Acta Informatica (2023) 60:385–451
https://doi.org/10.1007/s00236-023-00441-9

ORIGINAL ARTICLE

On first-order runtime enforcement of branching-time properties

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

Received: 2 July 2019 / Accepted: 1 June 2023 / Published online: 3 August 2023

Abstract

Runtime enforcement is a dynamic analysis technique that uses monitors to enforce the behaviour specified by some correctness property on an executing system. The enforceability of a logic captures the extent to which the properties expressible via the logic can be enforced at runtime for a specified operational model of enforcing monitors. We study the enforceability of branching-time, first-order properties expressed in the Hennessy–Milner Logic with Recursion (μHML) with respect to monitors that can enforce behaviour involving events that carry data. To this end, we develop an operational framework for first-order enforcement via suppressions, insertions and replacements. We then use this model to formalise the meaning of enforcing a branching-time property. We also show that a safety syntactic fragment of the logic is enforceable within this framework by providing an automated synthesis function that generates correct suppression monitors from any formula taken from this logical fragment.

The research work disclosed in this publication is partially supported by the projects "Developing Theoretical Foundations for Runtime Enforcement" (184776-051), "TheoFoMon: Theoretical Foundations for Monitorability" (163406-051) and "Mode(l)s of Verification and Monitorability" (MoVeMent) (217987-051) of the Icelandic Research Fund, by the BehAPI project funded by the EU H2020 RISE of the Marie Skłodowska-Curie action (778233) and by the Endeavour Scholarship Scheme (Malta), part-financed by the European Social Fund (ESF)—Operational Programme II—Cohesion Policy 2014–2020.

✉ Adrian Francalanza
adrian.francalanza@um.edu.mt

Luca Aceto
luca@ru.is; luca.aceto@gssi.it

Ian Cassar
ian.cassar.10@um.edu.mt; ianc17@ru.is

Anna Ingólfsdóttir
annai@ru.is

1 Reykjavik University, Reykjavik, Iceland

2 Gran Sasso Science Institute, L'Aquila, Italy

3 University of Malta, Msida, Malta

1 Introduction

Runtime monitoring [1–3] is a popular dynamic analysis technique. It uses code units called *monitors* to either aggregate system information, compare system execution against correctness specifications, or steer the execution of the observed system. *Runtime enforcement* (RE) [4–6] is a specialised monitoring technique, used to ensure that the behaviour of a system-under-scrutiny (SuS) is *always* in agreement with some correctness specification. It employs a specific kind of monitor (referred to as a *transducer* [7–9], *shield* [10] or an *edit-automaton* [4, 5]) to anticipate incorrect behaviour and counter it. Such a monitor thus acts as a proxy between the SuS and the surrounding environment interacting with it, encapsulating the system to form a composite (monitored) system. The behaviour of the composite system may vary from that of the SuS depending on the actions executed by the SuS in conjunction with a range of runtime *transformations* applied by the monitor, including action *suppressions*, *insertions* and *replacements*.

We extend a recent line of research [3, 11–16] and study the potential of extending RE approaches to first-order branching-time specifications. Understanding the effectiveness of RE over branching-time specifications is important for modern verification setups where RE is only *one* option from an arsenal of verification techniques that can be used, covering both pre- and post-deployment phases of the software development lifecycle [17–22]. In such cases, it is natural to consider correctness specifications describing the SuS *computation graph*, typically formalised by a branching-time logic. Practical specifications often also need to describe data relationships over the SuS event payloads, which is typically achieved using a first-order constructs. Although these specifications are best verified using a static technique like model checking, there are numerous situations where such a strategy is impractical (e.g., when an exhaustive static verification is prohibitively expensive, or when a sufficiently detailed SuS model cannot be obtained due to restrictive licensing agreements of third-party software components). In such cases, verification engineers need to resort to other techniques such RE.

The branching-time nature of the specifications considered departs substantially from that of linear-time specifications [23] used by the state of the art on RE. Whereas linear-time specifications describe properties of the *current* execution trace of the SuS, branching-time specifications describe properties such as what can/cannot be done by the SuS after some/all computations exhibiting a particular trace. As a result, the standard RE criteria of soundness (i.e., when the enforced behaviour satisfies the property to be enforced) and transparency (i.e., when the monitor should not intervene because the property is not violated), identified by Ligatti et al. [4] for a linear-time setup, are not immediately applicable to branching-time specifications.

The branching-time interpretation of a formula also affects the RE handling of certain logical constructs. For instance, consider the disjunction formula $\varphi_1 \vee \varphi_2$. The linear-time setting requires the *current trace* to either satisfy φ_1 or φ_2, and an RE setup can intervene to enforce the property whenever the monitor observes enough of this trace to determine that neither φ_1 nor φ_2 are satisfied. The situation is different for a branching-time interpretation since the subformulas φ_1 and φ_2 can, in principle, describe computation from *different parts of the computation tree*. In turn, although the current execution observed might provide enough information to determine that *either one* of φ_1 or φ_2 is violated, there would never be an execution that allows a (sound and transparent) monitor to determine when to intervene in cases where *both* subformulas are violated.

These are a few of the issues that are crucial for ensuring *monitor correctness*. Since any analysis tool ought to form part of the trusted computing base, a monitor synthesised from a specification for enforcement purposes should be, in and of itself, correct. However, it is unclear what guarantees are to be expected from a monitor that enforces a branching-time formula. Nor is it clear for which type of specifications this approach should be expected to work effectively; it has been well established that a number of properties are *not* monitorable [11, 12, 23–27] and it is therefore reasonable to expect similar limits in the case of enforceability [2, 28].

In order to conduct our investigation in a systematic manner, we insists on a *separation of concerns* between the correctness specification, describing *what* properties the SuS should satisfy, and the monitor, describing *how* to enforce these properties on the SuS. Our work considers data-oriented properties expressed in terms of a first-order extension of the logic μHML [29, 30] and explores what and how first-order branching-time properties can be enforced. By way of example, we formally demonstrate how these properties can be operationally enforced by monitors that are instrumented to execute in tandem with the SuS in order to suppress, insert and replace system events that carry a payload. A central element for the realisation of such an approach is the *synthesis* function which automates the translation from the *declarative* μHML specifications to *algorithmic* descriptions formulated as executable monitors.

This separation of concerns serves a number of purposes. First, the convenience of a highly expressive logic such as μHML (a reformulation of the modal μ-calculus) allows us to achieve a good degree of generality for our results; by employing this logic, our work also applies to other widely used logics (such as LTL and CTL [31]) that are embedded within μHML (see [23, 32] for examples of such embeddings). Second, since such a branching-time logic is verification-technique agnostic (compared to logics such as LTL_3 [33] tailored for runtime verification), it fits better with the realities of present-day software verification where, as stated earlier, a *variety* of techniques (e.g., model-checking and testing) straddling both pre- and post-deployment phases are used. In such cases, knowing which properties can be verified statically and which ones can be monitored and enforced at runtime is crucial for devising effective multi-pronged verification strategies [34–44]. Equipped with such knowledge, one could also employ standard techniques [45–47] to decompose a non-enforceable property into a collection of smaller properties, a subset of which can then be enforced at runtime. Within this setup, this paper makes the following contributions:

Modelling We develop a general framework for first-order enforcement instrumentation that is parametrisable by any system whose behaviour can be expressed via labelled transitions. The framework can handle enforcement of events carrying data via action suppression, insertion and replacement (Fig. 2).

Correctness We provide two formal definitions for asserting when a monitor correctly enforces a formula interpreted over labelled transition systems, namely enforcement, Definition 4, and weak enforcement, Definition 7, and formally compare the two Theorem 2; these definitions rely on novel interpretations for enforcement soundness, Definition 2, and transparency, Definitions 3 and 6. We also define a parametric definition for logic enforceability, Definition 8 (Enforceability), that manifests a *black-box* treatment of the SuS, and can also be instantiated to different criteria for correct enforcement. To our knowledge, all existing studies of RE target linear-time properties; we are also unaware of any study on the enforceability of logics with data.

Expressiveness We identify a subset of μHML formulas that can be mapped to our monitors enforcing data-dependent behaviour. In fact, we prove an even stronger result and show that suppression monitors are sufficiently expressive to conduct such correct enforcement for this logical subset. This result has benefits from a realisability standpoint, since suppression monitors are easier to implement in general; data-dependent insertions/replacements need to determine the payload carried by the inserted/replaced events, which is not always a function of the data observed by monitoring up to that point, and may not necessarily be in line with typical default values in the case of certain data domains (e.g., the value 0 is often chosen as the default value for the natural numbers but there many be properties for which this is inadequate). To assess the correctness of this mapping, we provide enforceability results, namely Theorems 3 (Enforcement) and 5 (Normalisation Equivalence) (but also Theorem 4 (Weak Enforcement)).

As a by-product of this study, we also develop a provably correct synthesis function, Definition 10, that can then be used for tool construction, along the lines of [48–53].

Structure of the paper: Sect. 2 revisits labelled transition systems and presents our touchstone logic, μHML, extended to a first-order setting. The operational model for data-oriented enforcement monitors and instrumentation is given in Sect. 3. In Sect. 4 we formalise the interdependent notions of correct enforcement and enforceability. These notions act as a foundation for the development of a synthesis function in Sect. 5, which produces *correct-by-construction* monitors from normalised safety formulas. In Sect. 6 we then show that when restricted to safety properties, our notions of correct enforcement from Sect. 4 coincide. Section 7 concludes and discusses related and future work. This article is an extended version of [54]; it includes expanded explanations and examples, complete proofs and additional results, including a comparison of our enforcement definitions, Theorem 2 in Sect. 4 and Theorem 6 in Sect. 6, and a detailed explanation in Sect. 5.2 of a result showing that every formula definable by a fragment of the safety subset of μHML can be normalised into an equivalent formula that adheres to a stricter syntax, Theorem 5.

2 Preliminaries

The Model: We assume image-finite systems that are described as *labelled transition systems* (LTSs), consisting of triples $\langle \textsc{Sys}, \textsc{Act} \cup \{\tau\}, \rightarrow \rangle$ defining a set of *system states*, $s, r, q \in \textsc{Sys}$, a set of *observable actions*, $\alpha, \beta \in \textsc{Act}$, and a distinguished silent action $\tau \notin \textsc{Act}$ along with a *transition* relation, $\longrightarrow \subseteq (\textsc{Sys} \times \textsc{Act} \cup \{\tau\} \times \textsc{Sys})$. We use the dedicated variable $\mu \in \textsc{Act} \cup \{\tau\}$ to range over both silent and observable actions. We write $s \xrightarrow{\mu} r$ in lieu of $(s, \mu, r) \in \rightarrow$, and $s \overset{\alpha}{\Rightarrow} r$ to denote weak transitions representing $s(\xrightarrow{\tau})^{*} \cdot \xrightarrow{\alpha} \cdot (\xrightarrow{\tau})^{*} r$ and refer to r as a α-derivative of s. The syntax of the regular fragment of CCS [55] is occasionally used to concisely describe LTSs in our examples. We include its syntax and LTS semantics for completeness. Apart from recursion, $\mathsf{rec}\, x.s$, the two main constructs of regular CCS are action prefixing, $\mu.s$, and n-ary choice, $\sum_{i \in I} s_i$ where $|I| = n$ (for the binary case when $n = 2$, we simply write $s_1 + s_2$). Their behaviour is fairly standard, as their

Syntax

$$\begin{array}{llllll} \varphi,\psi \in \mu\text{HML} ::= \mathsf{tt} & \text{(truth)} & \mid \mathsf{ff} & \text{(falsehood)} & \mid \bigvee_{i\in I}\psi_i & \text{(disjunction)} \\ \mid \bigwedge_{i\in I}\psi_i & \text{(conjunction)} & \mid \langle\{p,c\}\rangle\varphi & \text{(possibility)} & \mid [\{p,c\}]\varphi & \text{(necessity)} \\ \mid \mathsf{min}\,X.\varphi & \text{(least fp.)} & \mid \mathsf{max}\,X.\varphi & \text{(greatest fp.)} & \mid X & \text{(fp. variable)} \end{array}$$

Semantics

$$[\![\mathsf{tt},\rho]\!] \stackrel{\text{def}}{=} \textsc{Sys} \qquad [\![\mathsf{ff},\rho]\!] \stackrel{\text{def}}{=} \emptyset \qquad [\![X,\rho]\!] \stackrel{\text{def}}{=} \rho(X)$$

$$[\![\textstyle\bigwedge_{i\in I}\varphi_i,\rho]\!] \stackrel{\text{def}}{=} \textstyle\bigcap_{i\in I}[\![\varphi_i,\rho]\!] \qquad [\![\mathsf{max}\,X.\varphi,\rho]\!] \stackrel{\text{def}}{=} \bigcup\{S \mid S\subseteq [\![\varphi,\rho[X\mapsto S]]\!]\}$$

$$[\![\textstyle\bigvee_{i\in I}\varphi_i,\rho]\!] \stackrel{\text{def}}{=} \textstyle\bigcup_{i\in I}[\![\varphi_i,\rho]\!] \qquad [\![\mathsf{min}\,X.\varphi,\rho]\!] \stackrel{\text{def}}{=} \bigcap\{S \mid [\![\varphi,\rho[X\mapsto S]]\!]\subseteq S\}$$

$$[\![\,[\{p,c\}]\varphi,\rho]\!] \stackrel{\text{def}}{=} \{s \mid (\forall\alpha,r,\sigma\cdot s \stackrel{\alpha}{\Longrightarrow} r \text{ and } \mathsf{mtch}(p,\alpha){=}\sigma \text{ and } c\sigma \Downarrow \mathsf{true}) \text{ implies } r\in[\![\varphi\sigma,\rho]\!]\}$$

$$[\![\langle\{p,c\}\rangle\varphi,\rho]\!] \stackrel{\text{def}}{=} \{s \mid \exists\alpha,r,\sigma\cdot(s \stackrel{\alpha}{\Longrightarrow} r \text{ and } \mathsf{mtch}(p,\alpha){=}\sigma \text{ and } c\sigma \Downarrow \mathsf{true} \text{ and } r\in[\![\varphi\sigma,\rho]\!])\}$$

Fig. 1 The syntax and semantics for μHML

respective transition rules show (e.g., $\mu.s$ transitions to state s by emitting the action μ).

$$s,r \in \text{rCCS} ::= \quad \mathsf{nil} \quad \mid \mu.s \quad \mid \textstyle\sum_{i\in I} s_i \quad \mid \mathsf{rec}\,x.s \quad \mid x$$

$$\frac{}{\mu.s \xrightarrow{\mu} s} \qquad \frac{s_j \xrightarrow{\mu} r_j}{\sum_{i\in I} s_i \xrightarrow{\mu} r_j}\; j\in I \qquad \frac{s\{\mathsf{rec}\,x.s/x\} \xrightarrow{\mu} r}{\mathsf{rec}\,x.s \xrightarrow{\mu} r}$$

Traces $t,u \in \textsc{Act}^*$ range over (finite) sequences of observable actions, and we write $s \stackrel{t}{\Rightarrow} r$ to denote a sequence of weak transitions $s \stackrel{\alpha_1}{\Longrightarrow} \cdots \stackrel{\alpha_n}{\Longrightarrow} r$ for $t = \alpha_1,\ldots,\alpha_n$ for some $n \geq 0$. When $n = 0$, t is the empty trace ε and $s \stackrel{\varepsilon}{\Rightarrow} r$ means $s \xrightarrow{\tau}{}^{*} r$. We also assume the classic notions of *strong similarity*, $s \precsim r$, and *bisimilarity*, $s \sim r$, for our model [55, 56], using them as our touchstone system preorder and equivalence relations, respectively.

Definition 1 (*Strong similarity and bisimilarity*) A relation $\mathcal{R}$ over a set of system states is a *strong simulation* iff whenever $(s,r) \in \mathcal{R}$ for every action μ:

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

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

A relation $\mathcal{R}$ over a set of system states is a *strong bisimulation* iff whenever $(s,r) \in \mathcal{R}$ for every action μ, the following transfer properties are satisfied:

- every $s \xrightarrow{\mu} s'$ implies there exists a transition $r \xrightarrow{\mu} r'$ s.t. $(s',r') \in \mathcal{R}$; and
- every $r \xrightarrow{\mu} r'$ implies there exists a transition $s \xrightarrow{\mu} s'$ s.t. $(s',r') \in \mathcal{R}$.

Two system states s and r are *bisimilar*, $s \sim r$, iff there exists a *strong bisimulation* that relates them. □

The Logic: We consider a slightly generalised version of μHML [30, 57] that uses *symbolic actions* (*SAs*) of the form p,c, in contrast to the conventional concrete actions, α. *Patterns*, p, abstract over actions using *data variables* $d,e,f \in \textsc{DVar}$. Variables in a pattern may either occur free, d, or as binders, (d) where a *closed pattern* is one without any free variables. We use function $\mathbf{bv}(p)$ to denote the set of binders in p, and $\mathbf{fv}(c)$ to represent the set of free variables referenced in condition c.

We assume a (partial) *matching function* for *closed* patterns $\mathsf{mtch}(p,\alpha)$ that (when successful) returns a substitution σ mapping variables in p to the corresponding values in α. For instance, if we match the pattern $\mathsf{i}?(d)$ with the (concrete) action $\mathsf{i}?5$ using $\mathsf{mtch}(\mathsf{i}?(d), \mathsf{i}?5)$ we obtain the data substitution $\{d \mapsto 5\}$. The *filtering condition*, c, contains variables found in p and is evaluated with respect to the substitutions returned by successful matches, written as $c\sigma \Downarrow b$ where $b \in \{\mathsf{true}, \mathsf{false}\}$. Put differently, a *closed SA*, p, c, is one where p is closed and $\mathbf{fv}(c) \subseteq \mathbf{bv}(p)$; it denotes the *set* of actions $[\![p, c]\!] \overset{\text{def}}{=} \{ \alpha \mid \exists\sigma \cdot \mathsf{mtch}(p, \alpha) = \sigma \textit{ and } c\sigma \Downarrow \mathsf{true} \}$. The use of symbolic actions allows for more adequate reasoning about LTSs with infinite actions (e.g., actions carrying data from infinite domains).

Example 1 Symbolic action $(d)?(e), e{=}1$ is valid since $(\mathbf{fv}(e{=}1) = \{e\}) \subseteq (\mathbf{bv}((d)?(e)) = \{d, e\})$, but actions $(d)?e, e = 1$ and $(d)?1, e = 1$ are invalid since $\mathbf{fv}(e{=}1) \nsubseteq (\mathbf{bv}((d)?e) = \{d\})$. □

Two symbolic actions, p_1, c_1 and p_2, c_2, are said to be *equivalent* when $[\![p_1, c_1]\!] = [\![p_2, c_2]\!]$, and *pattern equivalent* when $[\![p_1, \mathsf{true}]\!] = [\![p_2, \mathsf{true}]\!]$.

The syntax of the logic is given in Fig. 1 and assumes a countably infinite set of logical variables $X, Y \in \text{LVAR}$. It provides standard logical constructs such as truth, falsehood, conjunctions and disjunctions: $\bigwedge_{i \in I} \varphi_i$ describes a *compound* conjunction, $\varphi_1 \wedge \ldots \wedge \varphi_n$, where $I = \{1, .., n\}$ is a finite set of indices, and similarly for disjunctions. It allows for defining recursive properties using the greatest and least fixpoints, $\mathsf{max}\, X.\varphi$ and $\mathsf{min}\, X.\varphi$, both of which bind free occurrences of X in φ. The logic also uses *universal* and *existential* modal operators defining symbolic actions, $[p, c]\varphi$ and $\langle p, c\rangle\varphi$, where $\mathbf{bv}(p)$ bind free data variables in c and φ. Formulas in μHML are interpreted over the system powerset domain where $S \in \mathcal{P}(\text{SYS})$. The semantic definition of Fig. 1, $[\![\varphi, \rho]\!]$, is given for *both* open and closed formulas. It employs a valuation from logical variables to sets of states, $\rho \in (\text{LVAR} \to \mathcal{P}(\text{SYS}))$, which permits an inductive definition on the structure of the formulas; $\rho' = \rho[X \mapsto S]$ denotes a valuation where $\rho'(X) = S$ and $\rho'(Y) = \rho(Y)$ for all other $Y \neq X$. The semantic definition of Fig. 1 uses also the substitution operation $\varphi\sigma$ substituting each free occurrence of data variables in the formula φ by their corresponding values, according to the substitution σ. The only non-standard cases are those for the modal formulas, due to the use of *SAs*.

Note, however, that we recover the standard logic for symbolic actions, p, c, when the data variables in pattern p are all equated to a single value in condition c, e.g., a concrete action $\alpha = \mathsf{i}?v$ is equivalent to symbolic action $(d)?(e), d = \mathsf{i} \wedge e = v$ which can alternatively be written as $\mathsf{i}?v, \mathsf{true}$ in shorthand notation. We refer to these as *singleton symbolic actions* and in such cases we simply write $[\alpha]\varphi$ and $\langle\alpha\rangle\varphi$ for short, thus eliding the condition "true". We assume *closed* formulas, i.e., without free logical and data variables, and write $[\![\varphi]\!]$ in lieu of $[\![\varphi, \rho]\!]$ since the interpretation of a closed φ is independent of the valuation ρ. A system s *satisfies* formula φ whenever $s \in [\![\varphi]\!]$; a formula φ is *satisfiable*, whenever there exists a system r such that $r \in [\![\varphi]\!]$, i.e., $[\![\varphi]\!] \neq \emptyset$.

In [58], Hennessy and Milner proved a powerful result linking the notion of strong bisimilarity to the logic used in this paper, by establishing that strong bisimilar image-finite systems satisfy the same set of properties (restated as Theorem 1 below). A consequence of this theorem is that non-bisimilar systems can be distinguished by finding a property that is satisfied by one but not the other. Although this result was originally given in relation to the Hennessy-Milner logic (without recursion), it still applies to the full μHML [59, 60].

Theorem 1 (Hennessy–Milner Theorem [58]) *Let s and r be two states of an image-finite LTS such that when $s \sim r$ then both s and r satisfy exactly the same μHML formulas.* □

Example 2 Consider two systems (a good system, $s_{\mathbf{g}}$, and a bad one, $s_{\mathbf{b}}$) implementing a server that interacts on port i, repeatedly accepting *requests* that are *answered* by outputting on the same port, and terminating the service once a *close* request is accepted (on the same port). Whereas $s_{\mathbf{g}}$ outputs a *single* answer (i!ans) for every request (i?req), $s_{\mathbf{b}}$ occasionally produces *multiple* answers for a given request (see the underlined branch in the description of $s_{\mathbf{b}}$ below). Both systems terminate with i?cls.

$$s_{\mathbf{g}} = \mathsf{rec}\, x.\big(\mathsf{i?req.i!ans}.x + \mathsf{i?cls.nil}\big)$$
$$s_{\mathbf{b}} = \mathsf{rec}\, x.\big(\mathsf{i?req.}(\mathsf{i!ans}.x\; +\; \underline{\mathsf{i!ans.i!ans}.x}) + \mathsf{i?cls.nil}\big)$$

We can specify that a request followed by two consecutive answers on port i indicates invalid behaviour via the μHML formula φ_0.

$$\varphi_0 \overset{\text{def}}{=} [\mathsf{i?req}]\mathsf{max}\, X.[\mathsf{i!ans}]([\mathsf{i!ans}]\mathsf{ff}\wedge[\mathsf{i?req}]X)$$

It defines an invariant property ($\mathsf{max}\, X.\,(\cdots)$) requiring that whenever the system interacting on port i outputs an answer following a request, it cannot output a subsequent answer, i.e., $[\mathsf{i!ans}]\mathsf{ff}$, unless it inputs a request beforehand, in which case the formula recurses, i.e., $[\mathsf{i?req}]X$.

Using symbolic actions, we can generalise φ_0 to a first-order setting by requiring the property to hold for *any* interaction happening on *any* port number *except* j.

$$\varphi_1 \overset{\text{def}}{=} [(d)\mathsf{?req},\, d{\neq}\mathsf{j}]\mathsf{max}\, X.[d\mathsf{!ans},\,\mathsf{true}]([d\mathsf{!ans},\,\mathsf{true}]\mathsf{ff}\wedge[d\mathsf{?req},\,\mathsf{true}]X)$$

In φ_1, $(d)\mathsf{?req}$ binds the free occurrences of d found in $d{\neq}j$ and in the continuation formula $\mathsf{max}\, X.[d\mathsf{!ans},\,\mathsf{true}]([d\mathsf{!ans},\,\mathsf{true}]\mathsf{ff}\wedge[d\mathsf{?req},\,\mathsf{true}]X)$. Using the semantics in Fig. 1, one can check that $s_{\mathbf{g}}{\in}[\![\varphi_1]\!]$, whereas $s_{\mathbf{b}}{\notin}[\![\varphi_1]\!]$ since $s_{\mathbf{b}} \xrightarrow{\mathsf{i?req}} \cdot \xrightarrow{\mathsf{i!ans}} \cdot \xrightarrow{\mathsf{i!ans}} \ldots$ □

3 An operational model for enforcement

Our operational mechanism for enforcing properties over systems uses the (symbolic) transducers $m, n \in \textsc{Trn}$ defined in Fig. 2. Transducers are a special kind of monitors that define *symbolic transformation triples*, p, c, p', consisting of the action *pattern* and condition, p and c *resp.*, along with the *transformation pattern* p'. The action pattern and condition determine whether or not the transformation should be applied to an action α, or if the monitor should act independent of the system. The transformation pattern specifies the kind of transformation that should be applied. Transformations therefore permit the transducer to suppress, replace or insert actions.

The syntax of our transducers assumes a well-formedness constraint where for every $p, c, p'.m$, $\mathbf{bv}(c){\cup}\mathbf{bv}(p') = \emptyset$. The transition rules in Fig. 2 assume closed terms, i.e., for every *transformation-prefix transducer* of the form $p, c, p'.m$, p is closed and $\big(\mathbf{fv}(c){\cup}\mathbf{fv}(p'){\cup}\mathbf{fv}(m)\big) \subseteq \mathbf{bv}(p)$, and yield an LTS with labels of the form $\gamma{\blacktriangleright}\gamma'$, where $\gamma, \gamma' \in (\textsc{Act} \cup \{\bullet\})$ and $\bullet$ is a monitor action – the matching function is lifted to these extended actions in the obvious way, where $\mathsf{mtch}(\bullet, \bullet){=}\emptyset$.

Intuitively, a transition $m \xrightarrow{\alpha\blacktriangleright\gamma} n$ denotes the fact that the transducer in state m *transforms* the visible action α (produced by the system) into action γ and transitions into state n. In this sense, the transducer action $\alpha{\blacktriangleright}\beta$ represents the *replacing* of α by β, and $\alpha{\blacktriangleright}\alpha$ denotes the *identity* transformation. Cases $\alpha{\blacktriangleright}\bullet$ and $\bullet{\blacktriangleright}\alpha$ *resp.* encode the *suppression* and *insertion* transformations of action α; in the former, $\bullet$ signifies the removal of action α from the

Syntax

$$m, n \in \textsc{Trn} ::= \quad \{p, c, p'\}.m \qquad \mid \textstyle\sum_{i \in I} m_i \qquad \mid \mathsf{rec}\, x.m \qquad \mid x$$

Dynamics

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

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

Instrumentation

$$\textsc{iTrn}\,\frac{s \xrightarrow{\alpha} s' \qquad m \xrightarrow{\alpha \blacktriangleright \beta} n}{m[s] \xrightarrow{\beta} n[s']} \qquad \textsc{iSup}\,\frac{s \xrightarrow{\alpha} s' \qquad m \xrightarrow{\alpha \blacktriangleright \bullet} n}{m[s] \xrightarrow{\tau} n[s]} \qquad \textsc{iIns}\,\frac{m \xrightarrow{\bullet \blacktriangleright \alpha} n}{m[s] \xrightarrow{\alpha} n[s]}$$

$$\textsc{iAsy}\,\frac{s \xrightarrow{\tau} s'}{m[s] \xrightarrow{\tau} m[s']} \qquad \textsc{iDef}\,\frac{s \xrightarrow{\alpha} s' \qquad m \overset{\alpha}{\not\rightarrow} \qquad m \overset{\bullet}{\not\rightarrow}}{m[s] \xrightarrow{\alpha} \mathsf{id}[s']}$$

where $\mathsf{id} \overset{\text{def}}{=} \mathsf{rec}\, y.\{(d)!(e), \mathsf{true}, d!e\}.y + \{(d)?(e), \mathsf{true}, d?e\}.y$.

Fig. 2 A model for transducers (I is a finite index set and $m \overset{\gamma}{\not\rightarrow}$ means $\nexists \gamma', n \cdot m \xrightarrow{\gamma \blacktriangleright \gamma'} n$)

execution of the monitored system, while in the latter it represents a monitor transition that introduces an action α that was not induced by the system.

The key transition rule in Fig. 2 is eTrn. It states that the transformation-prefix transducer p, c, $p'.m$ can transform an extended action γ into a different action γ', as long as the action matches with pattern p yielding substitution σ ($\neq$ undef), $\mathsf{mtch}(p, \gamma) = \sigma$, and the condition is satisfied by σ, $c\sigma \Downarrow \mathsf{true}$. In such a case, the transformed action is $\gamma' = p'\sigma$, i.e., the action γ' resulting from the instantiation of the free data variables in pattern p' with the corresponding values mapped by σ, and the transducer state reached is $m\sigma$. The remaining rules for recursion (eRec) and selection (eSel) are standard. We encode the identity monitor, id, as a recursive monitor defining identity transformations that match every possible action.

Figure 2 also describes an *instrumentation* relation, which relates the behaviour of the SuS s with the transformations of a transducer monitor m that *agrees* with the (observable) actions Act of s. The term $m[s]$ thus denotes the resulting *monitored system* whose behaviour is defined in terms of $\textsc{Act} \cup \{\tau\}$ from the system's LTS. Concretely, rule iTrn states that when a system s transitions with an observable action α to s' and the transducer m can *transform* this action into β and transition to n, the instrumented system $m[s]$ transitions with action β to $n[s']$. However, when s transitions with a silent action, rule iAsy allows it to do so independently of the transducer.

Rule iSup states that if the system performs an action α that the monitor can *suppress* into $\bullet$, the composite system transitions silently over τ. Dually, with rule iIns the composite system transitions over an action α when the transducer is able to *insert* α independently of the behaviour of s. Rule iDef is analogous to standard monitor instrumentation rules for premature termination of the transducer [11, 13, 61, 62], and accounts for underspecification of transformations. Thus, if a system s transitions with an observable action α to s', and the transducer m does not specify how to transform it ($m \overset{\alpha}{\not\rightarrow}$), nor can it transition to a new transducer state by inserting an action ($< m \overset{\bullet}{\not\rightarrow} >$), the system is still allowed to transition while the transducer defaults to acting like the identity monitor, id, from that point onwards. It is worth highlighting that the instrumentation is *evidence based*: the transitions of a monitored system only rely on actual transitions of the SuS and are never based on other SuS aspects such as the transitions it cannot do (as is the case for the monitor with premises $< m \overset{\alpha}{\not\rightarrow} >$ and $< m \overset{\bullet}{\not\rightarrow} >$ in rule IDEF). This manifests a black-box treatment of the SuS.

Example 3 Consider the insertion transducer $m_{\mathbf{i}}$ and the replacement transducer $m_{\mathbf{r}}$ below:

$$m_{\mathbf{i}} \overset{\text{def}}{=} (d)?\mathsf{req}, \mathsf{true}, d?\mathsf{req}.\bullet, \mathsf{true}, \mathsf{i}!\mathsf{ans}.\mathsf{id}$$

$$m_{\mathbf{r}} \overset{\text{def}}{=} \mathsf{rec}\, x. \begin{pmatrix} (d)?\mathsf{req}, \mathsf{true}, \mathsf{j}?\mathsf{req}.x \;+\; (d)!\mathsf{ans}, \mathsf{true}, \mathsf{j}!\mathsf{ans}.x \\ +\;\; (d)?\mathsf{cls}, \mathsf{true}, \mathsf{j}?\mathsf{cls}.x \end{pmatrix}.$$

When instrumented with a system, $m_{\mathbf{i}}$ inserts action $\mathsf{i}!\mathsf{ans}$, after the system inputs a request $\mathsf{i}?\mathsf{req}$, before behaving as the identity transducer. Concretely, the system $m_{\mathbf{i}}[s_{\mathbf{b}}]$, where $s_{\mathbf{b}}$ is from Example 2, can only start the computation as follows:

$$m_{\mathbf{i}}[s_{\mathbf{b}}] \xrightarrow{\mathsf{i}?\mathsf{req}} \bullet, \mathsf{true}, \mathsf{i}!\mathsf{ans}.\mathsf{id}[s'_{\mathbf{b}}] \xrightarrow{\mathsf{i}!\mathsf{ans}} \mathsf{id}[s'_{\mathbf{b}}] \xrightarrow{\mathsf{i}!\mathsf{ans}} \ldots$$
$$(\text{where } s'_{\mathbf{b}} = \mathsf{i}!\mathsf{ans}.s_{\mathbf{b}} + \mathsf{i}!\mathsf{ans}.\mathsf{i}!\mathsf{ans}.s_{\mathbf{b}}).$$

By contrast, $m_{\mathbf{r}}$ transforms input actions with either payload req or cls and output actions with payload ans on any port name, into the respective actions on port j. For instance, we have that:

$$m_{\mathbf{r}}[s_{\mathbf{b}}] \xrightarrow{j?\mathsf{req}} m_{\mathbf{r}}[s'_{\mathbf{b}}] \xrightarrow{j!\mathsf{ans}} m_{\mathbf{r}}[s_{\mathbf{b}}] \xrightarrow{j?\mathsf{cls}} m_{\mathbf{r}}[\mathsf{nil}].$$

Consider now the two suppression transducers $m_{\mathbf{s}}$ and $m_{\mathbf{t}}$ for actions on ports other than j:

$$m_{\mathbf{s}} \overset{\text{def}}{=} \mathsf{rec}\, x.\big(d?\mathsf{req}, \mathsf{true}, d?\mathsf{req}.x + (d)!\mathsf{ans}, d \neq \mathsf{j}, \bullet.x\big)$$
$$m_{\mathbf{t}} \overset{\text{def}}{=} (d)?\mathsf{req}, \mathsf{true}, d?\mathsf{req}.\mathsf{rec}\, x.\big(d!\mathsf{ans}, \mathsf{true}, d!\mathsf{ans}.$$
$$\mathsf{rec}\, y.\big(d!\mathsf{ans}, \mathsf{true}, \bullet.y + d?\mathsf{req}, \mathsf{true}, d?\mathsf{req}.x\big)\big).$$

Monitor $m_{\mathbf{s}}$ suppresses every answer on ports other than j, and continues to do so after every request on such ports. When instrumented with $s_{\mathbf{b}}$ from Example 2, we can observe the following behaviour:

$$m_{\mathbf{s}}[s_{\mathbf{b}}] \xrightarrow{\mathsf{i}?\mathsf{req}} m_{\mathbf{s}}[s'_{\mathbf{b}}] \xrightarrow{\tau} m_{\mathbf{s}}[s_{\mathbf{b}}] \xrightarrow{\mathsf{i}?\mathsf{req}} m_{\mathbf{s}}[s'_{\mathbf{b}}] \xrightarrow{\tau} m_{\mathbf{s}}[s_{\mathbf{b}}] \ldots$$

Note that $m_{\mathbf{s}}$ does not specify a transformation behaviour for when the monitored system produces inputs with payload other than req The instrumentation handles this underspecification by defaulting to the identity transducer; in the case of $s_{\mathbf{b}}$ we get $m_{\mathbf{s}}[s_{\mathbf{b}}] \xrightarrow{\mathsf{i}?\mathsf{cls}} \mathsf{id}[\mathsf{nil}]$.

