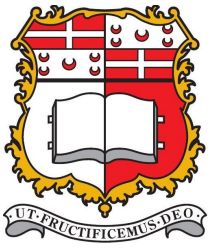

University of Malta &
Reykjavik University

PhD Dissertation Report

Developing Theoretical Foundations for Runtime Enforcement

Ian Cassar

Supervised by:

Prof. Adrian Francalanza,
Prof. Luca Aceto, and
Prof. Anna Ingólfsdóttir

Submitted in partial fulfillment of the requirements for the degree of PhD. in Computer Science on 24th January 2020.

University of Malta and Reykjavik University

Declaration

I, the undersigned, declare that the dissertation entitled:

Developing Theoretical Foundations for Runtime Enforcement

submitted is my work, except where acknowledged and referenced.

Ian Cassar

24th January 2020

The research work disclosed in this publication is partially funded by the Endeavour Scholarship Scheme (Malta). Scholarships are part-financed by the European Union - European Social Fund (ESF) - Operational Programme II – Cohesion Policy 2014-2020 "*Investing in human capital to create more opportunities and promote the well-being of society*".

European Union – European Structural and Investment Funds
Operational Programme II – Cohesion Policy 2014-2020
"Investing in human capital to create more opportunities and promote the well-being of society"
Scholarships are part-financed by the European Union - European Social Funds (ESF)
Co-financing rate: 80% EU Funds;20% National Funds

Acknowledgements

I would like to thank my parents for always being supportive of my decisions and for always encouraging me to keep on pursuing my academic goals. I will be forever grateful for their love and support. I would especially like to thank my girlfriend, and soon to be wife, Melanie for staying by my side through the tough times of this PhD. This academic venture proved to be a challenge for both of us since it required having to delay and sacrifice a lot of our couple goals, including that of getting married and buying a place of our own. It was an emotional challenge that only the two of us can truly understand.

I would also like to thank my friends at UOM and RU for making my life at both universities a fun and loving experience. In particular, I would like to thank Duncan for always having the right words to calm me down and cheer me up whenever I was stressed, Aaron for always keeping me good company and for giving me tips on how to improve my physical exercises at the gym, Antonis for always having my back while I was in Iceland, and Keith, Sandro and their awesome team of PhD students with whom I had the best laughs of my life. Moreover, I'd like to thank the Icelandic people in general for giving me a very warm and loving stay in such a cold country as Iceland. I have had many wonderful experiences during my stays in Iceland that I will cherish them for all my life. Finally, I'd like to thank my PhD advisers Luca, Anna and Adrian for always giving me excellent advice and guidance.

I further acknowledge that the research work disclosed in this dissertation was partially supported by the projects "TheoFoMon: Theoretical Foundations for Monitorability" (nr.163406-051) and "Developing Theoretical Foundations for Runtime Enforcement" (nr.184776-051) of the Icelandic Research Fund, and by the Endeavour Scholarship Scheme (Malta), part-financed by the European Social Fund (ESF) - Operational Programme II - 2014-2020.

"Somehow I can't believe that there are any heights that can't be scaled by a man who knows the secrets of making dreams come true. This special secret, it seems to me, can be summarized in four C s. They are curiosity, confidence, courage, and constancy, and the greatest of all is confidence. When you believe in a thing, believe in it all the way, implicitly and unquestionable. "

– Walt Disney

Nagato Uzumaki: *"Sometimes you must hurt in order to know, fall in order to grow, lose in order to gain because life's greatest lessons are learned through pain."*

– Masashi Kishimoto, *Naruto*

Abstract*

The ubiquitous reliance on software systems is increasing the need for ensuring their correctness. Runtime enforcement is a monitoring technique that uses monitors that can transform the actions of a system under scrutiny in order to alter its runtime behaviour and keep it in line with a correctness specification; these type of enforcement monitors are often called transducers. In runtime enforcement there is often no clear separation between the specification language describing the correctness criteria that a system must satisfy, and the monitoring mechanism that actually ensures that these criteria are met. We thus aim to adopt a separation of concerns between the correctness specification describing what properties the system should satisfy, and the monitor describing how to enforce these properties. In this thesis we study the enforceability of the highly expressive branching time logic μHML, in a bid to identify a subset of this logic whose formulas can be adequately enforced by transducers at runtime.

We conducted our study in relation to two different enforcement instrumentation settings, namely, a unidirectional setting that is simpler to understand and formalise but limited in the type of system actions it can transform at runtime, and a bidirectional one that, albeit being more complex, it allows transducers to effect and modify a wider set of system actions. During our investigation we define the behaviour of enforcement transducers and how they should be embedded with a system to achieve unidirectional and bidirectional enforcement. We also investigate what it means for a monitor to adequately enforce a logic formula, and define the necessary criteria that a monitor must satisfy in order to be adequate. Since enforcement monitors are highly intrusive, we also define a notion of optimality to use as a guide for identifying the least intrusive monitor that adequately enforces a formula.

Using these enforcement definitions, we identify a μHML fragment that can be adequately enforced via enforcement transducers that drop the execution of certain actions. We then show that this fragment is maximally expressive, *i.e.,* it is the largest subset that can be enforced via these type of enforcement monitors. We finally look into static alternatives to runtime enforcement and identify a static analysis technique that can also enforce the identified μHML fragment, but without requiring the system to execute.

*An electronic copy of this document can be retrieved from `http://bit.ly/ian-cassar-phd-2020`.

Contents

List of Figures

1. Introduction

Our dependence on software systems raises the demand for ensuring their correctness. Verifying software systems is, however, becoming harder due to their ever increasing complexity. For instance, systems nowadays are no longer simple monolithic processes that only produce a single output in reaction to an input. In fact, systems may now opt to collect data from multiple sources before providing the required response, or else they may supply multiple outputs in response to a single input. Moreover, most systems are also developed using concurrent interacting entities (such as processes, threads or actors) that may lead to non-deterministic behaviour.

Several techniques help facilitate the arduous task of verifying software by automating the process of deducing whether the system under scrutiny (SuS) satisfies a predefined set of correctness properties. Properties are either verified pre-deployment (statically), as in the case of model checking [11, 41], or post-deployment (dynamically), as per runtime verification [32, 55, 77]. In both cases, any error discovered during verification serves as guidance for identifying the invalid parts of the system that require adjustment.

Techniques, such as *Runtime Enforcement* (RE) [52, 78, 80], additionally attempt to automatically transform the invalid system into a valid one. Runtime enforcement is a dynamic verification technique that adopts an intrusive monitoring approach to ensure that the visible behaviour of the SuS is *always* in agreement with some correctness specification. It employs a specific kind of monitor (known as a *transducer* [12, 22, 95], *shield* [69] or *enforcement-automaton* [25, 52, 78, 80, 98]) that when instrumented with the SuS, produces a *well behaved* composite (monitored) system. At runtime, the composed monitor applies *transformations* to either *replace* invalid system actions by valid ones, *suppress* them, or *inserts* actions on behalf of the SuS.

The seminal work in RE [22, 24, 68, 78, 79, 95] models the behaviour of the SuS

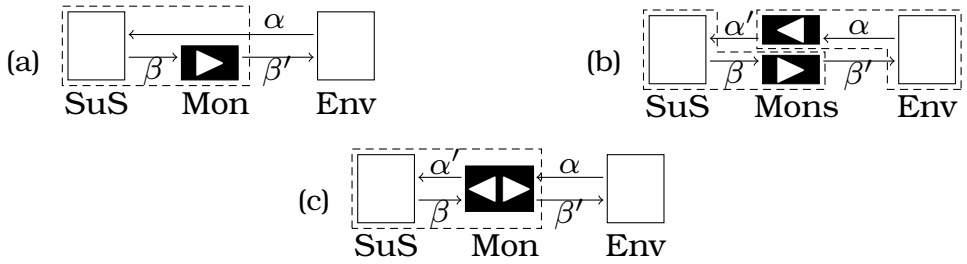

Figure 1.1: Enforcement instrumentation setups.

as a *trace* of *arbitrary* actions, *e.g.*, $\alpha_1.\alpha_2, \ldots$, and assumes that every action in the trace is instigated and controlled by the SuS and that the composed monitor can transform *any* trace action. This trace-based approach has thus been effectively used to implement *unidirectional* enforcement approaches [12, 50, 69] that monitor the trace of *outputs* produced by the SuS as illustrated by Figure 1.1 (a).

