,[\mathsf{fork}(j,h) \triangleleft \mathrm{tr}(t):\alpha]^{\bullet} \\ \parallel j[[\![\varphi_1]\!]^{\mathbf{m}}(l_{\mathrm{env}})]^{\bullet} \parallel h[[\![\varphi_2]\!]^{\mathbf{m}}(l_{\mathrm{env}})]^{\bullet}\end{pmatrix}\end{pmatrix}(\xrightarrow{\tau})^{n-3} \xrightarrow{\mathsf{fail!}}$$

Thus, by I.H. we obtain, for some $l \leq n-3$

$$(\nu\, i)(i_{\mathrm{mtr}}[\mathsf{mLoop}(i) \triangleleft \mathrm{tr}(t\alpha s')]^{*} \parallel i[[\![\varphi_1]\!]^{\mathbf{m}}(l_{\mathrm{env}})]^{\bullet})(\xrightarrow{\tau})^{l} \xrightarrow{\mathsf{fail!}}$$
$$\text{or}\quad (\nu\, i)(i_{\mathrm{mtr}}[\mathsf{mLoop}(i) \triangleleft \mathrm{tr}(t\alpha s')]^{*} \parallel i[[\![\varphi_2]\!]^{\mathbf{m}}(l_{\mathrm{env}})]^{\bullet})(\xrightarrow{\tau})^{l} \xrightarrow{\mathsf{fail!}}$$

The result follows since $s = \alpha s'$. □

Equipped with Lemma 12, we can now prove Lemma 13.

Lemma 13. *If* $A \overset{s}{\Longrightarrow}$, $l_{env} = \mathrm{enc}(\theta)$ *and* $(\nu\, i)(i_{mtr}[\mathit{mLoop}(i) \triangleleft \mathrm{tr}(s)]^{*} \parallel i[[\![\varphi]\!]^{\mathbf{m}}(l_{env})]^{\bullet}) \overset{\mathit{fail!}}{\Longrightarrow}$ *then* $A, s \models_{v} \varphi\theta$*, whenever* $\mathrm{fv}(\varphi) \subseteq \mathrm{dom}(\theta)$.

Proof. By strong induction on $(\nu\, i)(i_{\mathrm{mtr}}[\mathsf{mLoop}(i) \triangleleft \mathrm{tr}(s)]^{*} \parallel i[[\![\varphi]\!]^{\mathbf{m}}(l_{\mathrm{env}})]^{\bullet})(\xrightarrow{\tau})^{n} \xrightarrow{\mathsf{fail!}}$.

$n = 0$**:** By inspection of the definition for mLoop, and by case analysis of $[\![\varphi]\!]^{\mathbf{m}}(l_{\mathrm{env}})$ from Def. 6, it can never be the case that

$$(\nu\, i)(i_{\mathrm{mtr}}[\mathsf{mLoop}(i) \triangleleft \mathrm{tr}(s)]^{*} \parallel i[[\![\varphi]\!]^{\mathbf{m}}(l_{\mathrm{env}})]^{\bullet}) \xrightarrow{\mathsf{fail!}}$$

Thus the result holds trivially.

$n = k+1$**:** We proceed by case analysis on φ.

$\varphi = \mathsf{ff}$**:** The result holds immediately for any A and s by Def. 2.

$\varphi = [\alpha]\psi$**:** By Def. 6, we know that

$$(\nu\, i)(i_{\mathrm{mtr}}[\mathsf{mLoop}(i) \triangleleft \mathrm{tr}(s)]^{*} \parallel i[[\![[\alpha]\psi]\!]^{\mathbf{m}}(l_{\mathrm{env}})]^{\bullet})(\xrightarrow{\tau})^{k_1} \tag{23}$$
$$(\nu\, i)(i_{\mathrm{mtr}}[\mathsf{mLoop}(i) \triangleleft \mathrm{tr}(s_2)]^{*} \parallel i[[\![[\alpha]\psi]\!]^{\mathbf{m}}(l_{\mathrm{env}}) \triangleleft \mathrm{tr}(s_1)]^{\bullet}) \xrightarrow{\tau} \tag{24}$$
$$(\nu\, i)\begin{pmatrix} i_{\mathrm{mtr}}[\mathsf{mLoop}(i) \triangleleft \mathrm{tr}(s_2)]^{*} \parallel \\ i\left[\mathsf{rcv}\begin{pmatrix}\mathrm{tr}(\alpha) \rightarrow [\![\psi]\!]^{\mathbf{m}}(l_{\mathrm{env}})\,; \\ \_ \rightarrow \mathsf{ok}\end{pmatrix}\mathsf{end} \triangleleft \mathrm{tr}(s_1)\right]^{\bullet}\end{pmatrix}(\xrightarrow{\tau})^{k_2} \xrightarrow{\mathsf{fail!}} \tag{25}$$

where $k+1 = k_1 + k_2 + 1$ and $s = s_1 s_2$ (26)

From the analysis of the code in (25), the only way for the action $\mathsf{fail!}$ to be triggered is by choosing the guarded branch $\mathrm{tr}(\alpha) \rightarrow [\![\varphi]\!]^{\mathbf{m}}(l_{\mathrm{env}})$ in actor i. This means that (25) can be decomposed into the following reduction sequences.

$$(\nu\, i)\left(\begin{array}{l} i_{\mathrm{mtr}}[\mathsf{mLoop}(i) \triangleleft \mathrm{tr}(s_2)]^{*} \parallel \\ i \left[\mathsf{rcv} \begin{pmatrix} \mathrm{tr}(\alpha) \rightarrow [\![\psi]\!]^{\mathbf{m}}(l_{\mathrm{env}})\,; \\ \_ \rightarrow \mathsf{ok} \end{pmatrix} \mathsf{end} \triangleleft \mathrm{tr}(s_1) \right]^{\bullet} \end{array} \right) (\xrightarrow{\tau})^{k_3} \tag{27}$$

$$(\nu\, i)\left(\begin{array}{l} i_{\mathrm{mtr}}[\mathsf{mLoop}(i) \triangleleft \mathrm{tr}(s_4)]^{*} \parallel \\ i \left[\mathsf{rcv} \begin{pmatrix} \mathrm{tr}(\alpha) \rightarrow [\![\psi]\!]^{\mathbf{m}}(l_{\mathrm{env}})\,; \\ \_ \rightarrow \mathsf{ok} \end{pmatrix} \mathsf{end} \triangleleft \mathrm{tr}(s_1 s_3) \right]^{\bullet} \end{array} \right) \xrightarrow{\tau} \tag{28}$$

$$(\nu\, i)\left(\begin{array}{l} i_{\mathrm{mtr}}[\mathsf{mLoop}(i) \triangleleft \mathrm{tr}(s_4)]^{*} \parallel \\ i\; [[\![\psi]\!]^{\mathbf{m}}(l_{\mathrm{env}}) \triangleleft \mathrm{tr}(s_5)]^{\bullet} \end{array} \right) (\xrightarrow{\tau})^{k_4} \xrightarrow{\mathsf{fail!}} \tag{29}$$

$$\text{where } k_2 = k_3 + k_4 + 1 \text{ and } s_1 s_3 = \alpha s_5 \text{ and } s_2 = s_3 s_4 \tag{30}$$