Transducer $m_{\mathbf{t}}$ performs slightly more elaborate transformations. For interactions on ports other than j, it suppresses consecutive answers that are output by the system following any serviced request (i.e., a req input on i followed by an ans output on i) sequence. For $s_{\mathbf{b}}$ we can observe the following:

$$m_{\mathbf{t}}[s_{\mathbf{b}}] \xRightarrow{\mathsf{i}?\mathsf{req}\cdot\mathsf{i}!\mathsf{ans}} \mathsf{rec}\, y.\big(\mathsf{i}!\mathsf{ans}, \mathsf{true}, \bullet.y + \mathsf{i}?\mathsf{req}, \mathsf{true}, \mathsf{i}?\mathsf{req}.m'_{\mathbf{t}}\big)[\mathsf{i}!\mathsf{ans}.s_{\mathbf{b}}]$$
$$\xrightarrow{\tau} \mathsf{rec}\, y.\big(\mathsf{i}!\mathsf{ans}, \mathsf{true}, \bullet.y + \mathsf{i}?\mathsf{req}, \mathsf{true}, \mathsf{i}?\mathsf{req}.m'_{\mathbf{t}}\big)[s_{\mathbf{b}}]$$

where

$$m'_{\mathbf{t}} \overset{\text{def}}{=} \mathsf{rec}\, x.\big(\mathsf{i}!\mathsf{ans}, \mathsf{true}, \mathsf{i}!\mathsf{ans}.\mathsf{rec}\, y.\big(\mathsf{i}!\mathsf{ans}, \mathsf{true}, \bullet.y + \mathsf{i}?\mathsf{req}, \mathsf{true}, \mathsf{i}?\mathsf{req}.x\big)\big)$$

□

In the sequel, we find it convenient to refer to $\underline{p}$ as the transformation pattern p where all its binding occurrences are converted to free occurrences, e.g., $\underline{(d)!(e)}$ denotes $d!e$. As shorthand notation, we elide the second pattern p' in a transducer $\overline{p, c, p'.m}$ whenever $p'{=}\underline{p}$ and simply write $p, c.m$; note that if $\mathbf{bv}(p) = \emptyset$, then $\underline{p}{=}p$. Similarly, we elide c whenever $c{=}\mathsf{true}$. This allows us to express $m_{\mathbf{t}}$ from Example 3 as $(d)?\mathsf{req}, d \neq \mathsf{j}.\mathsf{rec}\,x.\big(d!\mathsf{ans}.\mathsf{rec}\,y.\big(d!\mathsf{ans}, \bullet.y + d?\mathsf{req}.x\big)\big)$.

4 Enforcement and enforceability

We investigate what it means for the monitors and instrumentation defined in Fig. 2 to enforce a branching-time property. We follow the template of previous work such as Ligatti et al. [4] and define enforcement in terms of two criteria:

(Enforcement) Soundness which requires that enforced behaviour should indeed satisfy the property being enforced; and
(Enforcement) Transparency which regulates the extent of intervention of the enforcing monitor whenever the system, or exhibited behaviour, already satisfies the property being enforced.

There are, however, important differences that are specific to our setting of Figs. 1 and 2 that prevent us from directly using existing definitions for these two criteria. For one, branching-time properties are defined over the computation graph of the SuS which might have several executions apart from the one that is currently being observed; by contrast, linear-time properties in prior RE investigations describe how the *current* execution is expected to be. For two, our monitor operational model is different from those assumed by other formal studies of enforcement. Concretely, it can handle first-order events where the data can be *learnt at runtime* whereas monitors used by other formal studies of enforcement cannot. In addition, we purposefully use an operational model that can potentially express *non-deterministic* monitor behaviour; As shown in prior work [25, 61–68], non-deterministic monitor behaviour is prone to arise in contexts such as first-order properties and automated monitor synthesis. Since we later consider automated monitor synthesis, we wanted to assume a framework that incorporates such behaviour in order to force our enforcement definitions to take it into consideration.

In the case of enforcement soundness, we should expect that whenever the monitor m *enforces* the property φ, then for *any* system s, the resulting composite system obtained from instrumenting m with it following the operational semantics of Fig. 2, $m[s]$, should satisfy the property of interest, φ. Note that a monitor m could, in principle, still satisfy soundness for the property φ even if it behaves non-deterministically, as long as all the possible non-deterministic enforcement operations employed all fall within the behaviour specified by φ. There is, of course, a caveat: the property being enforced *must be satisfiable*, i.e., $\llbracket\varphi\rrbracket \neq \emptyset$, for otherwise it would be impossible for the enforcing monitor to produce *any* satisfying behaviour.

Definition 2 (*Sound enforcement*) Monitor m *soundly enforces* a formula φ, denoted as $\mathsf{senf}(m, \varphi)$, iff for every LTS $\langle \textsc{Sys}, \textsc{Act}{\cup}\{\tau\}, \rightarrow\rangle$ and system states $s \in \textsc{Sys}$, $\llbracket\varphi\rrbracket \neq \emptyset$ implies $m[s] \in \llbracket\varphi\rrbracket$. □

Example 4 In general, showing that a monitor soundly enforces a formula requires showing this for *every* possible system. However, in this example we give an intuition based on systems $s_{\mathbf{g}}$ and $s_{\mathbf{b}}$. So recall φ_1, $s_{\mathbf{g}}$ and $s_{\mathbf{b}}$ from Example 2 where $s_{\mathbf{g}} \in \llbracket\varphi_1\rrbracket$ (hence φ_1 is satisfiable) and $s_{\mathbf{b}} \notin \llbracket\varphi_1\rrbracket$. For the monitors $m_{\mathbf{i}}$, $m_{\mathbf{r}}$, $m_{\mathbf{s}}$ and $m_{\mathbf{t}}$ presented in Example 3, we have that:

- $m_{\mathbf{i}}[s_{\mathbf{b}}] \notin [\![\varphi_1]\!]$, since $m_{\mathbf{i}}[s_{\mathbf{b}}] \xrightarrow{\mathsf{i?req}} (\bullet, \mathsf{i!ans.id})[s'_{\mathbf{b}}] \xrightarrow{\mathsf{i!ans}} \mathsf{id}[s'_{\mathbf{b}}] \xrightarrow{\mathsf{i!ans}} \mathsf{id}[s_{\mathbf{b}}]$. This counter-example implies that $\neg\mathsf{senf}(m_{\mathbf{i}}, \varphi_1)$.
- $m_{\mathbf{r}}[s_{\mathbf{g}}] \in [\![\varphi_1]\!]$ and $m_{\mathbf{r}}[s_{\mathbf{b}}] \in [\![\varphi_1]\!]$. Intuitively, this is because the ensuing instrumented systems only generate (replaced) actions that are not of concern to φ_1. Since this behaviour applies to any system $m_{\mathbf{r}}$ is composed with, we can conclude that $\mathsf{senf}(m_{\mathbf{r}}, \varphi_1)$.
- $m_{\mathbf{s}}[s_{\mathbf{g}}] \in [\![\varphi_1]\!]$ and $m_{\mathbf{s}}[s_{\mathbf{b}}] \in [\![\varphi_1]\!]$ because the resulting instrumented systems never produce outputs with ans on a port number other than j. We can thus conclude that $\mathsf{senf}(m_{\mathbf{s}}, \varphi_1)$.
- $m_{\mathbf{t}}[s_{\mathbf{g}}] \in [\![\varphi_1]\!]$ and $m_{\mathbf{t}}[s_{\mathbf{b}}] \in [\![\varphi_1]\!]$. Since the resulting instrumentation suppresses consecutive answers (if any) after any number of serviced requests on any port other than j, we can conclude that $\mathsf{senf}(m_{\mathbf{t}}, \varphi_1)$. □

By itself, sound enforcement is a relatively weak requirement for adequate enforcement as it does not regulate the *extent* of the induced enforcement. More concretely, consider the case of monitor $m_{\mathbf{s}}$ from Example 3. Although $m_{\mathbf{s}}$ manages to suppress the violating executions of system $s_{\mathbf{b}}$, thereby bringing it in line with property φ_1, it needlessly modifies the behaviour of $s_{\mathbf{g}}$ (namely it prohibits it from producing any outputs with ans on port numbers different from j), even though it satisfies φ_1. Thus, in addition to sound enforcement it is customary to also require a *transparency* condition for adequate enforcement. Since our properties of interest (i.e., first-order branching-time properties) describe execution graphs, one possible interpretation of such requirement dictates that, whenever a system s already satisfies the property φ, the assigned monitor m should not alter the behaviour of s. Put differently, the behaviour of the enforced system should be equivalent to that of the original system. Again, there are various possible candidates for what constitutes to be an adequate notion of behavioural equivalence, such as trace equivalence, mutual simulation, (strong) bisimulation and weak bisimulation [56, 69]. We here opt for the strongest possible definition from those mentioned, namely (strong) bisimulation (Definition 1), which also implies all of the other equivalences mentioned here (i.e., if two systems are strongly bisimilar, they are also weakly bisimilar, mutually similar and trace equivalent).

Definition 3 (*Transparent enforcement*) A monitor m is *transparent* when enforcing a formula φ, denoted as $\mathsf{tenf}(m, \varphi)$, iff for *all* LTSs $\langle \textsc{Sys}, \textsc{Act} \cup \{\tau\}, \rightarrow \rangle$ and system states $s \in \textsc{Sys}$, whenever $s \in [\![\varphi]\!]$ then $m[s] \sim s$. □

Example 5 We have already argued—via the counter-example $s_{\mathbf{g}}$—why $m_{\mathbf{s}}$ does *not* transparently enforce φ_1. We can also argue easily why $\neg\mathsf{tenf}(m_{\mathbf{r}}, \varphi_1)$ also holds: the simple system $\mathsf{i?req.i!ans.nil}$ trivially satisfies φ_1 but, clearly, we have the inequality $m_{\mathbf{r}}[\mathsf{i?req.i!ans.nil}] \not\sim \mathsf{i?req.i!ans.nil}$ since $m_{\mathbf{r}}[\mathsf{i?req.i!ans.nil}] \xrightarrow{\mathsf{j?req}} m_{\mathbf{r}}[\mathsf{nil}]$ and $\mathsf{i?req.i!ans.nil} \overset{j\mathsf{?req}}{\not\longrightarrow}$.

It turns out, however, that $\mathsf{tenf}(m_{\mathbf{t}}, \varphi_1)$ holds. Although this property is not as easy to show—due to the universal quantification over all systems—we can get a fairly good intuition for why this is the case via the example $s_{\mathbf{g}}$, since this system satisfies φ_1 and one can easily establish that $m_{\mathbf{t}}[s_{\mathbf{g}}] \sim s_{\mathbf{g}}$ holds. □

This brings us to our first formal definition of what "(monitor) m *enforces* (property) φ" can be interpreted to mean in a branching-time setting.

Definition 4 (*Enforcement*) A monitor m *enforces* property φ whenever it does so (i) *soundly*, as specified in Definition 2, and (ii) *transparently*, as specified in Definition 3. □

We note a few important aspects from Definition 4. First, the definition requires that, for a specific property, a monitor enforces *any* system *both* soundly and transparently. Put differently, we could have consolidated the respective universal quantifications in both Definitions 2 and 3 into a single outer quantification without changing the semantics of Definition 4. However, this format allows for better modularity since soundness and transparency can be understood in isolation. Second, our choice of process equivalence in Definition 3 restricts the non-deterministic behaviour of an enforced system since strong bisimulation is one of the finest equivalences; coarser choices for process equivalence would allow more non-deterministic behaviour on the part of the monitor. Third, the transparency requirement of Definition 4, by way of Definition 3, only restricts monitors from modifying the behaviour of satisfying systems, i.e., when $s \in [\![\varphi]\!]$, but fails to specify any enforcement behaviour for the cases when the SuS violates the property.

Example 6 Recall φ_1 and $s_\mathbf{b}$ from Example 2, and also $m_\mathbf{t}$ from Example 4. Even though $s_\mathbf{b} \notin [\![\varphi_1]\!]$, not all of its exhibited behaviours constitute violating traces: for instance, $s_\mathbf{b} \xRightarrow{\mathsf{i?req}\cdot\mathsf{i!ans}\cdot\mathsf{i?cls}} \mathsf{nil}$ is not a violating trace, meaning that a system that only executes this trace satisfies φ_1, e.g., $\mathsf{i?req.i!ans.i?cls.nil} \in [\![\varphi_1]\!]$. Correspondingly, we also have $m_\mathbf{t}[s_\mathbf{b}] \xRightarrow{\mathsf{i?req}\cdot\mathsf{i!ans}\cdot\mathsf{i?cls}} \mathsf{id}[\mathsf{nil}]$. □

We thus consider an alternative transparency requirement for a property φ that incorporates the expected enforcement behaviour for *both* satisfying and violating systems. More concretely, transparency can be redefined by quantifying over the *behaviours* exhibited by the system, i.e., their *traces*, rather than on the systems themselves. This trace-based version of transparency—hereinafter referred to as *trace transparency*—resembles the classical definitions that are prevalent in the runtime enforcement literature [4, 28, 70]. Monitors adhering to trace transparency must ensure that if a system trace is correct, regardless of whether it originates from a valid or invalid system, the monitor should refrain from modifying it. We define trace transparency, Definition 6, in terms of *trace-systems*, $\mathsf{sys}(t)$, as defined in Definition 5.

Definition 5 (*Trace system*) A system $\mathsf{sys}(t)$ is a *trace system* for a trace t if it can *only* execute t and all of its prefixes. Multiple trace systems for t are therefore *bisimilar*. □

Definition 6 (*Trace transparent enforcement*) A monitor m adheres to *trace transparency* when enforcing a formula φ, denoted as $\mathsf{ttenf}(m, \varphi)$ if for every trace t, when $\mathsf{sys}(t) \in [\![\varphi]\!]$ and $m[\mathsf{sys}(t)] \xRightarrow{t'} m'[\mathsf{sys}(t'')]$ then $t = t't''$. □

Going back to Example 6, a trace-transparent monitor $m_\mathbf{tt}$ ensures that although $s_\mathbf{b} \notin [\![\varphi_1]\!]$, its valid traces, such as $\mathsf{i?req.i!ans.i?cls}.\varepsilon$, would not be modified at runtime, that is, since $\mathsf{sys}(\mathsf{i?req.i!ans.i?cls}.\varepsilon) \in [\![\varphi_1]\!]$, every instrumented trace u where $m_\mathbf{tt}[\mathsf{sys}(\mathsf{i?req.i!ans.i?cls}.\varepsilon)] \xRightarrow{u}$, is a prefix of $\mathsf{i?req.i!ans.i?cls}.\varepsilon$.

Proving that a monitor adheres to trace-transparency is, however, not an easy task as a result of the universal quantification over all possible traces.

Example 7 Consider a monitor $m_1 = a, \mathsf{true}.\mathsf{rec}\, x.b, \mathsf{true}, \bullet.x$ and formula $\varphi_2 = \langle a\rangle[b]\mathsf{ff}$. To prove that $\mathsf{ttenf}(m_1, \varphi_2)$ holds we must show that for every trace t, if $\mathsf{sys}(t) \in [\![\varphi_2]\!]$ and $m_1[\mathsf{sys}(t)] \xRightarrow{t'} m_1'[\mathsf{sys}(t'')]$ then $t = t't''$. We thus inspect the following cases for t.

(*a*) $t = ab.u$ (for some suffix u): This case holds vacuously since $\mathsf{sys}(ab.u) \notin [\![\varphi_2]\!]$.
(*b*) $t \neq ab.u$: This case also holds since monitor m_1 is unable to modify any trace that is not prefixed by ab, which means that for all t' when $m_1[\mathsf{sys}(t)] \xRightarrow{t'} m_1'[\mathsf{sys}(t'')]$ then $t = t't''$ as required.

Hence, from (a) and (b) we can conclude that $\mathsf{ttenf}(m_1, \varphi_2)$ holds. □

Although Definition 3 (Transparency) and Definition 6 provide two different ways of defining transparency, our first main result shows that trace transparency is in fact a *weaker* instance of Definition 3.

Theorem 2 *(ttenf* vs. *tenf) For every monitor m and* μHML *formula* φ*,*

(i) *tenf*(m, φ) *implies pgttenf*(m, φ)*; and that*
(ii) *ttenf*(m, φ) *does not imply tenf*(m, φ)*.* □

Proof The proof for (i) follows immediately from Definitions 3 and 6 since trace systems are a subset of the possible system states of LTSs.

To prove (ii), it suffices to find a single monitor and formula that adhere to Definition 6 but not to Definition 3. Recall the result proven in Example 7 which states that $\mathsf{ttenf}(m_1, \varphi_2)$. Using this as a counter example entails showing that $\mathsf{tenf}(m_1, \varphi_2)$ is false. Hence, if we consider system $s_1 = a.b.\mathsf{nil} + a.c.\mathsf{nil}$, despite $s_1 \in [\![\varphi_2]\!]$, we also know that $s_1 \xrightarrow{a} \cdot \xrightarrow{b} \mathsf{nil}$ while $m_1[s_1] \overset{ab}{\not\Rightarrow}$. This proves that $\mathsf{tenf}(m_1, \varphi_2)$ *does not hold* as required, and we are done.

With this result we can thus give a weaker definition for "*m enforces* φ" then the one in Definition 4 by requiring sound enforcement, Definition 3, and trace transparency, Definition 6 (instead of the transparent enforcement of Definition 3). We formally detail this in Definition 7. Theorem 2 also suggests an important observation, namely that the enforcement of branching-time properties occasionally necessitates criteria that are more stringent than those for enforcement in linear-time settings, such as those in [4, 28, 70].

Definition 7 (*Weak enforcement*) A monitor m *enforces* formula φ whenever it adheres to (i) *soundness*, Definition 2, and (ii) *trace transparency*, Definition 6. □

Enforceability: Definitions 4 and 7 establish a relationship between the semantic behaviour specified by a behavioural correctness property on the one hand, and the ability of the operational mechanism (e.g., the transducers and instrumentation of Sect. 3) to enforce the specified behaviour on the other. Said definitions can form the foundation for establishing *enforceability*, a characteristic describing whether a correctness property can be enforced. This characteristic can be extended to a logic (or a logical fragment) that is providing a syntactic description of such properties. It could then be utilised by automation tools as a filtering principle when attempting to synthesise monitors from these syntactic descriptions of properties, as argued already in [32] for the case of runtime verification.

Definition 8 (*Enforceability*) A formula φ is *enforceable* iff there *exists* a transducer m such that m *enforces* φ. A logic $\mathcal{L}$ is enforceable iff *every* formula $\varphi \in \mathcal{L}$ is *enforceable*. □

We note a few aspects of Definition 8. First, a formula is enforceable only if (at least) *one* monitor can be identified to carry out *all* the necessary enforcement *irrespective of* which SuS it is composed with. Put differently, Definition 8 does not allow us to use prior knowledge about SuS to select the most appropriate monitor to carry out the enforcement; this exemplifies a black-box treatment of the SuS. Second, Definition 8 is parametric and depends on what is considered to be an adequate definition for "*m enforces* φ". Thus, both Definitions 4 and 7 can be plugged into Definition 8 to yield different definitions for what it means for a logic/property to be enforceable.

It is worth noting that the question of whether a logic is enforceable or not is challenging. More concretely, for reasonably expressive logics (such as μHML), it is usually the case that *not* every formula can be enforced, as the following example illustrates. This can be problematic from the point of view of a tool construction that aims to automatically synthesise monitors from specifications expressed as formulas of a logic of choice [32].

Example 8 Consider the μHML property φ_{or} (an instantiation of the formula discussed in the introduction), with the two systems s_4 and s_2:

$$\varphi_{\mathsf{or}} \overset{\text{def}}{=} [\mathsf{i}!v]\mathsf{ff} \vee [\mathsf{j}!w]\mathsf{ff} \qquad s_2 \overset{\text{def}}{=} \mathsf{i}!v.\mathsf{nil} \qquad s_3 \overset{\text{def}}{=} \mathsf{j}!w.\mathsf{nil} \qquad s_4 \overset{\text{def}}{=} s_2 + s_3$$

A system satisfies φ_{or} if *either* it cannot produce action $\mathsf{i}!v$ *or* it cannot produce action $\mathsf{j}!w$. Clearly, s_4 violates this property as it can produce both. This system can only be enforced by suppressing or replacing either one of the actions, because insertions would immediately break transparency. Without loss of generality, assume that our monitors suppress actions (the same applies for action replacement). The monitor $m_2 \overset{\text{def}}{=} \mathsf{rec}\, y.\big(\mathsf{i}!v, \bullet.y + \mathsf{j}!w, \bullet.y\big)$ would be able to suppress the offending actions produced by s_4, thus obtaining $m_2[s_4] \in [\![\varphi_{\mathsf{or}}]\!]$. However, it also suppresses the sole actions $\mathsf{i}!v$ and $\mathsf{j}!w$ produced by s_2 and s_3 *resp.* even though they both satisfy φ_{or}. This would, in turn, infringe the transparency criterion of Definition 3 since it needlessly suppresses the actions of s_2 and s_3, i.e., although $s_2, s_3 \in [\![\varphi_{\mathsf{or}}]\!]$ we have $m_2[s_2] \not\sim s_2$ and similarly for s_3. Note that a weaker version of m_2, such as $\mathsf{rec}\, y.\mathsf{i}!v, \bullet.y$ (*resp.* $\mathsf{rec}\, y.\mathsf{j}!w, \bullet.y$) still breaches transparency as it modifies s_2 (*resp.* s_3) unnecessarily. Similarly, m_2 also violates the weaker requirement of trace-transparency, Definition 6. Although every trace executable by s_2, s_3 and s_4, i.e., $t \in \{(\mathsf{i}!v)\varepsilon, (\mathsf{j}!w)\varepsilon\}$, is valid, $\mathsf{sys}(t) \in [\![\varphi_{\mathsf{or}}]\!]$, we can deduce that $m_2[\mathsf{sys}(t)] \overset{t}{\not\Rightarrow}$. The intuitive reason for this is that a monitor cannot, in principle, look into the computation graph of a system, but is limited to the current trace. □

5 Synthesising suppression monitors

Despite their merits, Definitions 4, 7 and 8 are not easy to work with. The universal quantifications over all systems (in all LTSs) in Definitions 2 and 3, and over all traces in Definition 6, make it hard to establish that a monitor correctly enforces a property. Moreover, according to Definition 8, in order to determine whether a particular property is enforceable or not, one would need to show the existence of a monitor that correctly enforces it; put differently, showing that a property is *not* enforceable entails another universal quantification, this time showing that no monitor can possibly enforce the property (recall that Example 8 has shown that this is not necessarily the case). Lifting the question of enforceability to the level of a (sub)logic entails a further universal quantification, this time on all the formulas of the logic.

We address these problems in two ways. First, we identify a non-trivial syntactic subset of μHML that is *guaranteed to be enforceable*; in a multi-pronged approach to system verification, this result could act as a guide for whether the property should be considered at a pre-deployment or post-deployment phase. Second, for *every* formula φ in this enforceable subset, we provide an *automated procedure* to *synthesise* a monitor m from it that correctly enforces φ when instrumented over arbitrary systems, according to Definition 4. This procedure can then be used as a basis for constructing tools that automate property enforcement, similar to what has been argued for the case runtime verification [32].

In the sequel, we sharpen our enforceability study to the use of *suppression monitors*, i.e., transducers that are only allowed to intervene by dropping system actions. Despite being more constrained, suppression monitors side-step problems associated with what data to use in a

$$\varphi, \psi \in \text{sHML} ::= \mathsf{tt} \quad | \quad \mathsf{ff} \quad | \quad \bigwedge_{i\in I} \varphi_i \quad | \quad [\{p, c\}]\varphi \quad | \quad X \quad | \quad \mathsf{max}\, X.\varphi$$

Fig. 3 The syntax for the safety μHML fragment, sHML

payload-carrying action generated by the monitor, as in the case of insertion and replacement monitors: the notion of a default value for certain data domains is not always immediate. This makes suppression monitors substantially easier to implement in practice. In our case, the resulting monitor model of Sect. 3 restricted to suppression yields one that is very similar to the models proposed for runtime verification in [1, 11, 13, 62], which have been implemented as part of the detectEr tool suite.[1] and shown to induce feasible overheads [51, 71, 72]. By extension, we conjecture that our suppression monitors also induce minimal overheads when implemented in programming language environments similar to that targetted by detectEr. This also means that the first-order logic we consider in this section can be enforced in a feasible manner in practice.

Intuitively, a suppression monitor would suppress the necessary actions as soon as it becomes apparent that a violation is about to be committed by the SuS. Such an intervention intrinsically relies on the *detection* of a violation. To this effect, we use a prior result from [11], which identified a maximally-expressive logical fragment of μHML that can be handled by violation-detecting (recogniser) monitors. We therefore limit our enforceability study to a variant of this maximal safety fragment, called sHML, since a *transparent* suppression monitor cannot judiciously suppress actions without first detecting a (potential) violation. In Fig. 3 we recall the syntax for sHML, which restricts the logic to *truth* and *falsehood* (tt and ff), conjunctions ($\bigwedge_{i\in I}\varphi$, for some finite, non-empty, index set I) and only allows for recursion to be expressed through greatest fixpoints ($\mathsf{max}\, X.\varphi$). The semantics for these constructs follows from that of Fig. 1.

A standard way how to achieve our aims would be to (i) define a (total) synthesis function $(\!|-|\!) : \text{sHML} \mapsto \text{TRN}$ from sHML formulas to suppression monitors and (ii) then show that for *any* $\varphi \in$ sHML, the synthesised monitor $(\!|\varphi|\!)$ enforces φ according to Definition 4 and Definition 7. Moreover, we would also require the synthesis function to be *compositional*, whereby the definition of the monitor for a composite formula is defined in terms of the monitors obtained for the constituent subformulas. There are a number of reasons for this requirement. For one, it would simplify our analysis of the produced monitors and allow us to use standard inductive proof techniques to prove properties about the synthesis function, such as the aforementioned criterion (ii). However, a naive approach to such a scheme is bound to fail, as discussed in the next example.

Example 9 Consider an equivalent reformulation of φ_1 from Example 2.

$$\varphi_4 \overset{\text{def}}{=} [(d)?\mathsf{req}, d{\neq}\mathsf{j}]\mathsf{max}\, X. \begin{pmatrix} [d!\mathsf{ans}, \mathsf{true}][d!\mathsf{ans}, \mathsf{true}]\mathsf{ff} \wedge \\ [d!\mathsf{ans}, \mathsf{true}][d?\mathsf{req}, \mathsf{true}]X \end{pmatrix}$$

At an intuitive level, the monitor that one expects to obtain for subformula $\varphi_2' \overset{\text{def}}{=} [d!\mathsf{ans}, \mathsf{true}][d!\mathsf{ans}, \mathsf{true}]\mathsf{ff}$ is $d!\mathsf{ans}.\mathsf{rec}\, y.d!\mathsf{ans}, \bullet.y$ (i.e., a monitor that repeatedly drops every output ans that follows a serviced request on the same port), whereas the monitor obtained for the subformula $\varphi_2'' \overset{\text{def}}{=} [d!\mathsf{ans}, \mathsf{true}][d?\mathsf{req}, \mathsf{true}]X$ is $d!\mathsf{ans}.d?\mathsf{req}.x$ (assuming some variable mapping from X to x). These monitors would then be combined in the synthesis for $[(d)?\mathsf{req}, d{\neq}\mathsf{j}]\mathsf{max}\, X.\varphi_2' \wedge \varphi_2''$ as

$$m_{\mathbf{b}} \overset{\text{def}}{=} (d)?\mathsf{req}, d{\neq}\mathsf{j}.\mathsf{rec}\, x.\big(\mathsf{rec}\, y.d!\mathsf{ans}.d!\mathsf{ans}, \bullet.y + d!\mathsf{ans}.d?\mathsf{req}.x\big).$$

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

One can easily see that $m_{\mathbf{b}}$ does *not* soundly enforce φ_4. For instance, for the violating system $\mathsf{i?req.i!ans.i!ans.nil} \notin [\![\varphi_4]\!](= [\![\varphi_1]\!])$ we can observe the transition sequence $m_{\mathbf{b}}[\mathsf{i?req.i!ans.i!ans.nil}] \xRightarrow{\mathsf{i?req}\cdot\mathsf{i!ans}} (\mathsf{i?req}, \mathsf{true}.m_{\mathbf{b}})[\mathsf{i!ans.nil}]$
$\xrightarrow{\mathsf{i!ans}} \mathsf{id}[\mathsf{nil}]$. □

Instead of complicating our synthesis function to cater for anomalies such as those presented in Example 9—also making it *less* compositional in the process—we opted for a two stage synthesis procedure. First, we consider a *normalised* subset for sHML formulas, which is amenable to a (straightforward) synthesis function definition that is compositional. This also facilitates the proofs for the conditions required by Definition 4 for any synthesised monitor. As a secondary result, we show that every sHML formula without data dependencies across necessities is logically equivalent to some formula in this normalised form. We are then able to show that our two-stage approach is expressive enough to show the enforceability for this fragment of sHML.

5.1 The synthesis function

The following grammar combines necessity operators with conjunctions into one construct $\bigwedge_{i\in I}[p_i, c_i]\varphi_i$ which is written as $[p_0, c_0]\varphi_0 \wedge \ldots \wedge [p_n, c_n]\varphi_n$ for $I = \{0, \ldots, n\}$. We simply write $[p, c]\varphi$ when $|\, I\,| = 1$.

Definition 9 (sHML *normal form*) The set of normalised sHML formulas is defined as follows (where $\varphi_i \in \mathrm{sHML}_{\mathbf{nf}}$ as well):

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

In addition, normalised sHML formulas are required to satisfy the following conditions:

1. Every symbolic action in $\bigwedge_{i\in I}[p_i, c_i]\varphi_i$, must satisfy $|I| \geq 1$ and must be *disjoint*, i.e., $\#_{i\in I}\, p_i, c_i$ which entails that for every $i, j \in I, i \neq j$ implies $[\![p_i, c_i]\!] \cap [\![p_j, c_j]\!] = \emptyset$.
2. For every $\mathsf{max}\, X.\varphi$ we have $X \in \mathbf{fv}(\varphi)$.
3. Every logical variable is *guarded* by a modal necessity. □

In a (closed) normalised sHML formula, the basic terms tt and ff can never appear unguarded unless they are at the top level (e.g., we can never have $\varphi\wedge\mathsf{ff}$ or $\mathsf{max}\, X_0.\ldots\mathsf{max}\, X_n.\mathsf{ff}$). Similarly, fixpoint variables, X, must also be guarded by a modal necessity (e.g., $\mathsf{max}\, X.([\alpha]\mathsf{ff}\wedge X)$ is invalid, unlike $\mathsf{max}\, X.([\beta]\mathsf{ff}\wedge[\alpha]X)$ in which X is guarded by $[\alpha]$). Moreover, in any conjunction of necessity subformulas, $\bigwedge_{i\in I}[p_i, c_i]\varphi_i$, the necessity guards are *disjoint* and *at most one* necessity guard can be matched by any particular action. This substantially facilitates the compositional implementation of a monitor enforcing the formula since the necessary enforcement required by a specific system execution can be determined by only considering one subformula in a conjunction of possibilities.

We proceed to define our synthesis function over normalised sHML formulas.

Definition 10 The synthesis function $(\!|-|\!) : \mathrm{sHML}_{\mathbf{nf}} \mapsto \mathrm{TRN}$ is defined inductively as:

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

$$(\!|\bigwedge_{i\in I}[p_i, c_i]\varphi_i|\!) \stackrel{\text{def}}{=} \mathsf{rec}\, y.\sum_{i\in I}\begin{cases} p_i, c_i, \bullet.y & \text{if } \varphi_i = \mathsf{ff} \\ p_i, c_i, \underline{p_i}.(\!|\varphi_i|\!) & \text{otherwise} \end{cases}$$

□

The synthesis function is compositional. It assumes a bijective mapping between formula variables and monitor recursion variables and converts logical variables X accordingly, whereas maximal fixpoints, $\mathsf{max}\, X.\varphi$, are converted into the corresponding recursive monitor. The synthesis also converts truth and falsehood formulas, tt and ff, into the identity monitor id. Normalised conjunctions, $\bigwedge_{i\in I}[p_i, c_i]\varphi_i$, are synthesised into a *recursive summation* of monitors, i.e., $\mathsf{rec}\, y.\sum_{i\in I} m_i$, where y is fresh, and every branch m_i can be either of the following:

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

(ii) when m_i is derived from a branch of the form $[p_i, c_i]\mathsf{ff}$, the synthesis produces a *suppression transformation*, $p_i, c_i, \bullet$, that drops every action matching p_i, c_i, followed by the recursive variable of the branch y, i.e., a branch of the form $[p_i, c_i]\mathsf{ff}$ is translated into $p_i, c_i, \bullet.y$.