In this setup, the monitor is instrumented with the SuS to form a *composite system* (represented by the dashed enclosure) and is tasked with transforming the output behaviour of the SuS to ensure its correctness. For instance, if the SuS executes an erroneous output β, this gets intercepted by the monitor and modified accordingly into β' to stop the error from propagating to the environment.

Despite its merits, unidirectional enforcement lacks the power to enforce properties that require modifying the *input* behaviour of the SuS. This happens because, unlike outputs, inputs are instigated and mainly controlled by the environment. In fact, although the SuS can control when certain inputs can be supplied (*e.g.*, by opening and reading from a file, communication port, *etc.*), at runtime the environment determines what payload will be provided. Enforcing properties that require modifying input actions thus requires using a more elaborate instrumentation setup that supports for transforming input actions.

For this reason, instead of dealing with the complexities of directly transforming inputs, several solutions [26, 38, 44, 64, 80] have been presented to circumvent this issue by using an extra unidirectional monitor. As shown in Figure 1.1 (b), this monitor is attached to the environment to scrutinise its outputs before they are forwarded as inputs to the SuS. While this approach may be viable in certain cases, it assumes that a monitor can actually be attached to the environment (which is often inaccessible).

By contrast, Figure 1.1 (c) presents a less explored *bi-directional enforcement* setup. Unlike in (b), the monitor in this setup scrutinises the *entire* behaviour of the SuS without instrumenting the environment. Even though the monitor is only attached to the SuS, along with transforming the outputs produced by the SuS, it can also regulate the type of inputs allowed by the system, *e.g.*, by sanitising the

input payload before it reaches the SuS [87]. This setup must therefore adopt more elaborate transformations that exploit the limited control of the SuS to modify its inputs as necessary.

In view of these different instrumentation setups, one may realise that it is quite cumbersome to specify correctness properties directly as monitors. Particularly, this would burden the specifier with not only having to specify *what* properties need to be enforced, but also with *how* to enforce them at runtime, and which instrumentation setup to use. Our main aim in this dissertation is therefore to extend a recent line of research [2, 3, 5, 55–57] and adopt a *separation of concerns* between the correctness specification, describing *what* properties the SuS should satisfy, and the monitor, describing *how* to enforce these properties. We study the *enforceability* of the well-studied process logic μHML [70, 75], and explore which properties can be operationally enforced by enforcement monitors in view of the different instrumentation setups illustrated by Figure 1.1. As setup (b) is just a reformulation of (a), we conjecture that any enforceability result attained for (a) can also be applied to (b) with minimal effort, and thus only explore enforceability vis-a-vis setups (a) and (c), *i.e.,* unidirectional and bidirectional enforcement.

There are, however, several issues that we must address throughout our investigation. (i) Since software analysis tools ought to form part of the trusted computing base, enforcement monitoring should be, in and of itself, *correct*. However, it is unclear what is to be expected of an enforcement monitor to adequately enforce a μHML 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 [3, 5, 6, 37, 39, 56, 89] and it is therefore reasonable to expect similar limits in the case of enforceability [47]. It is also desirable for monitors to be optimal, particularly by minimising their intrusion when correcting the behaviour of the SuS. However, we are unsure whether a property can be enforced by different monitors, and if so, whether some monitors are more optimal than the others.

(ii) Studying enforcement in view of its entire behaviour is quite challenging, especially since bidirectional enforcement has barely been explored. Moreover, since the SuS enjoys limited control over its input behaviour, it is unclear whether the traditional monitor transformations suffice to completely enforce properties concerning both inputs and outputs. In fact, since properties are violated when the system performs an invalid action, it may be too late for the monitor to prevent the violation if it allows the SuS to input a value that then turns out to be invalid. It is hence questionable whether the monitor can intercept and suppress (or replace) an invalid input that has already been provided by the environment. The monitor must therefore exploit the system's limited control over its inputs in the best possible way

to ensure that the resulting composite system can perform all the specified valid inputs, while preventing it from performing invalid ones. We thus conjecture that the monitor must adopt a different enforcement approach than the ones conventionally used for enforcing behaviour in unidirectional settings.

(iii) The current RE state of the art [24, 50, 78, 80, 86, 87] fails to distinguish between the transformations performed by the monitor (*i.e.,* suppressions, insertions and replacements) and their resulting effect on the visible behaviour of the composite system. In fact, it is commonly accepted that the monitor's transformations dictate directly what happens to an action once it gets transformed. For instance, if an action gets suppressed by the enforcement monitor, then it is assumed to be removed from the composite system's behaviour. However, this might not necessarily be the case in bidirectional enforcement, especially since enforcing inputs might require unconventional use of the monitor's transformations.

(iv) It is not known whether there exists a static analysis technique that can at least enforce the same properties as RE pre-deployment. Having a way to attain the same enforcement result statically is particularly useful in situations where runtime enforcement is not viable, *e.g.,* when runtime overheads from the monitor may impact a performance critical system. This, however, requires identifying a static technique that can correct a SuS prior to deployment, so that the corrected system becomes —in some sense— equivalent to a monitored version of the SuS.

To the best of our knowledge, the above mentioned issues, $(i) - (iv)$, have not been studied extensively.

1.1 Our objectives

The setting selected for our study serves a number of purposes. To begin with, the chosen logic, μHML, is a branching-time logic that allows us to investigate enforceability for properties describing computation graphs. Having a way to define correct system behaviour in view of its different execution branches is especially useful when investigating enforceability in a bidirectional context. Particularly when considering systems whose inputs may lead them into taking erroneous branches that produce invalid outputs, enforcement monitors may harness this branching time information and modify their inputs in order to steer their execution towards a more stable state.

Second, the use of a highly expressive logic allows us to achieve a good degree of generality for our results, and so, by working in relation to logics like μHML (which is a reformulation of the modal μ-calculus [70]), our work also applies to other widely used logics (such as LTL and CTL [40]) that are embedded within this logic.

Third, unlike logics such as RV-LTL [19] and LTL_3 [20] which semantics is defined specifically for runtime verification, our chosen logic μHML is verification-technique agnostic since it's semantics is not defined with respect to a specific verification technique. It therefore fits better with the realities of software verification in the present world, where a *variety* of techniques (such as 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 enforced dynamically is crucial for devising effective multi-pronged verification strategies. Equipped with such knowledge, one could also employ standard techniques [13, 71, 82] to decompose a non-enforceable property into a collection of smaller properties, a subset of which can then be enforced at runtime.