By (26) and (30) we derive

$$s = \alpha t \text{ where } t = s_5 s_4 \tag{31}$$

From the definition of mLoop we can derive

$$\begin{array}{r} (\nu\, i)(i_{\mathrm{mtr}}[\mathsf{mLoop}(i) \triangleleft \mathrm{tr}(t)]^{*} \parallel i[[\![\psi]\!]^{\mathbf{m}}(l_{\mathrm{env}})]^{\bullet})(\xrightarrow{\tau})^{k_5} \qquad\qquad\qquad\qquad \\ (\nu\, i)(i_{\mathrm{mtr}}[\mathsf{mLoop}(i) \triangleleft \mathrm{tr}(s_4)]^{*} \parallel i\; [[\![\psi]\!]^{\mathbf{m}}(l_{\mathrm{env}}) \triangleleft \mathrm{tr}(s_5)]^{\bullet}) \end{array} \tag{32}$$

where $k_5 \leq k_1 + k_3$. From (31) we can split $A \overset{s}{\Longrightarrow}$ as $A \overset{\alpha}{\Longrightarrow} A' \overset{t}{\Longrightarrow}$ and from (32), (29), the fact that $k_5 + k_4 < k + 1 = n$ from (26) and (30), and I.H. we obtain

$$A', t \models_{\mathrm{v}} \psi\theta \tag{33}$$

From (33), $A \overset{\alpha}{\Longrightarrow} A'$ and Def. 2 we thus conclude $A, s \models_{\mathrm{v}} ([\alpha]\psi)\theta$.

$\varphi = \varphi_1 \wedge \varphi_2$ From Def. 6, we can decompose the transition sequence as follows

$$(\nu\, i)(i_{\mathrm{mtr}}[\mathsf{mLoop}(i) \triangleleft \mathrm{tr}(s)]^{*} \parallel i[[\![\varphi_1 \wedge \varphi_2]\!]^{\mathbf{m}}(l_{\mathrm{env}})]^{\bullet})(\xrightarrow{\tau})^{k_1} \tag{34}$$

$$(\nu\, i)(i_{\mathrm{mtr}}[\mathsf{mLoop}(i) \triangleleft \mathrm{tr}(s_2)]^{*} \parallel i[[\![\varphi_1 \wedge \varphi_2]\!]^{\mathbf{m}}(l_{\mathrm{env}}) \triangleleft \mathrm{tr}(s_1)]^{\bullet}) \xrightarrow{\tau} \tag{35}$$

$$(\nu\, i)\left(\begin{array}{l} i_{\mathrm{mtr}}[\mathsf{mLoop}(i) \triangleleft \mathrm{tr}(s_2)]^{*} \\ \parallel i \left[\begin{array}{l} y_1 = \mathsf{spw}\,([\![\varphi_1]\!]^{\mathbf{m}}(l_{\mathrm{env}})), \\ y_2 = \mathsf{spw}\,([\![\varphi_2]\!]^{\mathbf{m}}(l_{\mathrm{env}})), \\ \mathsf{fork}(y_1, y_2) \end{array} \triangleleft \mathrm{tr}(s_1) \right]^{\bullet} \end{array} \right) (\xrightarrow{\tau})^{k_2} \tag{36}$$

$$(\nu\, i)\left(\begin{array}{l} i_{\mathrm{mtr}}[\mathsf{mLoop}(i) \triangleleft \mathrm{tr}(s_4)]^{*} \\ \parallel i \left[\begin{array}{l} y_1 = \mathsf{spw}\,([\![\varphi_1]\!]^{\mathbf{m}}(l_{\mathrm{env}})), \\ y_2 = \mathsf{spw}\,([\![\varphi_2]\!]^{\mathbf{m}}(l_{\mathrm{env}})), \\ \mathsf{fork}(y_1, y_2) \end{array} \triangleleft \mathrm{tr}(s_1 s_3) \right]^{\bullet} \end{array} \right) (\xrightarrow{\tau})^{2} \tag{37}$$

$$(\nu\, i)\left(\begin{array}{l} i_{\text{mtr}}[\mathsf{mLoop}(i) \triangleleft \operatorname{tr}(s_4)]^* \\ \parallel (\nu\, j)\left(\begin{array}{l} i\left[\begin{array}{l} y_2 = \mathsf{spw}\,(\llbracket\varphi_2\rrbracket^{\mathbf{m}}(l_{\text{env}})), \\ \mathsf{fork}(j, y_2) \end{array} \triangleleft \operatorname{tr}(s_1 s_3)\right]^{\bullet} \\ \parallel j[\llbracket\varphi_1\rrbracket^{\mathbf{m}}(l_{\text{env}})]^{\bullet} \end{array}\right) \end{array}\right)(\xrightarrow{\tau})^{k_3} \xrightarrow{\mathsf{fail!}} \tag{38}$$

$$\text{where } k+1 = k_1 + 1 + k_2 + 2 + k_3,\ s = s_1 s_2 \text{ and } s_2 = s_3 s_4 \tag{39}$$

From (38) we can deduce that there are two possible transition sequences how action fail! was reached:

1. If fail! was reached because

$$j[\llbracket\varphi_1\rrbracket^{\mathbf{m}}(l_{\text{env}})]^{\bullet}(\xrightarrow{\tau})^{k_4} \xrightarrow{\mathsf{fail!}}$$

   on its own, for some $k_4 \leq k_3$ then, by Par and Scp we deduce

$$(\nu\, i)(i_{\text{mtr}}[\mathsf{mLoop}(i) \triangleleft \operatorname{tr}(s)]^* \parallel j[\llbracket\varphi_1\rrbracket^{\mathbf{m}}(l_{\text{env}})]^{\bullet})(\xrightarrow{\tau})^{k_4} \xrightarrow{\mathsf{fail!}}$$

   From (39) we know that $k_4 < k+1 = n$, and by the premise $A \overset{s}{\Longrightarrow}$ and I.H. we obtain $A, s \models_{\text{v}} \varphi_1\theta$. By Def. 2 we then obtain $A, s \models_{\text{v}} (\varphi_1 \wedge \varphi_2)\theta$

2. Alternatively, (38) can be decomposed further as

$$(\nu\, i)\left(\begin{array}{l} i_{\text{mtr}}[\mathsf{mLoop}(i) \triangleleft \operatorname{tr}(s_4)]^* \\ \parallel (\nu\, j)\left(\begin{array}{l} i\left[\begin{array}{l} y_2 = \mathsf{spw}\,(\llbracket\varphi_2\rrbracket^{\mathbf{m}}(l_{\text{env}})), \\ \mathsf{fork}(j, y_2) \end{array} \triangleleft \operatorname{tr}(s_1 s_3)\right]^{\bullet} \\ \parallel j[\llbracket\varphi_1\rrbracket^{\mathbf{m}}(l_{\text{env}})]^{\bullet} \end{array}\right) \end{array}\right)(\xrightarrow{\tau})^{k_4} \tag{40}$$

$$(\nu\, i)\left(\begin{array}{l} i_{\text{mtr}}[\mathsf{mLoop}(i) \triangleleft \operatorname{tr}(s_6)]^* \\ \parallel (\nu\, j)\left(\begin{array}{l} i\left[\begin{array}{l} y_2 = \mathsf{spw}\,(\llbracket\varphi_2\rrbracket^{\mathbf{m}}(l_{\text{env}})), \\ \mathsf{fork}(j, y_2) \end{array} \triangleleft \operatorname{tr}(s_1 s_3 s_5)\right]^{\bullet} \\ \parallel j[\llbracket\varphi_1\rrbracket^{\mathbf{m}}(l_{\text{env}})]^{\bullet} \end{array}\right) \end{array}\right)(\xrightarrow{\tau})^{2} \tag{41}$$