Example 10 Recall formula φ_1 from Example 2:

$$\varphi_1 \overset{\text{def}}{=} [(d)?\mathsf{req}, d{\neq}\mathsf{j}]\mathsf{max}\, X.[d!\mathsf{ans}, \mathsf{true}]([d!\mathsf{ans}, \mathsf{true}]\mathsf{ff} \wedge [d?\mathsf{req}, \mathsf{true}]X).$$

Using the synthesis function defined in Definition 10, we generate monitor

$$(\!| \varphi_1 |\!) = \mathsf{rec}\, x'.(d)?\mathsf{req}, d \neq \mathsf{j}.\mathsf{rec}\, x.\mathsf{rec}\, z.\big(d!\mathsf{ans}.\mathsf{rec}\, y.d!\mathsf{ans}, \bullet.y + d?\mathsf{req}.x\big)$$

which can be optimised by removing redundant recursive constructs (e.g., $\mathsf{rec}\, z.\_$), obtaining: $(d)?\mathsf{req}, d \neq \mathsf{j}.\mathsf{rec}\, x.\big(d!\mathsf{ans}.\mathsf{rec}\, y.d!\mathsf{ans}, \bullet.y + d?\mathsf{req}.x\big) \;=\; m_{\mathbf{t}}$.

□

It is clear that the synthesis function of Definition 10 is total for sHML$_{\mathbf{nf}}$ formulas and yields exclusively suppression monitors.

Lemma 1 *For any $\varphi \in$ sHML$_{nf}$, $(\!| \varphi |\!)$ is defined and is a suppression monitor.*

Proof By induction on the structure of φ.

We now present the second main set of results to the paper. Theorem 3 follows as a corollary of Lemma 1 and a strengthening of the stated requirement that narrows down monitors to suppression monitors, Proposition 1.

Theorem 3 (Enforcement) *The (sub)logic* sHML$_{nf}$ *is enforceable with respect to Definition 4.*

Proposition 1 (Enforcement via suppression) *The (sub)logic* sHML$_{nf}$ *is enforceable with respect to Definition 4 using only suppression monitors.*

Proof By Definition 8, the result follows if we show that for all $\varphi \in$ sHML$_{\mathbf{nf}}$, $(\!| \varphi |\!)$ *enforces* φ in the sense of Definition 4. Hence, by Definition 4, this is a corollary of Definition 2 and 3 stated below.

Proposition 2 (Enforcement soundness) *For every LTSs $\langle$*Sys*, *Act *$\cup \{\tau\}, \rightarrow\rangle$, system $s \in$ *Sys *and $\varphi \in$ sHML$_{nf}$ then* $[\![\varphi]\!] \neq \emptyset$ implies $(\!| \varphi |\!)[s] \in [\![\varphi]\!]$.

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

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

Fig. 4 A satisfaction relation for sHML formulas

Proposition 3 (Enforcement transparency) *For every LTSs $\langle \text{Sys}, \text{Act}\cup\{\tau\}, \rightarrow\rangle$, system $s\in\text{Sys}$ and $\varphi\in\text{sHML}_{\mathbf{nf}}$ then $s\in[\![\varphi]\!]$ implies $(\!|\,\varphi\,|\!)[s]\sim s$.*

As our first result, Theorem 2, states that trace transparency (Definition 6) is inherently a weaker version of transparency (Definition 3), we can also prove that $\text{sHML}_{\mathbf{nf}}$ is enforceable in the sense of Definition 7.

Theorem 4 (Weak enforcement) *The (sub)logic $\text{sHML}_{\mathbf{nf}}$ is enforceable with respect to Definition 7.*

Proof By Definition 8, this follows by showing that for every $\text{sHML}_{\mathbf{nf}}$ formula φ, $(\!|\,\varphi\,|\!)$ *enforces* φ as defined by Definition 7. Hence, in the light of Theorem 2, this result becomes a corollary of Theorem 3.

To facilitate the proofs for Propositions 2 and 3, we use the satisfaction semantics for sHML from [73] which are defined in terms of the *satisfaction relation*, $\vDash$. When restricted to sHML, $\vDash$ is the *largest relation* $\mathcal{R}$ satisfying the implications defined in Fig. 4. As these semantics are well known to agree with the sHML semantics of Fig. 1, we use $s\vDash\varphi$ in lieu of $s\in[\![\varphi]\!]$. These proofs may safely be skipped upon first reading.

Proof (Proof for Proposition 2) We prove a stronger result stating that for every system r that can be simulated by $(\!|\,\varphi\,|\!)[s]$, i.e., $r \lesssim (\!|\,\varphi\,|\!)[s]$, if $[\![\varphi]\!]\neq\emptyset$ then $r\vDash\varphi$. We prove this result by showing that relation $\mathcal{R} \overset{\text{def}}{=} \left\{ (r,\varphi) \mid [\![\varphi]\!]\neq\emptyset \textit{ and } r\lesssim(\!|\,\varphi\,|\!)[s] \right\}$ is a *satisfaction relation* ($\vDash$) as defined by the rules in Fig. 4. We proceed by case analysis on the structure of φ.

Cases $\varphi\in\{X,\mathsf{ff}\}$. These cases do not apply as when $\varphi\in\{X,\mathsf{ff}\}$ then $[\![\varphi]\!]=\emptyset$.

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

Case $\varphi=\mathsf{max}\,X.\varphi$ and $X\in\mathbf{fv}(\varphi)$. Let's assume that $(r,\mathsf{max}\,X.\varphi)\in\mathcal{R}$ and so we have that

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

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

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

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

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

Case $\varphi = \bigwedge_{i\in I}[p_i, c_i]\varphi_i$ and $\#_{h\in I}\, p_h, c_h$. Now, let's start by assuming that $(r, \bigwedge_{i\in I}[p_i, c_i]\varphi_i) \in \mathcal{R}$ and so we have that

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

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

By the definition of $(\!| - |\!)$ we further know that

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

which can be further unfolded as

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

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

(i) *A violating branch*, $[p_j, c_j]\mathsf{ff}$:

To prove that $(r, [p_j, c_j]\mathsf{ff}) \in \mathcal{R}$ we must show that (a) $[\![[p_j, c_j]\mathsf{ff}]\!] \neq \emptyset$, (b) $r \lesssim (\!|[p_j, c_j]\mathsf{ff}|\!)[s]$, and (c) that for every action α, when $p_j, c_j(\alpha) = \sigma$, then there does not exist a system r' such that $r \overset{\alpha}{\Rightarrow} r'$. From (4) and the definition of $[\![-]\!]$, we can immediately infer that (a) holds, and so we have that

$$[\![[p_j, c_j]\mathsf{ff}]\!] \neq \emptyset. \tag{7}$$

We now note that since from (6) we know that branch $[p_j, c_j]\mathsf{ff}$ is synthesised into a *suppression monitor* $p_j, c_j, \bullet.m$, we infer that this branch can only suppress actions matching p_j, c_j, while monitor $m = (\!|\bigwedge_{i\in I}[p_i, c_i]\varphi_i|\!)$ can possibly suppress other actions as well. Hence, the composite system $m[s]$ (for any s) can *at most* perform the same actions as $(\!|[p_j, c_j]\mathsf{ff}|\!)[s]$ and so from (5) we can deduce that (b) holds since

$$r \lesssim (\!|\textstyle\bigwedge_{i\in I}[p_i, c_i]\varphi_i|\!)[s] \lesssim (\!|[p_j, c_j]\mathsf{ff}|\!)[s] \tag{8}$$

as required. Finally, from (6) we know that monitor m was synthesised from a normalised conjunction which is *disjoint* ($\#_{h\in I}\, p_h, c_h$) from which we conclude that whenever the system performs action α such that $p_j, c_j(\alpha) = \sigma$, only the suppression branch $p_j, c_j, \bullet.m$ (which is a single branch of m in (6)) can be selected via rule eSel. Once this branch is selected, the action is suppressed via rules eTrn and iSup which cause the composite system $m[s]$ to transition over a silent τ action to its recursive derivative m. This means that $m[s] \overset{\alpha}{\not\Rightarrow}$ and so from (5) we can deduce that (c) also holds since

$$\nexists r' \cdot r \overset{\alpha}{\Rightarrow} r' \tag{9}$$

which means that any modal necessity that precedes ff can never be satisfied by r as required. This case thus holds by (7), (8) and (9).

(ii) *A non-violating branch*, $[p_j, c_j]\varphi_j$ (*where* $\varphi_j \neq \mathsf{ff}$):

To prove that this branch is in $\mathcal{R}$, $(r, [p_j, c_j]\varphi_j) \in \mathcal{R}$, we must show that (a) $[\![[p_j, c_j]\varphi_j]\!] \neq \emptyset$, (b) $r \sqsubseteq (\![[p_j, c_j]\varphi_j]\!)[s]$ and then that (c) for every action α and derivative r', when $p_j, c_j(\alpha) = \sigma$ and $r \overset{\alpha}{\Rightarrow} r'$ then $(r', \varphi_j\sigma) \in \mathcal{R}$. From (4) and by the definition of $[\![-]\!]$ we can immediately determine that (a) holds, and so that

$$[\![[p_j, c_j]\varphi_j]\!] \neq \emptyset \tag{10}$$

and since $(\![[p_j, c_j]\varphi_j]\!) = \mathsf{rec}\, y.p_j, c_j.(\![\varphi]\!)$, from (6) we deduce that both monitors $m = (\![\bigwedge_{i\in I} [p_i, c_i]\varphi_i]\!)$ and $(\![[p_j, c_j]\varphi_j]\!)$ refrain from modifying actions matching p_j, c_j but m may suppress more actions. We can thus infer that for all s, $m[s] \sqsubseteq (\![[p_j, c_j]\varphi_j]\!)[s]$ and so from (5) we can deduce that (b) holds since

$$r \sqsubseteq m[s] \sqsubseteq (\![[p_j, c_j]\varphi_j]\!)[s] \tag{11}$$

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

$$p_j, c_j(\alpha) = \sigma \tag{12}$$

$$r \overset{\alpha}{\Rightarrow} r' \tag{13}$$

and so from (5) and (13) we can deduce that

$$m[s] \overset{\alpha}{\Rightarrow} q \;\; (\text{where } r' \sqsubseteq q). \tag{14}$$

Hence, by the definition of $\overset{\alpha}{\Rightarrow}$ we know that the weak transition in (14) is composed from zero or more τ-transitions followed by the α-transition, i.e., that

$$m[s] \xrightarrow{\tau} {}^{*} q' \xrightarrow{\alpha} q. \tag{15}$$

By the rules in our model we know that the τ-reductions in (15) could have been the result of either one of these instrumentation rules, namely ISUP or IASY. From (6) we, however, know that whenever an action is suppressed (via ISUP) the synthesised monitor m always recurses back to its original form m and in this case only s changes its state to some s'; the same effect occurs if rule IASY is applied instead. Hence, we know that $q' = m[s']$ (for some derivative s' of s), and so from (15) we have that

$$m[s'] \xrightarrow{\alpha} q. \tag{16}$$

From (12) we also know that the reduction in (16) can only be the result of rule ITRN, and so we can infer that $s' \xrightarrow{\alpha} s''$ and that

$$q = m'[s''] \tag{17}$$

$$m \xrightarrow{\alpha \blacktriangleright \alpha} m'. \tag{18}$$

Since we know that $[p_j, c_j]\varphi_j$ and $\varphi_j \neq \mathsf{ff}$, from (6) we know that m defines an *identity branch* of the form $p_j, c_j.(\![\varphi_j]\!)$ which is *completely disjoint* from the rest of the monitors. This is true since m is derived from a normalised conjunction in which $\#_{i\in I}\, p_i, c_i$. Hence, from (6), (12) and (18) we can deduce that

$$m' = (\![\varphi_j\sigma]\!). \tag{19}$$

Since from (10) and by the definition of $[\![-]\!]$ we know that $[\![\varphi_j\sigma]\!] \neq \emptyset$ and from (14), (17) and (19) we have that $r' \precsim (\!|\,\varphi_j\sigma\,|\!)[s'']$, by the definition of $\mathcal{R}$ we have that $(r', \varphi_j\sigma) \in \mathcal{R}$. From this we can conclude that (c) holds as well, which means that

$$\forall\alpha, r' \cdot \; \textit{if } p_j, c_j(\alpha) = \sigma \textit{ and } r \overset{\alpha}{\Rightarrow} r' \text{ then } (r', \varphi_j\sigma) \in \mathcal{R}. \tag{20}$$

This case is therefore done by (10), (11) and (20).

Proof for Proposition 3 To prove this proposition, we show that relation $\mathcal{R} \overset{\text{def}}{=} \{(s, (\!|\,\varphi\,|\!)[s]) \mid s \vDash \varphi\}$ is a *strong bisimulation relation* by showing that it satisfies the following transfer properties for each $(s, (\!|\,\varphi\,|\!)[s]) \in \mathcal{R}$:

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

We prove (a) and (b) separately by assuming that $s \vDash \varphi$ in both cases as defined by relation $\mathcal{R}$.

We also make reference to the τ-closure property of sHML, Proposition 4, proved in [73].

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

We now proceed to prove (a) by case analysis on φ.

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

Case $\varphi = \mathsf{tt}$ We now assume that $s \vDash \mathsf{tt}$ and that

$$s \xrightarrow{\mu} s' \tag{21}$$

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

- $\mu = \tau$: Since $\mu = \tau$, we can apply rule iAsy on (21) and get that

$$(\!|\,\mathsf{tt}\,|\!)[s] \xrightarrow{\tau} (\!|\,\mathsf{tt}\,|\!)[s'] \tag{22}$$

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

$$(s', (\!|\,\mathsf{tt}\,|\!)[s']) \in \mathcal{R} \tag{23}$$

  as required, which means that this case is done by (22) and (23).
- $\mu = \alpha$: Since id encodes the 'catch-all' monitor, $\mathsf{rec}\, y.(d)!(e), \mathsf{true}, d!e.y + (d)?(e), \mathsf{true}, d?e.y$, we can deduce that $\mathsf{id} \xrightarrow{\alpha \blacktriangleright \alpha} \mathsf{id}$ from rules eRec and eTrn and then rule iTrn, which we can further refine as

$$(\!|\,\mathsf{tt}\,|\!)[s] \xrightarrow{\alpha} (\!|\,\mathsf{tt}\,|\!)[s'] \tag{24}$$

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

$$(s', (\!|\,\mathsf{tt}\,|\!)[s']) \in \mathcal{R} \tag{25}$$

  as required, and so this case is done by (24) and (25).

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

$$s \vDash \bigwedge_{i\in I}[p_i, c_i]\varphi_i \tag{26}$$

$$s \xrightarrow{\mu} s' \tag{27}$$

and so by the definition of $\vDash$ and (26) we have that for every index $i \in I$ and action $\beta \in \text{ACT}$,

$$s \overset{\beta}{\Longrightarrow} s' \textit{ and } p_i, c_i(\beta) = \sigma \textit{ implies } s \vDash \bigwedge_{i\in I}[p_i, c_i]\varphi_i. \tag{28}$$

Since $\mu \in \{\tau, \alpha\}$, we must consider both possibilities for (27).

– $\mu = \tau$: Since $\mu = \tau$, we can apply rule IASY on (27) and obtain

$$(\!| \bigwedge_{i\in I}[p_i, c_i]\varphi_i |\!)[s] \xrightarrow{\tau} (\!| \bigwedge_{i\in I}[p_i, c_i]\varphi_i |\!)[s'] \tag{29}$$

as required. Since $\mu = \tau$, and since we know that sHML is τ-closed, from (26), (27) and Proposition 4, we can deduce that $s' \vDash \bigwedge_{i\in I}[p_i, c_i]\varphi_i$, so that by the definition of $\mathcal{R}$ we conclude

$$(s', (\!| \bigwedge_{i\in I}[p_i, c_i]\varphi_i |\!)[s']) \in \mathcal{R} \tag{30}$$

as required. This subcase is therefore done by (29) and (30).

– $\mu = \alpha$: Since $\mu = \alpha$, from (27) we know that

$$s \xrightarrow{\alpha} s' \tag{31}$$

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

$$(\!| \bigwedge_{i\in I}[p_i, c_i]\varphi_i |\!) = \mathsf{rec}\, y.\Big(\sum_{i\in I} \begin{cases} p_i, c_i, \bullet.y & \text{if } \varphi_i = \mathsf{ff} \\ p_i, c_i.(\!| \varphi_i |\!) & \text{otherwise} \end{cases}\Big). \tag{32}$$

Since the branches in the conjunction are all disjoint, $\#_{i\in I}\, p_i, c_i$, we know that *at most one* of the branches can match the same action α. Hence, we consider two cases, namely:

– *No matching branches* (i.e., $\forall i \in I \cdot p_i, c_i(\alpha) = \mathsf{undef}$): Since none of the symbolic transformations in (32) can match action α and since we do not synthesise insertion monitors, we know that the monitor can only default to id (via rule IDEF) and so from (31) we have that

$$(\!| \bigwedge_{i\in I}[p_i, c_i]\varphi_i |\!)[s] \xrightarrow{\alpha} (\!| \mathsf{tt} |\!)[s'] \quad (\text{since} \mathsf{id} = (\!| \mathsf{tt} |\!)) \tag{33}$$

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

$$(s', (\!| \mathsf{tt} |\!)[s']) \in \mathcal{R} \tag{34}$$

as required. This case is therefore done by (33) and (34).

– *One matching branch* (i.e., $\exists j \in I \cdot p_j, c_j(\alpha) = \sigma$):
From (32) we infer that the synthesised monitor can only suppress actions that are defined by violating necessities. However, from (28) we also deduce that s is *incapable* of executing such an action as otherwise would contradict assumption (26). Hence, since we now assume that $\exists j \in I \cdot p_j, c_j(\alpha) = \sigma$, from (32) we deduce that this action can only be transformed by an identity transformation and so by rule ETRN we have that

$$p_j, c_j.(\!| \varphi_j |\!) \xrightarrow{\alpha \blacktriangleright \alpha} (\!| \varphi_j\sigma |\!). \tag{35}$$

By applying rules ESEL, EREC on (35) and by (31), (32) and ITRN we get that

$$⦇ \bigwedge_{i\in I} [p_i, c_i]\varphi_i ⦈[s] \xrightarrow{\alpha} ⦇ \varphi_j\sigma ⦈[s'] \tag{36}$$

as required. By (28), (31) and since we assume that $\exists j \in I \cdot p_j, c_j(\alpha) = \sigma$ we have that $s' \vDash \varphi_j\sigma$, and so by the definition of $\mathcal{R}$ we conclude that

$$(s', ⦇ \varphi_j\sigma ⦈[s']) \in \mathcal{R} \tag{37}$$

as required. Hence, this subcase is done by (36) and (37).

Case $\varphi = \max X.\varphi$ *and* $X \in \mathbf{fv}(\varphi)$. Now, let's assume that

$$s \xrightarrow{\mu} s' \tag{38}$$

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

$$s \vDash \varphi\{\max X.\varphi/X\}. \tag{39}$$

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

$$\varphi = \max Y_0.\ldots\max Y_n.\bigwedge_{i\in I}[p_i, c_i]\varphi_i \tag{40}$$

and so by (39), (40) and the definition of $\vDash$ we have that

$$s \vDash (\textstyle\bigwedge_{i\in I}[p_i, c_i]\varphi_i)\{\cdot\cdot\} \quad \textit{where}$$
$$\{\cdot\cdot\} = \{\max X.\varphi/X, (\max Y_0.\ldots\max Y_n.\bigwedge_{i\in I}[p_i, c_i]\varphi_i)/Y_0, \ldots, (\max Y_n.\bigwedge_{i\in I}[p_i, c_i]\varphi_i)/Y_n\}. \tag{41}$$

Since we know (38) and (41), from this point onwards the proof proceeds as per the previous case. We thus omit this part of the proof and immediately deduce that

$$\exists m' \cdot ⦇ (\textstyle\bigwedge_{i\in I}[p_i, c_i]\varphi_i)\{\cdot\cdot\} ⦈[s] \xrightarrow{\mu} ⦇ m' ⦈[s'] \tag{42}$$

$$(s', ⦇ m' ⦈[s']) \in \mathcal{R} \tag{43}$$

and so since $⦇ (\bigwedge_{i\in I}[p_i, c_i]\varphi_i)\{\cdot\cdot\} ⦈$ synthesises the *unfolded equivalent* as per $⦇ \varphi\{\max X.\varphi/X\} ⦈$, from (42) we can conclude that

$$\exists m' \cdot ⦇ \varphi\{\max X.\varphi/X\} ⦈[s] \xrightarrow{\mu} ⦇ m' ⦈[s'] \tag{44}$$

as required, and so this case holds by (43) and (44).
These cases thus allow us to conclude that (*a*) holds. We now proceed to prove (*b*) using the same case analysis approach.

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

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

$$⦇ \mathsf{tt} ⦈[s] \xrightarrow{\mu} r'. \tag{45}$$

Since $\mu \in \{\tau, \alpha\}$, we must consider each case.

- $\mu = \tau$: Since $\mu = \tau$, the transition in (45) can be performed either via iSup, or iAsy. We must therefore consider these cases.
  - iAsy: From rule iAsy and (45) we thus know that $r' = (\!|\, \mathsf{tt} \,|\!)[s']$ and that $s \xrightarrow{\tau} s'$ as required. Also, since every system state satisfies tt, we know that $s' \vDash \mathsf{tt}$ as well, and so we are done since by the definition of $\mathcal{R}$ we know that $(s', (\!|\, \mathsf{tt} \,|\!)[s']) \in \mathcal{R}$.
  - iSup: This case does not apply since from rule iSup and (45) we know that: $r' = m'[s'], s \xrightarrow{\alpha} s'$ and that $(\!|\, \mathsf{tt} \,|\!) \xrightarrow{\alpha \blacktriangleright \bullet} m'$ which is a *false* assumption as $(\!|\, \mathsf{tt} \,|\!) = \mathsf{id}$.
- $\mu = \alpha$: Since $\mu = \alpha$, the transition in (45) can be performed either via iDef, iIns or iTrn. We consider each case.
  - iDef: This case does not apply since $(\!|\, \mathsf{tt} \,|\!) = \mathsf{id}$ which cannot ever reach a state n where $n \overset{\alpha}{\not\rightarrow}$ and $n \overset{\bullet}{\not\rightarrow}$.
  - iIns: This case does not apply since from (45) and by the definition of $(\!|\, - \,|\!)$ we know that the synthesised monitor does not include action insertions.
  - iTrn: By applying rule iTrn on (45) we know that $r' = m'[s']$ such that

$$s \xrightarrow{\beta} s' \tag{46}$$

$$(\!|\, \mathsf{tt} \,|\!) \xrightarrow{\alpha \blacktriangleright \beta} m'. \tag{47}$$

Since $(\!|\, \mathsf{tt} \,|\!) = \mathsf{id} = \mathsf{rec}\, y.(d)!(e), \mathsf{true}, d!e.y + (d)?(e), \mathsf{true}, d?e.y$, by applying rules eRec, eSel and eTrn to (47) we know that $\alpha = \beta$, $m' = \mathsf{id} = (\!|\, \mathsf{tt} \,|\!)$, meaning that $r' = (\!|\, \mathsf{tt} \,|\!)[s']$. Hence, since every system state satisfies tt we know that $s' \vDash \mathsf{tt}$, so that by the definition of $\mathcal{R}$ we conclude that

$$(s', (\!|\, \mathsf{tt} \,|\!)[s']) \in \mathcal{R}. \tag{48}$$

Hence, we are done by (46) and (48) since we know that $\alpha = \beta$.

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

$$s \vDash \textstyle\bigwedge_{i \in I} [p_i, c_i]\varphi_i \tag{49}$$

$$(\!|\, \textstyle\bigwedge_{i \in I} [p_i, c_i]\varphi_i \,|\!)[s] \xrightarrow{\mu} r'. \tag{50}$$

From (49) and by the definition of $\vDash$, we can deduce that

$$\forall i \in I, \beta \in \textsc{Act} \cdot s \overset{\beta}{\Longrightarrow} s' \textit{ and } p_i, c_i(\alpha) = \sigma \textit{ implies } s' \vDash \varphi_i\sigma \tag{51}$$

and from (50) and by the definition of $(\!|\, - \,|\!)$ we have that

$$\left(\mathsf{rec}\, y.\sum_{i \in I} \begin{cases} p_i, c_i, \bullet.y & \text{if } \varphi_i = \mathsf{ff} \\ p_i, c_i.(\!|\, \varphi_i \,|\!) & \text{otherwise} \end{cases}\right)[s'] \xrightarrow{\mu} r'. \tag{52}$$

From (52) we know that the synthesised monitor can only suppress an action β when this satisfies a violating necessity. However, we can also infer that s is *incapable* of performing β as otherwise it would contradict with assumption (51) since $s' \vDash \mathsf{ff}$ does not hold. Hence, we can safely conclude that the synthesised monitor in (52) does *not* suppress any actions of s, and so we conclude that

$$\forall \alpha \in \textsc{Act}, s' \in \textsc{Sys} \cdot s \xrightarrow{\alpha} s' \textit{ implies } (\!|\, \textstyle\bigwedge_{i \in I} [p_i, c_i]\varphi_i \,|\!) \overset{\alpha \blacktriangleright \bullet}{\not\longrightarrow}. \tag{53}$$

Since $\mu \in \{\tau, \alpha\}$, we must consider each case.

- $\mu = \tau$: Since $\mu = \tau$, from (50) we know that
$$(\!| \bigwedge_{i\in I} [p_i, c_i]\varphi_i |\!)[s] \xrightarrow{\tau} r' \tag{54}$$
The τ-transition in (54) can be the result of rules iAsy or iSup; we thus consider each eventuality.
  - iAsy: As we assume that the reduction in (54) is the result of rule iAsy, we know that $r' = (\!| \bigwedge_{i\in I} [p_i, c_i]\varphi_i |\!)[s']$ and that
$$s \xrightarrow{\tau} s' \tag{55}$$
as required. Also, since sHML is τ-closed, by (49), (55) and Proposition 4 we deduce that $s' \vDash \bigwedge_{i\in I} [p_i, c_i]\varphi_i$ as well, so that by the definition of $\mathcal{R}$ we conclude that
$$(s', (\!| \bigwedge_{i\in I} [p_i, c_i]\varphi_i |\!)[s']) \in \mathcal{R} \tag{56}$$
and so we are done by (55) and (56).
  - iSup: As we now assume that the reduction in (54) results from iSup, we have that $r' = m'[s']$ and that
$$s \xrightarrow{\alpha} s' \tag{57}$$
$$(\!| \bigwedge_{i\in I} [p_i, c_i]\varphi_i |\!) \xrightarrow{\alpha \blacktriangleright \bullet} m'. \tag{58}$$
This case does not apply since by (53) and (57) we can deduce that $(\!| \bigwedge_{i\in I} [p_i, c_i]\varphi_i |\!) \not\xrightarrow{\alpha \blacktriangleright \bullet}$ which contradicts with (58).
- $\mu = \alpha$: When $\mu = \alpha$, the transition in (52) can be performed via rules iDef, iIns or iTrn, we consider both possibilities.
  - iDef: If (52) results from iDef, we have that
$$r' = (\!| \mathsf{tt} |\!)[s'] \quad (\text{since } (\!| \mathsf{tt} |\!) = \mathsf{id}) \tag{59}$$
$$s \xrightarrow{\alpha} s'. \tag{60}$$
Consequently, as every system state satisfies tt, we know that $s' \vDash \mathsf{tt}$ and so by the definition of $\mathcal{R}$ we have that $(s', (\!| \mathsf{tt} |\!)[s']) \in \mathcal{R}$, so that from (59) we can conclude that
$$(s', r') \in \mathcal{R} \tag{61}$$
as required. Hence, this case is done by (60) and (61).
  - iIns: This case does not apply since from (52) and by the definition of $(\!| - |\!)$ we know that the synthesised monitor does not include action insertions.
  - iTrn: By assuming that (52) is obtained from rule iTrn, we know that
$$\left(\mathsf{rec}\, y.\sum_{i\in I} \begin{cases} p_i, c_i, \bullet.y & \text{if } \varphi_i = \mathsf{ff} \\ p_i, c_i.(\!| \varphi_i |\!) & \text{otherwise} \end{cases}\right) \xrightarrow{\beta \blacktriangleright \alpha} m' \tag{62}$$
$$s \xrightarrow{\beta} s' \tag{63}$$
$$r' = m'[s']. \tag{64}$$

Since from (53) we know that the synthesised monitor in (62) does not suppress any action performable by s, and since from the definition of $(\!| - |\!)$ we know that the synthesis cannot produce action replacing monitors, we can deduce that

$$\alpha = \beta. \tag{65}$$

With the knowledge of (65), from (63) we can thus deduce that

$$s \xrightarrow{\alpha} s' \tag{66}$$

as required. Knowing (65) we can also deduce that in (62) the monitor can only transform action β via an identity transformation synthesised from one of the *disjoint* conjunction branches, i.e., from a branch $p_j, c_j.(\!| \varphi_j |\!)$ for some $j \in I$. Hence, when we apply rules eRec, eSel and eTrn on (62) we deduce that

$$\exists j \in I \cdot p_j, c_j(\alpha) = \sigma \tag{67}$$

$$m' = (\!| \varphi_j \sigma |\!). \tag{68}$$

and so from (66), (67) and (51) we infer that $s' \vDash \varphi_j \sigma$ from which by the definition of $\mathcal{R}$ we have that $(s', (\!| \varphi_j \sigma |\!)[s']) \in \mathcal{R}$, and so from (64) and (68) we can conclude that

$$(s', r') \in \mathcal{R} \tag{69}$$

as required, and so this case is done by (66) and (69).

Case $\varphi = \max X.\varphi$ *and* $X \in \mathbf{fv}(\varphi)$. Now, let's assume that

$$(\!| \max X.\varphi |\!)[s] \xrightarrow{\mu} r' \tag{70}$$

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

$$s \vDash \varphi\{\max X.\varphi/X\}. \tag{71}$$

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

$$\varphi = \max Y_0 \ldots \max Y_n.\bigwedge_{i \in I} [p_i, c_i]\varphi_i \tag{72}$$

and so by (71), (72) and the definition of $\vDash$ we have that

$$s \vDash (\textstyle\bigwedge_{i \in I} [p_i, c_i]\varphi_i)\{\cdot\cdot\} \quad \textit{where}$$
$$\{\cdot\cdot\} = \{{}^{\max X.\varphi}/_{X}, {}^{(\max Y_0 \ldots \max Y_n.\bigwedge_{i \in I} [p_i, c_i]\varphi_i)}/_{Y_0}, \ldots, {}^{(\max Y_n.\bigwedge_{i \in I} [p_i, c_i]\varphi_i)}/_{Y_n}\}. \tag{73}$$

Since $(\!| (\bigwedge_{i \in I} [p_i, c_i]\varphi_i)\{\cdot\cdot\} |\!)$ synthesises the *unfolded equivalent* of $(\!| \max X.\varphi |\!)$, from (70) we know that

$$(\!| (\textstyle\bigwedge_{i \in I} [p_i, c_i]\varphi_i)\{\cdot\cdot\} |\!)[s] \xrightarrow{\mu} r'. \tag{74}$$

Hence, since we know (73) and (74), from this point onwards the proof proceeds as per the previous case. We thus omit showing the remainder of this proof.

From the above cases, we can therefore conclude that (b) holds as well. $\square$

In light of Theorems 3 and 4, in order to show that sHML is an enforceable logic, we only need to prove that for every $\varphi \in$ sHML there exists a corresponding $\psi \in$ sHML$_{\mathbf{nf}}$ with the same semantic meaning, i.e., $[\![\varphi]\!] = [\![\psi]\!]$. In fact, we go a step further and provide a constructive proof using a transformation $\langle\!\langle - \rangle\!\rangle$: sHML $\mapsto$ sHML$_{\mathbf{nf}}$ that constructs a semantically equivalent sHML$_{\mathbf{nf}}$ formula from an sHML one. As a result, from an arbitrary sHML formula φ we can then automatically synthesise a correct monitor using $(\!|\, \langle\!\langle \varphi \rangle\!\rangle \,|\!)$, which is useful for tool construction.

5.2 The normalisation algorithm

Our transformation relies on a number of steps, during which we assume sHML formulas that only use symbolic actions with *normalised* patterns p, i.e., patterns that do not use any data or free data variables (but they may use bound data variables) and necessity binding does not extend to other necessities, i.e., whenever $\varphi = [p, c]\varphi'$ then $\mathbf{bv}(p) \subseteq \mathbf{fv}(c)$ and $\mathbf{fv}(\varphi') = \emptyset$. Note that any symbolic action p, c can be easily converted into an equivalent one using normalised patterns as shown in the next example.

Example 11 Consider the symbolic action $d!\mathsf{ans}, d \neq j$ where d is free in the *SA* and ans is a data value. Such *SAs* can be converted to a corresponding normalised *SA* by replacing every occurrence of a data value and free data variable in the pattern by a fresh binding variable, and then add an equality constraint between the fresh variable and the data value or free variable it has replaced in the pattern, to the *SAs* condition. In our case, we would obtain $(e)!(f), d{\neq}j \wedge e{=}d \wedge f{=}\mathsf{ans}$ where e and f are fresh, and although d is free in the *SAs* condition, it no longer forms part of the pattern. $\square$

Our algorithm for converting *closed* sHML formulas (with normalised patterns) to sHML$_{\mathbf{nf}}$ formulas, $\langle\!\langle - \rangle\!\rangle$, is based on Aceto et al.'s work [67] for determinising (possibly open) sHML formulas defining concrete actions, and on Rabinovich's work [74] for determinising systems of equations, both of which rely on the standard powerset construction for converting NFAs into DFAs. With this algorithm we can prove the second main result of this paper.