Throughout this study we thus address the identified issues $(i) - (iv)$ by undertaking the following research objectives:

1. **Modelling:** We start addressing issue (i) by developing enforcement models that differentiate between the monitor's behaviour and that of the instrumentation. We first plan to develop a general framework for enforcement monitors that *solely* models their runtime behaviour. On top of this model, we can then define multiple instrumentation models, namely, one that achieves unidirectional enforcement and another for bidirectional enforcement; modelling the latter entails dealing with issue (ii). This separation of concerns also contributes towards addressing issue (iii) as it allows us to distinguish between the monitor's transformations and their resulting effect on the visible behaviour of the composite system.
2. **Correctness:** To evaluate our enforcement models we plan to further address issue (i) by developing formal definitions that specify what is to be expected of an enforcement monitor to adequately enforce a μHML formula. Since enforcement monitoring must be adequate and optimal irrespective of the applied instrumentation (unidirectional and bidirectional) approach, our definitions must be parametrisable with respect to any instrumentation setup.
3. **Expressiveness:** Having established the meaning of adequate enforcement in objective 1, we can use the formal models, alluded to by objective 2, to determine the type of specifications that can be adequately enforced at runtime. We thus continue addressing issue (i) by identifying a subset of μHML formulas that can be mapped to enforcement monitors via a *synthesis function*. To evaluate the quality of this mapping, we intend to provide *enforceability results* proving that the synthesised monitors adequately and optimally enforce their respective formula.

4. **Maximality:** To entirely address (i), we also aim to investigate whether the logic subset identified in objective 3, is in fact the *largest* enforceable subset that one can possibly find. This ensures that properties that do not form part of the identified enforceable subset cannot be enforced at runtime, unless they can be expressed as a formula pertaining to the identified subset.
5. **Static counterpart:** We tackle issue (iv) by exploring a static analysis technique called *Controlled System Synthesis* [14, 45, 88, 101]. Particularly, we aim to study how the behaviour of a system that was statically reformulated to satisfy a formula, compares to a monitored system that dynamically enforces the same formula. This enables us to investigate whether the identified maximally expressive μHML subset can also be enforced pre-deployment using this static technique.

We subdivide our study about the enforceability of μHML into two parts. For the first part we address all the above objectives (*i.e.,* 1-5) in a unidirectional context. We follow the more traditional, *trace-based* view of the SuS (as per [24, 49, 50, 68, 78, 79]) and assume that all actions can be treated *equally* and transformed in the same way. By contrast, in the second part we show that this assumption is too strong, and that the unidirectional model might not work as intended for certain actions. We thus commence our investigation into bidirectional enforcement by adopting a branching time view of the SuS and by distinguishing explicitly between its input and output actions. During this investigation we explore the first three objectives (*i.e.,* 1-3) in a bidirectional context. Although we do not address objectives 4 and 5 in the second part, we conjecture that addressing these objectives entails adopting similar methodologies to the ones employed when studying them in a unidirectional setting.

1.2 Document structure and outline of contributions

We structure the rest of the document as follows. We start with Chapter 2 where we provide preliminary material about how we model systems as labelled transition systems, and about our touchstone logic μHML. From this point onwards we subdivide the document into two parts, namely Part I exploring *unidirectional enforcement* in Chapters 3 to 8, and Part II discussing *bidirectional enforcement* in Chapters 9 to 12.

We start the investigation of Part I by addressing our first objective in Chapter 3. We thus present a formal model for unidirectional enforcement instrumentation (Figure 3.2) for which we prove zipping and unzipping results (Propositions 3.1

and 3.2). In Chapter 4 we then address the second objective by formalising the interdependent notions of enforceability (Definition 4.1), adequate enforcement (Definitions 4.4, 4.6 and 4.8) and optimal enforcement (Definition 4.9). As part of our investigation we also show how the different definitions of adequate enforcement relate to one another (Theorems 4.1 and 4.2 and Corollary 4.1). These notions act as a foundation for determining the type of μHML specifications that can be enforced at runtime while addressing objective 3 in Chapter 5. Specifically, in Chapter 5 we identify a μHML subset (Figure 2.4) that can be enforced by *action suppression monitors* when composed with a system via the unidirectional instrumentation model of Chapter 3. We define a synthesis function (Definition 5.2) that maps the formulas from the identified subset to suppression monitors. We then assess its correctness by providing *enforceability* results (Theorems 5.1 and 5.3) and by showing that the synthesised monitors are *optimal* (Theorem 5.2).

Having identified a μHML subset that is enforceable via action suppressions, in Chapter 6 we address our fourth objective. We first establish a notion of well structured suppression monitors, and show that ill-structured ones are unsound as they cannot enforce μHML formulas (Theorem 6.1). Using this result we then show that the μHML formulas that are enforceable via well structured suppression monitors, are either expressible using the μHML subset identified in Chapter 5, or else semantically equivalent to formulas that form part of the subset (Theorem 6.2). In Chapter 7 we finally address the last objective and identify the static analysis technique called Controlled System Synthesis as being the static counterpart to suppression enforcement (Theorems 7.2 to 7.4). Chapter 8 discusses related and future work about unidirectional enforcement and concludes Part I.

We start Part II by presenting a bidirectional instrumentation model of enforcement (Figure 9.2) in Chapter 9. Once again, we prove zipping and unzipping results (Propositions 9.1 and 9.2) for this new instrumentation model. In Chapter 10 we then present a sequence of examples to motivate how the enforcement definitions, introduced in Chapter 4 of Part I, are still relevant even with respect to the bidirectional setting of Chapter 9. Subsequently, in Chapter 11 we show that the same μHML subset that was identified to be maximally expressive in Part I, is also enforceable in a bidirectional context. To obtain this result we define yet another synthesis function that maps the formulas in the μHML subset to bidirectional enforcement monitors (Definition 11.2), and prove that the synthesised monitors enforce their respective formula adequately and optimally (Theorems 11.1 and 11.2). We conclude Part II in Chapter 12 and discuss related and future work about bidirectional enforcement.

Finally, we conclude the dissertation in Chapter 13 with some final remarks that

include an overview of the papers published by the author of this thesis during his PhD studies.

2. Preliminaries

In this chapter we give an overview of the preliminary material required to understand our contributions. In Section 2.1 we explain how we model systems as Labelled Transition Systems, and define the different notions of system equivalence that we consider throughout our work. In Section 2.2 we then introduce the syntax and semantics of the logic μHML.

2.1 Labelled Transition Systems

We assume systems described as *labelled transition systems* (LTSs) [66]. An LTS is a triple $\langle \text{Sys}, (\text{Act} \cup \{\tau\}), \rightarrow \rangle$ that consists of: a set of *system states* $s, r, q \in \text{Sys}$, a countably infinite set of *observable* (visible) actions $a, b, \ldots, \alpha, \beta \in \text{Act}$ along with a distinguished *silent* (invisible) action $\tau \notin \text{Act}$ (where $\mu \in \text{Act} \cup \{\tau\}$), and a *transition* relation, $\longrightarrow \subseteq (\text{Sys} \times (\text{Act} \cup \{\tau\}) \times \text{Sys})$. Note that apart from the Latin characters $a, b, \ldots \in \text{Act}$, we only use the Greek symbols α and β to denote *observable* system actions. We write $s \xrightarrow{\mu} r$ in lieu of $(s, \mu, r) \in \rightarrow$ and use $s \stackrel{\alpha}{\Longrightarrow} r$ to denote *delayed* transitions [96] representing $s(\xrightarrow{\tau})^* \cdot \xrightarrow{\alpha} r$; we often refer to the resulting system state r as the α-*derivative* of s. For each $\mu \in \text{Act} \cup \{\tau\}$ the notation $\hat{\mu}$ stands for ε if $\mu = \tau$ and for μ otherwise.

To concisely describe LTSs, we use the syntax of the regular fragments of CCS [85] (in Part I) and value-passing CCS [60] (in Part II) defined in Figure 2.1. These fragments allow for defining systems that are inactive nil, can perform a mutually-exclusive choice $\sum_{i \in I} s_i$ (which represents $s_1 + \ldots + s_n$ for some $1, \ldots, n \in I$), are recursive $\mathsf{rec}\, X.s$, and prefixed by an action $\mu.s$. The prefixing actions μ in CCS can be any action α or τ, whereas in value passing CCS they can be τ or else represent

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

Figure 2.1: The regular fragments of CCS and value-passing CCS.

the output of a value v on some communication port a *i.e.,* $\mathsf{a}!v$, or the input of some value x from a port a *i.e.,* $\mathsf{a}?x$.