$$(\nu\, i)\left(\begin{array}{l} i_{\text{mtr}}[\mathsf{mLoop}(i) \triangleleft \operatorname{tr}(s_6)]^* \\ \parallel (\nu\, j, h)\left(\begin{array}{l} i\,[\mathsf{fork}(j,h) \triangleleft \operatorname{tr}(s_1 s_3 s_5)]^{\bullet} \\ \parallel j[\llbracket\varphi_1\rrbracket^{\mathbf{m}}(l_{\text{env}})]^{\bullet} \\ \parallel h[\llbracket\varphi_2\rrbracket^{\mathbf{m}}(l_{\text{env}})]^{\bullet} \end{array}\right) \end{array}\right)(\xrightarrow{\tau})^{k_5} \xrightarrow{\mathsf{fail!}} \tag{42}$$

$$\text{where} k_3 = k_4 + 2 + k_5 \text{ and } s_4 = s_5 s_6 \tag{43}$$

   From (42) and Lemma 12 we know that either

$$(\nu\, i)(i_{\text{mtr}}[\mathsf{mLoop}(i) \triangleleft \operatorname{tr}(s_1 s_3 s_5 s_6)]^* \parallel i[\llbracket\varphi_1\rrbracket^{\mathbf{m}}(l_{\text{env}})]^{\bullet})(\xrightarrow{\tau})^{k_6} \xrightarrow{\mathsf{fail!}}$$

$$\text{or } (\nu\, i)(i_{\text{mtr}}[\mathsf{mLoop}(i) \triangleleft \operatorname{tr}(s_1 s_3 s_5 s_6)]^* \parallel i[\llbracket\varphi_2\rrbracket^{\mathbf{m}}(l_{\text{env}})]^{\bullet})(\xrightarrow{\tau})^{k_6} \xrightarrow{\mathsf{fail!}}$$

   where $k_6 \leq k_5$

   From (39) and (43) we know that $s = s_1 s_3 s_5 s_6$ and that $k_6 < k+1 = n$. By I.H., we obtain either $A, s \models_{\text{v}} \varphi_1\theta$ or $A, s \models_{\text{v}} \varphi_2\theta$, and in either case, by Def. 2 we deduce $A, s \models_{\text{v}} (\varphi_1 \wedge \varphi_2)\theta$.

$\varphi = X$ By Def. 6, we can deconstruct $(\nu\, i)(i_{\mathrm{mtr}}[\mathsf{mLoop}(i) \triangleleft \mathrm{tr}(s)]^* \parallel i[[\![X]\!]^{\mathbf{m}}(l_{\mathrm{env}})]^\bullet)(\xrightarrow{\tau})^{k+1} \xrightarrow{\mathsf{fail!}}$ as

$$(\nu\, i)(i_{\mathrm{mtr}}[\mathsf{mLoop}(i) \triangleleft \mathrm{tr}(s)]^* \parallel i[[\![X]\!]^{\mathbf{m}}(l_{\mathrm{env}})]^\bullet) \Longrightarrow\xrightarrow{\tau} \tag{44}$$

$$(\nu\, i)(i_{\mathrm{mtr}}[\mathsf{mLoop}(i) \triangleleft \mathrm{tr}(s_2)]^* \parallel i[y = \mathsf{lookUp}('X', l_{\mathrm{env}}), y(l_{\mathrm{env}}) \triangleleft \mathrm{tr}(s_1)]^\bullet) \Longrightarrow\xrightarrow{\tau} \tag{45}$$

$$(\nu\, i)(i_{\mathrm{mtr}}[\mathsf{mLoop}(i) \triangleleft \mathrm{tr}(s_4)]^* \parallel i[y = v, y(l_{\mathrm{env}}) \triangleleft \mathrm{tr}(s_1 s_3)]^\bullet) \Longrightarrow\xrightarrow{\tau} \tag{46}$$

$$(\nu\, i)(i_{\mathrm{mtr}}[\mathsf{mLoop}(i) \triangleleft \mathrm{tr}(s_6)]^* \parallel i[v(l_{\mathrm{env}}) \triangleleft \mathrm{tr}(s_1 s_3 s_5)]^\bullet) \Longrightarrow\xrightarrow{\mathsf{fail!}} \tag{47}$$

where $s = s_1 s_2$, $s_2 = s_3 s_4$ and $s_4 = s_5 s_6$

Since $X \in \mathrm{dom}(\theta)$, we know that

$$\theta(X) = \psi \tag{48}$$

for some ψ. By the assumption $l_{\mathrm{env}} = \mathrm{enc}(\theta)$ and Lemma 6 we obtain that $v = [\![\psi]\!]^{\mathbf{m}}$. Hence, by (44), (45), (46) and (47) we can reconstruct

$$(\nu\, i)(i_{\mathrm{mtr}}[\mathsf{mLoop}(i) \triangleleft \mathrm{tr}(s)]^* \parallel i[[\![\psi]\!]^{\mathbf{m}}(l_{\mathrm{env}})]^\bullet)(\xrightarrow{\tau})^{k_1}$$
$$(\nu\, i)(i_{\mathrm{mtr}}[\mathsf{mLoop}(i) \triangleleft \mathrm{tr}(s_6)]^* \parallel i[[\![\psi]\!]^{\mathbf{m}}(l_{\mathrm{env}}) \triangleleft \mathrm{tr}(s_1 s_3 s_5)]^\bullet)(\xrightarrow{\tau})^{k_2} \xrightarrow{\mathsf{fail!}} \tag{49}$$

where $k_1 + k_2 < k + 1 = n$. By (49) and I.H. we obtain $A, s \models_{\mathrm{v}} \psi$, which is the result required, since by (48) we know that $X\theta = \psi$.

$\varphi = \mathsf{max}(X, \psi)$ By Def. 6, we can deconstruct

$$(\nu\, i)(i_{\mathrm{mtr}}[\mathsf{mLoop}(i) \triangleleft \mathrm{tr}(s)]^* \parallel i[[\![\mathsf{max}(X, \psi)]\!]^{\mathbf{m}}(l_{\mathrm{env}})]^\bullet)(\xrightarrow{\tau})^{k+1} \xrightarrow{\mathsf{fail!}}$$

as follows:

$$(\nu\, i)(i_{\mathrm{mtr}}[\mathsf{mLoop}(i) \triangleleft \mathrm{tr}(s)]^* \parallel i[[\![\mathsf{max}(X, \psi)]\!]^{\mathbf{m}}(l_{\mathrm{env}})]^\bullet)(\xrightarrow{\tau})^{k_1} \xrightarrow{\tau}$$
$$(\nu\, i)(i_{\mathrm{mtr}}[\mathsf{mLoop}(i) \triangleleft \mathrm{tr}(s_2)]^* \parallel i[[\![\psi]\!]^{\mathbf{m}}(\{'X', \psi\} : l_{\mathrm{env}}) \triangleleft \mathrm{tr}(s_1)]^\bullet)(\xrightarrow{\tau})^{k_2} \xrightarrow{\mathsf{fail!}}$$

from which we can reconstruct the transition sequence

$$(\nu\, i)(i_{\mathrm{mtr}}[\mathsf{mLoop}(i) \triangleleft \mathrm{tr}(s)]^* \parallel i[[\![\psi]\!]^{\mathbf{m}}(\{'X', \psi\} : l_{\mathrm{env}})]^\bullet)(\xrightarrow{\tau})^{k_1+k_2} \xrightarrow{\mathsf{fail!}} \tag{50}$$