Theorem 5 (Normalisation equivalence) *For every closed* sHML *formula φ there exists a formula $\psi \in$* sHML$_{\mathbf{nf}}$ *such that $[\![\varphi]\!]{=}[\![\psi]\!]$.* $\square$

5.3 Reconstructing sHML into sHML$_{\mathbf{nf}}$ wrt. singleton symbolic actions

We first define the normalisation algorithm for *sHML* formulas that only define *singleton symbolic actions*. Since singleton *SAs* do not bind user data, these can be easily *distinguished statically* based on their syntactic form, e.g., $\mathsf{i!ans} \neq \mathsf{i?req}$ implies $[\![\mathsf{i!ans}]\!] \cap [\![\mathsf{i?req}]\!] = \emptyset$, unlike non-singleton ones, e.g., although $(d)?\mathsf{ans} \neq \mathsf{i}?(e)$ we have that $[\![(d)?\mathsf{ans}]\!] \neq [\![\mathsf{i}?(e)]\!] = \{\mathsf{i!ans}\}$.

We define the algorithm in terms of the *five constructions* given below; each construction is accompanied by a proof guaranteeing semantic preservation, i.e., that the result of each translation is equivalent to its input. The construction sequence is as follows:

§1. Unguarded fixpoint variable removal: the formula is modified to ensure that the fixpoint variables in the formula are all guarded (Sect. 5.3.1).

$$\varphi \in \text{sHML}_{\mathbf{1}} ::= \ \mathsf{tt} \mid \mathsf{ff} \mid \mathsf{max}\, X.\varphi \mid \bigwedge_{i\in I}\varphi_i \mid [\eta]\psi \quad (\textit{where } \psi ::= X \mid \varphi)$$

Fig. 5 The sHML$_{\mathbf{1}}$ syntax

$$\langle\!\langle \varphi \rangle\!\rangle_1 \stackrel{\text{def}}{=} \begin{cases} \mathsf{max}\, X.(\psi \wedge \bigwedge f(\varphi')\backslash\{X\}) & \textit{if } \varphi = \mathsf{max}\, X.\varphi' \textit{ and } \langle\!\langle \varphi' \rangle\!\rangle_1 = \psi \wedge \bigwedge f(\varphi') \\ \psi_1 \wedge \psi_2 \wedge \bigwedge f(\psi_1) \wedge \bigwedge f(\psi_2) & \textit{if } \varphi = \psi_1 \wedge \psi_2 \textit{ and } \langle\!\langle \psi_1 \rangle\!\rangle_1 = \psi_1 \wedge \bigwedge f(\psi_1) \\ & \textit{and } \langle\!\langle \psi_2 \rangle\!\rangle_1 = \psi_2 \wedge \bigwedge f(\psi_2) \\ [\eta]\langle\!\langle \varphi' \rangle\!\rangle_1 & \textit{if } \varphi = [\eta]\varphi' \\ X \wedge \mathsf{tt} & \textit{if } \varphi = X \\ \varphi & \textit{otherwise} \end{cases}$$

$$\text{where } f(\varphi) \stackrel{\text{def}}{=} \bigwedge_{X_i \in S} X_i \qquad \text{and } S = \{X \mid \text{if } X \text{ is free and unguarded in } \varphi\}.$$

Fig. 6 The unguarded fixpoint removal algorithm

§2. Equation construction: the formula is reformulated into a system of equations to enable easier manipulation in later stages (Sect. 5.3.2).
§3. Powerset construction: the resultant system of equations is restructured into an equivalent system that defines syntactically disjoint conjunctions (Sect. 5.3.3).
§4. Formula reconstruction: the system of equations is converted back into an sHML formula with disjoint conjunctions which may define redundant fixpoints (Sect. 5.3.4).
§5. Redundant fixpoint removal: finally, fixpoint variable declarations, $\mathsf{max}\, X.\varphi$, are removed whenever variable X is not used in φ (i.e., $X \notin \mathbf{fv}(\varphi)$) – this produces the required sHML$_{\mathbf{nf}}$ formula (Sect. 5.3.5).

For conciseness, we use notation η to refer to an arbitrary symbolic action p, c, $p[d]$ for an arbitrary pattern that *binds* variable d, and $c[d]$ for a condition whose evaluation depends on the value of variable d.

5.3.1 Unguarded fixpoint variable removal

We start the normalisation procedure by converting the sHML formula into a semantically equivalent sHML$_{\mathbf{1}}$ formula, i.e., an sHML formula in which every fixed point variable is *guarded* by a modal necessity as specified in Fig. 5.

Example 12 Formula $\mathsf{max}\, X.([\alpha]X \wedge X)$ can be rewritten as $\mathsf{max}\, X.([\alpha]X)$, and $\mathsf{max}\, X.(\mathsf{max}\, Y.([\alpha]Y \wedge X))$ into $\mathsf{max}\, X.(\mathsf{max}\, Y.([\alpha]Y))$. □

Function $\langle\!\langle - \rangle\!\rangle_1$: sHML $\rightarrow$ sHML$_{\mathbf{1}}$ in Fig. 6 compositionally analyses a formula and removes every unguarded fixpoint variable. Specifically, when analysing a fixpoint, $\mathsf{max}\, X.\varphi'$, it is recursively applied to the fixpoint body φ' such that $\langle\!\langle \varphi' \rangle\!\rangle_1$ returns $\psi \wedge \bigwedge f(\varphi')$ where $f(\varphi')$ contains all free and unguarded fixpoint variables defined in φ'. If $X \in f(\varphi')$ it means that X is unguarded in φ' and is thus removed from the resulting formula, i.e., $\mathsf{max}\, X.(\psi \wedge \bigwedge f(\varphi')\backslash\{X\})$. Conjunct formulas, $\psi_1 \wedge \psi_2$, are analysed separately and the free and unguarded variables of each branch are grouped at the top level. The remaining cases are unremarkable.

Example 13 Consider $\varphi_5 \stackrel{\text{def}}{=} \mathsf{max}\, X_0.([\mathsf{i?req}]([\mathsf{i!ans}][\mathsf{i!ans}]\mathsf{ff}) \wedge ([\mathsf{i!ans}]X_0) \wedge \underline{X_0})$, a reformulated version of φ_0 from Example 2. By applying $\langle\!\langle - \rangle\!\rangle_1$ to φ_5,
we obtain $\psi_3 \stackrel{\text{def}}{=} \mathsf{max}\, X_0.[\mathsf{i?req}]([\mathsf{i!ans}][\mathsf{i!ans}]\mathsf{ff}) \wedge ([\mathsf{i!ans}]X_0)$ where $\psi_3 \in$ sHML$_{\mathbf{1}}$ as it does not define any unguarded fixpoint variables. □

$$\varphi \in \text{sHML}_{\mathbf{eq}} ::= \mathsf{tt} \quad | \quad \mathsf{ff} \quad | \quad \textstyle\bigwedge_{i\in I} [\eta]X_i$$

Fig. 7 A syntactic restriction for equated formulas

Lemma 2 *For every* sHML *formula φ we have that $[\![\langle\!\langle\varphi\rangle\!\rangle_1]\!] = [\![\varphi]\!]$.*

Proof The proof follows from *Lemma 8* in [67]. Although *Lemma 8* is proven *wrt.* a version of sHML that only allows for defining concrete actions, the proof of this lemma still applies to our setting, since $\langle\!\langle - \rangle\!\rangle_1$ pays no regard to the type of actions described in the modal necessities. Adapting the proof for our setting thus only requires minor syntactic changes.

□

5.3.2 Equation construction

This construction produces a system of equations from a given sHML formula. *Systems of equations* (*SoEs*) provide an alternative way for defining recursive sHML formulas without resorting to maximal fixpoints.

Definition 11 (*System of equations*) A system of equations is defined as a triple $(Eq,\ X,\ \mathcal{Y})$, where X represents the *principal logical variable* which identifies the starting equation, $\mathcal{Y}$ is a finite set of *free logical variables*, and *Eq* is an *n-tuple of equations*, i.e., $\{X_1 = \psi_1, X_2 = \psi_2, \ldots, X_n = \varphi_n\}$, where for $1 \le i < j \le n$, X_i is different from X_j, and each φ_i is a $\text{sHML}_{\mathbf{eq}}$ expression as defined in Fig. 7. □

Two systems of equations are *equivalent* (written as $\equiv$) when their largest solution assigns the same meaning to their principal variable. We abuse notation and use *Eq* as a map where $Eq(X_i) = \varphi_i$ when $X_i = \varphi_i \in Eq$. A maximal fixpoint $\mathsf{max}\, X.\varphi$ is represented in a *SoE* by the X-component of the greatest solution of the *SoE* over $(2^{\text{Sys}})^n$ (where n refers to the number of equations in the equation tuple). A *SoE* is closed when $\mathcal{Y}$ is empty.

Example 14 A recursive formula such as $\mathsf{max}\, X_0.[\text{i?3}]([\text{i!4}]X_0 \wedge [\text{i!5}]\mathsf{ff})$ can be represented as a system of four equations $(Eq,\ X_0,\ \mathcal{Y})$ where X_0 is the principal variable, $Eq \stackrel{\text{def}}{=} \{X_0{=}[\text{i?3}]X_1,\ X_1{=}[\text{i!4}]X_2 \wedge [\text{i!5}]X_3,\ X_2{=}[\text{i?3}]X_1,\ X_3{=}\mathsf{ff}\}$, where $X_1{=}X_0$ and $\mathcal{Y}{=}\emptyset$ as all the logical variables defined in the system are *bound*, i.e., equated to some $\text{sHML}_{\mathbf{eq}}$ formula. Notice how recursion is represented by referring to X_1 in the penultimate equation.
□

Function $\langle\!\langle - \rangle\!\rangle_2 : \text{sHML}_{\mathbf{1}} \to Eq \times \text{Var} \times \mathcal{P}(\text{Var})$, in Fig. 8, compositionally inspects a given closed $\text{sHML}_{\mathbf{1}}$ formula φ and translates it into an equivalent *SoE*. *Truth*, tt, and *falsehood*, ff, are, respectively, translated into equations $X_j = \mathsf{tt}$ and $X_j = \mathsf{ff}$, with j being a fresh index and X_j being the principal variable of the resultant *SoE*. Logical variables Y are initially translated into a *SoE* defining: equation $X_j = Y$, X_j as the principal variable, and $\mathcal{Y} = \{Y\}$, signifying that Y is free. Although equation $X_j = Y$ does not comply to $\text{sHML}_{\mathbf{eq}}$ (and is thus invalid), since we assume closed formulas, this equation gets fixed when $\langle\!\langle - \rangle\!\rangle_2$ recurses back to the binding fixpoint.

Fixpoints, $\mathsf{max}\, Y.\varphi$, are converted into equation $Y = Eq(X_i)$, where X_i is the principal variable of the *SoE* obtained from the recursive application on the continuation φ', i.e., $\langle\!\langle \varphi' \rangle\!\rangle_2 = (Eq,\ X_i,\ \mathcal{Y})$. This is added to the equation set *Eq*. Variable Y is then removed from $\mathcal{Y}$, denoting that although Y is free in φ', this is no longer the case in $\varphi = \mathsf{max}\, Y.\varphi'$.

$$\langle\!\langle \varphi \rangle\!\rangle_2 \stackrel{\text{def}}{=} \begin{cases} (\{X_j = \mathsf{tt}\},\ X_j,\ \emptyset) & \text{if } \varphi = \mathsf{tt} \\ (\{X_j = \mathsf{ff}\},\ X_j,\ \emptyset) & \text{if } \varphi = \mathsf{ff} \\ (\{X_j = Y\},\ X_j,\ \{Y\}) & \text{if } \varphi = Y \\ \left(\bigcup_{i\in I} Eq_i \cup \{X_j = \bigwedge_{i\in I} Eq_i(X_i)\},\ X_j,\ \bigcup_{i\in I} \mathcal{Y}_i\right) & \text{if } \varphi = \bigwedge_{i\in I}\varphi_i \ \textit{and} \\ & \forall i \in I \cdot \langle\!\langle \varphi_i \rangle\!\rangle_2 = (Eq_i,\ X_i,\ \mathcal{Y}_i) \\ (Eq \cup \{X_j = [\eta]X_k\},\ X_j,\ \mathcal{Y}) & \text{if } \varphi = [\eta]\psi \ \textit{and} \\ & \langle\!\langle \psi \rangle\!\rangle_2 = (Eq,\ X_k,\ \mathcal{Y}) \\ \left(\{Y = Eq(X_i)\} \cup \left\{\begin{matrix} X_j = Eq(X_i) & \text{if } X_j = Y \in Eq \\ X_k = \varphi_k & \text{if } X_k = \varphi_k \in Eq \end{matrix}\right\},\ Y,\ \mathcal{Y}\backslash\{Y\}\right) & \text{if } \varphi = \max Y.\varphi' \ \textit{and} \\ & \langle\!\langle \varphi' \rangle\!\rangle_2 = (Eq,\ X_i,\ \mathcal{Y}) \end{cases}$$

where variable X_j is *fresh in all cases.*

Fig. 8 The conversion algorithm from a sHML$_\mathbf{1}$ formula to a *SoE*

Equations of the sort $X_j = Y$ in *Eq* are reformulated into valid equations as $X_j = Eq(X_i)$ where X_i points to the same equation as Y; this ensures that every logical variable is *guarded* by a modal necessity.

Modal necessities, $[\eta]\varphi$, are reformed as a *SoE* defining equation set $\{X_j = [\eta]X_k\} \cup Eq$, where X_k and *Eq* are the principal variable and equation set obtained from $\langle\!\langle \varphi \rangle\!\rangle_2$, respectively. Conjunctions, $\bigwedge_{i\in I}\varphi_i$, are converted into a *SoE* containing the equations obtained from analysing every conjunct formula φ_i, i.e., Eq_i for every $i \in I$, along with equation $X_j = \bigwedge_{i\in I} Eq_i(X_i)$, where X_i is the principal variable of every *SoE* obtained from $\langle\!\langle \varphi_i \rangle\!\rangle_2$ (for every $i \in I$). Note that since the introduced variables are chosen to be *fresh*, the equation sets Eq_i are defined over pairwise disjoint sets of bound variables.

Example 15 Recall $\psi_3 \stackrel{\text{def}}{=} \max X_0.[\mathsf{i?req}]([\mathsf{i!ans}][\mathsf{i!ans}]\mathsf{ff}) \wedge ([\mathsf{i!ans}]X_0)$ from Example 13. From $\langle\!\langle \psi_3 \rangle\!\rangle_2$ we obtain $(Eq,\ X_0,\ \emptyset)$ where

$$Eq = \left\{ \begin{matrix} X_0 = Eq(X_1) = [\mathsf{i?req}]X_2, & X_1 = [\mathsf{i?req}]X_2, & X_3 = [\mathsf{i!ans}]X_5, \\ X_2 = Eq(X_3) \wedge Eq(X_4) = [\mathsf{i!ans}]X_5 \wedge [\mathsf{i!ans}]X_6, & X_4 = [\mathsf{i!ans}]X_6, & \\ X_5 = [\mathsf{i!ans}]X_7, \ \ X_6 = X_0 = Eq(X_1) = [\mathsf{i?req}]X_2, \ \ X_7 = \mathsf{ff} & & \end{matrix} \right\}.$$

The greyed formulas are not reachable from the principal equation and are thus redundant. We ignore them in forthcoming examples. □

Lemma 3 *For every closed* sHML$_\mathit{1}$ *formula* φ*, the SoE obtained from* $\langle\!\langle \varphi \rangle\!\rangle_2$ *has the same meaning as* φ*.*

Proof The proof follows from *Lemma 10* given in [67]. Although this lemma is proven in relation to formulas that define concrete actions, it still applies for formulas defining symbolic actions, since the construction is independent of the type of action described in the modal necessities.

5.3.3 Powerset construction

In this step we convert a *SoE* into an equivalent *SoE* in which every equated formula meets the restrictions of sHML$^\#_\mathbf{eq}$ in Fig. 9. Conjunctions in the equated formulas are now required to be guarded by *disjoint* modal necessities. Figure 10 presents $\langle\!\langle - \rangle\!\rangle_3 : (Eq \times \text{VAR} \times \mathcal{P}(\text{VAR})) \rightarrow (Eq_\# \times \text{VAR} \times \mathcal{P}(\text{VAR}))$ where for every logical variable X, $Eq_\#(X) \in$ sHML$^\#_\mathbf{eq}$. This function generates a new *SoE* containing the powerset combinations of the equations from the original *SoE*. Intuitively, it takes two or more equations and combines the equated formulas

$$\varphi \in \mathrm{sHML}^{\#}_{\mathbf{eq}} ::= \mathsf{tt} \quad | \quad \mathsf{ff} \quad | \quad \bigwedge_{i\in I} [\eta]X_i \ \mathit{and}\ \#_{i\in I}\, \eta_i$$

Fig. 9 A disjointness requirement, $\#_{i\in I}\,\eta_i$, for equated formulas entailing that for every $i, j \in I$, $[\![\eta_i]\!] \cap [\![\eta_j]\!] = \emptyset$

$$\langle\!\langle (\mathit{Eq},\, X_i,\, \mathcal{Y}) \rangle\!\rangle_3 \overset{\text{def}}{=} \big(\mathit{Eq}_{\#},\, X_{\{i\}},\, \mathcal{Y}\big)$$

$$\begin{aligned}\mathit{Eq}_{\#} \overset{\text{def}}{=}\ & \{\ X_I = \mathsf{ff} \mid \ \mathit{if}\ I \subseteq I(\mathit{Eq})\ \mathit{and}\ \exists j{\in}I \cdot \mathit{Eq}(X_j) = \mathsf{ff}\ \} \\ & \cup \{\ X_I = \textstyle\bigwedge_{\eta\in G(I,\mathit{Eq})} [\eta]X_{CI(I,\mathit{Eq},\eta)} \ \big|\ \ \mathit{if}\ I \subseteq I(\mathit{Eq})\ \mathit{and}\ \nexists j{\in}I \cdot \mathit{Eq}(X_j) = \mathsf{ff}\ \}\end{aligned}$$

$$\begin{aligned}\text{where} \qquad I(\mathit{Eq}) &\overset{\text{def}}{=} \Big\{ \mathsf{i} \ \Big|\ (X_i = \varphi_i) \in \mathit{Eq} \Big\} \\ G(I, \mathit{Eq}) &\overset{\text{def}}{=} \textstyle\bigcup_{i\in I} \Big\{ \eta_j \ \Big|\ \mathit{if}\ \mathit{Eq}(X_i) \bigwedge_{j\in I'} [\eta_j]X_j \ (\text{for some } I') \Big\} \\ CI(I, \mathit{Eq}, \eta) &\overset{\text{def}}{=} \textstyle\bigcup_{i\in I} \Big\{ j \ \Big|\ \mathit{if}\ \mathit{Eq}(X_i) = [\eta]X_j {\wedge} \varphi \ (\text{for some } \varphi) \Big\}\end{aligned}$$

Fig. 10 The powerset construction for systems of equations

with a conjunction. This technique mimics the classic powerset construction for determinising automata in automata theory [74].

Specifically, $\langle\!\langle - \rangle\!\rangle_3$ creates a new equation set in which the index of each equation is $I \subseteq I(\mathit{Eq})$, i.e., an element of the powerset of all indices defined by the equation set *Eq* of the given *SoE*. The formula φ_I of a new equation $X_I = \varphi_I$ is constructed by analysing every equation $X_i = \varphi_i$ where $i \in I$. If there exists at least one index $j \in I$ so that $X_j = \mathsf{ff}$, then X_I is immediately set to ff. This is done since if ff is used to reconstruct a conjunction along with the other formulas φ_i (where $i \neq j$), the resultant conjunction would still be semantically equivalent to ff. Otherwise, X_I is reconstructed as the merged conjunction $\bigwedge_{\eta\in G(I,\mathit{Eq})} [\eta]X_{CI(I,\mathit{Eq},\eta)}$ which is created using functions G and CI in Fig. 9.

The former function is used to retrieve the set of all the *syntactically unique SAs*, η, defined by the equated formulas φ_i for each $i{\in}I$. The latter returns the set of indices containing the index j of every variable X_j that is guarded by a modal necessity defining *SA* η in φ_i. Hence, every branch in the resultant conjunction $\bigwedge_{\eta\in G(I,\mathit{Eq})} [\eta]X_{CI(I,\mathit{Eq},\eta)}$ is guarded by a *syntactically disjoint* modal necessity.

Remark 1 Function $\langle\!\langle - \rangle\!\rangle_3$ makes a crucial assumption that actions that vary syntactically are also semantically disjoint, and so if $\eta_1 \neq \eta_2$ then no action can match both *SAs*. For now, this assumption holds since we are only considering *singleton SAs*. In Sect. 5.4 we will see how additional transformations are required to ensure this for non-singleton *SAs*. □

Example 16 Recall the *SoE* obtained in Example 15, i.e., $(\mathit{Eq},\, X_0,\, \emptyset)$ where

$$\mathit{Eq} = \left\{ \begin{array}{l} X_0 = [\mathsf{i?req}]X_2,\ \ X_2 = [\mathsf{i!ans}]X_5 {\wedge} [\mathsf{i!ans}]X_6, \\ X_5 = [\mathsf{i!ans}]X_7,\ \ X_6 = [\mathsf{i?req}]X_2,\ \ X_7 = \mathsf{ff} \end{array} \right\}.$$

When $\langle\!\langle - \rangle\!\rangle_3$ is applied, it generates every possible combination and merges the modal necessities where necessary. From $\langle\!\langle (\mathit{Eq},\, X_0,\, \emptyset) \rangle\!\rangle_3$ we therefore obtain $\big(\mathit{Eq}_{\#},\, X_{\{0\}},\, \emptyset\big)$ where $\mathit{Eq}_{\#} = \big\{X_{\{0\}} = [\mathsf{i?req}]X_{\{2\}}, X_{\{2\}} = [\mathsf{i!ans}]X_{\{5,6\}}\big\} \cup \mathit{Eq}'_{\#}$. Notice how continuations X_5 and X_6 in $X_2 = [\mathsf{i!ans}]X_5 {\wedge} [\mathsf{i!ans}]X_6$ were combined into a single continuation in $X_{\{2\}} = [\mathsf{i!ans}]X_{\{5,6\}}$. The algorithm constructs all the formula combinations including those for $X_{\{5,6\}}$ as per $\mathit{Eq}'_{\#}$:

$$\mathit{Eq}'_{\#} = \big\{ X_{\{5,6\}} = [\mathsf{i!ans}]X_{\{7\}} {\wedge} [\mathsf{i?req}]X_{\{2\}},\, X_{\{7\}} = \mathsf{ff},\, \ldots \big\}.$$

$$\varphi \in \text{sHML}_{\mathbf{2}} ::= \mathsf{tt} \mid \mathsf{ff} \mid \mathsf{max}\, X.\varphi \mid \bigwedge_{i\in I} [\eta_i]\psi_i \quad (\text{where } \#_{i\in I}\, \eta_i \text{ and } \psi ::= X \mid \varphi)$$

Fig. 11 The sHML$_{\mathbf{2}}$ syntax

$$\langle\!\langle (\, Eq\,,\, X_i,\, \mathcal{Y})\rangle\!\rangle_4 \stackrel{\text{def}}{=} \sigma_{\mathsf{shml}}(X_i, Eq)$$

$$\sigma_{\mathsf{shml}}(\varphi, Eq) \stackrel{\text{def}}{=} \begin{cases} \varphi & \text{if } \mathbf{fv}(\varphi){=}\emptyset \\ \sigma_{\mathsf{shml}}(\varphi\sigma, Eq) & \text{if } \mathbf{fv}(\varphi){=}S \text{ then } \sigma{=}\left\{ (\mathsf{max}\, X.\varphi)/X \;\middle|\; \begin{array}{l} (X{=}\varphi) \in Eq \\ \text{and } \; X{\in}S \end{array} \right\} \end{cases}$$

Fig. 12 Converting a *SoE* in conj. normal form into an sHML$_{\mathbf{2}}$ formula

We omit the redundant combinations that are *not reachable* from the new principal variable $X_{\{0\}}$, from the resultant equation set. □

Lemma 4 *For every SoE* $(Eq,\, X_0,\, \mathcal{Y})$*, if* $\langle\!\langle (Eq,\, X_0,\, \mathcal{Y})\rangle\!\rangle_3 = \big(Eq',\, X_{\{0\}},\, \mathcal{Y}\big)$ *then* $(Eq,\, X_0,\, \mathcal{Y}) \equiv \big(Eq',\, X_{\{0\}},\, \mathcal{Y}\big)$ *and for every* $(X_i = \varphi_i){\in}Eq'$*,* $\varphi_i{\in}\text{sHML}^{\#}_{\boldsymbol{eq}}$.

Proof In [67] the authors present a version of $\langle\!\langle - \rangle\!\rangle_3$ which processes equations that equate formulas which only specify concrete actions. By definition syntactically different concrete actions are also disjoint, which is not always the case with *SAs*. As for now we are assuming that our formulas can only include singleton *SAs*, semantic preservation is ensured by *Lemma 11* in [67]. In Sect. 5.4 we will present the necessary steps for ensuring that this criterion holds for every kind of *SA*.

5.3.4 Formula reconstruction

With this step we convert the *SoE* back to a formula that adheres to the restrictions imposed by sHML$_{\mathbf{2}}$ in Fig. 11. sHML$_{\mathbf{2}}$ requires conjunctions to be guarded by disjoint modal necessities, but allows for defining redundant fixpoint declarations.

Figure 12 presents $\langle\!\langle - \rangle\!\rangle_4 \colon (\, Eq_{\#}\,,\, \textsc{Var},\, \mathcal{P}(\textsc{Var})\,) \to \text{sHML}_{\mathbf{2}}$, which internally employs $\sigma_{\mathsf{shml}} \colon (\text{sHML}_{\mathbf{2}} \times Eq) \to \text{sHML}_{\mathbf{2}}$ to construct the corresponding sHML$_{\mathbf{2}}$ formula. Initially, σ_{shml} takes as input the principal variable X_{i} along with the equation set Eq. Since X_i is an open term, $\mathbf{fv}(X_i) = \{X_i\}$, the function searches for equation $X_i{=}\varphi_i$ in Eq and converts it into a substitution environment which substitutes variable X_i with $\mathsf{max}\, X_i.\varphi_i$, i.e., $\{\mathsf{max}\, X_i.\varphi_i/X_i\}$. This substitution is then applied to X_i and the function recurses with the substituted value, $\sigma_{\mathsf{shml}}(\mathsf{max}\, X_i.\varphi_i, Eq)$; recursion stops when the resultant formula φ becomes closed, $\mathbf{fv}(\varphi) = \emptyset$, in which case it is returned.

Example 17 Recall the *SoE* $\big(Eq_{\#}\,,\, X_{\{0\}},\, \emptyset\big)$ obtained in Example 16, where

$$Eq_{\#} = \left\{ \begin{array}{l} X_{\{0\}} = [\mathsf{i?req}]X_{\{2\}},\;\; X_{\{2\}} = [\mathsf{i!ans}]X_{\{5,6\}}, \\ X_{\{5,6\}} = [\mathsf{i!ans}]X_{\{7\}}{\wedge}[\mathsf{i?req}]X_{\{2\}},\;\; X_{\{7\}} = \mathsf{ff} \end{array} \right\}.$$

and so by applying $\langle\!\langle - \rangle\!\rangle_4$ we obtain $\psi_4 \in \text{sHML}_{\mathbf{2}}$ where

$$\begin{aligned} \psi_4 &= \sigma_{\mathsf{shml}}(X_{\{0\}}, Eq_{\#}) \\ &= \mathsf{max}\, X_{\{0\}}.\, \big([\mathsf{i?req}]\mathsf{max}\, X_{\{2\}}.\, \big([\mathsf{i!ans}]\mathsf{max}\, X_{\{5,6\}}.(\psi_4'{\wedge}\psi_4'')\big)\big) \end{aligned}$$

$$\textit{where} \quad \psi_4' = [\mathsf{i!ans}]\mathsf{max}\, X_{\{7\}}.\mathsf{ff} \quad \textit{and} \quad \psi_4'' = [\mathsf{i?req}]X_{\{2\}}.$$

□

Fig. 13 Converting sHML$_\mathbf{2}$ formulas into sHML$_\mathbf{nf}$

$$\langle\!\langle \varphi \rangle\!\rangle_5 \overset{\text{def}}{=} \begin{cases} \varphi & \text{if } \varphi \in \{\mathsf{ff}, \mathsf{tt}\} \\ \langle\!\langle \varphi' \rangle\!\rangle_5 & \text{if } \varphi = \mathsf{max}\, X.\varphi' \text{ and } X \notin \mathbf{fv}(\varphi') \\ \mathsf{max}\, X.\langle\!\langle \varphi' \rangle\!\rangle_5 & \text{if } \varphi = \mathsf{max}\, X.\varphi' \text{ and } X \in \mathbf{fv}(\varphi') \\ \bigwedge_{i\in I} [\eta_i]\langle\!\langle \varphi_i \rangle\!\rangle_5 & \text{if } \varphi = \bigwedge_{i\in I} [\eta_i]\varphi_i \end{cases}$$

Lemma 5 *For every SoE* $(Eq,\ X_{\{0\}},\ \mathcal{Y})$*, if* $\langle\!\langle (Eq,\ X_{\{0\}},\ \mathcal{Y}) \rangle\!\rangle_3 = \varphi$ *then* φ *conveys the same meaning as* $(Eq,\ X_{\{0\}},\ \mathcal{Y})$ *and that* $\varphi \in$ sHML$_\mathbf{2}$.

Proof Since construction $\langle\!\langle - \rangle\!\rangle_4$ is independent of the type of actions defined in the modal necessities of the given *SoE*, we refer to *Lemma 12* from [67] as proof that $\langle\!\langle - \rangle\!\rangle_4$ always produces a semantically equivalent formula $\varphi \in$sHML$_\mathbf{2}$.

5.3.5 Removing redundant fixpoints

The final construction produces a sHML$_\mathbf{nf}$ formula in which every logical variable X defined by a fixpoint $\mathsf{max}\, X.\varphi$ is free in the continuation formula φ (i.e., $X \in \mathbf{fv}(\varphi)$), meaning that X is used at least once in φ. We formalise this construction as function $\langle\!\langle - \rangle\!\rangle_5$: sHML$_\mathbf{2}$ $\to$ sHML$_\mathbf{nf}$ in Fig. 13. This function compositionally inspects a given formula φ and removes maximal fixpoint declarations whenever their variable is not free (and so never used) in φ.

Example 18 The redundant fixpoints in ψ_4 from Example 17 can be removed via function $\langle\!\langle - \rangle\!\rangle_5$, thus obtaining the following sHML$_\mathbf{nf}$ formula:

$$\psi_5 \overset{\text{def}}{=} [\mathsf{i?req}]\mathsf{max}\, X_{\{2\}}.[\mathsf{i!ans}]([\mathsf{i!ans}]\mathsf{ff} \wedge [\mathsf{i?req}]X_{\{2\}}).$$

Notice that the obtained formula ψ_5 is identical to φ_0 (modulo α-renaming) from Example 2, and are both definable via the sHML$_\mathbf{nf}$ syntax, and thus in *normal form.* □

Lemma 6 *For every formula* $\varphi \in$ sHML$_\mathbf{2}$, $[\![\langle\!\langle \varphi \rangle\!\rangle_5]\!] = [\![\varphi]\!]$.

Proof We prove that for every system s,

(a) $s \in [\![\langle\!\langle \varphi \rangle\!\rangle_5]\!]$ *implies* $s \in [\![\varphi]\!]$; and
(b) $s \in [\![\varphi]\!]$ *implies* $s \in [\![\langle\!\langle \varphi \rangle\!\rangle_5]\!]$.

The proofs for both of these cases are provided in Appendix A.1.

We have presented a sequence of constructions that transform sHML formulas defining singleton *SAs* into their normalised equivalent in sHML$_\mathbf{nf}$. We thus conclude that when we *only* consider *singleton SAs*, Theorem 5 holds as a result of Lemmas 2 and 6.

5.4 Reconstructing sHML into sHML$_\mathbf{nf}$ wrt. *any* symbolic action

Up until now we have only considered normalising sHML formulas defining singleton *SAs* as these events are easy to statically differentiate from each other which is a crucial requirement for merging branches in **§3.** However, modal necessities in general can also describe non-singleton *SAs* for which syntactic difference does not necessarily reflect disjointness. For instance, although $(d)!(e), e = 5$ and $(d)!(e), d = \mathsf{i}$ differ syntactically, they define *intersecting sets* of actions, $[\![(d)!(e), e = 5]\!] \cap [\![(d)!(e), d = \mathsf{i}]\!] = \{\mathsf{i!5}\}$, meaning that both can match the same system action i!5.

As shown in Example 19, normalising a non-singleton symbolic formula using the algorithm in Sect. 5.3, may sometimes fail to produce a normalised equivalent formula.

Example 19 Consider φ_6 a variant of φ_4 from Example 9.

$$\varphi_6 \overset{\text{def}}{=} \mathsf{max}\, X_0. \left([(d^1)?\mathsf{req}, \mathsf{true}]\left(\begin{array}{l}[(d^2)!\mathsf{ans}, d^2{\neq}\mathsf{h}{=}d^1][(d^4)!\mathsf{ans}, d^4{=}d^2]\mathsf{ff} \wedge \\ [(d^3)!\mathsf{ans}, d^3{\neq}\mathsf{j}{=}d^1]X_0\end{array}\right)\right).$$

By applying **§1.** and **§2.**, we construct $(Eq_4\,,\ X_0,\ \emptyset)$ where

$$Eq_4 = \left\{\begin{array}{c} X_0 = [(d^1)?\mathsf{req}, \mathsf{true}]X_1, \\ X_1 = [(d^2)!\mathsf{ans}, d^2{\neq}\mathsf{h}{=}d^1]X_2 \wedge [(d^3)!\mathsf{ans}, d^3{\neq}\mathsf{j}{=}d^1]X_3, \quad \ldots \end{array}\right\}.$$

However, when we apply **§3.** the algorithm fails to combine symbolic actions $(d^2)!\mathsf{ans}, d^2{\neq}\mathsf{h}{=}d^1$ and $(d^3)!\mathsf{ans}, d^3{\neq}\mathsf{j}{=}d^1$ as despite not being disjoint, they still differ syntactically, and so the equations defining these actions remain unmerged. We thus end up with $\left(Eq^4_{\#}\,,\ X_{\{0\}},\ \emptyset\right)$ where

$$Eq^4_{\#} = \left\{\begin{array}{c} X_{\{0\}} = [(d^1)?\mathsf{req}, \mathsf{true}]X_{\{1\}}, \\ X_{\{1\}} = [(d^2)!\mathsf{ans}, d^2{\neq}\mathsf{h}{=}d^1]X_{\{2\}} \wedge [(d^3)!\mathsf{ans}, d^3{\neq}\mathsf{j}{=}d^1]X_{\{3\}}, \quad \ldots \end{array}\right\}.$$

This error propagates through to steps **§4.** and **§5.** which produce a formula that despite being semantically equivalent to the original formula φ_5, it is still not in normal form due to its non-disjoint conjunctions. The current algorithm thus fails in the general case. □

When dealing with non-singleton *SAs*, we must introduce additional constructions to ensure that **§3.** correctly merges the conjunctions within a formula.