We assume that traces $t, u \in \text{Act}^*$ range over (finite) sequences of visible actions and write $s \stackrel{t}{\Longrightarrow} r$ to denote a sequence of *delayed* transitions $s \stackrel{\alpha_1}{\Longrightarrow} \ldots \stackrel{\alpha_n}{\Longrightarrow} r$ for $t = \alpha_1, \ldots, \alpha_n$ (where $n \geq 1$), but when $t = \varepsilon$, $s \stackrel{\varepsilon}{\Longrightarrow} r$ means $s \stackrel{\tau}{\longrightarrow}^* r$. We also denote the set of traces executable from system state s as *traces*(s), meaning that $t \in$ *traces*(s) iff $s \stackrel{t}{\Longrightarrow} r$ for some r. System states that execute the same set of traces are said to be *trace equivalent*.

Definition 2.1 (Trace Equivalence)**.** Two LTS states s and r are *trace equivalent* iff they produce the *same* set of traces, *i.e., traces*$(s) =$ *traces*(r). □

Additionally, we represent system runs as *explicit traces* that include τ-actions, $t_\tau, u_\tau \in (\text{Act} \cup \{\tau\})^*$. We abuse notation and write $s \xrightarrow{t_\tau} r$ to denote a sequence of strong transitions $s \xrightarrow{\mu_1} \ldots \xrightarrow{\mu_n} r$ when $t_\tau = \mu_1 \ldots \mu_n$. Occasionally, we use the notation $t_\tau; u_\tau$ to denote the concatenation of two traces, and use the function $\mathsf{sys}(t_\tau)$ to produce a trace system *i.e.,* a system that *only* executes the sequence of actions defined in the system run t_τ. For instance, $\mathsf{sys}(\alpha_1.\tau.\alpha_2)$ produces $\alpha_1.\tau.\alpha_2.\mathsf{nil}$.

Definition 2.2 (Trace System)**.** A trace system $\mathsf{sys}(t_\tau)$ is nil when $t_\tau = \varepsilon$ and $\mu.\mathsf{sys}(t'_\tau)$ when $t_\tau = \mu.t'_\tau$. □

To denote system equivalence we also assume the notions of *strong* and *delay bisimilarity* [60, 96, 97], where the former is a stronger instance of the latter.

Definition 2.3 (Strong Bisimilarity)**.** A relation $\mathcal{R} \subseteq \text{Sys} \times \text{Sys}$ is a *strong bisimulation* iff whenever $(s, r) \in \mathcal{R}$ for every action μ, the following properties hold:

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

System states s and r are *strongly bisimilar*, denoted by $s \sim r$, iff they can be related by a *strong bisimulation*. □

Definition 2.4 (Delay Bisimilarity)**.** A relation $\mathcal{R} \subseteq \text{Sys} \times \text{Sys}$ is a *delay bisimulation* equivalence relation iff whenever $(s, r) \in \mathcal{R}$ for every action μ, the following transfer properties hold:

- $s \xrightarrow{\mu} s'$ implies there exists a *delayed* transition $r \stackrel{\hat{\mu}}{\Longrightarrow} r'$ so that $(s', r') \in \mathcal{R}$,
- $r \xrightarrow{\mu} r'$ implies there exists a *delayed* transition $s \stackrel{\hat{\mu}}{\Longrightarrow} s'$ so that $(s', r') \in \mathcal{R}$.

Two system states s and r are *delay bisimilar* (*a.k.a.* observationally equivalent), denoted by $s \approx r$, iff there exists a delay bisimulation that relates them. □

Syntax

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

Semantics

$$\begin{array}{c}[\![\mathsf{tt},\rho]\!] \stackrel{\text{def}}{=} \textsc{Sys} \qquad [\![\mathsf{ff},\rho]\!] \stackrel{\text{def}}{=} \varnothing \qquad [\![X,\rho]\!] \stackrel{\text{def}}{=} \rho(X)\\ [\![\textstyle\bigwedge_{i\in I}\varphi_i,\rho]\!] \stackrel{\text{def}}{=} \bigcap_{i\in I}[\![\varphi_i,\rho]\!] \qquad [\![\max X.\varphi,\rho]\!] \stackrel{\text{def}}{=} \bigcup\{S \mid S \subseteq [\![\varphi,\rho[X\mapsto S]]\!]\}\\ [\![\textstyle\bigvee_{i\in I}\varphi_i,\rho]\!] \stackrel{\text{def}}{=} \bigcup_{i\in I}[\![\varphi_i,\rho]\!] \qquad [\![\min X.\varphi,\rho]\!] \stackrel{\text{def}}{=} \bigcap\{S \mid [\![\varphi,\rho[X\mapsto S]]\!] \subseteq S\}\\ [\![[\{p,c\}]\varphi,\rho]\!] \stackrel{\text{def}}{=} \{s \mid \forall\alpha,r\cdot(s \stackrel{\alpha}{\Longrightarrow} r \text{ and } (\exists\sigma\cdot \mathsf{mtch}(p,\alpha){=}\sigma \text{ and } c\sigma \Downarrow \mathsf{true})) \text{ implies } r \in [\![\varphi\sigma,\rho]\!]\}\\ [\![\langle\{p,c\}\rangle\varphi,\rho]\!] \stackrel{\text{def}}{=} \{s \mid \exists\alpha,r,\sigma\cdot(s \stackrel{\alpha}{\Longrightarrow} r \text{ and } \mathsf{mtch}(p,\alpha){=}\sigma \text{ and } c\sigma \Downarrow \mathsf{true} \text{ and } r \in [\![\varphi\sigma,\rho]\!])\}\end{array}$$

Figure 2.2: The syntax and semantics for μHML.

Functions $\mathbf{bv}(p)$ and $\mathbf{fv}(c)$

$$\mathbf{bv}(p) \stackrel{\text{def}}{=} \left\{ x \;\middle|\; x \text{ is bound in } p. \right\} \quad \mathbf{fv}(c) \stackrel{\text{def}}{=} \left\{ x \;\middle|\; x \text{ is free in } c. \right\}$$

The matching function.

$$\mathsf{mtch}(p,\alpha) \stackrel{\text{def}}{=} \begin{cases} \{^{x}/_{v},^{y}/_{w}\} & \text{if } p = (x)?(y) \text{ and } \alpha = v?w\\ \{^{x}/_{v},^{y}/_{w}\} & \text{if } p = (x)!(y) \text{ and } \alpha = v!w\\ \cdots & \end{cases}$$

Figure 2.3: Auxiliary functions.

2.2 Hennessy Milner Logic with Recursion (μHML)

We consider a generalised version of μHML [11, 61, 75] that uses *symbolic actions* of the form $\{p, c\}$, defining a pattern p and a condition c. Symbolic actions abstract over visible actions using *data variables* $x, y, z \in \textsc{DVar}$ that occur free in the condition c or as binders in the pattern p, denoted as (x), where a *closed pattern* is one without free variables. Although symbolic actions can define any pattern p, we mainly use two types of patterns, namely, $(x)!(y)$ and $(x)?(y)$ that respectively denote an output and input action pattern, each of which binds variables x and y. We also use function $\mathbf{bv}(p)$ to denote the set of binding variables in a pattern p, and $\mathbf{fv}(c)$ to represent the set of free variables referenced in a condition c which we define in Figure 2.3.