By the assumption $l_{\mathrm{env}} = \Gamma(\theta)$ we deduce that $\{'X', \psi\} : l_{\mathrm{env}} = \mathrm{enc}(\{^{\mathsf{max}(X,\psi)}/\}\theta)$ and, since $k_1 + k_2 < k + 1 = n$, we can use (50), $A \overset{s}{\Longrightarrow}$ and I.H. to obtain $A, s \models_{\mathrm{v}} \psi\{^{\mathsf{max}(X,\psi)}/_X\}\theta$. By Def. 2 we then conclude $A, s \models_{\mathrm{v}} \mathsf{max}(X, \psi)\theta$. □

We are now in a position to prove Lemma 3; we recall that the lemma was stated *wrt.* closed sHML formulas.

Lemma 3 (Violation Detection). *Whenever* $A \overset{s}{\Longrightarrow}$ *then :*

$$A, s \models_v \varphi \quad \textit{iff} \quad i_{mtr}[\mathsf{Mon}(\varphi) \triangleleft \mathrm{tr}(s)]^* \overset{\mathsf{fail!}}{\Longrightarrow}$$

Proof. For the *only-if* case, we assume $A \overset{s}{\Longrightarrow}$ and $A, s \models_\mathrm{v} \varphi$ and are required to prove $i_\mathrm{mtr}[\mathsf{Mon}(\varphi) \triangleleft \mathrm{tr}(s)]^* \overset{\mathsf{fail!}}{\Longrightarrow}$. We recall from Sec. 5 that Mon was defined as

$$\lambda x_\mathrm{frm}.z_\mathrm{pid} = \mathsf{spw}\,([\![x_\mathrm{frm}]\!]^\mathbf{m}(\mathsf{nil})), \mathsf{mLoop}(z_\mathrm{pid}). \tag{51}$$

and as a result we can deduce (using rules such as App, Spw and Par) that

$$i_\mathrm{mtr}[\mathsf{Mon}(\varphi) \triangleleft \mathrm{tr}(s)]^* \Longrightarrow (\nu\, i)(i_\mathrm{mtr}[\mathsf{mLoop}(i) \triangleleft \mathrm{tr}(s)]^* \parallel i[[\![\varphi]\!]^\mathbf{m}(\mathsf{nil})]^\bullet) \tag{52}$$

Assumption $A, s \models_\mathrm{v} \varphi$ can be rewritten as $A, s \models_\mathrm{v} \varphi\theta$ for $\theta = \epsilon$, and thus, by Def. 8 we know $\mathsf{nil} = \mathrm{enc}(\theta)$. By Lemma 9 we obtain

$$(\nu\, i)(i_\mathrm{mtr}[\mathsf{mLoop}(i) \triangleleft \mathrm{tr}(s)]^* \parallel i[[\![\varphi]\!]^\mathbf{m}(\mathsf{nil})]^\bullet) \overset{\mathsf{fail!}}{\Longrightarrow} \tag{53}$$

and the result thus follows from (52) and (53).

For the *if* case, we assume $A \overset{s}{\Longrightarrow}$ and $i_\mathrm{mtr}[\mathsf{Mon}(\varphi) \triangleleft \mathrm{tr}(s)]^* \overset{\mathsf{fail!}}{\Longrightarrow}$ and are required to prove $A, s \models_\mathrm{v} \varphi$.

Since φ is closed, we can assume the empty list of substitutions $\theta = \epsilon$ where, by default, $\mathrm{fv}(\varphi) \subseteq \mathrm{dom}(\theta)$ and, by Def. 8, $\mathsf{nil} = \mathrm{enc}(\theta)$. By (51) we can decompose the transition sequence $i_\mathrm{mtr}[\mathsf{Mon}(\varphi) \triangleleft \mathrm{tr}(s)]^* \overset{\mathsf{fail!}}{\Longrightarrow}$ as

$$\begin{aligned} & i_\mathrm{mtr}[\mathsf{Mon}(\varphi) \triangleleft \mathrm{tr}(s)]^* (\xrightarrow{\tau})^3 \\ & \qquad (\nu\, i)(i_\mathrm{mtr}[\mathsf{mLoop}(i) \triangleleft \mathrm{tr}(s)]^* \parallel i[[\![\varphi]\!]^\mathbf{m}(\mathsf{nil})]^\bullet) \overset{\mathsf{fail!}}{\Longrightarrow} \end{aligned} \tag{54}$$

The result, *i.e.*, $A, s \models_\mathrm{v} \varphi$, follows from (54) and Lemma 13. □

Detection Preservation In order to prove Lemma 4, we are able to require a stronger guarantee, *i.e.*, confluence under weak transitions (Lemma 16) for the concurrent monitors described in Def. 6. Lemma 16 relies heavily on Lemma 15.

Definition 9 (Confluence modulo Inputs with Identical Recipients).

$$\mathrm{cnf}(A) \overset{\mathit{def}}{=} \; A \xrightarrow{\gamma_1} A' \textit{ and } A \xrightarrow{\gamma_2} A'' \;\; \textit{implies} \;\; \begin{cases} \gamma_1 = i?v_1, \gamma_2 = i?v_2 \;\; \textit{or;} \\ \gamma_1 = \gamma_2, A' = A'' \;\; \textit{or;} \\ A' \xrightarrow{\gamma_2} A''', A'' \xrightarrow{\gamma_1} A''' \textit{ for some } A''' \end{cases}$$

Before we embark on showing that our synthesised monitors (Def. 6) remain confluent after a sequence of silent transitions, Lemma 15 and Lemma 16, we find it convenient to prove a technical result, Lemma 14, identifying the possible structures a monitor can be in after an arbitrary number of silent actions; the lemma also establishes that the only possible external action that a synthesised monitors can perform is the *fail* action: this property helps us reason about the possible interactions that concurrent monitors may engage in when proving Lemma 15.

Lemma 14 (Monitor Reductions and Structure). *For all* $\varphi \in \mathrm{sHML}, q \in (\textsc{Val})^*$ *and* $\theta :: \textsc{LVar} \rightharpoonup \mathrm{sHML}$ *if* $i[[\![\varphi]\!]^{\mathbf{m}}(\mathrm{enc}(\theta)) \triangleleft q]^\bullet(\xrightarrow{\tau})^n A$ *then*

1. $A \xrightarrow{\alpha} B$ *implies* $\alpha = \mathit{fail}!$ *and;*
2. *A has the form* $i[[\![\varphi]\!]^{\mathbf{m}}(\mathrm{enc}(\theta)) \triangleleft q]^\bullet$ *or, depending on* φ*:*

$\varphi = \mathsf{ff}$**:** $A \equiv i[\mathit{fail}! \triangleleft q]^\bullet$ *or* $A \equiv i[\mathit{fail} \triangleleft q]^\bullet$