Example 20 To give some intuition of the necessary steps, consider again actions $(d^1)?(e^1), e^1 = 5$ and $(d^2)?(e^2), d^2 = \mathsf{i}$. Despite being syntactically different, these *SAs* are not disjoint as both can match i?5. The information they convey can, however, be encoded into 4 *SAs* (amounting to 3 disjoint ones) as follows:
– $(d^1)?(e^1), e^1{=}5$ becomes $(d)?(e), e{=}5 \wedge d{=}\mathsf{i}$ and $(d)?(e), e{=}5 \wedge d{\neq}\mathsf{i}$, while
– $(d^2)?(e^2), d^2{=}\mathsf{i}$ becomes $(d)?(e), e{=}5 \wedge d{=}\mathsf{i}$ and $(d)?(e), e{\neq}5 \wedge d{=}\mathsf{i}$
where d and e are fresh variables. Since these newly encoded *SAs* differ syntactically and are also disjoint, they can be distinguished via a simple syntactic check. For instance, $(d)?(e), e{=}5 \wedge d{=}\mathsf{i}$ and $(d)?(e), e{\neq}5 \wedge d{=}\mathsf{i}$ are not only syntactically different, but their contradicting conditions, $e{=}5$ and $e{\neq}5$, also guarantee their disjointness. □

5.4.1 Additional steps for normalising necessities defining symbolic actions

We formally define two additional constructions that must be applied between steps **§2.** and **§3.** They convert conjunctions that are guarded by necessities defining non-disjoint *SAs*, into equivalent conjunctions guarded by *syntactically disjoint necessities*, i.e., necessities describing *SAs* that are syntactically (hence semantically) disjoint. The additional steps include:

§i.step:alpha-equiv-cons **Conversion to uniform** *SAs*: we inspect modal necessities defined at the *same modal depth* within a conjunction and substitute their data variables with the *same* fresh variable whenever they define pattern equivalent *SAs* (Sect. 5.4.2).
§i.step:truth-combos-cons **Condition reformulation of conjunct** *SAs*: once uniformed, the conjunctions are recomposed to define branches that are prefixed by modal necessities specifying syntactically disjoint *SAs* (Sect. 5.4.3).

Traversal Functions.

$$\mathsf{traverse}(Eq, I, \lambda, \delta) \overset{\text{def}}{=} \begin{cases} \mathsf{traverse}(Eq', I', \lambda, \delta') & \text{if } Eq{\neq}\emptyset \text{ and } I{\neq}\emptyset \text{ then } \delta'{=}\lambda(Eq, I, \delta) \\ & \text{and } Eq'{=}Eq\backslash Eq_{//I} \\ & \text{and } I'{=}\bigcup_{j\in I} \mathsf{child}(Eq, j) \\ \delta & \text{otherwise} \end{cases}$$

$$\mathsf{child}(Eq, \mathsf{i}) \overset{\text{def}}{=} \left\{ j \;\middle|\; Eq(X_i){=}\bigwedge_{j\in I}[\eta_j]X_j{\wedge}\varphi \text{ and } j{\neq}i \text{ and } X_j{\in}\mathbf{dom}(Eq) \right\}$$

$$Eq_{//I} \overset{\text{def}}{=} \{\ X_i{=}\varphi_i \mid (X_i{=}\varphi_i) \in Eq \text{ and } i{\in}I\}$$

Fig. 14 The breadth first traversal algorithm

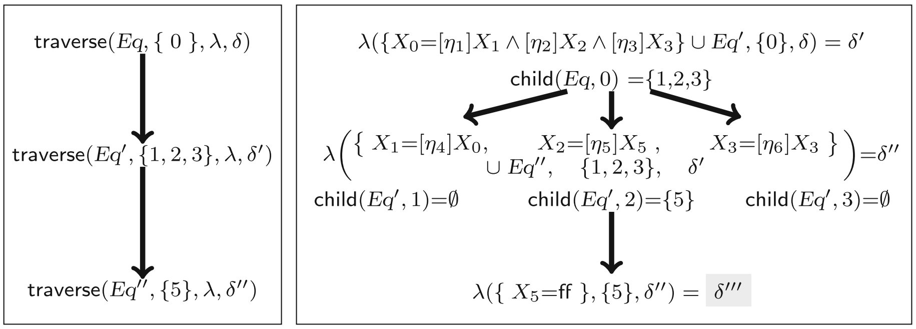
traverse(Eq, { 0 }, λ, δ)
traverse(Eq', {1, 2, 3}, λ, δ')
traverse(Eq'', {5}, λ, δ'')
$\lambda(\{X_0{=}[\eta_1]X_1 \wedge [\eta_2]X_2 \wedge [\eta_3]X_3\} \cup Eq', \{0\}, \delta) = \delta'$
child(Eq, 0) ={1,2,3}
$\lambda\Big(\{\ X_1{=}[\eta_4]X_0,\quad X_2{=}[\eta_5]X_5\ ,\quad X_3{=}[\eta_6]X_3\ \} \cup Eq'',\ \{1,2,3\},\ \delta'\Big){=}\delta''$
child(Eq', 1)=$\emptyset$
child(Eq', 2)={5}
child(Eq', 3)=$\emptyset$
$\lambda(\{\ X_5{=}\mathsf{ff}\ \}, \{5\}, \delta'') = \delta'''$

Fig. 15 A pictorial view of an example equation set traversal

Example 21 Recall $(d^1)?(e^1)$, $e^1 = 5$ and $(d^2)?(e^2)$, $d^2 = \mathsf{i}$ from Example 20. Construction **§i.** uniforms the *SAs* by assigning the same fresh variables to both *SAs*, and so they become $(d)?(e)$, $e = 5$ and $(d)?(e)$, $d = \mathsf{i}$. Construction **§i.** then reformulates the conditions of the resulting *SAs* to obtain $(d)?(e)$, $e{=}5 \wedge d{=}\mathsf{i}$, $(d)?(e)$, $e{\neq}5 \wedge d{=}\mathsf{i}$ and $(d)?(e)$, $e{=}5 \wedge d{\neq}\mathsf{i}$ which are disjoint. □

Internally, constructions **§i.** and **§i.** both use the traverse function defined in Fig. 14 to process the given set of equations in a *tree-like* manner. traverse : $(Eq \times \mathcal{P}(\textsc{Index}) \times \textsc{Fun} \times \textsc{Acc}) \rightarrow \textsc{Acc}$ is a *higher order function* which takes as input: a set of equations *Eq*, a set of indices I, an arbitrary projection function λ, and an accumulator argument δ.

It conducts a *breadth first* traversal on an equation set, starting from the equation of the principal variable as the root of the tree traversal. For instance, in Fig. 15 equation $X_0{=}[\eta_1]X_1 \wedge [\eta_2]X_2 \wedge [\eta_3]X_3$ is the root of the traversal since X_0 is the principal variable of $(Eq,\ X_0,\ \mathcal{Y})$.

The children of the root are calculated via the child:$(Eq \times \textsc{Index}) {\rightarrow} \mathcal{P}(\textsc{Index})$ function. It takes as input an equation set *Eq* along with the index i of the parent equation, e.g., index 0 for equation $X_0{=}[\eta_1]X_1 \wedge [\eta_2]X_2 \wedge [\eta_3]X_3$. It then scans the equated formula and returns the set containing the indices of every branch, defined in the equated formula, which is prefixed by a modal necessity. For example in Fig. 15, the children of $X_0{=}[\eta_1]X_1 \wedge [\eta_2]X_2 \wedge [\eta_3]X_3$ are $\{1, 2, 3\}$, and so branches $[\eta_1]X_1$, $[\eta_2]X_2$ and $[\eta_3]X_3$ are *siblings* as they are defined at the *same modal depth* of the conjunction.

Cycles in the traversal are avoided since the child function is always executed *wrt.* a restricted set of equations, i.e., one which *does not* include the parent equation. Cycles to the (immediate) parent are also avoided by removing the parent's index from the returned set of child indices.

Example 22 While analysing equation $X_1{=}[\eta_4]X_0$ in Fig. 15, traverse is evaluated *wrt.* Eq' which does not include the parent equation, i.e., since $Eq'{=}Eq\backslash Eq_{//\{0\}}$ where $Eq_{//\{0\}} = \{X_0{=}[\eta_1]X_1 \wedge [\eta_2]X_2 \wedge [\eta_3]X_3\}$. In this way, when computing the children of X_1 (via $\mathsf{child}(Eq', 1)$) index 0 is not added to the resultant set of child indices, since $X_0 \notin \mathbf{dom}(Eq')$; this avoids cycling back to some (grand) parent equation. Moreover, when evaluating $\mathsf{child}(Eq', 3)$ to retrieve the child indices of equation $X_3{=}[\eta_6]X_3$, index 3 is removed thus avoiding the creation of a loop in the traversal. □

While traversing the equation set, the traverse function can apply an arbitrary projection function λ. As mentioned above, despite being an arbitrary function, λ must adhere to a specific type, namely, $\lambda : (Eq \times \mathcal{P}(\textsc{Index}) \times \textsc{Acc}) \rightarrow \textsc{Acc}$. It must take three inputs including: the current equation set Eq, a set of indices I and an accumulator value δ, and must return an updated accumulator δ'.

Upon termination, the traversal returns the latest version of the accumulator. The traversal terminates when either all the equations in Eq have been processed such that the traverse function is applied *wrt.* $Eq{=}\emptyset$, or whenever no further children can be visited, i.e., for every branch i, $\mathsf{child}(Eq, \mathsf{i}){=}\emptyset$. The latter is an optimisation which omits the redundant processing of equations that are not reachable from the principal equation.

With this mechanism in place, we can now define steps **§i.** and **§i.** in Sects. 5.4.2 and 5.4.3.

5.4.2 Uniformity of symbolic actions

Intuitively, this part of the normalisation algorithm *renames the data variables* of *pattern equivalent* sibling modal necessities, to the *same variable names*. This produces a *uniform system of equations*.

Definition 12 (*Uniform system of equations*) An equation is *uniform* when every *pattern equivalent SA* defined by *sibling* necessities within a conjunction, defines the exact *same* data variable names. A system of equations is uniform when all of its equations are uniform. □

Example 23 The *SAs* in $X_0 = [(d^1)?(d^2), c_1[d^1, d^2]]X_1 {\wedge} [(e^1)?(e^2), c_2[e^1, e^2]]X_2$ are both pattern equivalent, yet *not uniform* as they *do not* define the same variable names. Uniformity can be attained by renaming d^1 and e^1 to the same f^1 and similarly d^2 and e^2 to a fresh variable f^2, so to obtain $X_0 = [(f^1)?(f^2), c_1[f^1, f^2]]X_1 {\wedge} [(f^1)?(f^2), c_2[f^1, f^2]]X_2$. □

Figure 16 presents $\langle\!\langle - \rangle\!\rangle_{(\mathsf{i})} : (Eq,\ \textsc{Var},\ \mathcal{P}(\textsc{Var})) \rightarrow \big(Eq^{\mathsf{uni}},\ \textsc{Var},\ \mathcal{P}(\textsc{Var})\big)$. This internally uses the uni function to create the required uniform set of equations Eq^{uni} from a given equation set. Specifically, uni reconstructs the equation set by performing a linear scan during which it converts equations of the form $X_i{=}\bigwedge_{j\in I}[\eta_j]X_j{\wedge}\varphi$ to $X_i{=}\bigwedge_{j\in I}[\eta_j\zeta(j)]X_j{\wedge}\varphi$ where $\zeta : \textsc{Index}{\rightarrow}\sigma$ is a map that provides a substitution environment σ for a given index j. For the reconstruction to be correct, the ζ must be *well-formed*.

Definition 13 (*A well-formed ζ map*) We say that ζ is a *well-formed map* for an equation set Eq, whenever it provides a set of mappings which allow for

(i) uniformly renaming the data variables of pattern equivalent sibling necessities, defined in Eq, by setting them to the *same* set of fresh variables, and for

$$\langle\!\langle (\, Eq \,,\, X_0 ,\; \mathcal{Y}) \rangle\!\rangle_{(\mathsf{i})} \overset{\text{def}}{=} (\,\mathsf{uni}(Eq, \zeta)\,,\, X_0,\; \mathcal{Y}) \\ \text{where } \zeta{=}\mathsf{traverse}(Eq, \{0\}, \mathsf{partition}, \emptyset)$$

$$\mathsf{uni}(Eq, \zeta) \overset{\text{def}}{=} \left\{ X_i{=}\bigwedge_{j\in I} [\eta_j \zeta(j)] X_j {\wedge} \varphi \;\middle|\; X_i{=}\bigwedge_{j\in I} [\eta_j] X_j {\wedge} \varphi \in Eq \right\}$$

$$\mathsf{partition}(Eq, I, \zeta) \overset{\text{def}}{=} \left\{ \begin{array}{c} j \mapsto \zeta(i) \,\dot{\cup}\, \{f^n/d^n\} \\ \\ k \mapsto \zeta(l) \,\dot{\cup}\, \{f^n/e^n\} \end{array} \;\middle|\; \begin{array}{l} \forall i, l \in I \cdot \textit{if } Eq(i){=}\bigwedge_{j\in I} [\{p_j[d^n], c_j\}] X_j {\wedge} \varphi' \\ \textit{and } Eq(l){=}\bigwedge_{k\in I''} [\{p_k[e^n], c_k\}] X_k {\wedge} \varphi'' \textit{ and} \\ i \neq l \textit{ and } [\![\{p_j[d^n], \mathsf{true}\}]\!] = [\![\{p_k[e^n], \mathsf{true}\}]\!] \\ \textit{(pattern equiv.) then we assign the } \text{same} \\ \textit{fresh variables } f^1, f^2. \end{array} \right\} \cup \zeta$$

where $\sigma \,\dot{\cup}\, \{f/e\} = \sigma \cup \{f/e\}$ *iff* $e \notin \mathbf{dom}(\sigma)$.

Fig. 16 The uniformity algorithm for symbolic actions

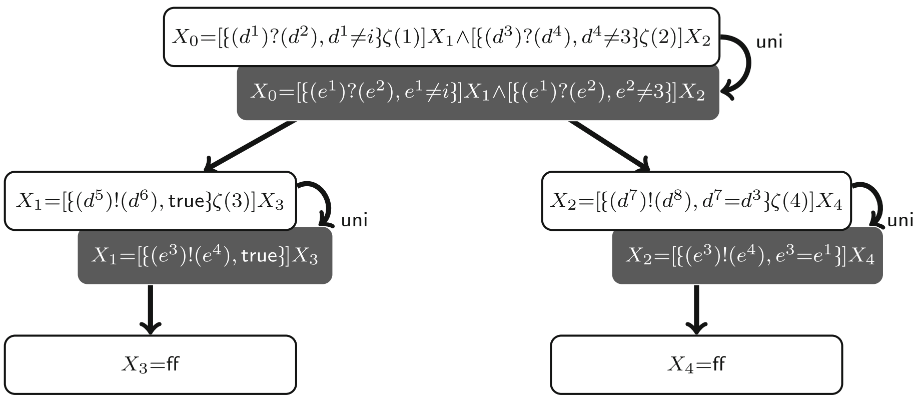
$X_0{=}[\{(d^1)?(d^2), d^1{\neq}i\}\zeta(1)]X_1{\wedge}[\{(d^3)?(d^4), d^4{\neq}3\}\zeta(2)]X_2$
uni
$X_0{=}[\{(e^1)?(e^2), e^1{\neq}i\}]X_1{\wedge}[\{(e^1)?(e^2), e^2{\neq}3\}]X_2$
$X_1{=}[\{(d^5)!(d^6), \mathsf{true}\}\zeta(3)]X_3$
uni
$X_1{=}[\{(e^3)!(e^4), \mathsf{true}\}]X_3$
$X_2{=}[\{(d^7)!(d^8), d^7{=}d^3\}\zeta(4)]X_4$
uni
$X_2{=}[\{(e^3)!(e^4), e^3{=}e^1\}]X_4$
$X_3{=}\mathsf{ff}$
$X_4{=}\mathsf{ff}$

Fig. 17 A Tree representation of the uni traversal performed on *Eq*

(ii) renaming any data variable reference that is bound to a renamed parent modal necessity defined in *Eq*.

We assume that by default $\zeta(i) = \emptyset$ when i is the index of the root equation. □

Example 24 Consider the following system of equations $(\,Eq,\; X_0,\; \emptyset\,)$ where

$$Eq = \left\{ \begin{array}{c} X_0{=}[(d^1)?(d^2), d^1{\neq}i]X_1{\wedge}[(d^3)?(d^4), d^4{\neq}3]X_2, \;\; X_3{=}\mathsf{ff}, \\ X_1{=}[(d^5)!(d^6), \mathsf{true}]X_3, \;\; X_2{=}[(d^7)!(d^8), d^7{=}d^3]X_4, \;\; X_4{=}\mathsf{ff} \end{array} \right\}.$$

For convenience, we also represent these equations as a tree starting from the principal equation $X_0{=}[(d^1)?(d^2), d^1{\neq}i]X_1{\wedge}[(d^3)?(d^4), d^4{\neq}3]X_2$ as the root of the tree. We also assume the knowledge of a *well-formed* ζ map:

$$\zeta = \left\{ \begin{array}{c} 0 \mapsto \{\emptyset\}, \quad 1 \mapsto \zeta(0) \,\dot{\cup}\, \{d^1/e^1, d^2/e^2\}, \quad 2 \mapsto \zeta(0) \,\dot{\cup}\, \{d^3/e^1, d^4/e^2\}, \\ 3 \mapsto \zeta(1) \,\dot{\cup}\, \{d^5/e^3, d^6/e^4\}, \quad 4 \mapsto \zeta(2) \,\dot{\cup}\, \{d^7/e^3, d^8/e^4\} \end{array} \right\}.$$

As shown by the tree representation in Fig. 17, actions $(d^1)?(d^2), d^1{\neq}i$ and $(d^3)?(d^4), d^4{\neq}3$ are pattern equivalent and defined by sibling necessities in the conjunction of equation X_0. For these to be uniformed, the substitution map ζ projects indices 1 and 2 onto substitutions $\{d^1/e^1{,}d^2/e^2\}$ and $\{d^3/e^1{,}d^4/e^2\}$ *resp.* Once the substitution is applied

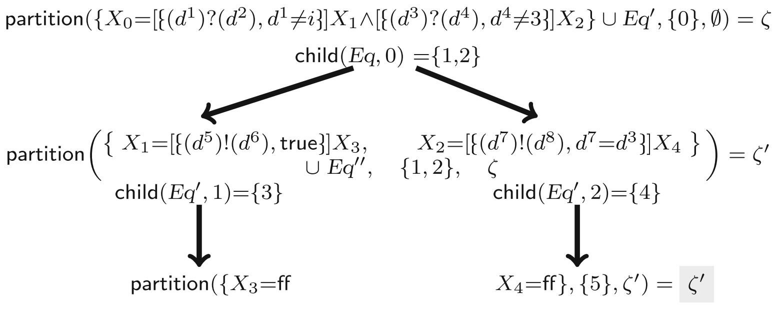
partition({X_0=[{$(d^1)?(d^2), d^1{\neq}i$}]$X_1 \wedge$[{$(d^3)?(d^4), d^4{\neq}3$}]X_2} ∪ Eq', {0}, ∅) = ζ
child(Eq, 0) ={1,2}
partition({ X_1=[{$(d^5)!(d^6)$, true}]X_3, X_2=[{$(d^7)!(d^8), d^7{=}d^3$}]X_4 } ∪ Eq'', {1,2}, ζ) = ζ'
child(Eq', 1)={3}
child(Eq', 2)={4}
partition({X_3=ff X_4=ff}, {5}, ζ') = ζ'

Fig. 18 A breadth first traversal using partition to obtain ζ

to both *SAs* we obtain $(e^1)?(e^2), e^1{\neq}i$ and $(e^1)?(e^2), e^2{\neq}3$. Notice how the patterns in both of the necessities are now *syntactically equal*, meaning that the resulting equation $X_0{=}[(e^1)?(e^2), e^1{\neq}i]X_1 \wedge [(e^1)?(e^2), e^2{\neq}3]X_2$ is now *uniform*.

Since $(d^5)!(d^6)$, true and $(d^7)!(d^8), d^7{=}d^3$ are pattern equivalent siblings in X_0, to achieve uniformity ζ provides mappings $3 \mapsto \zeta(1) \mathbin{\dot{\cup}} \{d^5/e^3, d^6/e^4\}$ and $4 \mapsto \zeta(2) \mathbin{\dot{\cup}} \{d^7/e^3, d^8/e^4\}$ that rename these *SAs* to $(e^3)!(e^4)$, true and $(e^3)!(e^4), e^3{=}e^1$. Notice how condition $d^7{=}d^3$ in $(d^7)!(d^8), d^7{=}d^3$ was also renamed to $e^3{=}e^1$ as variable d^3 was substituted by e^1 when its binding *SA* $(d^3)?(d^4), d^4{\neq}3$ was uniformed into $(e^1)?(e^2), e^2{\neq}3$. This substitution was possible since mapping $\zeta(4)$ includes the substitutions returned by the parent's index, i.e., $\zeta(2)$ that allows for applying the substitutions performed upon the parent, to its children, thus keeping the *SoE* closed. □

So far we have assumed the existence of a well-formed ζ map that provides all the necessary information, without having any knowledge as to how it is created. The ζ map is created as a result of conducting a breadth first traversal, via the traverse function, on the given equation set, using the partition function (defined in Fig. 16) as the λ projection function for traverse. The function partition:$(Eq \times \mathcal{P}(\text{INDEX}) \times \text{ACC}) \to \text{ACC}$ follows the format dictated by λ, i.e., it takes as input a set of equations *Eq*, a set of indices I and an accumulator – in this case ζ – and returns an updated version of ζ as a result. To update ζ, partition inspects the sibling equations denoted by the indices in I and as a result creates a *substitution environment* which renames the variable names of each pattern equivalent sibling necessity, to the same fresh set of variables.

Example 25 Recall $(Eq,\ X_0,\ \emptyset)$ from Example 24 where

$$Eq = \left\{ \begin{array}{c} X_0{=}[(d^1)?(d^2), d^1{\neq}i]X_1 \wedge [(d^3)?(d^4), d^4{\neq}3]X_2,\ \ X_3{=}\mathsf{ff}, \\ X_1{=}[(d^5)!(d^6), \mathsf{true}]X_3,\ \ X_2{=}[(d^7)!(d^8), d^7{=}d^3]X_4,\ \ X_4{=}\mathsf{ff} \end{array} \right\}.$$

Fig. 18 depicts the breadth first traversal performed by the traverse function in which the projection function partition was applied on each set of siblings. Notice that when partition is applied on the root equation, the initially empty ζ map gets extended by two entries, namely $\zeta{=}\emptyset{\cup}\{1 \mapsto \emptyset \mathbin{\dot{\cup}} \{e^1/d^1, e^2/d^2\}, 2 \mapsto \emptyset \mathbin{\dot{\cup}} \{e^1/d^3, e^2/d^4\}\}$. As shown in Example 24, this allows for the sibling necessities defined in X_0 to be uniformed. The ζ map is further extended into $\zeta'{=}\zeta{\cup}\{3 \mapsto \zeta(1) \mathbin{\dot{\cup}} \{e^3/d^5, e^4/d^6\}, 4 \mapsto \zeta(2) \mathbin{\dot{\cup}} \{e^3/d^7, e^4/d^8\}\}$, since the partition function recognises that sibling *SAs* $(d^5)!(d^6)$, true and $(d^7)!(d^8), d^7{=}d^3$ are also pattern equivalent. It therefore maps variables d^5, d^7 to the same fresh variable e^3, and d^6, d^8 to e^4. □

Lemma 7 *For every* $SoE(Eq, X_0, \mathcal{Y})$ *if* $\langle\!\langle (Eq, X_0, \mathcal{Y}) \rangle\!\rangle_{(i)} = \left(Eq', X_0', \mathcal{Y}'\right)$ *then* $(Eq, X_0, \mathcal{Y}) \equiv \left(Eq', X_0', \mathcal{Y}'\right)$ *and* $\left(Eq', X_0', \mathcal{Y}'\right)$ *is* uniform.

Proof To prove this statement, we assume knowledge of Lemmas 8 and 9 both of which are proved in Appendix A.

Lemma 8 *For every equation set Eq if* $\mathsf{traverse}(Eq, \{0\}, \mathsf{partition}, \emptyset){=}\zeta$ *then* ζ *is a* well-formed *map for Eq.*

Lemma 9 *For every* ζ *map, and equation set Eq, if* ζ *is a* well-formed *map for Eq then* $\mathsf{uni}(Eq, \zeta){\equiv}Eq$ *and every equation* $(X_k{=}\psi_k) \in \mathsf{uni}(Eq, \zeta)$ *is* Uniform.

Now assume that $\langle\!\langle (Eq, X_0, \mathcal{Y}) \rangle\!\rangle_{(i)} = \left(Eq', X_0', \mathcal{Y}'\right)$ and so by the definition of $\langle\!\langle - \rangle\!\rangle_{(i)}$ we have that $X_0' = X_0, \mathcal{Y}' = \mathcal{Y}$ and $Eq' = \mathsf{uni}(Eq, \zeta)$ where $\zeta = \mathsf{traverse}(Eq, \{0\}, \mathsf{partition}, \emptyset)$ from which by Lemma 8 we can deduce that ζ is a *well-formed* map for *Eq*. This means that from Lemma 9 we can infer that

$$\mathsf{uni}(Eq, \zeta){\equiv}Eq \tag{75}$$

$$\text{every equation } (X_k{=}\psi_k) \in \mathsf{uni}(Eq, \zeta) \text{ is} uniform \tag{76}$$

and so since from (75) we know that the uniformed equation set is equivalent to *Eq* and from (76) we have that every equation is uniform, we conclude that

$$(Eq, X_0, \mathcal{Y}) \equiv \left(Eq', X_0', \mathcal{Y}'\right) \text{ and that } \left(Eq', X_0', \mathcal{Y}'\right) \text{ is } \textit{uniform} \tag{77}$$

as required, and so we are done.

5.4.3 Condition reformulation of sibling symbolic actions

By reformulating the conditions of sibling symbolic actions in a uniform *SoE*, we aim to obtain its *equi-disjoint* equivalent.

Definition 14 (*System of equi-disjoint equations*) An equation is *equi-disjoint* when it is *uniform*, and its sibling necessities *cannot be satisfied* by the same concrete action α, unless they are *syntactically equal*. A *SoE* is *equi-disjoint* when all of its equations are *equi-disjoint*. □

Example 26 As per Definition 14, we can thus infer that equation

$$X_0 = [(d)?(e), e{>}5]X_1 \wedge [(d)?(e), e{>}5]X_2 \wedge [(d)?(e), e{\leq}5]X_3$$

is *equi-disjoint* since there does not exist a system action that can satisfy both $(d)?(e), e{>}5$ and $(d)?(e), e{\leq}5$. The only two branches that are satisfied by common actions are $[(d)?(e), e{>}5]X_1$ and $(d)?(e), e{>}5X_2$ but they are both prefixed by *syntactically equal* necessities. However, for equation $X_1 = [(d^1)?(e^1), \mathsf{true}]X_4 \wedge [(d^1)?(e^1), e^1 \neq 5]X_5$ we can immediately conclude that it is *not* equi-disjoint. □

Figure 19 presents function $\langle\!\langle - \rangle\!\rangle_{(ii)}\colon \left(Eq^{\mathsf{uni}}, \textsc{Var}, \mathcal{P}(\textsc{Var})\right) \rightarrow Eq^{\mathsf{ed}}$ for recomposing uniform *SoEs* into equi-disjoint ones. Internally, this function uses the $\mathsf{traverse}$ function to perform a breadth first traversal on the given uniform equation set, Eq^{uni}, starting from the principal equation, i.e., with $I{=}\{0\}$. While conducting the traversal, it applies the $\mathsf{cond\_comb}$

$$\langle\!\langle (\, Eq\,,\ X_0,\ \mathcal{Y}) \rangle\!\rangle_{\text{(ii)}} \overset{\text{def}}{=} (\, \mathsf{traverse}(Eq, \{0\}, \mathsf{cond\_comb}, \emptyset)\,,\ X,\ \mathcal{Y})$$

$$\mathsf{cond\_comb}(Eq, I, \omega) \overset{\text{def}}{=} \left\{ X_i {=} \bigwedge_{c_k \in \mathbb{C}(j,I')} [\{p, c_k\}] X_j \wedge \varphi \;\middle|\; \begin{array}{l} (X_i {=} \bigwedge_{j \in I''} [\{p, c_j\}] X_j \wedge \varphi) \in Eq_{//I} \\ \text{and } I' {=} \bigcup_{l \in I} \mathsf{child}(Eq, l) \\ \textit{such that } I'' \subseteq I' \end{array} \right\} \overset{+}{\cup}\, \omega$$

$$\mathbb{C}(j, I) \overset{\text{def}}{=} \left\{ \begin{array}{c} \underline{c_j} \wedge c_i \ldots \wedge c_n, \\ \underline{c_j} \wedge \neg c_i \ldots \wedge c_n, \\ \underline{c_j} \wedge \neg c_i \ldots \wedge \neg c_n \end{array} \;\middle|\; \begin{array}{c} \forall i \ldots n \in I \text{ where } j \neq i \neq \ldots \neq n \\ \text{such that } p_j = p_i = \ldots = p_n \end{array} \right\}$$

Fig. 19 The Conjunction Reformulation Algorithm

function to reconstruct the uniform conjunctions, defined in $(X_i{=}\varphi_i) \in Eq^{\mathsf{uni}}$, into equi-disjoint ones, thereby producing an *equi-disjoint* equation set Eq^{ed} at the end of the traversal.

The function $\mathsf{cond\_comb}{:}(Eq^{\mathsf{uni}} \times \mathcal{P}(\textsc{Index}) \times \textsc{Acc}){\rightarrow}\textsc{Acc}$ is a projection function that takes as input a uniform equation set Eq^{uni}, a set of indices I, and an accumulator ω. The accumulator ω contains a partial equi-disjoint set of equations which is first initialised to $\emptyset$ and is constantly extended by recursive $\mathsf{cond\_comb}$ applications until the traversal is complete, in which case ω is returned as the resultant equi-disjoint equation set. In order to update ω, the $\mathsf{cond\_comb}$ function inspects the sibling equations denoted by the indices in I, i.e., $(X_i{=}\varphi_i) \in Eq_{//I}$, and computes the *truth combinations* of the *conditions* defined by sibling symbolic necessities defining syntactically equal patterns.

To compute these truth combinations, the $\mathsf{cond\_comb}$ function starts by computing the child indices of the current sibling equations, denoted by I, by using the child function, i.e., $I'{=}\bigcup_{l \in I} \mathsf{child}(Eq, l)$. It then inspects the conjunctions defined in the selected equations, i.e., $\bigwedge_{j \in I''} [p_j, c_j]X_j \wedge \varphi$, and reconstructs them into $\bigwedge_{c_k \in \mathbb{C}(j,I')} [p_j, c_k]X_j \wedge \varphi$. Notice that c_k is a *truth combination* of all the filtering conditions that are defined by modal necessities that specify *syntactically equal patterns* and which are defined by the branches identified by the indices in I'. For instance, if $I'{=}\{1, 2, 3\}$, then one possible truth combination c_k is $c_1{\wedge}\neg c_2{\wedge}c_3$.

The truth combinations, such as c_k, are generated through the *combinatorial function* $\mathbb{C}{:}(\textsc{Index} \times \mathcal{P}(\textsc{Index}))$. It takes as input the index j of the branch that is being analysed, along with the indices of all the sibling branches specified in I'. As a result, $\mathbb{C}(j, I')$ returns the truth combinations in which the filtering condition, c_j, of the branch that is currently being reconstructed is *true*. For instance, $\mathbb{C}(1, \{1, 2, 3\})$ provides combinations $\big\{(c_1{\wedge}c_2{\wedge}c_3),\ (c_1{\wedge}c_2{\wedge}\neg c_3),\ (c_1{\wedge}\neg c_2{\wedge}c_3),\ (c_1{\wedge}\neg c_2{\wedge}\neg c_3)\big\}$ where c_1 is always true. These truth combinations are then used to reconstruct the existing branch into a collection of equi-disjoint branches.

The resultant equations are thus *equi-disjoint* as the truth combination conditions ensure that a concrete system event α can *never* satisfy multiple symbolic necessities in the reconstructed branches, unless these are *syntactically equal*. Note that the truth combinations generated by function $\mathbb{C}(j, I')$ *do not include the cases where* c_j *is false*. This is essential to ensure that none of the reconstructed branches can be satisfied when the original condition c_j is false, thereby preserving the semantics of the original branch.

Once the traversal completes, the construction outputs the final accumulator value ω containing the required equi-disjoint equation set.