We also define a (partial) *matching function* for *closed* patterns $\mathsf{mtch}(p, \alpha)$ in Figure 2.3 that (when successful) returns a substitution σ mapping variables in p to the corresponding values in α *i.e.,* if we instantiate every binder (x) in p with $\sigma(x)$ we obtain α. Although we only define this matching function with respect to input, output actions and patterns, it can easily be extended for other types of actions and patterns if necessary. The *filtering condition*, c, may refer to variables bound in p and is evaluated with respect to the substitutions returned by successful matches,

written as $c\sigma \Downarrow b$ (where $b \in \{\mathsf{true}, \mathsf{false}\}$) which follows standard condition evaluation semantics.

Put differently, a *closed* symbolic action, $\{p, c\}$, is one where p is closed and $\mathbf{fv}(c) \subseteq \mathbf{bv}(p)$; it denotes the *set* of actions that match pattern p with substitution σ, and satisfy condition $c\sigma$ *i.e.,* $[\![\{p, c\}]\!] \stackrel{\text{def}}{=} \{\, \alpha \mid \exists \sigma \cdot \mathsf{mtch}(p, \alpha) = \sigma \textit{ and } c\sigma \Downarrow \mathsf{true} \,\}$. The use of symbolic actions allows for more adequate reasoning about LTSs with infinitely many actions (*e.g.,* actions carrying data from infinite domains).

Example 2.1. Action $\{(x)!(y), y{=}1\}$ is correct since $\mathbf{fv}(y{=}1) \subseteq \mathbf{bv}((x)!(y))$ because $\mathbf{fv}(y{=}1) = \{y\}$ and $\mathbf{bv}((x)!(y)) = \{x, y\}$. However, $\{(x)!y, y = 1\}$ and $\{(x)!1, y = 1\}$ are incorrect since $\mathbf{fv}(y{=}1) \not\subseteq (\mathbf{bv}((x)!y) = \{x\})$. □

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 $[\![\{p_1, \mathsf{true}\}]\!] = [\![\{p_2, \mathsf{true}\}]\!]$.

Figure 2.2 presents the syntax of μHML that 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: where $\bigwedge_{i \in I} \varphi_i$ describes a *compound* conjunction, $\varphi_1 \wedge \ldots \wedge \varphi_n$, where $I = \{1, .., n\}$ is a finite set of indices, 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. Fixpoint variables, X, are assumed to be guarded by a prefixing modal operator *i.e.,* $[\{\alpha\}]X$ — it is well known that this assumption does not hinder the expressiveness of the logic (see [16, 102]).

The logic also uses the *necessity* (universal) and *possibility* (existential) modal operators defining symbolic actions, $[\{p, c\}]\varphi$ and $\langle\{p, c\}\rangle\varphi$, where the binders $\mathbf{bv}(p)$ bind free data variables in c and φ. A formula $[\{p, c\}]\varphi$ is satisfied by any system that *cannot* perform an action α that matches p and satisfies condition c. It is also satisfied by any system that performs a matching α with substitution σ, provided that *all* of the system's α-derivative states satisfy the continuation $\varphi\sigma$. By contrast, a formula $\langle\{p, c\}\rangle\varphi$ is only satisfied by systems that may perform an action α that matches p with a matching substitution σ so that $c\sigma$ holds, and that has an α-derivative that satisfies $\varphi\sigma$. To improve presentation, in Part II we occasionally use the notation $(\_)$ to denote "don't care" binders in the pattern p, whose variables are not referenced in c and φ.

The formulas in μHML are interpreted over the system powerset domain where $S \in \mathcal{P}(\text{Sys})$. The semantic definition of Figure 2.2, $[\![\varphi, \rho]\!]$, is given for *both* open and closed formulas. It employs a valuation from logical variables to sets of states, $\rho \in (\text{LVar} \to \mathcal{P}(\text{Sys}))$, which permits an inductive definition on the structure of the formulas; $\rho' = \rho[X \mapsto S]$ denotes a valuation where $\rho'(X) = S$ and $\rho'(Y) = \rho(Y)$ for all other $Y \neq X$.

As a shorthand, whenever a condition c in a symbolic action $\{p, c\}$ equates a bound variable to a specific value, we embed the equated value within the pattern, *e.g.,* $\{(x)!(y), x = \mathsf{a} \wedge y = 3\}$ becomes $\{\mathsf{a}!3, \mathsf{true}\}$, we also elide the condition when it is true, and just write $\{\mathsf{a}!3\}$. We refer to symbolic actions that only define a single visible action as *singleton symbolic actions*, *e.g.,* $\{\mathsf{a}!3\}$ is singleton since $[\![\{\mathsf{a}!3\}]\!] \stackrel{\text{def}}{=} \{\mathsf{a}!3\}$. Unless stated explicitly, 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 $s \in [\![\varphi]\!]$, and φ is *satisfiable*, whenever there exists a system r such that $r \in [\![\varphi]\!]$, *i.e.,* $[\![\varphi]\!] \neq \varnothing$.

Example 2.2. Consider the following μHML formulas.

$$\varphi_a \stackrel{\text{def}}{=} \max X.[\{\mathsf{a}!(x), x > 0\}]X \wedge \langle\{\mathsf{b}!(y), y > 0\}\rangle \mathsf{tt} \qquad \varphi_b \stackrel{\text{def}}{=} [\{\mathsf{a}!3\}]\mathsf{ff} \wedge \langle\{\mathsf{a}!3\}\rangle \mathsf{tt}$$

The recursive formula φ_a states that a system s is correct when it can output a number $x > 0$ on port a, and when *all* the states that are reachable after the system outputs a number $y > 0$ on port b (if any), satisfy φ_a as well. For instance, systems $\mathsf{a}!1.(\mathsf{b}!2.\mathsf{nil} + \mathsf{a}!1.\mathsf{b}!2.\mathsf{nil}) + \mathsf{b}!2.\mathsf{nil}$ and $\mathsf{b}!2.\mathsf{nil}$ satisfy φ_a, but $\mathsf{a}!1.(\mathsf{b}!2.\mathsf{nil} + \mathsf{a}!1.\mathsf{b}!2.\mathsf{nil}) \notin [\![\varphi_a]\!]$ since it cannot initially perform action $\mathsf{b}!2$. Since there exists a system that satisfies φ_a (*i.e.,* $\mathsf{b}!2.\mathsf{nil}$ and $\mathsf{a}!1.(\mathsf{b}!2.\mathsf{nil} + \mathsf{a}!1.\mathsf{b}!2.\mathsf{nil}) + \mathsf{b}!2.\mathsf{nil}$), we know that φ_a is *satisfiable*.

By contrast, formula φ_b is *not* satisfiable because it requires that a system can able perform an action $\mathsf{a}!3$, and at the same time, not be able to perform it. Due to this contradiction, there does not exist a system that satisfies φ_b. □

In [62], Hennessy and Milner proved a powerful result – now known as the Hennessy-Milner Theorem – that links the notion of strong bisimilarity to μHML. More specifically, it determines that strong bisimilar *image-finite* systems (as defined by Definition 2.5) satisfy the same set of μHML properties. 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.

Definition 2.5 (Image-finite LTS)**.** An LTS state is *image-finite* if for every action μ there are a *finite* number of μ-derivative states. An LTS is image-finite if so are all of its system states. □

Theorem 2.1 (Hennessy-Milner Theorem)**.** 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. □

This result [62] was originally given in relation to the *standard* Hennessy-Milner logic *i.e.,* a version of μHML that does not support recursion and only allows for defining concrete visible actions α in its modal operators (instead of symbolic actions). However, it still applies for our version of μHML for two reasons. First, since

$$\varphi \in \text{sHML} ::= \mathsf{tt} \mid \mathsf{ff} \mid \bigwedge_{i\in I} \varphi_i \mid [\{p,c\}]\varphi \mid X \mid \max X.\varphi$$
$$\psi \in \text{cHML} ::= \mathsf{tt} \mid \mathsf{ff} \mid \bigvee_{i\in I} \psi_i \mid \langle\{p,c\}\rangle\psi \mid X \mid \min X.\psi$$

Figure 2.4: The safety and co-safety fragments.

symbolic actions represent a set of visible actions *i.e.,* $[\![\{p,c\}]\!] \stackrel{\text{def}}{=} \{\alpha_1, \alpha_2, \ldots\}$, our symbolic modal operators can be easily encoded into standard ones *e.g.,* $[\{p,c\}]\varphi'$ and $\langle\{p,c\}\rangle\varphi'$ can be encoded as $[\alpha_1]\varphi' \wedge [\alpha_2]\varphi' \wedge \ldots$ and $\langle\alpha_1\rangle\varphi' \vee \langle\alpha_2\rangle\varphi' \vee \ldots$ respectively. Second, it was shown in [99, 100] that the proven result also applies when recursion is supported via greatest and least fixpoints.