$\varphi = [\alpha]\psi$**:** $A \equiv i[\mathsf{rcv}\ (\mathrm{tr}(\alpha) \rightarrow [\![\psi]\!]^{\mathbf{m}}(\mathrm{enc}(\theta))\,;\, \_ \rightarrow \mathit{ok})\ \mathsf{end} \triangleleft q]^\bullet$ *or*
($A \equiv B$ *where* $i[[\![\psi]\!]^{\mathbf{m}}(\mathrm{enc}(\theta)) \triangleleft r]^\bullet(\xrightarrow{\tau})^k B$ *for some* $k < n$ *and* $q = \mathrm{tr}(\alpha) : r$)
or
$A \equiv i[\mathit{ok} \triangleleft r]^\bullet$ *where* $q = u : r$

$\varphi = \varphi_1 \wedge \varphi_2$**:** $A \equiv i\begin{bmatrix} y_1 = \mathsf{spw}\,([\![\varphi_1]\!]^{\mathbf{m}}(\mathrm{enc}(\theta))), & \\ y_2 = \mathsf{spw}\,([\![\varphi_2]\!]^{\mathbf{m}}(\mathrm{enc}(\theta))), \mathit{fork}(y_1, y_2) & \triangleleft\ q \end{bmatrix}^\bullet$
or
$A \equiv (\nu\, j_1)\Big(\ i[e \triangleleft q]^\bullet \parallel (\nu \widetilde{h_1})(j_1[e_1 \triangleleft q_1]^\bullet \parallel B)\ \Big)$ *where*

- *e is* $y_1 = j_1, y_2 = \mathsf{spw}\,([\![\varphi_2]\!]^{\mathbf{m}}(\mathrm{enc}(\theta))), \mathit{fork}(y_1, y_2)$ *or*
  $y_2 = \mathsf{spw}\ ([\![\varphi_2]\!]^{\mathbf{m}}(\mathrm{enc}(\theta)))\,, \mathit{fork}(j_1, y_2)$
- $j_1[[\![\varphi_1]\!]^{\mathbf{m}}(\mathrm{enc}(\theta))]^\bullet\ (\xrightarrow{\tau})^k\ (\nu \widetilde{h_1})(j_1[e_1 \triangleleft q_1]^\bullet \parallel B)$ *for some* $k < n$

or
$A \equiv (\nu\, j_1, j_2)\begin{pmatrix} i[y_2 = j_2, \mathit{fork}(j_1, y_2) \triangleleft q]^\bullet \\ \parallel\ (\nu \widetilde{h_1})(j_1[e_1 \triangleleft q_1]^\bullet \parallel B)\ \parallel\ (\nu \widetilde{h_2})(j_2[e_2 \triangleleft q_2]^\bullet \parallel C) \end{pmatrix}$
where

- $j_1[[\![\varphi_1]\!]^{\mathbf{m}}(\mathrm{enc}(\theta))]^\bullet\ (\xrightarrow{\tau})^k\ (\nu \widetilde{h_1})(j_1[e_1 \triangleleft q_1]^\bullet \parallel B)$ *for some* $k < n$
- $j_2[[\![\varphi_2]\!]^{\mathbf{m}}(\mathrm{enc}(\theta))]^\bullet\ (\xrightarrow{\tau})^l\ (\nu \widetilde{h_2})(j_2[e_2 \triangleleft q_2]^\bullet \parallel C)$ *for some* $l < n$

or
$A \equiv (\nu\, j_1, j_2)\Big(\ i[e \triangleleft r]^\bullet\ \parallel\ (\nu \widetilde{h_1})(j_1[e_1 \triangleleft q_1']^\bullet \parallel B)\ \parallel\ (\nu \widetilde{h_2})(j_2[e_2 \triangleleft q_2']^\bullet \parallel C)\ \Big)$
where

- *e is either* $\mathit{fork}(j_1, j_2)$ *or* $(\mathsf{rcv}\ z \rightarrow j_1!z, j_2!z\,\mathsf{end}, \mathit{fork}(j_1, j_2))$
  or $j_1!u, i_2!u, \mathit{fork}(j_1, j_2)$ *or* $j_2!u, \mathit{fork}(j_1, j_2)$
- $j_1[[\![\varphi_1]\!]^{\mathbf{m}}(\mathrm{enc}(\theta)) \triangleleft q_1]^\bullet\ (\xrightarrow{\tau})^k\ (\nu \widetilde{h_1})(j_1[e_1 \triangleleft q_1']^\bullet \parallel B)$ *for* $k < n$, $q_1 < q$
- $j_2[[\![\varphi_2]\!]^{\mathbf{m}}(\mathrm{enc}(\theta)) \triangleleft q_2]^\bullet\ (\xrightarrow{\tau})^l\ (\nu \widetilde{h_2})(j_2[e_2 \triangleleft q_2']^\bullet \parallel C)$ *for* $l < n$, $q_2 < q$

$\varphi = X$**:** $A \equiv i[y = \mathit{lookUp}('X', \mathrm{enc}(\theta')), y(\mathrm{enc}(\theta)) \triangleleft q]^\bullet$ *where* $\theta' < \theta$ *or*

$$A \equiv i\left[y = \begin{pmatrix} \mathsf{case}\ \ \mathrm{enc}(\theta')\ \mathsf{of}\ \{'X', z_{mon}\} : \_ \rightarrow z_{mon}; \\ \_ : z_{tl} \rightarrow \mathit{lookUp}('X', z_{tl}); \\ \mathit{nil} \rightarrow \mathit{exit}; \\ \mathsf{end} \end{pmatrix},\ \ y(\mathrm{enc}(\theta)) \triangleleft q \right]^\bullet$$

where $\theta' < \theta$
or
$A \equiv B$ *where*

- $i[y = [\![\psi]\!]^{\mathbf{m}}, y(\mathrm{enc}(\theta)) \triangleleft q]^\bullet\ (\xrightarrow{\tau})^k\ B$
- $\theta(X) = \psi$

or $A \equiv i[y = \mathit{exit}, y(\mathrm{enc}(\theta)) \triangleleft q]^\bullet$ *or* $A \equiv i[\mathit{exit} \triangleleft q]^\bullet$

$\varphi = \mathsf{max}(X, \psi)$**:** $A \equiv B$ *where* $i[[\![\psi]\!]^{\mathbf{m}}(\{'X', [\![\psi]\!]^{\mathbf{m}}\} : \mathrm{enc}(\theta)) \triangleleft q]^\bullet(\xrightarrow{\tau})^k B$
for $k < n$.

Proof. The proof is by strong induction on $i[[\![\varphi]\!]^{\mathbf{m}}(l_{\text{env}}) \triangleleft q]^{\bullet}(\xrightarrow{\tau})^n A$. The inductive case involved a long and tedious list of case analysis exhausting all possibilities. See [15] for details.

Lemma 15 (Translation Confluence). *For all* $\varphi \in \text{sHML}$, $q \in (\text{VAL})^*$ *and* $\theta :: \text{LVAR} \rightharpoonup \text{sHML}$, $i[[\![\varphi]\!]^{\mathbf{m}}(\text{enc}(\theta)) \triangleleft q]^{\bullet} \Longrightarrow A$ *implies* $\text{cnf}(A)$.

Proof. Proof by strong induction on $i[[\![\varphi]\!]^{\mathbf{m}}(\text{enc}(\theta)) \triangleleft q]^{\bullet}(\xrightarrow{\tau})^n A$.