Example 27 Consider equation $X_0 = [p, c_1]X_1 \wedge [p, c_2]X_2 \wedge [p, c_3]X_3$, using the truth combinations provided by $\mathbb{C}(1, \{1, 2, 3\})$ we can reconstruct branch $[p, c_1]X_1$ into:

$$[p, \underline{c_1} \wedge c_2 \wedge c_3]X_1 \wedge [p, \underline{c_1} \wedge c_2 \wedge \neg c_3]X_1 \wedge [p, \underline{c_1} \wedge \neg c_2 \wedge c_3]X_1 \wedge [p, \underline{c_1} \wedge \neg c_2 \wedge \neg c_3]X_1.$$

Similarly, with $\mathbb{C}(2, \{1, 2, 3\})$ and $\mathbb{C}(3, \{1, 2, 3\})$, we can reconstruct branches $[p, c_2]X_2$ and $[p, c_3]X_3$ in the same way such that the resultant equation is:

$$X_0 = \begin{pmatrix} [p, \underline{c_1} \wedge c_2 \wedge c_3]\underline{X_1} \wedge [p, \underline{c_1} \wedge c_2 \wedge \neg c_3]\underline{X_1} \wedge \\ [p, \underline{c_1} \wedge \neg c_2 \wedge c_3]\underline{X_1} \wedge [p, \underline{c_1} \wedge \neg c_2 \wedge \neg c_3]\underline{X_1} \wedge \\ \\ [p, c_1 \wedge \underline{c_2} \wedge c_3]\underline{X_2} \wedge [p, c_1 \wedge \underline{c_2} \wedge \neg c_3]\underline{X_2} \wedge \\ [p, \neg c_1 \wedge \underline{c_2} \wedge c_3]\underline{X_2} \wedge [p, \neg c_1 \wedge \underline{c_2} \wedge \neg c_3]\underline{X_2} \wedge \\ \\ [p, c_1 \wedge c_2 \wedge \underline{c_3}]\underline{X_3} \wedge [p, \neg c_1 \wedge c_2 \wedge \underline{c_3}]\underline{X_3} \wedge \\ [p, c_1 \wedge \neg c_2 \wedge \underline{c_3}]\underline{X_3} \wedge [p, \neg c_1 \wedge \neg c_2 \wedge \underline{c_3}]\underline{X_3} \end{pmatrix}$$

Notice that logical variables X_1, X_2 and X_3 can only be evaluated when their prefixing modal necessities are satisfied by some system action, meaning that continuation X_1 is only reachable when c_1 is true, and *resp.* X_2 and X_3 when c_2 and c_3 are true. Hence, in the reconstructed equation, these (underlined) conditions are *never negated* when prefixing the *resp.* logical variable. □

Lemma 10 *For every system of equations,* $(Eq, X_0, \mathcal{Y})$*, if* $(Eq, X_0, \mathcal{Y})$ *is* uniform *then* $\langle\!\langle(Eq, X_0, \mathcal{Y})\rangle\!\rangle_{(ii)} \equiv (Eq, X_0, \mathcal{Y})$ *and* $\langle\!\langle(Eq, X_0, \mathcal{Y})\rangle\!\rangle_{(ii)}$ *is* equi-disjoint.

Proof For this proof we assume the knowledge of Lemma 11 which is proved in Appendix A.

Lemma 11 *For every equation* $(X_j{=}\varphi_j) \in Eq$*, if* $X_j{=}\varphi_j$ *is* uniform *then we have that* $Eq \equiv \textit{traverse}(Eq, \{0\}, \textit{cond\_comb},)\emptyset$ *and that every eqn.* $(X_k{=}\psi_k) \in \textit{traverse}(Eq, \{0\}, \textit{cond\_comb}, \emptyset)$ *is* equi-disjoint.

Now, let's assume that $(Eq, X_0, \mathcal{Y})$ is *uniform* which means that every equation $(X_j{=}\varphi_j) \in Eq$ is uniform, and so by Lemma 11 we deduce that

$$Eq \equiv \mathsf{traverse}(Eq, \{0\}, \mathsf{cond\_comb}, \emptyset) \tag{78}$$

$$\forall (X_k{=}\psi_k) \in \mathsf{traverse}(Eq, \{0\}, \mathsf{cond\_comb}, \emptyset) \cdot \ \mathsf{eqn}\ (X_k{=}\psi_k) \text{ is } \textit{equi-disjoint}. \tag{79}$$

Now since $\langle\!\langle(Eq, X_0, \mathcal{Y})\rangle\!\rangle_{(\mathrm{ii})} = (\mathsf{traverse}(Eq, \{0\}, \mathsf{cond\_comb}, \emptyset), X_0, \mathcal{Y})$ by (78) and (79) we can thus conclude that

$$\langle\!\langle(Eq, X_0, \mathcal{Y})\rangle\!\rangle_{(\mathrm{ii})} \equiv (Eq, X_0, \mathcal{Y}) \ \textit{and} \ \langle\!\langle(Eq, X_0, \mathcal{Y})\rangle\!\rangle_{(\mathrm{ii})} \text{ is } \textit{equi}-\textit{disjoint}$$

as required, and so we are done.

In Example 19 we had shown that the algorithm presented in Sect. 5.3 fails when dealing with non-singleton *SAs*. This can now be resolved by applying steps **§i.** and **§i.** prior to applying **§3.** – we leave this as an exercise to the reader.

With the extended normalisation algorithm we can finally conclude that Theorem 5 also holds for any sHML formula (defining any kind of *SAs*) as a result of Lemmas 2 and 3 followed by Lemmas 7 and 10, and then by Lemmas 5 and 6.

$$after_\varphi(\varphi,\alpha) \overset{\text{def}}{=} \begin{cases} \varphi & \text{if } \varphi \in \{\mathsf{tt},\mathsf{ff}\} \\ after_\varphi(\varphi\{\max X.\varphi/X\},\alpha) & \text{if } \varphi = \max X.\varphi \\ \bigwedge_{i\in I} after_\varphi(\varphi_i,\alpha) & \text{if } \varphi = \bigwedge_{i\in I}\varphi_i \\ \psi\sigma & \text{if } \varphi = [\{p,c\}]\psi \textit{ and } \mathsf{mtch}(p,\alpha){=}\sigma \textit{ and } c \Downarrow \mathsf{true} \\ \mathsf{tt} & \text{if } \varphi = [\{p,c\}]\psi \textit{ and otherwise} \end{cases}$$

Fig. 20 Defining the $after_\varphi$ function

6 Restricting weak enforcement to sHML

Although in Sect. 4 we prove that Definition 7 is inherently weaker than Definition 4 (i.e., Theorem 2), both definitions become *equally powerful* when restricted to sHML. As both are defined in terms of Definition 2 (Soundness) and only vary with respect to the transparency definition, to ensure this result it suffices to prove Theorem 6, i.e., that the Definitions 3 (Transparency) and 6 (Trace Transparency) coincide with respect to sHML formulas.

Theorem 6 *For every monitor m and formula $\varphi \in$ sHML, $\mathsf{tenf}(m,\varphi)$ iff $\mathsf{ttenf}(m,\varphi)$.* □

Since the if-case has already been proven to hold for the full μHML (in Theorem 2), this result implicitly applies for sHML, so no additional proofs are required. For the only-if case we, however, require an additional proof that uses the following lemmas whose proofs are provided in Appendix B.

Lemma 12 *For every system s, sHML formula φ and trace $t \in traces(s)$ when $s \in [\![\varphi]\!]$ then $\mathsf{sys}(t) \in [\![\varphi]\!]$ (where $traces(s) \overset{\text{def}}{=} \{\, t \mid s \overset{t}{\Rightarrow} \}$).*

Lemma 13 *For every system transition $s \overset{\alpha}{\Rightarrow} s'$ and sHML formula φ, if $s \in [\![\varphi]\!]$ then $s' \in [\![\mathsf{after}_\varphi(\varphi,\alpha)]\!]$ (where $\mathsf{after}_\varphi(\varphi,\alpha)$ is defined in Fig. 20).*

Lemma 14 *For every action α, sHML formula φ and trace t, if $\mathsf{sys}(t) \in [\![\mathsf{after}_\varphi(\varphi,\alpha)]\!]$ then $\mathsf{sys}(\alpha t) \in [\![\varphi]\!]$.*

Proof We prove Theorem 6 coinductively by showing that relation $\mathcal{R} \overset{\text{def}}{=} \{(m[s],s) s \in [\![\varphi]\!] \text{ and } \mathsf{ttenf}(m,\varphi)\}$ is a *strong bisimulation relation* and thus satisfies the following transfer properties, i.e., for each $(m[s],s) \in \mathcal{R}$:

(a) if $m[s] \xrightarrow{\mu} r'$ then $s \xrightarrow{\mu} s'$ and $(r',s') \in \mathcal{R}$
(b) if $s \xrightarrow{\mu} s'$ then $m[s] \xrightarrow{\mu} r'$ and $(r',s') \in \mathcal{R}$.

To prove (a), assume that

$$m[s] \xrightarrow{\mu} r' \tag{80}$$

$$s \in [\![\varphi]\!] \tag{81}$$

and that $\mathsf{ttenf}(m,\varphi)$ from which by Definition 6 we have that

$$\textit{if } \mathsf{sys}(t) \in [\![\varphi]\!] \textit{ and } m[\mathsf{sys}(t)] \overset{t'}{\Rightarrow} m'[\mathsf{sys}(t'')] \textit{ then } t = t't'' \tag{82}$$

and so by Lemma 12 from (81) and we infer that

$$\forall t \in traces(s) \cdot \mathsf{sys}(t) \in [\![\varphi]\!]. \tag{83}$$

From (82) and (83), we can thus conclude that monitor m does not modify any of the behaviours (traces) of s and so we know that

$$\forall t \in \mathit{traces}(s) \cdot m[\mathsf{sys}(t)] \overset{t}{\Rightarrow} . \tag{84}$$

We now explore all the possible instrumentation rules by which the reduction in (80) can occur.

- iAsy: From (80) and rule iAsy, we have that $\mu = \tau$ and that

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

$$r' = m[s']. \tag{86}$$

  Since by Proposition 4 we know that sHML is agnostic of τ-actions, from (81) and (85) we also know that $s' \in [\![\varphi]\!]$ and so since from (86) we know that m remains unmodified by the transition, from (82) and the definition of $\mathcal{R}$ we conclude that

$$(m'[s'], s') \in \mathcal{R} \tag{87}$$

  as required. Hence, this case holds by (85) and (87).
- iDef: From (80) and rule iDef, we have that $\mu = \alpha$ and that

$$s \xrightarrow{\alpha} s' \tag{88}$$

$$r' = \mathsf{id}[s']. \tag{89}$$

  Since id can only apply identity transformations, we can simply infer that for any formula ψ, $\mathsf{ttenf}(\mathsf{id}, \psi)$, and so we conclude that

$$\mathsf{ttenf}(\mathsf{id}, \mathit{after}_\varphi(\varphi, \alpha)). \tag{90}$$

  Finally, by (81), (88) and Lemma 13 we deduce that $s' \in [\![\mathit{after}_\varphi(\varphi, \alpha)]\!]$, and so knowing (90) and by the definition of $\mathcal{R}$ we conclude that

$$(\mathsf{id}[s'], s') \in \mathcal{R} \tag{91}$$

  as required. Hence, this case holds by (88) and (91).
- iTrn (identity): From (80) and rule iTrn we have that

$$s \xrightarrow{\alpha} s' \tag{92}$$

$$m \xrightarrow{\alpha \blacktriangleright \alpha} m'' \tag{93}$$

$$r' = m''[s'] \tag{94}$$

  and so by (81), (92) and Lemma 13 we can immediately deduce that

$$s' \in [\![\mathit{after}_\varphi(\varphi, \alpha)]\!]. \tag{95}$$

  Now, assume that for every trace u, we have that

$$\mathsf{sys}(u) \in [\![\mathit{after}_\varphi(\varphi, \alpha)]\!] \tag{96}$$

$$m''[\mathsf{sys}(u)] \xRightarrow{u'} m'[\mathsf{sys}(u'')]. \tag{97}$$

Knowing (96), by Lemma 14 we have that

$$\mathsf{sys}(\alpha u) \in [\![\varphi]\!] \tag{98}$$

and so from (82) and (98) we can infer that

$$\textit{if } m[\mathsf{sys}(\alpha u)] \xRightarrow{\alpha u'} m'[\mathsf{sys}(u'')] \textit{ then } \alpha u = \alpha u' u'' \tag{99}$$

and thus from (93), (97) and (99) we can conclude that

$$u = u'u''. \tag{100}$$

Hence, from assumptions (96), (97) and deduction (100) we can introduce an implication so that by Definition 6 we conclude that

$$\mathsf{ttenf}(m'', \mathit{after}_{\varphi}(\varphi, \alpha)) \tag{101}$$

and so by (95), (101) and the definition of $\mathcal{R}$ we have that

$$(m''[s'], s') \in \mathcal{R} \tag{102}$$

as required, and so we are done by (92) and (102).
- iSup, iIns, iTrn (replacement): These cases do not apply since these rules modify the trace actions executed by s, and so if (80) is the result of any these rules, it would contradict with (84).

These cases thus allow us to conclude that (a) holds.

We now proceed to prove (b). Assume that

$$s \xrightarrow{\mu} s' \tag{103}$$

$$s \in [\![\varphi]\!] \tag{104}$$

$$\mathsf{ttenf}(m, \varphi) \tag{105}$$

and so since $\mu \in \{\tau, \alpha\}$ we consider each case separately.

- $\mu = \tau$: Since $s \xrightarrow{\tau} s'$, by (104) and since sHML is agnostic of τ-actions (Proposition 4), we know that

$$s' \in [\![\varphi]\!] \tag{106}$$

and by rule iAsy we can also deduce that

$$m[s] \xrightarrow{\tau} m[s']. \tag{107}$$

Hence, by (105), (106) and the definition of $\mathcal{R}$ we can conclude that

$$(m[s'], s') \in \mathcal{R} \tag{108}$$

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

– $\mu = \alpha$: Since $s \xrightarrow{\alpha} s'$ from (104) and Lemma 13 we have that

$$s' \in [\![\mathit{after}_{\varphi}(\varphi, \alpha)]\!] \tag{109}$$

as required. From (105) and by Definition 6, we know that for every trace t

$$\textit{if}\ \mathsf{sys}(t) \in [\![\varphi]\!]\ \textit{and}\ m[\mathsf{sys}(t)] \overset{t'}{\Longrightarrow} m'[\mathsf{sys}(t'')]\ \textit{then}\ t = t't'' \tag{110}$$

and by Lemma 12 from (104) we infer that for every trace u that can be executed by s, i.e., $u \in \mathit{traces}(s)$, $\mathsf{sys}(u) \in [\![\varphi]\!]$ and so since $s \xrightarrow{\alpha} s'$ we know that $\mathsf{sys}(\alpha u') \in [\![\varphi]\!]$ where $u' \in \mathit{traces}(s')$. Hence, from (110) we can infer that

$$\textit{if}\ m[\mathsf{sys}(\alpha u')] \overset{\alpha u''}{\Longrightarrow} m'[\mathsf{sys}(u''')]\ \textit{then}\ \alpha u' = \alpha u'' u''' \tag{111}$$

which means that m is unable to modify any of the α-prefixed behaviours of s, and so since $s \xrightarrow{\alpha} s'$ we have that

$$\exists m'' \cdot m[s] \xrightarrow{\alpha} m''[s'] \tag{112}$$

as required. Finally, let's assume that for every trace v,

$$\mathsf{sys}(v) \in [\![\mathit{after}_{\varphi}(\varphi, \alpha)]\!] \tag{113}$$

$$m''[\mathsf{sys}(v)] \overset{v'}{\Longrightarrow} m'[\mathsf{sys}(v'')]. \tag{114}$$

Since by (113) and Lemma 14 we have that $\mathsf{sys}(\alpha v) \in [\![\varphi]\!]$, from (110) we can infer that

$$\textit{if}\ m[\mathsf{sys}(\alpha v)] \overset{\alpha v'}{\Longrightarrow} m'[\mathsf{sys}(v'')]\ \textit{then}\ \alpha v = \alpha v' v'' \tag{115}$$

and thus from (114) and (115) we can conclude that

$$v = v'v''. \tag{116}$$

Hence, from assumptions (113), (114) and deduction (116) we can introduce an implication so that by Lemma 6 we conclude that

$$\mathsf{ttenf}(m'', \mathit{after}_{\varphi}(\varphi, \alpha)) \tag{117}$$

and so by (109), (117) and the definition of $\mathcal{R}$ we have that

$$(m''[s'], s') \in \mathcal{R} \tag{118}$$

as required. Hence, this case holds by (112) and (118).

Remark 2 Although we have carried out our investigation for a branching time setting, it is natural to ask what the relation between enforceable branching-time properties and linear-time properties is. Intuitively, one might expect that a process satisfies a property φ in sHML if, and only if, each of its traces does so in the linear-time interpretation of φ. This intuition is formalised in [23, Proposition 5.11] for a setting without data. A version of that result should also hold true also for the version of sHML studied in this paper.

7 Conclusion

This paper presents a preliminary investigation of the enforceability of first-order branching-time properties expressed in a process logic with data bindings and constraints. We have focussed on a highly expressive and standard logic, μHML, and studied the ability to enforce μHML properties via a specific kind of monitor that performs suppression-based enforcement. We concluded that the safety fragment of μHML, i.e., sHML, is enforceable via these kind of monitors. To show this, we first defined enforceability for logics and system descriptions interpreted over labelled transition systems. Although enforceability builds upon soundness and transparency requirements that have been considered in other work, our branching-time framework required us to consider a broader design spaced for these requirements, resulting in new definitions for soundness and transparency. We also contend that the definitions that we develop for the enforcement framework are fairly modular: e.g., the instrumentation relation is independent of the specific language constructs defining our transducer monitors and it functions as expected as long as the transition semantics of the transducer and the system are in agreement. Based on this notion of enforcement, we devise a two-phase procedure to synthesise correct enforcement monitors. We first identify a syntactic subset of our target logic sHML that affords certain structural properties and permits a compositional definition of the synthesis function. We then show that, by augmenting existing rewriting techniques to our setting, we can convert any sHML formula into this syntactic subset. This yields one of the first *syntactic* studies of logic enforceability. Although our logic is declarative in nature (describing *what*), we are able to demonstrate how its syntactic constructs can still be used to define a synthesis procedure that generates operational descriptions detailing *how* a property is enforced. The flip-side of this approach is that we are then able to precisely describe the properties that we are able to enforce in terms of the *grammar of the logic fragment* considered. This modus operandi is essential for ensuring correct tool construction [32, 51]. Unfortunately this method is rarely used in the literature either because the properties to be enforced are never defined syntactically or because they are defined in terms of automata, which already have a strong operational flavour.

Related Work.

In his seminal work [2], Schneider regards a property (in a linear-time setting) to be enforceable if its *violation* can be *detected* by a *truncation automaton*, and prevents its occurrence via system termination; by preventing misbehaviour, these monitors can only enforce safety properties. In [4], Ligatti et al. extended this work via *edit automata*—an enforcement mechanism capable of *suppressing* and *inserting* system actions. A property is thus enforceable if it can be expressed as an edit automaton that *transforms* invalid executions into valid ones via suppressions and insertions. Edit automata are capable of enforcing instances of safety and liveness properties, along with other properties such as infinite renewal properties [4, 75]. As a means to assess the correctness of these automata, the authors introduced *soundness* and *transparency* along the lines of our trace transparency, Definition 6. All this work is pitched at a linear-time setting where properties are defined in terms of traces. They are never characterised syntactically, and they never discuss edit-automata synthesis. Moreover, first-order properties are not considered, limiting traces to a *finite* set of actions.

Könighofer et al. in [10] present a synthesis algorithm that produces action replacement transducers called *shields* from safety properties encoded as automata-based specifications. Shields analyse the inputs and outputs of a reactive system and enforce properties by modifying the least amount of output actions whenever the system deviates from the specified behaviour. By definition, shields should adhere to two desired properties, namely correctness and minimum deviation. Although these two criteria can be viewed as analogous to soundness and transparency, respectively, they are different from the ones we consider in our work, Definition 2 and Definition 3, since we operate within a branching-time setting. Moreover, Könighofer et al. do not study the enforceability of the logic. Falcone et al. in [6, 28], also propose synthesis procedures to translate properties—expressed as Streett automata—into the *resp.* monitors. The authors show that most of the property classes defined within the *safety-progress hierarchy* [76] are enforceable, as they can be encoded as Streett automata and subsequently converted into enforcement automata. Although this is one of the first bodies of work to coin the term enforceability, their investigation of property enforceability is very different from ours in two respects: they do not consider a declarative logic and consider linear-time properties defined over traces. Neither Könighofer et al. nor Falcone et al. consider first-order enforcement.

In [77], Pinisetty et al. consider first-order enforcement of timed properties. Apart from the timing aspect, which is not considered by our work, Pinisetty et al. study linear-time properties. This work does not define any automated synthesis procedures nor does it present any correctness proofs for the monitors considered. Instead the authors focus on providing an empirical assessment of the performance of their monitors. In other work [78, 79], Pinisetty et al. study the enforcement of input–output properties. Although they provide correctness guarantees for the enforcement monitors they define in terms of criteria such as soundness and transparency, they do not attempt to syntactically characterise any enforceable subset of properties. Crucially, the authors do not consider first-order properties and work in a linear-time setting.

Lanotte et al. [80] employ a process-based approach for the runtime enforcement of security properties that is very similar to our model of process monitors and instrumentation. Although their implementations handle the enforcement of data-based properties, their formalism does not. Their work does study the problem of logic enforceability.

Bielova et al. [70, 75] remark that soundness and transparency do not specify to what extent a transducer should modify an invalid execution. They thus introduce a *predictability* criterion to prevent transducers from transforming invalid executions arbitrarily. More concretely, a transducer is *predictable* if one can predict the number of transformations that it will apply in order to transform an invalid execution into a valid one, thereby preventing monitors from applying unnecessary transformations over an invalid execution. Using this notion, Bielova et al. thus devise a more stringent notion of enforceability. Although we do not explore this avenue, Definition 6 may be viewed as an attempt to constrain transformations of violating systems in a branching-time setup, and should be complementary to these predictability requirements. Importantly, the work by Bielova et al. is limited to the regular properties and does not study the enforcement of first-order computation.

To the best of our knowledge, the only other work that tackles enforceability for the modal μ-calculus [29] (a reformulation of μHML) is that of Martinelli et al. in [81, 82]. Their approach is, however, different from ours. In addition to the μ-calculus formula to enforce, their synthesis function also takes a "witness" system satisfying the formula as a parameter. This witness system is then used as the behaviour that is mimicked by the instrumentation via suppression, insertion or replacement mechanisms. Although the authors do not explore automated correctness criteria such as the ones we study in this work, it would be interesting to explore the applicability of our methods to their setting.

Bocchi et al. [19] adopt *multi-party session types* to project the global protocol specifications of distributed networks to *local types* defining a local protocol for every process in the network that are then either verified statically via typechecking or enforced dynamically via suppression monitors. To implement this enforcement strategy, the authors define a dynamic monitoring semantics for the local types that suppress process interactions so as to conform to the assigned local specification. They prove local soundness and transparency for monitored processes that, in turn, imply global soundness and transparency by construction. Their local enforcement is closely related to the suppression enforcement studied in our work with the following key differences: (i) well-formed branches in a session type are, by construction, *explicitly disjoint* via the use of distinct choice labels (i.e., similar to our normalised subset $\mathrm{sHML}_{\mathbf{nf}}$), whereas we can synthesise monitors for *every* sHML formula using a normalisation procedure; (ii) they give an LTS semantics to their local specifications (which are session types) which allows them to state that a process satisfies a specification when its behaviour is bisimilar to the operational semantics of the local specification—we do not change the semantics of our formulas, which is left in its original denotational form; (iii) our monitor descriptions sit at a lower level of abstraction than theirs using a dedicated language, whereas theirs have a session-type syntax with an LTS semantics (e.g., repeated suppressions have to be encoded in our case using the recursion construct while this is handled by their high-level instrumentation semantics). Although they consider first-order enforcement, they do not investigate the enforceability of session types along the lines of Burlo et al. [16].

In [83], Castellani et al. adopt session types to define reading and writing privileges amongst processes in a network as global types for information flow purposes. These global types are projected into local monitors capable of preventing read and write violations by adapting certain aspects of the network. They operate in a first-order setting and their monitors occasionally adapt the network by suppressing messages or by replacing messages with messages carrying a default nonce value, but their work targets adaptation [3, 84], rather than enforcement.

Future work. We plan to extend this work along two different avenues. On the one hand, we will attempt to extend the enforceable fragment of μHML. For a start, we intend to investigate maximality results for suppression monitors, along the lines of [11, 12], and find out whether sHML is the largest μHML subset that is enforceable via action suppressions. We also plan to consider more expressive enforcement mechanisms such as insertion and replacement actions. Finally, we also want to identify and investigate different classes of system actions that might require more elaborate instrumentation setups to enforce. For instance, the mechanism required for suppressing an input action might differ from that of an output action. Such setups may include the ones explored in [13], that can reveal refusals in addition to the actions performed by the system.

On the other hand, we also plan to study the implementability and feasibility of our framework. We will consider target languages for our monitor descriptions that are closer to an actual implementation (e.g., an actor-based language along the lines of [85]). We could then employ refinement analysis techniques and use our existing monitor descriptions as the abstract specifications that are refined by the concrete monitor descriptions. The more concrete synthesis can then be used for the construction of tools that are more amenable towards showing correctness guarantees.

A Missing proofs from Sect. 5.2

We provide the proofs for Lemmas 6, 8, 9 and 11 which were omitted from the main text.

A.1 Proving Lemma 6

To prove that for every $\varphi \in \mathrm{sHML}_{\mathbf{2}}$, $[\![\langle\!\langle \varphi \rangle\!\rangle_5]\!] = [\![\varphi]\!]$ we must prove that

(a) $\forall s \in \textsc{Sys} \cdot s \vDash \langle\!\langle \varphi \rangle\!\rangle_5$ *implies* $s \vDash \varphi$; and
(b) $\forall s \in \textsc{Sys} \cdot s \vDash \varphi$ *implies* $s \vDash \langle\!\langle \varphi \rangle\!\rangle_5$.

In order to prove (a) and (b) we rely on the following lemmas:

Lemma 15 *For every* $\varphi \in \mathrm{sHML}_{\mathit{2}}$ *if* $X \in \mathbf{\mathit{fv}}(\varphi)$ *then* $X \in \mathbf{\mathit{fv}}(\langle\!\langle \varphi \rangle\!\rangle_5)$.

Lemma 16 *For every* $\varphi \in \mathrm{sHML}_{\mathit{2}}$ *if* $X \in \mathbf{\mathit{fv}}(\varphi)$ and $X \in \mathbf{\mathit{fv}}(\langle\!\langle \psi \rangle\!\rangle_5)$ *then* $\langle\!\langle \varphi\{\mathsf{max}\, X.\psi/X\} \rangle\!\rangle_5 = \langle\!\langle \varphi \rangle\!\rangle_5\{\mathsf{max}\, X.\langle\!\langle \psi \rangle\!\rangle_5/X\}$

We provide the proofs for these lemmas after the proofs for (a) and (b).

Proof for (a) Let $\mathcal{R} \overset{\text{def}}{=} \{\, (s, \varphi) \mid s \vDash \langle\!\langle \varphi \rangle\!\rangle_5 \,\}$, we must prove that $\mathcal{R}$ is a satisfaction relation by showing that it obeys the rules of Fig. 4. We conduct this proof by case analysis on φ.
Cases $\varphi \in \{\mathsf{ff}, X\}$. These cases do not apply since $\langle\!\langle \varphi \rangle\!\rangle_5 = \varphi$ and so the assumption that $s \vDash \langle\!\langle \varphi \rangle\!\rangle_5$ does not hold when $\varphi \in \{\mathsf{ff}, X\}$.
Case $\varphi = \mathsf{tt}$. This case is satisfied trivially since any process satisfies tt which confirms that $(s, \mathsf{tt}) \in \mathcal{R}$.
Cases $\varphi = \bigwedge_{i\in I} [\eta_i]\varphi_i$. In order to prove this case, we must confirm that $(s, \bigwedge_{i\in I} [\eta_i]\varphi_i) \in \mathcal{R}$ by showing that for every α and $i \in I$, if $s \overset{\alpha}{\Rightarrow} s'$ *s.t.* $\eta_i(\alpha) = \sigma$ then $(s', \langle\!\langle \varphi_i\sigma \rangle\!\rangle_5) \in \mathcal{R}$. Hence, we assume that $s \vDash \langle\!\langle \bigwedge_{i\in I} [\eta_i]\varphi_i \rangle\!\rangle_5$ and since by the definition of $\langle\!\langle - \rangle\!\rangle_5$ we know that $s \vDash \bigwedge_{i\in I} [\eta_i]\langle\!\langle \varphi_i \rangle\!\rangle_5$ then by the definition of $\vDash$ we have that

$$\forall i \in I, \alpha \in \textsc{Act} \cdot \; \textit{if}\; s \overset{\alpha}{\Rightarrow} s' \textit{ s.t.}\; \eta_i(\alpha) = \sigma \text{ then } s' \vDash \langle\!\langle \varphi_i\sigma \rangle\!\rangle_5. \tag{119}$$

Hence, by (119) and the definition of $\mathcal{R}$ we can finally conclude that

$$\forall i \in I, \alpha \in \textsc{Act} \cdot \; \textit{if}\; s \overset{\alpha}{\Rightarrow} s' \text{ s.t. } \eta_i(\alpha) = \sigma \text{ then } (s', \varphi_i\sigma) \in \mathcal{R}$$

as required.

Case $\varphi = \mathsf{max}\, X.\varphi$. In order to prove this case, we must confirm that $(s, \mathsf{max}\, X.\varphi) \in \mathcal{R}$ by showing that $(s, \varphi\{\mathsf{max}\, X.\varphi/X\}) \in \mathcal{R}$ as well. Hence, we assume that

$$s \vDash \langle\!\langle \mathsf{max}\, X.\varphi \rangle\!\rangle_5 \tag{120}$$

and consider the following two subcases for $\langle\!\langle \mathsf{max}\, X.\varphi \rangle\!\rangle_5$.

– when $X \in \mathbf{fv}(\varphi)$: Since $X \in \mathbf{fv}(\varphi)$, from (120) and the definition of $\langle\!\langle - \rangle\!\rangle_5$ we have that $s \vDash \mathsf{max}\, X.\langle\!\langle \varphi \rangle\!\rangle_5$ and so by the definition of $\vDash$ we can deduce that

$$s \vDash \langle\!\langle \varphi \rangle\!\rangle_5\{\mathsf{max}\, X.\langle\!\langle \varphi \rangle\!\rangle_5/X\}. \tag{121}$$

Since $X \in \mathbf{fv}(\varphi)$ and by Lemma 15 we have that $X \in \mathbf{fv}(\langle\!\langle \varphi \rangle\!\rangle_5)$, and so by Lemma 16, from (121) we deduce that

$$s \vDash \langle\!\langle \varphi\{\mathsf{max}\, X.\varphi/X\} \rangle\!\rangle_5. \tag{122}$$

Hence, by (122) and the definition of $\mathcal{R}$ we deduce that

$$(s, \varphi\{\mathsf{max}\, X.\varphi/X\}) \in \mathcal{R}$$

as required.

– $X \notin \mathbf{fv}(\varphi)$: Since $X \notin \mathbf{fv}(\varphi)$, from (120) and the definition of $\langle\!\langle - \rangle\!\rangle_5$ we have that

$$s \vDash \langle\!\langle \varphi \rangle\!\rangle_5. \tag{123}$$

and so since $X \notin \mathbf{fv}(\varphi)$ from (123) we infer that $\langle\!\langle \varphi \rangle\!\rangle_5$ is equivalent to $\langle\!\langle \varphi\{\mathsf{max}\, X.\varphi/X\} \rangle\!\rangle_5$ since X is unused in φ which means that from (123) we can deduce that

$$s \vDash \langle\!\langle \varphi\{\mathsf{max}\, X.\varphi/X\} \rangle\!\rangle_5. \tag{124}$$

Hence, from (124) and the definition of $\mathcal{R}$ we conclude that

$$(s, \varphi\{\mathsf{max}\, X.\varphi/X\}) \in \mathcal{R}$$

as required, and so we are done.

□

Proof for (b) Let $\mathcal{R} \overset{\text{def}}{=} \{\ (s, \langle\!\langle \varphi \rangle\!\rangle_5) \mid s \vDash \varphi\ \}$, once again we must prove that $\mathcal{R}$ is a satisfaction relation and conduct this proof by case analysis on φ.

Cases $\varphi \in \{\mathsf{ff}, X\}$. These cases do not apply since the assumption that $s \vDash \varphi$ does not hold when $\varphi \in \{\mathsf{ff}, X\}$.

Case $\varphi = \mathsf{tt}$ This cases holds trivially since $\langle\!\langle \mathsf{tt} \rangle\!\rangle_5$=$\mathsf{tt}$ and since any process satisfies tt which allows us to affirm that $(s, \langle\!\langle \mathsf{tt} \rangle\!\rangle_5) \in \mathcal{R}$.

Case $\varphi = \bigwedge_{i \in I} [\eta_i]\varphi_i$. In order to prove this case, we must confirm that $(s, \langle\!\langle \bigwedge_{i \in I} [\eta_i]\varphi_i \rangle\!\rangle_5) \in \mathcal{R}$. Since $\langle\!\langle \bigwedge_{i \in I} [\eta_i]\varphi_i \rangle\!\rangle_5 = \bigwedge_{i \in I} [\eta_i]\langle\!\langle \varphi_i \rangle\!\rangle_5$, we instead confirm that $(s, \bigwedge_{i \in I} [\eta_i]\langle\!\langle \varphi_i \rangle\!\rangle_5) \in \mathcal{R}$ by showing that for every α and $i \in I$, if $s \overset{\alpha}{\Rightarrow} s'$ *s.t.* $\eta_i(\alpha) = \sigma$ then $(s', \langle\!\langle \varphi_i\sigma \rangle\!\rangle_5) \in \mathcal{R}$. Hence, we start by assuming that $s \vDash \bigwedge_{i \in I} [\eta_i]\varphi_i$ and so by the definition of $\vDash$ we have that

$$\forall i \in I, \alpha \in \textsc{Act} \cdot\ \textit{if}\ s \overset{\alpha}{\Rightarrow} s'\ \textit{s.t.}\ \eta_i(\alpha) = \sigma \text{ then } s' \vDash \varphi_i\sigma \tag{125}$$

and so by (125) and the definition of $\mathcal{R}$ we conclude that

$$\forall i \in I, \alpha \in \textsc{Act} \cdot\ \textit{if}\ s \overset{\alpha}{\Rightarrow} s'\ \textit{s.t.}\ \eta_i(\alpha) = \sigma \text{ then } (s', \langle\!\langle \varphi_i\sigma \rangle\!\rangle_5) \in \mathcal{R}$$

as required.