In order to use this powerful result in our work, we will limit our study to systems that can be defined as *image-finite* LTSs.

2.2.1 Prominent μHML fragments in runtime monitoring

Any specification language that is considerably expressive can express properties that cannot be monitored for at runtime [2, 5, 39, 50, 55, 56, 77, 81, 89] – μHML is no exception. Several works [2, 5, 56] have focussed on identifying specific μHML fragments that represent a set of properties that can be verified at runtime using runtime verification monitors. The most prominent fragments are the *safety* and *co-safety* fragments, known as sHML and cHML respectively (defined in Figure 2.4). Informally, safety allows for defining a property requiring the system not to execute some specific behaviour, while co-safety is generally regarded as being the opposite as it requires that the system is able to exhibit a specific behaviour [46, 48].

The sHML syntax restricts the logic to truth (tt), falsehood (ff), conjunctions ($\bigwedge_{i\in I}\varphi$), the modal necessity ($[\{p,c\}]\varphi$) and only allows for recursion to be expressed through greatest fixpoints ($\max X.\varphi$) – the semantics for these constructs follows from that of Figure 2.2. This fragment only allows for expressing safety properties that define what constitutes *incorrect* system behaviour. Put differently, these properties specify the behaviour that a system *must not* exhibit.

Example 2.3. The sHML formula $[\{a\}][\{b\}]\mathsf{ff}$ states that a system is *correct* if it *cannot* perform an action a followed by an action b, *e.g.,* systems $a.\mathsf{nil}$ and $c.\mathsf{nil}$ are correct, unlike $a.b.\mathsf{nil}$ and $a.(c.\mathsf{nil} + b.\mathsf{nil})$. □

Dually, cHML expresses co-safety properties that define the system's *correct* behaviour. Specifically, it restricts the logic to truth (tt), falsehood (ff), disjunctions ($\bigvee_{i\in I}\varphi$), the modal possibility ($\langle\{p,c\}\rangle\varphi$) and recursion via least fixpoints ($\min X.\varphi$). This means that a cHML formula specifies the behaviour that a system *must* be able to perform; failure to do so results in violating the property.

Example 2.4. The cHML formula $\langle\{a\}\rangle\langle\{b\}\rangle\mathtt{tt}$ regards a system as being *correct* if it *can* perform an action a followed by an action b, *e.g.,* in this case systems $a.b.\mathsf{nil}$ and $a.(c.\mathsf{nil} + b.\mathsf{nil})$ are correct, while $a.\mathsf{nil}$ and $c.\mathsf{nil}$ are not. □

These fragments are known to be monitorable in the sense of runtime verification; it is well known [5, 56, 57] that runtime verification monitors can be used *detect* systems that satisfy cHML formulas or violate sHML properties. However, since runtime verification monitors are incapable of modifying the execution of a system, the enforceability of these μHML fragments and that of the full μHML, has not yet been explored.

2.2.2 The *after* function

As we aim to use our logic in conjunction with monitoring, it is sometimes useful to know the constraint that a system must satisfy after performing a number of steps. For instance, if $\alpha.\beta.s$ satisfies formula $[\{\alpha\}][\{\beta\}]\psi$ then its derivative s must also satisfy formula ψ after it performs actions α and β. We thus adopt an approach akin to Brzozowski's derivatives [27] and define the *after* function to denote how μHML formulas evolve in reaction to an action μ.

Definition 2.6. We define the function $\mathit{after}\colon(\mu\text{HML}\times\text{Act}\cup\{\tau\})\to\mu\text{HML}$ as:

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

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

□

When applied to a greatest or least fixpoint, the function unfolds the formula and reapplies itself to the unfolded equivalent $\varphi'\{^{\varphi}/_{X}\}$. Our assumption that formulas are guarded ensures that $\varphi'\{^{\varphi}/_{X}\}$ has fewer top level occurrences of greatest and least fixpoint operators than $\max X.\varphi'$ and $\min X.\varphi'$. In the case of conjunctions and disjunctions the function recurses for each individual branch. It returns formula $\psi\sigma$ when α matches successfully the symbolic action of a modal necessity or possibility that precedes ψ (where σ is created by the successful match). However, on an unsuccessful match, it returns $\mathtt{tt}$ when the modal operator is a necessity, and ff when it is a possibility, thereby signifying a trivial satisfaction and violation respectively. As no visible system action can alter truth and falsehood, they also

do not change in the *after* function. Silent τ actions do not affect the formula φ as well.

In addition to defining the *after* function, it is crucial to provide some kind of semantic justification that ensures that the formula produced by $\mathit{after}(\varphi, \alpha)$ truly reflects the constraint that a system must satisfy after performing action α. We thus introduce two semantic justifications given by Definitions 2.7 and 2.8 and we deem the *after* function to be justified for a sublogic $\mathcal{L} \subseteq \mu\text{HML}$ if it adheres to either one of these definitions.

Definition 2.7. The *after* function is *semantically justified* vis-a-vis the semantics of a logic $\mathcal{L} \subseteq \mu\text{HML}$ iff for every state s and s', formula $\varphi \in \mathcal{L}$ and action α, if $s \stackrel{\alpha}{\Longrightarrow} s'$ and $s \in [\![\varphi]\!]$ then $s' \in [\![\mathit{after}(\varphi, \alpha)]\!]$. □

Definition 2.8. The *after* function is also *justified* vis-a-vis the semantics of a logic $\mathcal{L} \subseteq \mu\text{HML}$ iff for every state r and r', formula $\psi \in \mathcal{L}$ and action α, if $r \stackrel{\alpha}{\Longrightarrow} r'$ and $r' \in [\![\mathit{after}(\psi, \alpha)]\!]$ then $r \in [\![\psi]\!]$. □

However, as supported by Propositions 2.1 and 2.2, it turns out that for the full μHML semantics, the *after* function can neither be semantically justified as per Definition 2.7 nor per Definition 2.8.

Proposition 2.1. There exist a system state s, μHML formula φ and action α so that $s \in [\![\varphi]\!]$, $s \stackrel{\alpha}{\Longrightarrow} s'$ and $s' \notin [\![\mathit{after}(\varphi, \alpha)]\!]$. □

Proof. We prove this proposition by using a counter example. Consider the μHML formula $\varphi_b \stackrel{\text{def}}{=} \langle\{a\}\rangle[\{b\}]\mathsf{ff}$ and system $s = a.b.\mathsf{nil} + a.c.\mathsf{nil}$. Although $s \in [\![\varphi_b]\!]$, when $s \stackrel{a}{\Longrightarrow} b.\mathsf{nil}$, the function $\mathit{after}(\varphi_b, a)$ reduces to $[\{b\}]\mathsf{ff}$ where $b.\mathsf{nil} \notin [\![[\{b\}]\mathsf{ff}]\!]$. □

Proposition 2.2. There exist a system state r, μHML formula ψ and action α so that $r' \in [\![\mathit{after}(\psi, \alpha)]\!]$, $r \stackrel{\alpha}{\Longrightarrow} r'$ and $r \notin [\![\psi]\!]$. □

Proof. Consider formula $\psi_b \stackrel{\text{def}}{=} [a]\langle b\rangle\mathsf{tt}$ and system $r = a.b.\mathsf{nil} + a.c.\mathsf{nil}$. Although $r \stackrel{a}{\Longrightarrow} b.\mathsf{nil}$ and $b.\mathsf{nil} \in [\![\langle b\rangle\mathsf{tt}]\!]$ (where $\mathit{after}(\psi_b, a) = \langle b\rangle\mathsf{tt}$), it turns out that $r \notin [\![\psi_b]\!]$. □

The *after* function can, however, be justified for the safety and co-safety subsets (sHML and cHML). We therefore justify our definition of the *after* function vis-a-vis the semantics of Figure 2.2 via Proposition 2.3 for sHML and via Proposition 2.4 for cHML. The proofs for these propositions are given in Appendix A on page 147.