$n = 0$: The only possible τ-action that can be performed by $i[[\![\varphi]\!]^{\mathbf{m}}(\text{enc}(\theta)) \triangleleft q]^{\bullet}$ is that for the function application of the monitor definition, *i.e.,*

$$i[[\![\varphi]\!]^{\mathbf{m}}(\text{enc}(\theta)) \triangleleft q]^{\bullet} \xrightarrow{\tau} i[e \triangleleft q]^{\bullet} \text{ for some } e. \tag{55}$$

Apart from that $i[[\![\varphi]\!]^{\mathbf{m}}(\text{enc}(\theta)) \triangleleft q]^{\bullet}$ can also only perform input action at i, *i.e.,*

$$i[[\![\varphi]\!]^{\mathbf{m}}(\text{enc}(\theta)) \triangleleft q]^{\bullet} \xrightarrow{i?v} i[[\![\varphi]\!]^{\mathbf{m}}(\text{enc}(\theta)) \triangleleft q : v]^{\bullet}$$

On the one hand, we can derive $i[e \triangleleft q]^{\bullet} \xrightarrow{i?v} i[e \triangleleft q : v]^{\bullet}$. Moreover, from (55) and Lemma 11 we can deduce $i[[\![\varphi]\!]^{\mathbf{m}}(\text{enc}(\theta)) \triangleleft q : v]^{\bullet} \xrightarrow{\tau} i[e \triangleleft q : v]^{\bullet}$ which allows us to close the confluence diamond.

$n = k + 1$: We proceed by case analysis on the property φ, using Lemma 14 to infer the possible structures of the resulting process. Again, most involving cases are those for conjunction translations, as they generate more than one concurrent actor; we discuss one of these below:

$\varphi = \varphi_1 \wedge \varphi_2$: By Lemma 14, A can have any of 4 general structures, one of which is

$$A \equiv (\nu\, j_1, j_2)\left(i[j_2!u, \mathsf{fork}(j_1, j_2) \triangleleft q]^{\bullet} \begin{array}{l} \parallel (\nu\, \widetilde{h_1})(j_1[e_1 \triangleleft q_1']^{\bullet} \parallel B) \\ \parallel (\nu\, \widetilde{h_2})(j_2[e_2 \triangleleft q_2']^{\bullet} \parallel C) \end{array} \right) \tag{56}$$

where

$$j_1[[\![\varphi_1]\!]^{\mathbf{m}}(l_{\text{env}}) \triangleleft q_1]^{\bullet} (\xrightarrow{\tau})^k (\nu\, \widetilde{h_1})(j_1[e_1 \triangleleft q_1']^{\bullet} \parallel B) \text{ for } k < n, q_1 < q \tag{57}$$

$$j_2[[\![\varphi_2]\!]^{\mathbf{m}}(l_{\text{env}}) \triangleleft q_2]^{\bullet} (\xrightarrow{\tau})^l (\nu\, \widetilde{h_2})(j_2[e_2 \triangleleft q_2']^{\bullet} \parallel C) \text{ for } l < n, q_2 < q \tag{58}$$

By Lemma 14, (57) and (58) we also infer that the only external action that can be performed by the processes $(\nu\, \widetilde{h_1})(j_1[e_1 \triangleleft q_1']^{\bullet} \parallel B)$ and $(\nu\, \widetilde{h_2})(j_2[e_2 \triangleleft q_2']^{\bullet} \parallel C)$ is $\mathsf{fail}!$. Moreover by (57) and (58) we can also show that

$$\text{fId}\Big((\nu\, \widetilde{h_1})(j_1[e_1 \triangleleft q_1']^{\bullet} \parallel B)\Big) = \{j_1\} \quad \text{fId}\Big((\nu\, \widetilde{h_2})(j_2[e_2 \triangleleft q_2']^{\bullet} \parallel C)\Big) = \{j_2\}$$

Thus these two subactors cannot communicate with each other or send messages to the actor at i. This also means that the remaining possible actions that

A can perform are:

$$A \xrightarrow{\tau} (\nu\, j_1, j_2)\left(i[u, \mathsf{fork}(j_1, j_2) \triangleleft q]^\bullet \begin{array}{l} \parallel (\nu\,\widetilde{h_1})(j_1[e_1 \triangleleft q_1']^\bullet \parallel B) \\ \parallel (\nu\,\widetilde{h_2})(j_2[e_2 \triangleleft q_2' : u]^\bullet \parallel C) \end{array} \right) \tag{59}$$

$$A \xrightarrow{\tau} (\nu\, j_1, j_2)\left(i[j_2!u, \mathsf{fork}(j_1, j_2) \triangleleft q]^\bullet \begin{array}{l} \parallel (\nu\,\widetilde{h_1'})(j_1[e_1' \triangleleft q_1'']^\bullet \parallel B') \\ \parallel (\nu\,\widetilde{h_2})(j_2[e_2 \triangleleft q_2']^\bullet \parallel C) \end{array} \right)$$

because

$$(\nu\,\widetilde{h_1})(j_1[e_1 \triangleleft q_1']^\bullet \parallel B) \xrightarrow{\tau} (\nu\ \widetilde{h_1'}\ )(j_1[e_1' \triangleleft q_1'']^\bullet \parallel B') \tag{60}$$

$$A \xrightarrow{\tau} (\nu\, j_1, j_2)\left(i[j_2!u, \mathsf{fork}(j_1, j_2) \triangleleft q]^\bullet \begin{array}{l} \parallel (\nu\,\widetilde{h_1})(j_1[e_1 \triangleleft q_1']^\bullet \parallel B) \\ \parallel (\nu\,\widetilde{h_2'})(j_2[e_2' \triangleleft q_2'']^\bullet \parallel C') \end{array} \right)$$

because

$$(\nu\,\widetilde{h_2})(j_2[e_2 \triangleleft q_2']^\bullet \parallel C) \xrightarrow{\tau} (\nu\ \widetilde{h_2'}\ )(j_2[e_2' \triangleleft q_2'']^\bullet \parallel C') \tag{61}$$

$$A \xrightarrow{i?v} (\nu\, j_1, j_2)\left(\begin{array}{l} i[j_2!u, \mathsf{fork}(j_1, j_2) \triangleleft q : v]^\bullet \\ \parallel (\nu\,\widetilde{h_1})(j_1[e_1 \triangleleft q_1']^\bullet \parallel B) \\ \parallel (\nu\,\widetilde{h_2})(j_2[e_2 \triangleleft q_2']^\bullet \parallel C) \end{array} \right) \tag{62}$$

We consider actions (59) and (61) and leave the other combinations for the interested reader. From (61) and Lemma 11 we derive

$$(\nu\,\widetilde{h_2})(j_2[e_2 \triangleleft q_2' : u]^\bullet \parallel C) \xrightarrow{\tau} (\nu\ \widetilde{h_2'}\ )(j_2[e_2' \triangleleft q_2'' : u]^\bullet \parallel C')$$

and by Par and Scp we obtain

$$\begin{array}{l} (\nu\, j_1, j_2)\left(i[u, \mathsf{fork}(j_1, j_2) \triangleleft q]^\bullet \begin{array}{l} \parallel (\nu\,\widetilde{h_1})(j_1[e_1 \triangleleft q_1']^\bullet \parallel B) \\ \parallel (\nu\,\widetilde{h_2})(j_2[e_2 \triangleleft q_2' : u]^\bullet \parallel C) \end{array} \right) \xrightarrow{\tau} \\ \qquad (\nu\, j_1, j_2)\left(i[u, \mathsf{fork}(j_1, j_2) \triangleleft q]^\bullet \begin{array}{l} \parallel (\nu\,\widetilde{h_1})(j_1[e_1 \triangleleft q_1']^\bullet \parallel B) \\ \parallel (\nu\,\widetilde{h_2'})(j_2[e_2' \triangleleft q_2'' : u]^\bullet \parallel C') \end{array} \right) \end{array} \tag{63}$$