Case $\varphi = \mathsf{max}\, X.\varphi$. To prove this case, we must confirm that $(s, \langle\!\langle \mathsf{max}\, X.\varphi \rangle\!\rangle_5) \in \mathcal{R}$ and so we start by assuming that $s \vDash \mathsf{max}\, X.\varphi$ from which by the definitions of $\vDash$ and $\mathcal{R}$ we deduce that

$$(s, \langle\!\langle \varphi\{\mathsf{max}\, X.\varphi/X\} \rangle\!\rangle_5) \in \mathcal{R}. \tag{126}$$

We now consider two subcases for $\langle\!\langle \mathsf{max}\, X.\varphi \rangle\!\rangle_5$.

– $\langle\!\langle \mathsf{max}\, X.\varphi \rangle\!\rangle_5 = \mathsf{max}\, X.\langle\!\langle \varphi \rangle\!\rangle_5$ when $X \in \mathbf{fv}(\varphi)$: To confirm that $(s, \langle\!\langle \mathsf{max}\, X.\varphi \rangle\!\rangle_5) \in \mathcal{R}$, in this case we must affirm that $(s, \mathsf{max}\, X.\langle\!\langle \varphi \rangle\!\rangle_5) \in \mathcal{R}$ by showing that $(s, \langle\!\langle \varphi \rangle\!\rangle_5\{\mathsf{max}\, X.\langle\!\langle \varphi \rangle\!\rangle_5/X\}) \in \mathcal{R}$ as well. Hence, since we assume that $X \in \mathbf{fv}(\varphi)$, by Lemma 15 we deduce that $X \in \mathbf{fv}(\langle\!\langle \varphi \rangle\!\rangle_5)$ and so by Lemma 16 and from (126) we can conclude that

$$(s, \langle\!\langle \varphi \rangle\!\rangle_5\{\mathsf{max}\, X.\langle\!\langle \varphi \rangle\!\rangle_5/X\}) \in \mathcal{R}$$

as required.

- $\langle\!\langle \mathsf{max}\, X.\varphi \rangle\!\rangle_5 = \langle\!\langle \varphi \rangle\!\rangle_5$ when $X \notin \mathbf{fv}(\varphi)$: Hence, to confirm that $(s, \langle\!\langle \mathsf{max}\, X.\varphi \rangle\!\rangle_5) \in \mathcal{R}$, we must now affirm that $(s, \langle\!\langle \varphi \rangle\!\rangle_5) \in \mathcal{R}$. Since we now assume that $X \notin \mathbf{fv}(\varphi)$, we know that $\varphi\{\mathsf{max}\, X.\varphi/X\} \equiv \varphi$ and so from (126) we confirm that $(s, \langle\!\langle \varphi \rangle\!\rangle_5) \in \mathcal{R}$ as required.

□

Proof for Lemma 15 We conduct this proof by structural induction on φ.

Cases $\varphi \in \{\mathsf{ff}, \mathsf{tt}\}$. These cases do not apply since $X \notin \mathbf{fv}(\varphi)$ when $\varphi \in \{\mathsf{ff}, \mathsf{tt}\}$.

Case $\varphi = \bigwedge_{i\in I} [\eta_i]\varphi_i$. We first assume that $X \in \mathbf{fv}(\bigwedge_{i\in I} [\eta_i]\varphi_i)$ and so by the definition of $\mathbf{fv}(-)$ we know that for every $i \in I$, $X \in \mathbf{fv}(\varphi_i)$ and so by applying the inductive hypothesis for every $i \in I$ we infer that $X \in \mathbf{fv}(\langle\!\langle \varphi_i \rangle\!\rangle_5)$. With this result and by the definitions of $\mathbf{fv}(-)$ and $\langle\!\langle - \rangle\!\rangle_5$, we thus conclude that $X \in \mathbf{fv}(\langle\!\langle \bigwedge_{i\in I} [\eta_i]\varphi_i \rangle\!\rangle_5)$ as required, and so we are done.

Case $\varphi = Y$. We start by assuming that $X \in \mathbf{fv}(\varphi)$ and consider the following cases:

- when $Y = X$: This case holds trivially since $\langle\!\langle Y \rangle\!\rangle_5 = Y = X$ and so since $X \in \mathbf{fv}(X)$ we can infer that $X \in \mathbf{fv}(\langle\!\langle Y \rangle\!\rangle_5)$ as required.
- when $Y \neq X$: This case does not apply since $X \notin \mathbf{fv}(Y)$ when $Y \neq X$.

Case $\varphi = \mathsf{max}\, Y.\varphi$. We assume that

$$X \in \mathbf{fv}(\mathsf{max}\, Y.\varphi) \tag{127}$$

and consider the following cases:

- when $Y = X$: This case does not apply since $X \notin \mathbf{fv}(\mathsf{max}\, Y.\varphi)$ when $Y = X$.
- when $Y \neq X$: From (127) and by the definition of $\mathbf{fv}(-)$ we can deduce that

$$X \in \mathbf{fv}(\varphi) \tag{128}$$

  and so by the inductive hypothesis we have that $X \in \mathbf{fv}(\langle\!\langle \varphi \rangle\!\rangle_5)$ from which we can deduce that

$$X \in \mathbf{fv}(\mathsf{max}\, Y.\langle\!\langle \varphi \rangle\!\rangle_5). \tag{129}$$

  Finally, since $Y \in \mathbf{fv}(\langle\!\langle \varphi \rangle\!\rangle_5)$ from (129) and the definition of $\langle\!\langle - \rangle\!\rangle_5$ we can conclude that

$$X \in \mathbf{fv}(\langle\!\langle \mathsf{max}\, Y.\varphi \rangle\!\rangle_5) \tag{130}$$

  as required, and so we are done.

□

Proof for Lemma 16 We conduct this proof by structural induction on φ.

Cases $\varphi \in \{\mathsf{ff}, \mathsf{tt}\}$. These cases do not apply since $X \notin \mathbf{fv}(\varphi)$ when $\varphi \in \{\mathsf{ff}, \mathsf{tt}\}$.

Case $\varphi = \bigwedge_{i\in I} [\eta_i]\varphi_i$ We first assume that

$$X \in \mathbf{fv}(\textstyle\bigwedge_{i\in I} [\eta_i]\varphi_i) \tag{131}$$

$$X \in \mathbf{fv}(\langle\!\langle \psi \rangle\!\rangle_5) \tag{132}$$

so that by (131) and the definition of $\mathbf{fv}(-)$ we know that

$$\forall i \in I \cdot X \in \mathbf{fv}(\varphi_i). \tag{133}$$

Hence, by (132) we can apply the inductive hypothesis for every $i \in I$ and infer that

$$\forall i \in I \cdot \langle\!\langle \varphi_i\{\mathsf{max}\,X.\psi/X\}\rangle\!\rangle_5 = \langle\!\langle \varphi_i\rangle\!\rangle_5\{\mathsf{max}\,X.\langle\!\langle\psi\rangle\!\rangle_5/X\} \tag{134}$$

and by (134) and the definition of $\langle\!\langle - \rangle\!\rangle_5$ we thus conclude that

$$\langle\!\langle \textstyle\bigwedge_{i\in I}[\eta_i]\varphi_i\varphi_i\{\mathsf{max}\,X.\psi/X\}\rangle\!\rangle_5 = \langle\!\langle \textstyle\bigwedge_{i\in I}[\eta_i]\varphi_i\rangle\!\rangle_5\{\mathsf{max}\,X.\langle\!\langle\psi\rangle\!\rangle_5/X\}$$

as required.

Case $\varphi = Y$. We start by assuming that

$$X \in \mathbf{fv}(Y) \tag{135}$$

$$X \in \mathbf{fv}(\langle\!\langle\psi\rangle\!\rangle_5) \tag{136}$$

and consider the following cases:

- when $Y \neq X$: This case does not apply since (135) does not hold when $Y \neq X$.
- when $Y = X$: Since $Y = X$ we can thus unfold $Y\{\mathsf{max}\,X.\psi/X\}$ into $\mathsf{max}\,X.\psi$ such that we have that

$$\langle\!\langle Y\{\mathsf{max}\,X.\psi/X\}\rangle\!\rangle_5 = \langle\!\langle X\{\mathsf{max}\,X.\psi/X\}\rangle\!\rangle_5 = \langle\!\langle \mathsf{max}\,X.\psi\rangle\!\rangle_5. \tag{137}$$

  Since $\langle\!\langle Y\rangle\!\rangle_5 = Y$ and $Y = X$ we can deduce that

$$\langle\!\langle Y\rangle\!\rangle_5\{\mathsf{max}\,X.\langle\!\langle\psi\rangle\!\rangle_5/X\} = X\{\mathsf{max}\,X.\langle\!\langle\psi\rangle\!\rangle_5/X\} = \mathsf{max}\,X.\langle\!\langle\psi\rangle\!\rangle_5. \tag{138}$$

  Since by (136) and the definition of $\langle\!\langle - \rangle\!\rangle_5$ we know that $\langle\!\langle \mathsf{max}\,X.\psi\rangle\!\rangle_5 = \mathsf{max}\,X.\langle\!\langle\psi\rangle\!\rangle_5$ and so from (137) and (138) we can conclude that

$$\langle\!\langle Y\{\mathsf{max}\,X.\psi/X\}\rangle\!\rangle_5 = \langle\!\langle Y\rangle\!\rangle_5\{\mathsf{max}\,X.\langle\!\langle\psi\rangle\!\rangle_5/X\}.$$

  as required.

Case $\varphi = \mathsf{max}\,Y.\varphi$. We assume that

$$X \in \mathbf{fv}(\mathsf{max}\,Y.\varphi) \tag{139}$$

$$X \in \mathbf{fv}(\langle\!\langle\psi\rangle\!\rangle_5) \tag{140}$$

and consider the following cases:

- when $Y = X$: This case does not apply since $X \notin \mathbf{fv}(\mathsf{max}\,Y.\varphi)$ when $Y = X$.
- when $Y \neq X$: From (139) and by the definition of $\mathbf{fv}(-)$ we can deduce that $X \in \mathbf{fv}(\varphi)$ and so by (140) and the inductive hypothesis we have that

$$\langle\!\langle\varphi\rangle\!\rangle_5\{\mathsf{max}\,X.\langle\!\langle\psi\rangle\!\rangle_5/X\} = \langle\!\langle\varphi\{\mathsf{max}\,X.\psi/X\}\rangle\!\rangle_5. \tag{141}$$

  Hence, by applying the definition of $\langle\!\langle - \rangle\!\rangle_5$ on both sides of equation (141) we get that

$$\langle\!\langle \mathsf{max}\,Y.\varphi\{\mathsf{max}\,X.\psi/X\}\rangle\!\rangle_5 = \langle\!\langle \mathsf{max}\,Y.\varphi\rangle\!\rangle_5\{\mathsf{max}\,X.\langle\!\langle\psi\rangle\!\rangle_5/X\}. \tag{142}$$

  as required, and so we are done.

□

A.2 Proving Lemma 8.

$$\text{if } \mathsf{traverse}(Eq, \{0\}, \mathsf{partition}, \emptyset){=}\zeta \text{ then } \zeta \text{ is a } \textit{well-formed} \text{ map for } Eq.$$

To prove Lemma 8, we rely on Lemma 17.

Lemma 17 *For every set of indices I, ζ map, and equation sets Eq and Eq', if $Eq' \subseteq Eq$ and $\mathsf{traverse}(Eq', I, \mathsf{partition}, \zeta){=}\zeta'$ and ζ is a* well-formed *map for $Eq_{//\mathbf{dom}(\zeta)}$ then ζ' is a* well-formed *map for Eq.*

We provide the proof for this lemma at the end of this section.

Proof for Lemma 8 Assume that

$$\mathsf{traverse}(Eq, \{0\}, \mathsf{partition}, \emptyset){=}\zeta \tag{143}$$

and since by the definition of $Eq_{//I}$ we know that $Eq_{//\mathbf{dom}(\emptyset)} = \emptyset$ by the definition of a *well-formed* map we infer that

$$\emptyset \text{ is a } \textit{Well-formed} \text{ map for } Eq_{//\mathbf{dom}(\emptyset)} \tag{144}$$

and hence by (143), (144) and Lemma 17 we can conclude that

$$\zeta \text{ is a } well - formed \text{ map for } Eq$$

as required.

Proof for Lemma17 We proceed by induction on the structure of Eq'.

Case $Eq' = \emptyset$ Initially we assume that $\emptyset \subseteq Eq$ and that

$$\mathsf{traverse}(\emptyset, I, \mathsf{partition}, \zeta){=}\zeta' \tag{145}$$

$$\zeta \text{ is a } well - formed \text{ map for } Eq_{//\mathbf{dom}(\zeta)}. \tag{146}$$

Since $Eq'{=}\emptyset$, by (145) and the definition of $\mathsf{traverse}$ we have that $\zeta = \zeta'$ and so from (146) we can deduce that

$$\zeta' \text{ is a } well - formed \text{ map for } Eq_{//\mathbf{dom}(\zeta')}. \tag{147}$$

From (145) and the definition of $\mathsf{traverse}$, we know that the traversal starts from the full equation set, i.e., $Eq' = Eq$, using an empty ζ map. With every recursive application of $\mathsf{traverse}$, the equation set Eq' becomes smaller since when $\mathsf{traverse}$ recurses it does so *wrt.* Eq'', i.e., a smaller version of the current Eq' which is computed via $Eq''{=}Eq' \setminus Eq'_{//I}$. By contrast, with every recursive application of $\mathsf{traverse}$, the ζ accumulator becomes larger as it is updated with new mappings for each index specified by the set of indices I, i.e., with the indices of the equations that are removed from Eq' when creating Eq''. Hence, when the $\mathsf{traverse}$ function is recursively applied *wrt.* some $Eq'''{=}\emptyset$, it means that all the equations specified in Eq have been analysed by the traversal and their indices were thus added as maps in the resultant ζ'. Hence, we can deduce that $Eq_{//\mathbf{dom}(\zeta')} = Eq$ so that from (147) we can conclude that

$$\zeta' \text{ is a } \textit{well-formed} \text{ map for } Eq$$

as required.

Case $Eq' \neq \emptyset$. Now, assume that

$$\mathsf{traverse}(Eq', I, \mathsf{partition}, \zeta){=}\zeta' \tag{148}$$

$$\zeta \text{ is a } \textit{well-formed} \text{ map for } Eq_{/\!/\mathbf{dom}(\zeta)} \tag{149}$$

$$Eq' \subseteq Eq \tag{150}$$

and consider the following two subcases for the set of indices I.

- $\boldsymbol{I = \emptyset}$ **:** Since $I{=}\emptyset$, by (148) and the definition of $\mathsf{traverse}$ we know that $\zeta = \zeta'$ and so from (149) we can deduce that

  $$\zeta' \text{ is a } \textit{well-formed} \text{ map for } Eq_{/\!/\mathbf{dom}(\zeta')}. \tag{151}$$

  Since $I{=}\emptyset$, this means that the traversal has reached a point where no more children can be computed, which means that all the *relevant equations* (i.e., those reachable from the principle variable) have been analysed. This means that any other equation in Eq (that is not in $Eq_{/\!/\mathbf{dom}(\zeta')}$, if any) is *redundant* and *irrelevant*. Hence, since from (151) we know that ζ' is a *well-formed* map for the *relevant subset* of equations in Eq, i.e., $Eq_{/\!/\mathbf{dom}(\zeta')}$, then it is also *well-formed* for the full blown subset of equations Eq (i.e., including any unreachable, redundant equations). Therefore, we can conclude that

  $$\zeta' \text{ is a } \textit{well-formed} \text{ map for } Eq$$

  as required.
- $\boldsymbol{I{\neq}\emptyset}$ **:** By the definition of $\mathsf{traverse}$ and from (148) we can infer that

  $$\zeta'' = \mathsf{partition}(Eq', I, \zeta) \tag{152}$$

  $$Eq'' = Eq' \setminus Eq'_{/\!/I} \tag{153}$$

  $$I' = \bigcup_{j \in I} \mathsf{child}(Eq', j) \tag{154}$$

  $$\mathsf{traverse}(Eq'', I', \mathsf{partition}, \zeta''){=}\zeta' \tag{155}$$

  By (149) and the definition of a *well-formed* map, we know that ζ provides a set of mappings which allow for:

  - renaming the *data variables* of each *pattern equivalent sibling necessity*, defined in $Eq_{/\!/\mathbf{dom}(\zeta)}$, to the *same* set of fresh variables. (156)
  - renaming any *reference* to a data variable that is bound by a *renamed parent necessity* defined in $Eq_{/\!/\mathbf{dom}(\zeta)}$ (157)

  and by the definition of $\mathsf{partition}$ from (152) we have that

$$\zeta'' = \zeta \mathbin{\dot{\cup}} \left\{ \begin{array}{l|l} j \mapsto \zeta(i) \mathbin{\dot{\cup}} \{f^1/d^1, f^2/d^2\} & \forall i,l \in I \cdot Eq(i) {=} \bigwedge_{j \in I'} [(d^1)\$(d^2), c_j]X_j \wedge \varphi' \\ & \text{and } Eq(l) {=} \bigwedge_{k \in I''} [(e^1)\$(e^2), c_k]X_k \wedge \varphi'' \text{ s.t.} \\ k \mapsto \zeta(l) \mathbin{\dot{\cup}} \{f^1/e^1, f^2/e^2\} & \text{if } (d^1)\$(d^2), c_j \text{ is } \mathit{patternequivalent} \text{ to} \\ & (e^1)\$(e^2), c_k, \text{ then we assign the } \mathit{same} \\ & \text{fresh variables } f^1 \text{ and } f^2. \end{array} \right\} \tag{158}$$

From (158) we know that ζ'' includes a mapping for each sibling branch that defines a pattern equivalent *SA*. The added mappings map the child indices of the conjunction branches (i.e., $j, k{\in}I'$ since from (154) we know that I'' and I''' are subsets of I') that are defined by the equations identified by the parent indices (i.e., $i{\in}I$) specified in I, to a substitution environment. This mapped substitution renames the *resp.* variable names of these conjunct pattern equivalent sibling necessities, to the same fresh set of variable names, thereby making the equivalent sibling patterns, syntactically equal. Hence, from (156) we can deduce that ζ'' provides a set of mappings which allow for

- renaming the $\mathit{data\ variables}$ of each $\mathit{pattern\ equivalent\ sibling\ necessity}$, defined in $Eq_{//\mathbf{dom}(\zeta)\cup I'}$, to the same set of fresh variables. (159)

Similarly, from (158) we also know that the mappings in ζ'' include the substitutions performed upon the parent necessities. This means that in each mapping $j \mapsto \sigma_j$, the mapped substitution environment σ_j also includes $\zeta(i)$ where $i \in I$ is the parent index of $j \in I'$. Hence, from (157) we can deduce that the mappings provided by ζ'' also allow for

- renaming any $\mathit{reference}$ to a data variable that is bound by a $\mathit{renamed\ parent\ necessity}$ defined in $Eq_{//\mathbf{dom}(\zeta)\cup I'}$. (160)

Hence, by (159), (160) and the definition of a *well-formed* map we can infer that

$$\zeta'' \text{ is a } \mathit{well-formed} \text{ map for } Eq_{//\mathbf{dom}(\zeta)\cup I'}. \tag{161}$$

From (158) we know that ζ'' includes a mapping for each child branch, identified by $j \in I''$ and $k \in I'''$ (where I'' and I''' are both subsets of I'), that is defined in the equation identified by index $i \in I$ and which defines a pattern equivalent necessity. Hence, we know that the domain of ζ'' is an extension of the domain of ζ which additionally contains the child indices defined in I', such that we can deduce that $\mathbf{dom}(\zeta'') = \mathbf{dom}(\zeta) \cup I'$. Hence, from (161) we can infer that

$$\zeta'' \text{ is a } \textit{well-formed} \text{ map for } Eq_{//\mathbf{dom}(\zeta'')}. \tag{162}$$

Finally, since from (153) and (150) we have that $Eq'' \subseteq Eq$, by (155), (162) and the inductive hypothesis we can conclude that

$$\zeta' \text{ is a } \textit{well-formed} \text{ map for } Eq$$

as required, and so we are done.

□

A.3 Proving Lemma 9.

For every ζ map, and equation set *Eq*, if ζ is a *well-formed* map for *Eq* then $\mathsf{uni}(Eq, \zeta)\equiv Eq$ and every equation $(X_k{=}\psi_k) \in \mathsf{uni}(Eq, \zeta)$ is *Uniform.*

Proof for Lemma 9 We conduct this proof by induction on the structure of *Eq*.

Case $Eq = \emptyset$. This case holds trivially since $Eq = \emptyset = \mathsf{uni}(\emptyset, \zeta)$.

Case $Eq = \left\{X_i{=}\bigwedge_{j\in I}[\eta_j]\varphi_j \wedge \varphi\right\} \dot{\cup} Eq'$. We start by assuming that

$$\zeta \text{ is a } well-formed \text{ map for } Eq \tag{163}$$

and so by (163) and the definition of a *well-formed* map we know that ζ provides a set of mappings which allow for

- renaming the *data variables* of each *pattern equivalent sibling necessity*, defined in *Eq*, to the *same* set of fresh variables. (164)
- renaming any *reference* to a data variable that is bound by a *renamed parent necessity* defined in *Eq*. (165)

By applying the uni function on *Eq* and ζ we obtain

$$\begin{aligned}&\mathsf{uni}(\left\{X_i{=}\bigwedge_{j\in I}[\eta_j]\varphi_j \wedge \varphi\right\} \dot{\cup} Eq', \zeta)\\ &= \left\{X_i{=}\bigwedge_{j\in I}[\eta_j\underline{\zeta(j)}]\varphi_j \wedge \varphi\right\} \dot{\cup} \mathsf{uni}(Eq', \zeta)\end{aligned}. \tag{166}$$

Now if we assume that η_j defines an arbitrary pattern $(d^1)\$(d^2)$ (where d^1 and d^2 are newly bound variables), along with some condition $c_j[d^1, d^2, e^m_{<i}]$ whose evaluation depends on d^1, d^2 and the values of m variables $e^m_{<i}$ that are bound by parent modal necessities. Hence, from (164) we can deduce that mapping $\zeta(j)$ in (166) produces a substitution environment which renames the data bindings d^1 and d^2 to some fresh variables f^1 and f^2, which are the *same* for all the other conjunct sibling necessities that are pattern equivalent to η_j. From (165) we can also deduce that any reference being made to some variable $e^m_{<i}$ will also be renamed accordingly by $\zeta(j)$. Hence, by the definition of a *uniform equation*, we can deduce that

$$\text{equation } X_i{=}\bigwedge_{j\in I}[\eta_j]\varphi_j \wedge \varphi \text{ is } uniform. \tag{167}$$

Moreover, from (164) and (165) we can deduce that equation $X_i{=}\bigwedge_{j\in I}[\eta_j]\varphi_j \wedge \varphi$ is *semantically equivalent* to the equation reconstructed by the uni function in (166), i.e., $X_i{=}\bigwedge_{j\in I}[\eta_j\zeta(j)]\varphi_j \wedge \varphi$. This holds since when the substitution environment, returned by $\zeta(j)$, is applied to the equated formula, it only substitutes the variable names in η_j and so if η_j has an arbitrary form $(d^1)\$(d^2), c_j[d^1, d^2, e^m_{<i}]$ this will become $(f^1)\$(f^2), c_j[f^1, f^2, f^m_{<i}]$.

Notice that the new pattern $(f^1)\$(f^2)$ is *equivalent* to the original one $(d^1)\$(d^2)$ since it only varies by the name of the data variables it binds. The new condition $c_j[f^1, f^2, f^m_{<i}]$ is also equivalent to $c_j[d^1, d^2, e^m_{<i}]$ since by (165) we know that $\zeta(j)$ (where $\zeta(j)$ also contains

$\zeta(i)$ where i is the parent of j) renames d^1 and d^2 to f^1 and f^2 and $e^m_{<i}$ to the variable names, $f^m_{<i}$, bound by the renamed parent necessities. This preserves the semantics of the equation by keeping it closed *wrt.* data variables. Hence, we can deduce

$$\begin{aligned}
& X_i = \bigwedge_{j\in I} [\eta_j]\varphi_j \wedge \varphi \\
\equiv\ & X_i = \bigwedge_{j\in I} [(d^1)\$(d^2), c_j[d^1, d^2, e^m_{<i}]]\varphi_j \wedge \varphi \\
\equiv\ & X_i = \bigwedge_{j\in I} [(f^1)\$(f^2), c_j[f^1, f^2, f^m_{<i}]]\varphi_j \wedge \varphi \\
\equiv\ & X_i = \bigwedge_{j\in I} [(d^1)\$(d^2), c_j[d^1, d^2, e^m_{<i}]\zeta(j)]\varphi_j \wedge \varphi \\
\equiv\ & X_i = \bigwedge_{j\in I} [\eta_j\zeta(j)]\varphi_j \wedge \varphi.
\end{aligned} \tag{168}$$

Now since $Eq' \subset Eq$ from (163) we can infer that ζ is also a *well-formed* map for Eq' which allows us to apply the inductive hypothesis and deduce that

$$\text{every equation } (X_k{=}\psi_k) \in \mathsf{uni}(Eq', \zeta) \text{ is } \textit{uniform}, \text{ and that} \tag{169}$$

$$\mathsf{uni}(Eq', \zeta) \equiv Eq'. \tag{170}$$

Hence, by (166), (169) and (167) we can conclude that

$$\text{every equation } (X_k{=}\psi_k) \in \mathsf{uni}(Eq, \zeta) \text{ is } \textit{uniform} \tag{171}$$

and by (166), (170) and (168) we can conclude

$$\Big\{X_i = \bigwedge_{j\in I} [\eta_j]\varphi_j \wedge \varphi\Big\} \overset{+}{\cup} Eq' \equiv \Big\{X_i = \bigwedge_{j\in I} [\eta_j \underline{\zeta(j)}]\varphi_j \wedge \varphi\Big\} \overset{+}{\cup} \mathsf{uni}(Eq', \zeta) \tag{172}$$

as required, and so this case is done by (171) and (172). □

A.4 Proving Lemma 11.

For every eqn. $(X_j{=}\varphi_j) \in Eq$, if $X_j{=}\varphi_j$ is *uniform* then $Eq \equiv \mathsf{traverse}(Eq, \{0\}, \mathsf{cond\_comb}, \emptyset)$ and every eqn. $(X_k{=}\psi_k) \in \mathsf{traverse}(Eq, \{0\}, \mathsf{cond\_comb}, \emptyset)$ is *equi-disjoint*.

The proof for Lemma 11 depends on Lemma 18. This new lemma states that one can obtain an *equi-disjoint* equation set, ω', that is *semantically equivalent* to the original equation set Eq, by conducting a traversal upon a *uniform* subset of Eq (i.e., Eq'). This traversal is conducted *wrt.* an *equi-disjoint* accumulator equation set ω, where ω must be *semantically equivalent* to a subset of Eq that is restricted to the indices associated to the logical variables specified by the domain of ω, i.e., $\omega \equiv Eq_{//\mathsf{dom}_{\mathsf{ind}}(\omega)}$, where $\mathsf{dom}_{\mathsf{ind}}(\omega) \overset{\text{def}}{=} \{\, i \mid X_i \in \mathbf{dom}(\omega)\,\}$.

Lemma 18 *For every index set I, equi-disjoint set ω and equation sets Eq and Eq', if $Eq' \subseteq Eq$ and $\mathsf{traverse}(Eq', I, \mathsf{cond\_comb}, \omega){=}\omega'$ and $Eq_{//\mathsf{dom}_{ind}(\omega)} \equiv \omega$ and every equation $(X_j{=}\varphi_j) \in Eq'$ is* uniform *and every equation $(X_k{=}\psi_k) \in \omega$ is* equi-disjoint *then every equation $(X_k{=}\psi_k) \in \omega'$ is* equi-disjoint *and $Eq \equiv \omega'$.*

We provide the proof for this lemma at the end of this section.

Proof for Lemma 11 Assume that

$$\forall (X_j{=}\varphi_j) \in Eq \cdot \text{ equation } X_j{=}\varphi_j \text{ is } \textit{uniform}. \tag{173}$$

By applying the $\mathsf{traverse}$ function on Eq starting from $I{=}\{0\}$ and $\omega{=}\emptyset$, we know that

$$\mathsf{traverse}(Eq, \{0\}, \mathsf{cond\_comb}, \omega) = \omega' \tag{174}$$

and so since $\omega{=}\emptyset$, by the definition of $Eq_{//I}$ we have that $Eq_{//\mathbf{dom}(\emptyset)} = \emptyset = \omega$ which means that we can also deduce that every equation $(X_k{=}\psi_k) \in \omega$ is *equi-disjoint*. With this new information along with (173) and (174), we can use Lemma 18 to infer that

$$Eq \equiv \omega' \text{ and that every equation } (X_k{=}\psi_k) \in \omega' \text{ is } \textit{equi-disjoint}$$

as required, and so we are done. □

Proof for Lemma 18 We proceed by induction on the structure of I.

Case $I = \emptyset$ Let's start by assuming that

$$Eq' \subseteq Eq, \tag{175}$$

$$\mathsf{traverse}(Eq', \emptyset, \mathsf{cond\_comb}, \omega){=}\omega', \tag{176}$$

$$Eq_{//\mathsf{dom}_{\mathsf{ind}}(\omega)}{\equiv}\omega, \tag{177}$$

$$\text{every equation } (X_j{=}\varphi_j) \in Eq' \text{ is } \textit{uniform}, \text{ and that} \tag{178}$$

$$\text{every equation } (X_k{=}\psi_k) \in \omega \text{ is } \textit{equi-disjoint}. \tag{179}$$

By (176) and the definition of $\mathsf{traverse}$, we know that $\omega = \omega'$ and so from (177) and (179) we can deduce that

$$\text{every equation } (X_k{=}\psi_k) \in \omega' \text{ is } \textit{equi-disjoint} \tag{180}$$

$$Eq_{//\mathsf{dom}_{\mathsf{ind}}(\omega')}{\equiv}\omega'. \tag{181}$$

Since $I{=}\emptyset$, by the definition of $\mathsf{traverse}$ and (176) we know the traversal has reached a point where no more children can be computed, which means that all the *relevant equations* (i.e., those reachable from the principle variable) have been analysed. This implies that any other equation in Eq (if any) is *redundant* and *irrelevant*. Hence, since from (181) we know that the equations in ω' are *equivalent to the relevant subset of equations in* Eq, i.e., $Eq_{//\mathsf{dom}_{\mathsf{ind}}(\omega')}$, and hence, we can conclude that

$$\omega' \equiv Eq \tag{182}$$

as required, and so this case is done by (180) and (182).

Case $I \neq \emptyset$. Let us now assume that

$$Eq' \subseteq Eq \tag{183}$$

$$\mathsf{traverse}(Eq', I, \mathsf{cond\_comb}, \omega){=}\omega' \tag{184}$$

$$Eq_{//\mathsf{dom}_{\mathsf{ind}}(\omega)}{\equiv}\omega \tag{185}$$

$$\text{every equation } (X_j{=}\varphi_j) \in Eq' \text{ is } \textit{uniform} \tag{186}$$

$$\text{every equation } (X_k{=}\psi_k) \in \omega \text{ is } \textit{equi-disjoint} \tag{187}$$

and let's proceed by case analysis on Eq'.

– $Eq' = \emptyset$: Since $Eq' = \emptyset$, by (184) and the definition of traverse we know that $\omega = \omega'$ and so from (185) and (187) we can deduce that

$$Eq_{//\mathsf{dom}_{\mathsf{ind}}(\omega')} {\equiv} \omega', \text{ and that} \tag{188}$$

$$\text{every equation } (X_k{=}\psi_k) \in \omega' \text{ is } \textit{equi-disjoint.} \tag{189}$$

By (184) and the definition of traverse, we know that the traversal starts from the full equation set, i.e., $Eq' = Eq$, using an empty accumulator, i.e., $\omega{=}\emptyset$, that would eventually contain the resultant equi-disjoint equation set. Every recursive application of the traverse function is then performed *wrt.*: a *smaller* version Eq, i.e., $Eq'{=}Eq\backslash Eq_{//I}$, and a *larger* accumulator ω' containing the reformulated, equi-disjoint equations whose indices are defined in I (and which where removed from Eq'). Hence, when Eq' becomes $\emptyset$ it means that $\mathsf{dom}_{\mathsf{ind}}(\omega') = \mathsf{dom}_{\mathsf{ind}}(Eq)$ and so by the definition of $Eq_{//I}$ we can deduce that $Eq_{//\mathsf{dom}_{\mathsf{ind}}(\omega)} = Eq_{//\mathsf{dom}_{\mathsf{ind}}(Eq)} = Eq$ which means that from (188) we can conclude that

$$Eq \equiv \omega' \tag{190}$$

as required, and so this case holds by (189) and (190).