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

Proposition 2.4. For every system state s and s', cHML formula φ and action α, if $s \stackrel{\alpha}{\Longrightarrow} s'$ and $s' \in [\![\mathit{after}(\varphi, \alpha)]\!]$ then $s \in [\![\varphi]\!]$. □

Example 2.5. Consider the sHML formula $\varphi_s \stackrel{\text{def}}{=} \max X.[\{a\}]X \wedge [\{b\}]\mathsf{ff} \wedge [\{c\}]X$ and system $s = a.c.\mathsf{nil} + c.a.\mathsf{nil}$. Formula φ_s states that a system is *incorrect* if either the initial system state or any other state reachable over a sequence of a or c actions, can perform an action b. Since $s \in [\![\varphi_s]\!]$, Proposition 2.3 ensures that for every action α and α-derivative state s', if $s \stackrel{\alpha}{\Longrightarrow} s'$ then $s' \in [\![\mathit{after}(\varphi_s, \alpha)]\!]$. This is in fact *true* since s can only reduce either via $s \stackrel{a}{\longrightarrow} c.\mathsf{nil}$ where $c.\mathsf{nil} \in [\![\mathit{after}(\varphi_s, a)]\!] (= [\![\varphi_s]\!])$, or via $s \stackrel{c}{\longrightarrow} a.\mathsf{nil}$ in which case $a.\mathsf{nil} \in [\![\mathit{after}(\varphi_s, c)]\!] (= [\![\varphi_s]\!])$.

Similarly, consider the cHML formula $\varphi_c \stackrel{\text{def}}{=} \min X.\langle\{a\}\rangle\mathsf{tt} \vee \langle\{c\}\rangle X$ which states that a system is *correct* if either the initial state, or a state reachable over a sequence of c actions, can perform action a. From Proposition 2.4 we know that whenever s reduces over some α to s' where $s' \in [\![\mathit{after}(\varphi_c, \alpha)]\!]$ then $s \in [\![\varphi_c]\!]$ as well. In fact, when $s \stackrel{a}{\longrightarrow} c.\mathsf{nil}$ and $c.\mathsf{nil} \in [\![\mathit{after}(\varphi_c, a)]\!]$ (which is true since $\mathit{after}(\varphi_c, a) = \mathsf{tt}$), $s \in [\![\varphi_c]\!]$ also holds. The same applies in the case when we assume that $s \stackrel{c}{\longrightarrow} a.\mathsf{nil}$ and $a.\mathsf{nil} \in [\![\mathit{after}(\varphi_c, c)]\!]$. □

We abuse notation and lift the *after* function to (explicit) traces in the obvious way, *i.e.,* $\mathit{after}(\varphi, t_\tau)$ is equal to $\mathit{after}(\mathit{after}(\varphi, \mu), u_\tau)$ when $t_\tau = \mu u_\tau$ and to φ when $t_\tau = \varepsilon$.

Part I

Unidirectional Enforcement

3. A unidirectional enforcement model

Figure 3.1: Our unidirectional enforcement setup.

In our unidirectional enforcement approach we follow the seminal work in RE [23, 46, 50, 78] and adopt a trace based view of the runtime behaviour of the SuS which can be altered using the conventional monitor transformations, *i.e.,* action insertion, suppression and replacement. Figure 3.1 shows how the monitor transforms the system run $\alpha_1.\alpha_2.\alpha_3$ into $\beta_1.\tau.\beta_4.\alpha_3$. Specifically, the monitor first replaces α_1 by β_1, then suppresses α_2 into τ and finally inserts action β_4 before allowing action α_3 to go through without any modifications.

In this chapter we formalise this unidirectional instrumentation approach along with the behaviour of enforcement transducers. We also prove that our instrumentation model supports for unzipping and zipping functionality that permit us to decompose a monitored execution into two individual executions, and to recompose them back together as necessary. Beforehand, however, we will present two example systems that we will be referring to throughout the first part of this dissertation.

Example 3.1. Consider two systems (a good system, s_{g}, and a bad one, s_{b}) implementing a server that repeatedly accepts *requests* on port a and outputs an *answer* on the same port in response; the serviced request is logged by an output on a special port b. The server terminates once it accepts a *close* request from port b.

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

More specifically, for every request (a?req), system s_{g} outputs a *single* answer (a!ans) and logs it (b!log), whereas s_{b} occasionally produces *multiple* answers for a given request before behaving as per s_{g} (see the underlined branch in the description of s_{b} above). Both systems terminate with b?cls. □

3.1 The model

In Figure 3.2, we formalise our operational mechanism for enforcing properties in a unidirectional setting (that we highlighted in Figure 3.1) in terms of the (symbolic) transducers $m, n \in \textsc{Trn}$. Transducers are a special kind of monitors that define *symbolic transformation triples*, $\{p, c, p'\}$, consisting of the action *pattern* p, the *filtering condition* c and a *transformation pattern* p'. Conceptually, the action pattern and condition determine the range of visible actions upon which the transformation should be applied, while the transformation pattern specifies the kind of transformation that should be applied, *i.e.,* a suppression, replacement or insertion.

The symbolic transformation patterns p and p' are extended versions of those definable in symbolic actions. In addition to describing visible system actions, these extended patterns may also specify the symbol $\bullet$ and use it as follows. When $p = \bullet$, the transformation pattern represents a point where the monitor can act independently from the system to insert the action specified by p', but when $p' = \bullet$, it represents the suppression of the action specified by p. For our symbolic transformations we also assume a well-formedness constraint where, for every $\{p, c, p'\}.m$, either p or p' is $\bullet$ but not both *e.g.,*$\{\bullet, \mathsf{true}, \mathsf{a}?v\}$ and $\{(x)!(y), \mathsf{true}, \bullet\}$ are well-formed since only one of their patterns is $\bullet$. The matching function is lifted to these patterns in the obvious way, where $\mathsf{mtch}(\bullet, \bullet) = \varnothing$.

The syntax of our transducers assumes a well-formedness constraint where for every $\{p, c, p'\}.m$, $\mathbf{bv}(c) \cup \mathbf{bv}(p') = \varnothing$. The monitor transition rules in Figure 3.2 assume closed terms, *i.e.,* for every *transformation-prefix transducer* of the form $\{p, c, p'\}.m$, it is required that $\big(\mathbf{fv}(c) \cup \mathbf{fv}(p') \cup \mathbf{fv}(m)\big) \subseteq \mathbf{bv}(p) \cup \mathcal{V}$, where $\mathcal{V}$ is the set of variables that are bound by priorly defined transformation prefixes. Each transformation-prefix transducer yields an LTS with labels of the form $\gamma \blacktriangleright \gamma'$, where $\gamma, \gamma' \in (\textsc{Act} \cup \{\bullet\})$. Intuitively, transition $m \xrightarrow{\gamma \blacktriangleright \gamma'} n$ denotes the way that a transducer in state m *transforms* action γ into γ' and reduces to state n. In this sense, the transducer action $\alpha \blacktriangleright \beta$ represents the *replacing* of α by β, and $\alpha \blacktriangleright \alpha$ denotes the *identity* transformation. Cases $\alpha \blacktriangleright \bullet$ and $\bullet \blacktriangleright \alpha$ respectively encode the *suppression* and *insertion* transformations of action α.