Using Com, Str, Par and Scp we can derive

$$\begin{array}{l} (\nu\, j_1, j_2)\left(i[j_2!u, \mathsf{fork}(j_1, j_2) \triangleleft q]^\bullet \begin{array}{l} \parallel (\nu\,\widetilde{h_1})(j_1[e_1 \triangleleft q_1']^\bullet \parallel B) \\ \parallel (\nu\,\widetilde{h_2'})(j_2[e_2' \triangleleft q_2'']^\bullet \parallel C') \end{array} \right) \xrightarrow{\tau} \\ \qquad (\nu\, j_1, j_2)\left(i[u, \mathsf{fork}(j_1, j_2) \triangleleft q]^\bullet \begin{array}{l} \parallel (\nu\,\widetilde{h_1})(j_1[e_1 \triangleleft q_1']^\bullet \parallel B) \\ \parallel (\nu\,\widetilde{h_2'})(j_2[e_2' \triangleleft q_2'' : u]^\bullet \parallel C') \end{array} \right) \end{array} \tag{64}$$

thus we close the confluence diamond by (63) and (64). □

Lemma 16 (Weak Confluence). *For all $\varphi \in$ sHML, $q \in$ VAL**

$$i_{mtr}[\mathsf{Mon}(\varphi) \triangleleft q]^* \Longrightarrow A \;\; \textit{implies} \;\; \mathrm{cnf}(A)$$

Proof. By strong induction on n, the number of reductions $i_{\mathrm{mtr}}[\mathsf{Mon}(\varphi) \triangleleft q]^* \, (\xrightarrow{\tau})^n \, A$.

$n = 0$ We know $A = i_{\mathrm{mtr}}[\mathsf{Mon}(\varphi) \triangleleft q]^*$. It is confluent because it can perform either of two actions, namely a τ-action for the function application (see App in Fig. 2), or else an external input at i_{mtr}, (see RcvU in Fig. 2). The matching moves can be constructed by RcvU on the one hand, and by Lemma 11 on the other, analogously to the base case of Lemma 15.

$n = k + 1$ By performing a similar analysis to that of Lemma 14, but for $i_{\mathrm{mtr}}[\mathsf{Mon}(\varphi) \triangleleft q]^*$, we can determine that this actor can only weakly transition to either of the following forms:

(i) $A = i_{\mathrm{mtr}}[M = \mathsf{spw}\,([\![\varphi]\!]^{\mathbf{m}}(\mathsf{nil})), \mathsf{mLoop}(M) \triangleleft q]^*$

(ii) $A \equiv (\nu\, i)(i_{\mathrm{mtr}}[\mathsf{mLoop}(i) \triangleleft q]^* \parallel B)$ where $i[[\![\varphi]\!]^{\mathbf{m}}(l_{\mathrm{env}}) \triangleleft r]^{\bullet} \Longrightarrow B$ for some r.

(iii) $A \equiv (\nu\, i)(i_{\mathrm{mtr}}[\mathsf{rcv}\; z \to i!z\; \mathsf{end}, \mathsf{mLoop}(i) \triangleleft q]^* \parallel B)$ where $i[[\![\varphi]\!]^{\mathbf{m}}(l_{\mathrm{env}}) \triangleleft r]^{\bullet} \Longrightarrow B$ for some r.

(iv) $A \equiv (\nu\, i)(i_{\mathrm{mtr}}[i!v, \mathsf{mLoop}(i) \triangleleft q]^* \parallel B)$ where $i[[\![\varphi]\!]^{\mathbf{m}}(l_{\mathrm{env}}) \triangleleft r]^{\bullet} \Longrightarrow B$ for some r.

(v) $A \equiv (\nu\, i)(i_{\mathrm{mtr}}[v, \mathsf{mLoop}(i) \triangleleft q]^* \parallel B)$ where $i[[\![\varphi]\!]^{\mathbf{m}}(l_{\mathrm{env}}) \triangleleft r]^{\bullet} \Longrightarrow B$ for some r.

We here focus on the 4th case of monitor structure; the other cases are analogous. From $i[[\![\varphi]\!]^{\mathbf{m}}(l_{\mathrm{env}}) \triangleleft r]^{\bullet} \Longrightarrow B$ and Lemma 14 we know that

$$B \xrightarrow{\gamma} \quad \text{implies } \gamma = \mathsf{fail!} \text{ or } \gamma = \tau$$
$$B \equiv (\nu\, \tilde{h})(i[e \triangleleft r]^{\bullet} \parallel C) \quad \text{where } \mathrm{fId}(B) = i$$

This means that $(\nu\, i)(i_{\mathrm{mtr}}[i!v, \mathsf{mLoop}(i) \triangleleft q]^* \parallel B)$ can only exhibit the following actions:

$$(\nu\, i)(i_{\mathrm{mtr}}[i!v, \mathsf{mLoop}(i) \triangleleft q]^* \parallel B) \xrightarrow{i_{\mathrm{mtr}}?u} (\nu\, i)(i_{\mathrm{mtr}}[i!v, \mathsf{mLoop}(i) \triangleleft q : u]^* \parallel B) \tag{65}$$

$$(\nu\, i)(i_{\mathrm{mtr}}[i!v, \mathsf{mLoop}(i) \triangleleft q]^* \parallel B) \xrightarrow{\tau} (\nu\, i)(i_{\mathrm{mtr}}[v, \mathsf{mLoop}(i) \triangleleft q]^* \parallel (\nu\, \tilde{h})(i[e \triangleleft r : v]^{\bullet} \parallel C)) \tag{66}$$

$$(\nu\, i)(i_{\mathrm{mtr}}[i!v, \mathsf{mLoop}(i) \triangleleft q]^* \parallel B) \xrightarrow{\tau} (\nu\, i)(i_{\mathrm{mtr}}[i!v, \mathsf{mLoop}(i) \triangleleft q]^* \parallel C) \tag{67}$$

Most pairs of action can be commuted easily by Par and Scp as they concern distinct elements of the actor system. The only non-trivial case is the pair of actions (66) and (67), which can be commuted using Lemma 11, in analogous fashion to the proof for the base case. □

Lemma 16 allows us to prove Lemma 17, and subsequently Lemma 18; the latter Lemma implies Detection Preservation, Lemma 4, used by Theorem 2.

Lemma 17. *For all* $\varphi \in \text{sHML}$, $q \in \textsc{Val}^*$

$$i_{mtr}[\mathsf{Mon}(\varphi) \triangleleft q]^* \Longrightarrow A, A \overset{\mathsf{fail!}}{\Longrightarrow} \text{ and } A \xrightarrow{\tau} B \text{ implies } B \overset{\mathsf{fail!}}{\Longrightarrow}$$

Proof. From $i_{\text{mtr}}[\mathsf{Mon}(\varphi) \triangleleft q]^* \Longrightarrow A$ and Lemma 16 we know that $\text{cnf}(A)$. The proof is by induction on $A(\xrightarrow{\tau})^n \cdot \xrightarrow{\mathsf{fail!}}$.