– $Eq' \neq \emptyset$: By (184) and the definition of traverse we have that

$$\mathsf{cond\_comb}(Eq', I, \omega){=}\omega'' \tag{191}$$

$$Eq'' = Eq' \setminus Eq'_{//I} \tag{192}$$

$$I' = \bigcup_{l \in I} \mathsf{child}(Eq, l) \tag{193}$$

$$\mathsf{traverse}(Eq'', I', \mathsf{cond\_comb}, \omega'') = \omega', \tag{194}$$

By applying definition of cond_comb to (191), we deduce that

$$\omega'' = \omega \mathbin{\dot{\cup}} \left\{ X_i{=}\bigwedge_{c_k \in \mathbb{C}(j,I')} [p, c_k]X_j {\wedge} \varphi ({=}\ \psi_i) \;\middle|\; \begin{array}{l} (X_i{=}\bigwedge_{j \in I''} [p, c_j]X_j {\wedge} \varphi){\in}Eq_{//I} \\ \text{and } I'{=}\bigcup_{l \in I} \mathsf{child}(Eq, l) \\ \textit{such that } I'' \subseteq I' \end{array} \right\}. \tag{195}$$

Now from (195) and the definition of $\mathbb{C}(j, I')$, we know that the conjunctions in the reformulated equations (i.e., in every ψ_i) now include an additional branch for each condition $c_k \in \mathbb{C}(j, I')$ where c_k is a *compound condition*, e.g., $c_0 \wedge c_1 \wedge \ldots \wedge c_n$ or $c_0 \wedge \neg c_1 \wedge \ldots \wedge \neg c_n$. These compound conditions consist in a *truth combination* of the filtering conditions of the sibling *SAs* which specify *syntactically equal patterns*. This is guaranteed since from (186) we know that the equations in Eq' are *uniform*, meaning that all sibling pattern equivalent *SAs* are guaranteed to be syntactically equal as well. Hence, the reconstructed *SAs* in these new branches are *unable* to match the same concrete event α unless they are define the same pattern and condition. This is so as despite their pattern being syntactically equal, *only one* compound filtering condition can at most be satisfied by the matching concrete event α. Therefore, from (195) and the definition of *equi-disjoint*, we can deduce that

$$\text{every equation } (X_k{=}\psi_k) \in \left\{ X_i{=}\bigwedge_{c_k \in \mathbb{C}(j,I')} [p, c_k] X_j \wedge \varphi (= \psi_i) \;\middle|\; \begin{array}{l} (X_i{=}\bigwedge_{j \in I''} [p, c_j] X_j \wedge \varphi) \in Eq_{//I} \\ \text{and } I'{=}\bigcup_{l \in I} \mathsf{child}(Eq, l) \\ \textit{such that } I'' \subseteq I' \end{array} \right\} \text{ is } \textit{equi-disjoint} \tag{196}$$

which means that from (187), (195) and (196) we can conclude that

$$\text{every equation } (X_k{=}\psi_k) \in \omega'' \text{ is } \textit{equi-disjoint} \tag{197}$$

as required. We also argue that the reconstructed equations in (195) (i.e., $X_i{=}\psi_i$) are in fact *semantically equivalent* to the original ones (i.e., $(X_i{=}\varphi_i) {\in} Eq_{//I}$), since whenever a guarded branch, $[p, c_i]X_i$, is reconstructed into (possibly) multiple branches, $[p, c_i {\wedge} c_j \ldots c_k]X_i \wedge [p, c_i {\wedge} \neg c_j \ldots c_k]X_i \wedge \ldots \wedge [p, c_i {\wedge} \neg c_j \ldots \neg c_k]X_i$, via the truth combination function $\mathbb{C}(i, I')$, the condition, c_i, of the original branch is *never negated*. This guarantees that continuation X_i can only be reached when the original condition c_i is *true*, and thus preserves the original semantics of the branch. Therefore, we conclude that

$$\left\{ X_i{=}\bigwedge_{c_k \in \mathbb{C}(j,I')} [p, c_k] X_j \wedge \varphi (= \psi_i) \;\middle|\; \begin{array}{l} (X_i{=}\bigwedge_{j \in I''} [p, c_j] X_j \wedge \varphi) \in Eq_{//I} \\ \text{and } I'{=}\bigcup_{l \in I} \mathsf{child}(Eq, l) \\ \textit{such that } I'' \subseteq I' \end{array} \right\} \equiv Eq_{//I}$$

which means that from (185) and (195) we can infer that

$$Eq_{//\mathsf{dom}_{\mathsf{ind}}(\omega'')} \equiv \omega''. \tag{198}$$

Finally, since from (183) and (192) we know that $Eq'' \subseteq Eq$, from (186) we can infer that every equation $(X_j{=}\varphi_j) \in Eq''$ is *uniform*. Hence, with this result along with (194), (197) and (198) we can apply the inductive hypothesis and conclude that

$$Eq \equiv \omega' \text{ and that every equation } (X_k{=}\psi_k) \in \omega' \text{ is } \textit{equi-disjoint}$$

as required, and so we are done.

□

B Missing proofs from Sect. 6

B.1 Proving Lemma 12

We need to prove that for every system s, sHML formula φ and trace $t \in traces(s)$ when $s \in [\![\varphi]\!]$ then $\mathsf{sys}(t) \in [\![\varphi]\!]$.

Proof Since when restricted to sHML $s \in [\![\varphi]\!]$ can be defined in terms of the coinductive satisfaction rules of Fig. 4, we prove that $\mathcal{R} \overset{\text{def}}{=} \left\{ (\mathsf{sys}(t), \varphi) \;\middle|\; s \vDash \varphi \textit{ and } t \in traces(s) \right\}$ is a satisfaction relation that follows the rules of Fig. 4. We proceed by case analysis on φ.

Cases $\varphi \in \{\mathsf{ff}, X\}$. These cases do not apply since $s \nvDash \varphi$ when $\varphi \in \{\mathsf{ff}, X\}$.

Case $\varphi = \mathsf{tt}$. This case is satisfied trivially since $\varphi = \mathsf{tt}$.

Case $\varphi = \bigwedge_{i\in I} \varphi_i$. Assume that $s \vDash \bigwedge_{i\in I} \varphi_i$ from which by the definition of $\vDash$ we have that for every $i \in I$, $s \vDash \varphi_i$ and so by applying the definition of $\mathcal{R}$ for every $i \in I$ we get that $\forall i \in I \cdot (\mathsf{sys}(t), \varphi_i) \in \mathcal{R}$ as required.

Case $\varphi = \max X.\varphi$. Assume that $s \vDash \max X.\varphi$ from which by the definition of $\vDash$ we have that $s \vDash \varphi\{\max X.\varphi/X\}$ and so by applying the definition of $\mathcal{R}$ we get that $(\mathsf{sys}(t), \varphi\{\max X.\varphi/X\}) \in \mathcal{R}$ as required.

Case $\varphi = [p, c]\varphi$ Assume that

$$t \in \mathit{traces}(s) \tag{199}$$

and that $s \vDash [p, c]\varphi$ from which by the definition of $\vDash$ we have that

$$s \overset{\alpha}{\Longrightarrow} s' \tag{200}$$

$$\mathsf{mtch}(p, \alpha) = \sigma \textit{ and } c\sigma \Downarrow \mathsf{true} \tag{201}$$

$$s' \vDash \varphi\sigma. \tag{202}$$

Since from (200) we know that s transitions to s' over α, from (199) we can infer that $\alpha t' \in \mathit{traces}(s)$ where $t' \in \mathit{traces}(s')$ which means that by (202) and the definition of $\mathcal{R}$ we have that

$$(\mathsf{sys}(t'), \varphi\sigma) \in \mathcal{R}. \tag{203}$$

Therefore, this case holds by (201), (203) and since $\mathsf{sys}(\alpha t') \overset{\alpha}{\Longrightarrow} \mathsf{sys}(t')$ and so we are done. □

B.2 Proving Lemma 13

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

Proof We proceed by rule induction on after_φ.

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

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

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

$$s \overset{\alpha}{\Longrightarrow} s' \tag{204}$$

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

$$s' \notin [\![\bigwedge_{i\in I} \mathit{after}_\varphi(\varphi_i, \alpha)]\!] \equiv \exists j \in I \cdot s' \notin [\![\mathit{after}_\varphi(\varphi_j, \alpha)]\!]. \tag{205}$$

Hence, by (204) and (205) we can apply the inductive hypothesis and deduce that there exists a $j \in I$ such that $s \notin [\![\varphi_j]\!]$ which means that $s \notin \bigcap_{i\in I}[\![\varphi_i]\!] = [\![\bigwedge_{i\in I} \varphi_i]\!]$ as required.

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

$$s \overset{\alpha}{\Rightarrow} s' \tag{206}$$

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

$$s' \notin [\![\mathit{after}_\varphi(\varphi\{\mathsf{max}\, X.\varphi/X\}, \alpha)]\!] \tag{207}$$

and since by (206), (207) and the inductive hypothesis we have that $s \notin [\![\varphi\{\mathsf{max}\, X.\varphi/X\}]\!]$ and $[\![\varphi\{\mathsf{max}\, X.\varphi/X\}]\!] = [\![\mathsf{max}\, X.\varphi]\!]$ we can conclude that $s \notin [\![\mathsf{max}\, X.\varphi]\!]$ as required.

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

$$s \overset{\alpha}{\Rightarrow} s' \tag{208}$$

$$s' \notin [\![\mathit{after}_\varphi([p, c]\varphi, \alpha)]\!]. \tag{209}$$

Now consider the following two cases:

– $\mathsf{mtch}(p, \alpha) = \sigma$ and $c\sigma \Downarrow \mathsf{true}$: By (209) and the definition of after_φ we know that

$$s' \notin [\![\varphi\sigma]\!] \tag{210}$$

and so from (208), (210) and by the definition of $[\![-]\!]$ we can infer that $s \notin [\![[p, c]\varphi]\!]$ since there exists a transition, i.e., (208), that leads to a violation, i.e., (210).

– Otherwise: This case does not apply since $\mathit{after}_\varphi([p, c]\varphi, \alpha) = \mathsf{tt}$ which contradicts assumption (209).

□

B.3 Proving Lemma 14

We need to prove that for every action α, sHML formula φ and trace t, if $\mathsf{sys}(t) \in [\![\mathit{after}_\varphi(\varphi, \alpha)]\!]$ then $\mathsf{sys}(\alpha t) \in [\![\varphi]\!]$.

Proof We proceed by rule induction on after_φ.

Case $\mathit{after}_\varphi(\mathsf{ff}, \alpha)$. This case does not apply since $\mathit{after}_\varphi(\mathsf{ff}, \alpha) = \mathsf{ff}$ and so the assumption that $\mathsf{sys}(t) \in [\![\mathit{after}_\varphi(\mathsf{ff}, \alpha)]\!]$ is invalid.

Case $\mathit{after}_\varphi(\mathsf{tt}, \alpha)$. This case holds trivially since $\mathsf{sys}(\alpha t) \in [\![\mathsf{tt}]\!]$.

Case $\mathit{after}_\varphi(\bigwedge_{i\in I} \varphi_i, \alpha)$. Assume that $\mathsf{sys}(t) \in [\![\mathit{after}_\varphi(\bigwedge_{i\in I} \varphi_i, \alpha)]\!]$ from which by the definition of after_φ we have that

$$\mathsf{sys}(t) \in [\![\textstyle\bigwedge_{i\in I} \mathit{after}_\varphi(\varphi_i, \alpha)]\!] \equiv \forall i \in I \cdot \mathsf{sys}(t) \in [\![\mathit{after}_\varphi(\varphi_i, \alpha)]\!]. \tag{211}$$

Hence, knowing (211) we can apply the inductive hypothesis for every $i \in I$ and deduce that $\mathsf{sys}(\alpha t) \in [\![\varphi_i]\!]$ which means that $\mathsf{sys}(\alpha t) \in \bigcap_{i\in I}[\![\varphi_i]\!] = [\![\bigwedge_{i\in I} \varphi_i]\!]$ as required.

Case $\mathit{after}_\varphi(\mathsf{max}\, X.\varphi, \alpha)$. Assume that $\mathsf{sys}(t) \in [\![\mathit{after}_\varphi(\mathsf{max}\, X.\varphi, \alpha)]\!]$ from which by the definition of after_φ we know that

$$\mathsf{sys}(t) \in [\![\mathit{after}_\varphi(\varphi\{\mathsf{max}\, X.\varphi/X\}, \alpha)]\!] \tag{212}$$

and since by (212) and the inductive hypothesis we have that $\mathsf{sys}(\alpha t) \in [\![\varphi\{\max X.\varphi/X\}]\!]$ and $[\![\varphi\{\max X.\varphi/X\}]\!] = [\![\max X.\varphi]\!]$ we can conclude that $\mathsf{sys}(\alpha t) \in [\![\max X.\varphi]\!]$ as required.

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

$$\mathsf{sys}(t) \in [\![\mathit{after}_\varphi([p,c]\varphi,\alpha)]\!] \tag{213}$$

and consider the following two cases:

– $\mathsf{mtch}(p,\alpha) = \sigma$ and $c\sigma \Downarrow \mathsf{true}$: By (213) and the definition of after_φ we have that

$$\mathsf{sys}(t) \in [\![\varphi\sigma]\!]. \tag{214}$$

Since $\mathsf{sys}(\alpha t)$ is a trace process that can only perform α and transition to $\mathsf{sys}(t)$, i.e., $\mathsf{sys}(\alpha t) \overset{\alpha}{\Rightarrow} \mathsf{sys}(t)$, and since from (214) we know that $\mathsf{sys}(t)$ satisfies $\varphi\sigma$, by the definition of $[\![-]\!]$ we can thus conclude that $\mathsf{sys}(\alpha t) \in [\![[p,c]\varphi]\!]$ as required.

– Otherwise: This case is trivially satisfied since knowing that $\mathsf{sys}(\alpha t) \overset{\alpha}{\Rightarrow} \mathsf{sys}(t)$ and that $\mathsf{mtch}(p,\alpha) = \mathsf{undef}$ or $c \Downarrow \mathsf{ff}$, by the definition of $[\![-]\!]$ we can immediately conclude that $\mathsf{sys}(\alpha t) \in [\![[p,c]\varphi]\!]$ as required.

References

1. Francalanza, A.: A theory of monitors. Inf. Comput. **281**, 104704 (2021). https://doi.org/10.1016/j.ic.2021.104704
2. Schneider, F.B.: Enforceable security policies. ACM Trans. Inf. Syst. Secur. (TISSEC) **3**(1), 30–50 (2000)
3. Francalanza, A., Aceto, L., Achilleos, A., Attard, D.P., Cassar, I., Della Monica, D., Ingólfsdóttir, A.: A foundation for runtime monitoring. In: Lahiri, S., Reger, G. (eds.) Runtime Verification, pp. 8–29. Springer, Cham (2017)
4. Ligatti, J., Bauer, L., Walker, D.: Edit automata: enforcement mechanisms for run-time security policies. Int. J. Inf. Secur. **4**(1), 2–16 (2005)
5. Ligatti, J., Reddy, S.: A theory of runtime enforcement, with results. In: Gritzalis, D., Preneel, B., Theoharidou, M. (eds.) CESORICS, pp. 87–100. Springer, Berlin (2010)
6. Falcone, Y., Mounier, L., Fernandez, J.-C., Richier, J.-L.: Runtime enforcement monitors: composition, synthesis, and enforcement abilities. Formal Methods Syst. Des. **38**(3), 223–262 (2011)
7. Berstel, J., Boasson, L.: Transductions and context-free languages. Ed. Teubner, pp. 1–278 (1979)
8. Sakarovitch, J.: Elements of Automata Theory. Cambridge University Press, New York (2009)
9. Alur, R., Černý, P.: Streaming transducers for algorithmic verification of single-pass list-processing programs. In: Proceedings of the 38th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, pp. 599–610. ACM, ISBN 978-1-4503-0490-0 (2011)
10. Könighofer, B., Alshiekh, M., Bloem, R., Humphrey, L., Könighofer, R., Topcu, U., Wang, C.: Shield synthesis. Formal Methods Syst. Des. **51**(2), 332–361 (2017)
11. Francalanza, A., Aceto, L., Ingólfsdóttir, A.: Monitorability for the Hennessy–Milner logic with recursion. Formal Methods Syst. Des. **51**(1), 87–116 (2017)
12. Aceto, L., Achilleos, A., Francalanza, A., Ingólfsdóttir, A.: Monitoring for silent actions. In: Lokam, S., Ramanujam, R. (eds.) FSTTCS 2017: Foundations of Software Technology and Theoretical Computer Science, volume 93 of LIPIcs, p. 7:1-7:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Dagstuhl (2018)
13. Aceto, L., Achilleos, A., Francalanza, A., Ingólfsdóttir, A.: A framework for parameterized monitorability. In: Baier, C., Dal Lago, U. (eds.) Foundations of Software Science and Computation Structures, pp. 203–220. Springer, Cham (2018)
14. Aceto, L., Cassar, I., Francalanza, A., Ingólfsdóttir, A.: On bidirectional runtime enforcement. In: Peters, K., Willemse, T.A.C. (eds.) FORTE, volume 12719 of Lecture Notes in Computer Science, pp. 3–21. Springer, Cham (2021)
15. Aceto, L., Cassar, I., Francalanza, A., Ingólfsdóttir, A.: Comparing controlled system synthesis and suppression enforcement. Int. J. Softw. Tools Technol. Transf. **23**(4), 601–614 (2021)
16. Burlò, C.B., Francalanza, A., Scalas, A.: On the monitorability of session types, in theory and practice. In: Møller, A., Sridharan, M. (eds.) 35th European Conference on Object-Oriented Programming, ECOOP

2021, July 11–17, 2021, Aarhus, Denmark (Virtual Conference), volume 194 of LIPIcs, p. 20:1-20:30. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Dagstuhl (2021)
17. Artho, C., Barringer, H., Goldberg, A., Havelund, K., Khurshid, S., Lowry, M.R., Pasareanu, C.S., Rosu, G., Sen, K., Visser, W., Washington, R.: Combining test case generation and runtime verification. Theoret. Comput. Sci. **336**(2–3), 209–234 (2005)
18. Desai, A., Dreossi, T., Seshia, S.A.: Combining model checking and runtime verification for safe robotics. In: Lahiri, S., Reger, G. (eds.) Runtime Verification (RV), LNCS, pp. 172–189. Springer, Cham (2017)
19. Bocchi, L., Chen, T.-C., Demangeon, R., Honda, K., Yoshida, N.: Monitoring networks through multiparty session types. Theor. Comput. Sci. **669**, 33–58 (2017)
20. Jia, L., Gommerstadt, H., Pfenning, F.: Monitors and blame assignment for higher-order session types. In: Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. ACM, New York, pp. 582–594 (2016)
21. Ferrando, A., Dennis, L.A., Ancona, D., Fisher, M., Mascardi, V.: Verifying and validating autonomous systems: towards an integrated approach. In: Colombo, C., Leucker, M. (eds.) Runtime Verification—18th International Conference, RV 2018, volume 11237 of Lecture Notes in Computer Science, pp. 263–281. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-03769-7_15
22. Kejstová, K., Ročkai, P., Barnat, J.: From model checking to runtime verification and back. In: Lahiri, S., Reger, G. (eds.) Runtime Verification RV 2017. Springer, Cham (2017)
23. Aceto, L., Achilleos, A., Francalanza, A., Ingólfsdóttir, A., Lehtinen, K.: Adventures in monitorability: from branching to linear time and back again. Proc. ACM Program. Lang. **3**, 52:1-52:29 (2019). https://doi.org/10.1145/3290365
24. Chang, E., Manna, Z., Pnueli, A.: The safety-progress classification. In: Bauer, F.L., et al. (eds.) Logic and Algebra of Specification, pp. 143–202. Springer, Berlin (1993)
25. Pnueli, A., Zaks, A.: PSL model checking and run-time verification via testers. In: Misra, J., Nipkow, T., Sekerinski, E. (eds.) International Symposium on Formal Methods, pp. 573–586. Springer, Berlin (2006)
26. Francalanza, A., Cini, C.: Computer says no: verdict explainability for runtime monitors using a local proof system. J. Log. Algebraic Methods Program. **119**, 100636 (2021). https://doi.org/10.1016/j.jlamp.2020.100636
27. Aceto, L., Achilleos, A., Francalanza, A., Ingólfsdóttir, A., Lehtinen, K.: The best a monitor can do. In: Baier, C., Goubault-Larrecq, J. (eds.) 29th EACSL Annual Conference on Computer Science Logic, CSL 2021, January 25–28, 2021, Ljubljana, Slovenia (Virtual Conference), volume 183 of LIPIcs, p. 7:1-7:23. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Dagstu (2021). https://doi.org/10.4230/LIPIcs.CSL.2021.7
28. Falcone, Y., Fernandez, J.-C., Mounier, L.: What can you verify and enforce at runtime? Int. J. Softw. Tools Technol. Transf. **14**(3), 349 (2012)
29. Kozen, D.C.: Results on the propositional μ-calculus. Theoret. Comput. Sci. **27**, 333–354 (1983)
30. Larsen, K.G.: Proof systems for satisfiability in Hennessy–Milner logic with recursion. Theor. Comput. Sci. **72**(2), 265–288 (1990)
31. Clarke, E.M., Emerson, E.A.: Design and synthesis of synchronization skeletons using branching time temporal logic. In: Kozen, D. (ed.) 25 Years of Model Checking, pp. 196–215. Springer, Berlin (2008)
32. Aceto, L., Achilleos, A., Francalanza, A., Ingólfsdóttir, A., Lehtinen, K.: An operational guide to monitorability with applications to regular properties. Softw. Syst. Model. **20**(2), 335–361 (2021). https://doi.org/10.1007/s10270-020-00860-z
33. Bauer, A., Leucker, M., Schallhart, C.: The good, the bad, and the ugly, but how ugly is ugly? In: Sokolsky, O., Tasiran, S. (eds.) International Workshop on Runtime Verification, pp. 126–138. Springer, Berlin (2007)
34. Artho, C., Barringer, H., Goldberg, A., Havelund, K., Khurshid, S., Lowry, M.R., Pasareanu, C.S., Rosu, G., Sen, K., Visser, W., Washington, R.: Combining test case generation and runtime verification. Theor. Comput. Sci. **336**(2–3), 209–234 (2005)
35. Leucker, M.: Sliding between model checking and runtime verification. In: Qadeer, S., Tasiran, S. (eds.) RV, volume 7687 of Lecture Notes in Computer Science, pp. 82–87. Springer, Berlin (2012)
36. Decker, N., Leucker, M., Thoma, D.: junit$^{\mathrm{rv}}$-adding runtime verification to junit. In: Brat, G., Rungta, N., Venet, A. (eds.) NASA Formal Methods, volume 7871 of Lecture Notes in Computer Science, pp. 459–464. Springer, Berlin (2013)
37. Desai, A., Dreossi, T., Seshia, S.A.: Combining model checking and runtime verification for safe robotics. In: Lahiri, S., Reger, G. (eds.) RV, volume 10548 of Lecture Notes in Computer Science, pp. 172–189. Springer, Cham (2017)
38. Kejstová, K., Rockai, P., Barnat, J.: From model checking to runtime verification and back. In: Lahiri, S., Reger, G. (eds.) RV, volume 10548 of Lecture Notes in Computer Science, pp. 225–240. Springer, Cham (2017)

39. Aceto, L., Achilleos, A., Francalanza, A., Ingólfsdóttir, A., Lehtinen, K.: Testing equivalence vs. runtime monitoring. In: Boreale, M., Corradini, F., Loreti, M., Pugliese, R. (eds.) Models, Languages, and Tools for Concurrent and Distributed Programming, volume 11665 of Lecture Notes in Computer Science, pp. 28–44. Springer, Berlin (2019)
40. Monica, D.D, Francalanza, A.L.: Pushing runtime verification to the limit: may process semantics be with us. In: OVERLAY@AI*IA, volume 2509 of CEUR Workshop Proceedings, pp. 47–52. CEUR-WS.org (2019)
41. Havelund, K., Peled, D.: Bdds for representing data in runtime verification. In: Deshmukh, J., Nickovic, D. (eds.) RV, volume 12399 of Lecture Notes in Computer Science, pp. 107–128. Springer, Cham (2020)
42. Guzmán, M., Riganelli, O., Micucci, D., Mariani, L.: Test4enforcers: test case generation for software enforcers. In: Deshmukh, J., Nickovic, D. (eds.) RV, volume 12399 of Lecture Notes in Computer Science, pp. 279–297. Springer, Cham (2020)
43. Burlò, C.B., Francalanza, A., Scalas, A.: Towards a hybrid verification methodology for communication protocols (short paper). In: Gotsman, A., Sokolova, A. (eds.) FORTE, volume 12136 of Lecture Notes in Computer Science, pp. 227–235. Springer, Cham (2020)
44. Shijubo, J., Waga, M., Suenaga, K.: Efficient black-box checking via model checking with strengthened specifications. In: Feng, L., Fisman, D. (eds.) RV, volume 12974 of Lecture Notes in Computer Science, pp. 100–120. Springer, Cham (2021)
45. Martinelli, F., Matteucci, I.: Partial model checking, process algebra operators and satisfiability procedures for (automatically) enforcing security properties. In: Foundations of Computer Security. Citeseer, pp. 133–144 (2005)
46. Andersen, H.R.: Partial model checking. In: Proceedings of Tenth Annual IEEE Symposium on Logic in Computer Science. IEEE, pp. 398–407 (1995)
47. Lang, F., Mateescu, R.: Partial model checking using networks of labelled transition systems and Boolean equation systems. In: Flanagan, C., König, B. (eds.) TACAS, pp. 141–156. Springer, Berlin (2012)
48. Attard, D.P., Francalanza, A.: A monitoring tool for a branching-time logic. In: Falcone, Y., Sanchez, C. (eds.) Runtime Verification, pp. 473–481. Springer, Cham (2016)
49. Attard, D.P., Cassar, I., Francalanza, A., Aceto, L., Ingolfsdottir, A.: A Runtime Monitoring Tool for Actor-Based Systems, pp. 49–74. River Publishers, Aalborg (2017)
50. Francalanza, A., Xuereb, J.: On implementing symbolic controllability. In: Bliudze, S., Bocchi, L. (eds.) COORDINATION, volume 12134 of Lecture Notes in Computer Science, pp. 350–369. Springer, Cham (2020)
51. Attard, D.P., Aceto, L., Achilleos, A., Francalanza, A., Ingólfsdóttir, A., Lehtinen, K.: Better late than never or: verifying asynchronous components at runtime. In: Peters, K., Willemse, T.A.C. (eds.) Formal Techniques for Distributed Objects, Components, and Systems—41st IFIP WG 6.1 International Conference, FORTE 2021, Held as Part of the 16th International Federated Conference on Distributed Computing Techniques, DisCoTec 2021, Valletta, Malta, June 14–18, 2021, Proceedings, volume 12719 of Lecture Notes in Computer Science, pp. 207–225. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-78089-0_14
52. Achilleos, A., Exibard, L., Francalanza, A., Lehtinen, K., Xuereb, J.: A synthesis tool for optimal monitors in a branching-time setting. In: ter Beek, M.H., Sirjani, M. (eds.) COORDINATION, volume 13271 of Lecture Notes in Computer Science, pp. 181–199. Springer, Cham (2022)
53. Aceto, L., Achilleos, A., Attard, D.P., Exibard, L., Francalanza, A., Ingólfsdóttir, A.: A monitoring tool for linear-time μhml. In: ter Beek, M.H., Sirjani, M. (eds.) COORDINATION, volume 13271 of Lecture Notes in Computer Science, pp. 200–219. Springer, Cham (2022)
54. Aceto, L., Cassar, I., Francalanza, A., Ingólfsdóttir, A.: On runtime enforcement via suppressions. In: 29th International Conference on Concurrency Theory, CONCUR 2018, September 4–7, 2018, Beijing, China, pp. 34:1–34:17 (2018). https://doi.org/10.4230/LIPIcs.CONCUR.2018.34
55. Milner, R., Parrow, J., Walker, D.: A calculus of mobile processes. I. Inf. Comput. **100**(1), 1–40 (1992)
56. Sangiorgi, D.: Introduction to Bisimulation and Coinduction. Cambridge University Press, New York (2011)
57. Aceto, L., Ingólfsdóttir, A., Larsen, K.G., Srba, J.: Reactive Systems: Modelling, Specification and Verification. Cambridge University Press, New York (2007)
58. Hennessy, M., Milner, R.: Algebraic laws for nondeterminism and concurrency. J. ACM **32**(1), 137–161 (1985)
59. Stirling, C.: Handbook of logic in computer science, vol. 2. Modal and Temporal Logics, pp. 477–563. Oxford University Press, Inc., New York (1992)
60. Stirling, C.: Model checking and other games. In: Notes for Mathfit Workshop on Finite Model Theory. University of Wales, Swansea (1996)

61. Francalanza, A.: A Theory of Monitors (extended abstract). In: International Conference on Foundations of Software Science and Computation Structures. Springer, pp. 145–161 (2016)
62. Francalanza, A.: Consistently-detecting monitors. In: 28th International Conference on Concurrency Theory (CONCUR 2017), volume 85 of Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, pp. 8:1–8:19 (2017)
63. d'Amorim, M., Roşu, G.: Efficient monitoring of ω-languages. In: CAV, pp. 364–378 (2005)
64. Wolff, E.M., Topcu, U., Murray, R.M.: Efficient reactive controller synthesis for a fragment of linear temporal logic. In: 2013 IEEE International Conference on Robotics and Automation, pp. 5033–5040, May (2013). https://doi.org/10.1109/ICRA.2013.6631296
65. Dolzhenko, E., Ligatti, J., Reddy, S.: Modeling runtime enforcement with mandatory results automata. Int. J. Inf. Secur. **14**(1), 47–60 (2015). https://doi.org/10.1007/s10207-014-0239-8
66. Debois, S., Hildebrandt, T., Slaats, T.: Safety, liveness and run-time refinement for modular process-aware information systems with dynamic sub processes. In: Bjørner, N., de Boer, F. (eds.) FM 2015: Formal Methods, pp. 143–160. Springer, Cham (2015)
67. Aceto, L., Achilleos, A., Francalanza, A., Ingólfsdóttir, A., Kjartansson, S.Ö.: Determinizing monitors for HML with recursion. J. Log. Algebraic Methods Program. **111**, 100515 (2020). https://doi.org/10.1016/j.jlamp.2019.100515
68. van Hulst, A.C., Reniers, M.A., Fokkink, W.J.: Maximally permissive controlled system synthesis for non-determinism and modal logic. Discrete Event Dyn. Syst. **27**(1), 109–142 (2017)
69. Milner, R.: Communication and Concurrency. PHI Series in Computer Science, Prentice Hall, Upper Saddle River (1989)
70. Bielova, N., Massacci, F.: Predictability of enforcement. In: Erlingsson, U., Wieringa, R., Zannone, N. (eds.) International Symposium on Engineering Secure Software and Systems, pp. 73–86. Springer, Berlin (2011)
71. Attard, D.P., Francalanza, A.: Trace partitioning and local monitoring for asynchronous components. In: Cimatti, A., Sirjani, M. (eds.) Software Engineering and Formal Methods—15th International Conference, SEFM 2017, Trento, Italy, September 4–8, 2017, Proceedings, volume 10469 of Lecture Notes in Computer Science, pp. 219–235. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-66197-1_14
72. Aceto, L., Attard, D.P., Francalanza, A., Ingólfsdóttir, A.: On benchmarking for concurrent runtime verification. In: Guerra, E., Stoelinga, M. (eds.) Fundamental Approaches to Software Engineering—24th International Conference, FASE 2021, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2021, Luxembourg City, Luxembourg, March 27–April 1 (2021), Proceedings, volume 12649 of Lecture Notes in Computer Science, pp. 3–23. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-71500-7_1
73. Aceto, L., Ingólfsdóttir, A.: Testing Hennessy–Milner logic with recursion. In: Thomas, W. (ed.) Foundations of Software Science and Computation Structures, pp. 41–55. Springer, Berlin (1999)
74. Rabinovich, A.M.: A complete axiomatisation for trace congruence of finite state behaviors. In: Brookes, S., Main, M., Melton, A., Mislove, M., Schmidt, D. (eds.) Proceedings of the 9th International Conference on Mathematical Foundations of Programming Semantics, pp. 530–543. Springer, London (1994)
75. Bielova, N.: A theory of constructive and predictable runtime enforcement mechanisms. Ph.D. Thesis, University of Trento (2011)
76. Pnueli, Z.M.A.: A hierarchy of temporal properties. In: Proceedings of the 2nd Symposium. ACM of Principle Of Distributed Computer (1990)
77. Pinisetty, S., Falcone, Y., Jéron, T., Marchand, H.: Runtime enforcement of parametric timed properties with practical applications. In: IEEE International Workshop on Discrete Event Systems, Cachan, France, May, pp. 46–53 (2014)
78. Pinisetty, S., Roop, P.S., Smyth, S., Tripakis, S., von Hanxleden, R.: Runtime enforcement of reactive systems using synchronous enforcers. CoRR, arxiv:1612.05030 (2016)
79. Pinisetty, S., Roop, P.S., Smyth, S., Allen, N., Tripakis, S., Hanxleden, R.V.: Runtime enforcement of cyber-physical systems. ACM Trans. Embed. Comput. Syst. **16**(5), 178:1-178:25 (2017)
80. Lanotte, R., Merro, M., Munteanu, A.: Runtime enforcement for control system security. In: 33rd IEEE Computer Security Foundations Symposium, CSF 2020, Boston, MA, USA, June 22–26, 2020. IEEE, pp. 246–261 (2020). https://doi.org/10.1109/CSF49147.2020.00025
81. Martinelli, F., Matteucci, I.: Through modeling to synthesis of security automata. Electron. Not. Theor. Comput. Sci. **179**, 31–46 (2006)
82. Martinelli, F., Matteucci, I.: An approach for the specification, verification and synthesis of secure systems. Electron. Not. Theor. Comput. Sci. **168**, 29–43 (2007)
83. Castellani, I., Dezani-Ciancaglini, M., Pérez, J.A.: Self-adaptation and secure information flow in multiparty communications. Formal Asp. Comput. 28 (4): 669-696 (2016)

84. Cassar, I., Francalanza, A.: On implementing a monitor-oriented programming framework for actor systems. In: Abraham, E., Huisman, M. (eds.) International Conference on Integrated Formal Methods, pp. 176–192. Springer, Cham (2016)
85. Francalanza, A., Seychell, A.: Synthesising correct concurrent runtime monitors (extended abstract). In: Legay, A., Bensalem, S. (eds.) RV, volume 8174 of Lecture Notes in Computer Science, vol. 8174, pp. 112–129. Springer, Cham (2013)