$n = 0$**:** We have $A \xrightarrow{\mathsf{fail!}} A'$ (for some A'). By $A \xrightarrow{\tau} B$ and $\text{cnf}(A)$ we obtain $B \xrightarrow{\mathsf{fail!}} B'$ for some B' where $A' \xrightarrow{\tau} B'$.

$n = k + 1$**:** We have $A \xrightarrow{\tau} A'(\xrightarrow{\tau})^k \cdot \xrightarrow{\mathsf{fail!}}$ (for some A'). By $A \xrightarrow{\tau} A'$, $A \xrightarrow{\tau} B$ and $\text{cnf}(A)$ we either know that $B = A'$, in which case the result follows immediately, or else obtain

$$B \xrightarrow{\tau} A'' \tag{68}$$

$$A' \xrightarrow{\tau} A'' \quad \text{for some } A'' \tag{69}$$

In such a case, by $A \xrightarrow{\tau} A'$ and $i_{\text{mtr}}[\mathsf{Mon}(\varphi) \triangleleft q]^* \Longrightarrow A$ we deduce that

$$i_{\text{mtr}}[\mathsf{Mon}(\varphi) \triangleleft q]^* \Longrightarrow A',$$

and subsequently, by (69), $A'(\xrightarrow{\tau})^k \cdot \xrightarrow{\mathsf{fail!}}$ and I.H. we obtain $A'' \overset{\mathsf{fail!}}{\Longrightarrow}$; the required result then follows from (68). □

Lemma 18 (Detection Confluence). *For all* $\varphi \in \text{sHML}$, $q \in \textsc{Val}^*$

$$i_{mtr}[\mathsf{Mon}(\varphi) \triangleleft q]^* \Longrightarrow A, A \overset{\mathsf{fail!}}{\Longrightarrow} \text{ and } A \Longrightarrow B \text{ implies } B \overset{\mathsf{fail!}}{\Longrightarrow}$$

Proof. By induction on $A(\xrightarrow{\tau})^n B$ and Lemma 17. □

Lemma 4 (Detection Preservation). *For all* $\varphi \in \text{sHML}$, $q \in \textsc{Val}^*$

$$i_{mtr}[\mathsf{Mon}(\varphi) \triangleleft q]^* \overset{\mathsf{fail!}}{\Longrightarrow} \text{ and } i_{mtr}[\mathsf{Mon}(\varphi) \triangleleft q]^* \Longrightarrow B \text{ implies } B \overset{\mathsf{fail!}}{\Longrightarrow}$$

Proof. Immediate, from Lem. 18, for the special case where $i_{\text{mtr}}[\mathsf{Mon}(\varphi) \triangleleft q]^* \Longrightarrow i_{\text{mtr}}[\mathsf{Mon}(\varphi) \triangleleft q]^*$. □

Monitor Separability With the body of supporting lemmata proved thus far, the proof for Lemma 5 turns out to be relatively straightforward. In particular, we make use of Lemma 2, relating the behaviour of a monitored system to the same system when unmonitored, Lemma 11 delineating behaviour preservation after extending mailbox contents at specific actors, and Lemma 14, so as to reason about the structure and generic behaviour of synthesised monitors.

Lemma 5 (Monitor Separability). *For all basic actors $\varphi \in$ sHML, $A \in$ Actr where i_{mtr} is fresh to A, and $s \in \text{Act}^* \setminus \{\textsf{fail!}\}$,*

$$(\nu\, i_{mtr})(\lceil A \rceil \parallel i_{mtr}[\textsf{Mon}(\varphi)]^*) \overset{s}{\Longrightarrow} B \textit{ implies } \exists B', B'' \textit{s.t.} \begin{cases} B \equiv (\nu\, i_{mtr})(B' \parallel B'') \\ A \overset{s}{\Longrightarrow} A' \textit{ s.t. } B' = \lceil A' \rceil \\ i_{mtr}[\textsf{Mon}(\varphi) \triangleleft \text{tr}(s)]^* \Longrightarrow B'' \end{cases}$$

Proof. By induction on n in $(\nu\, i_{\text{mtr}})(\lceil A \rceil \parallel i_{\text{mtr}}[\textsf{Mon}(\varphi)]^*)(\xrightarrow{\gamma_k})^n B$, the length of the sequence of actions:

$n = 0$**:** We know that $s = \epsilon$ and $A = (\nu\, i_{\text{mtr}})(\lceil A \rceil \parallel i_{\text{mtr}}[\textsf{Mon}(\varphi)]^*)$. Thus the conditions hold trivially.

$n = k + 1$**:** We have $(\nu\, i_{\text{mtr}})(\lceil A \rceil \parallel i_{\text{mtr}}[\textsf{Mon}(\varphi)]^*)(\xrightarrow{\gamma_k})^k C \xrightarrow{\gamma} B$. By I.H. we know that

$$C \equiv (\nu\, i_{\text{mtr}})(C' \parallel C'') \tag{70}$$

$$A \overset{t}{\Longrightarrow} A'' \textit{ s.t. } C' = \lceil A'' \rceil \tag{71}$$

$$i_{\text{mtr}}[\textsf{Mon}(\varphi) \triangleleft \text{tr}(t)]^* \Longrightarrow C'' \tag{72}$$

$$\gamma = \tau \text{ implies } t = s \quad \text{and} \quad \gamma = \alpha \text{ implies } t\alpha = s \tag{73}$$

and by (72) and Lemma 14 we know that

$$C'' \equiv (\nu\, \tilde{h})(i_{\text{mtr}}[e \triangleleft q]^* \parallel C''') \tag{74}$$

$$\text{fId}(C'') = \{i_{\text{mtr}}\} \tag{75}$$

We proceed by considering the two possible subcases for the structure of γ:

$\gamma = \alpha$**:** By (73) we know that $s = t\alpha$. By (75) and (74), it must be the case that $C \equiv (\nu\, i_{\text{mtr}})(C' \parallel C'') \xrightarrow{\alpha} B$ happens because

$$\text{for some } B'\ C' \xrightarrow{\alpha} B' \tag{76}$$

$$B \equiv (\nu\, i_{\text{mtr}})(B' \parallel (\nu\, \tilde{h})(i_{\text{mtr}}[e \triangleleft q : \text{tr}(\alpha)]^* \parallel C''')) \tag{77}$$

By (76), (71) and Lemma 2 we know that $\exists A'$ such that $\lceil A' \rceil = B'$ and that $A'' \xrightarrow{\alpha} A'$. Thus by (71) and $s = t\alpha$ we obtain

$$A \overset{s}{\Longrightarrow} A' \textit{ s.t. } B' = \lceil A' \rceil$$

By (72), (74) and repeated applications of Lemma 11 we also know that

$$i_{\text{mtr}}[\textsf{Mon}(\varphi) \triangleleft \text{tr}(t) : \text{tr}(\alpha)]^* = i_{\text{mtr}}[\textsf{Mon}(\varphi) \triangleleft \text{tr}(s)]^* \Longrightarrow \\ (\nu\, \tilde{h})(i_{\text{mtr}}[e \triangleleft q : \text{tr}(\alpha)]^* \parallel C''') = B''$$

The result then follows from (77).

$\gamma = \tau$**:** Analogous to the other case, where we also have the case that the reduction is instigated by C'', in which case the results follows immediately. □