The key transition rule in Figure 3.2 is $\textsc{eTrn}$. It states that the transformation-prefix transducer $\{p, c, p'\}.m$ transforms action γ into a (potentially) different action γ'

Syntax

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

Dynamics

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

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

Instrumentation

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

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

Figure 3.2: A model for transducers and unidirectional enforcement instrumentation.

and reduces to state $m\sigma$, whenever γ matches pattern p, *i.e.*, $\mathsf{mtch}(p,\gamma){=}\sigma$, and satisfies condition c, *i.e.*, $c\sigma \Downarrow \mathsf{true}$. Action γ' results from instantiating the free variables in p' with the corresponding values mapped by σ, *i.e.*, $\gamma'{=}p\sigma$. The remaining rules eSel and eRec respectively define the standard selection and recursion operations. A sum of transducers $\sum_{i\in I} m_i$ can reduce via eSel to some n_j over some action $\gamma\blacktriangleright\gamma'$, whenever there exists a transducer m_j in the summation that reduces to n_j over the same action. Rule eRec enables a recursion transducer $\mathsf{rec}\, X.m$ to reduce to some n when its unfolded instance $m\{{}^{\mathsf{rec}\, X.m}/_{X}\}$ reduces to n as well. We encode the identity monitor, id, as a recursive monitor defining identity transformations that match every action.

Figure 3.2 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 $\textsc{Act}\cup\{\tau\}$. Concretely, rule iTrn states that when a system s transitions with an observable action α to s' and the transducer m can *transform* α into β and transition to n, the instrumented system $m[s]$ transitions with action β to $n[s']$. 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* an action α independently of the behaviour of s. Since

the monitor can act independently, action insertion may potentially introduce non-deterministic behaviour in the resulting composite system. For instance, a monitor might be required to non-deterministically choose between inserting two or more actions, *e.g.,* $\{\bullet, \alpha\}.m_1 + \{\bullet, \beta\}.m_2$ must non-deterministically insert either action α and reduce to m_1, or β and transition to m_2. Due to their potential of introducing non-determinism, insertions are therefore more complex to understand and work with when compared to other enforcement transformations and so one should pay extra care when using them.

Rule IDEF is analogous to standard monitor instrumentation rules for premature termination of the transducer [2, 53, 54, 56], and accounts for underspecification of transformations. Thus, if a system s transitions with an observable action α to s', and the transducer m is neither able to transform it ($m \overset{\alpha}{\not\rightarrow}$), nor 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 (where id is shorthand for $\mathsf{rec}\, Y.\{(x)!(y), \mathsf{true}, x!y\}.Y + \{(x)?(y), \mathsf{true}, x?y\}.Y$ and $m \overset{\gamma}{\not\rightarrow}$ means $\nexists \gamma', n \cdot m \xrightarrow{\gamma \blacktriangleright \gamma'} n$). Finally, when s transitions with a silent action, rule IASY allows it to do so independently of the transducer.

We occasionally use the notation $m \xrightarrow{\kappa} m'$ to represent a sequence of monitor transformations, *i.e.,* $m \xrightarrow{\gamma_1 \blacktriangleright \gamma_1'} \ldots \xrightarrow{\gamma_n \blacktriangleright \gamma_n'} m'$ where κ is a trace of transformations $(\gamma_1 \blacktriangleright \gamma_1') \ldots (\gamma_n \blacktriangleright \gamma_n')$ for some $n > 0$ (but $\kappa{=}\varepsilon$ when $n = 0$).

Example 3.2. Consider the insertion and replacement transducers given below:

$$m_{\mathbf{i}} \overset{\text{def}}{=} \{(x)?\mathsf{req}, \mathsf{true}, x?\mathsf{req}\}.\{\bullet, \mathsf{true}, x!\mathsf{ans}\}.\mathsf{id}$$
$$m_{\mathbf{r}} \overset{\text{def}}{=} \mathsf{rec}\, X.\,(\{(x)?(y), \mathsf{true}, \mathsf{b}?y\}.X + \{(x)!(y), \mathsf{true}, \mathsf{b}!y\}.X\,).$$

When instrumented with a system, $m_{\mathbf{i}}$ inserts action $x!\mathsf{ans}$, after the system inputs a request $x?\mathsf{req}$ from some port x. It then behaves as the identity transducer. Concretely, the system $m_{\mathbf{i}}[s_{\mathsf{b}}]$, where s_{b} is from Example 3.1, can only start the computation as follows:

$$m_{\mathbf{i}}[s_{\mathsf{b}}] \xrightarrow{\mathsf{a?req}} \{\bullet, \mathsf{true}, \mathsf{a!ans}\}.\mathsf{id}[s_{\mathsf{b}}'] \xrightarrow{\mathsf{a!ans}} \mathsf{id}[s_{\mathsf{b}}'] \xrightarrow{\mathsf{a!ans}} \ldots$$

(where $s_{\mathsf{b}}' = \mathsf{a!ans.b!log}.s_{\mathsf{b}} + \mathsf{a!ans.a!ans.b!log}.s_{\mathsf{g}}$). By contrast, $m_{\mathbf{r}}$ intercepts all the system's inputs and outputs and redirects them through the special port b. For instance, we have that:

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

Consider now the following suppression transducers $m_{\mathbf{s}}$, $m_{\mathbf{t}}$ and $m_{\mathbf{et}}$ for actions

made on any port except b:

$$m_{\mathbf{s}} \stackrel{\text{def}}{=} \mathsf{rec}\, X.\big(\{(x)?\mathsf{req}, x \neq \mathsf{b}, x?\mathsf{req}\}.X + \{(x)!\mathsf{ans}, x \neq \mathsf{b}, \bullet\}.X + \{\mathsf{b}!\mathsf{log}, \mathsf{true}, \mathsf{b}!\mathsf{log}\}.X\big)$$

$$m_{\mathbf{t}} \stackrel{\text{def}}{=} \mathsf{rec}\, X.\big(\{(x)?\mathsf{req}, x \neq \mathsf{b}, x?\mathsf{req}\}.\{x!\mathsf{ans}, \mathsf{true}, x!\mathsf{ans}\}.\big(\{x!\mathsf{ans}, \mathsf{true}, \bullet\}.\mathsf{sup} + \{\mathsf{b}!\mathsf{log}, \mathsf{true}, \mathsf{b}!\mathsf{log}\}.X\big)\big)$$

$$m_{\mathbf{et}} \stackrel{\text{def}}{=} \mathsf{rec}\, X.\big(\{(x)?\mathsf{req}, x \neq \mathsf{b}, x?\mathsf{req}\}.\{x!\mathsf{ans}, \mathsf{true}, x!\mathsf{ans}\}.\mathsf{rec}\, Y.\big(\{x!\mathsf{ans}, \mathsf{true}, \bullet\}.Y + \{\mathsf{b}!\mathsf{log}, \mathsf{true}, \mathsf{b}!\mathsf{log}\}.X\big)\big)$$

where sup is a recursive monitor that recursively suppresses every system action. Monitor $m_{\mathbf{s}}$ suppresses every answer on ports other than b, and continues to do so after every request on such ports. When instrumented with s_{b} from Example 3.1, we can observe the following behaviour:

$$m_{\mathbf{s}}[s_{\mathsf{b}}] \xrightarrow{\mathsf{a}?\mathsf{req}} m_{\mathbf{s}}[s'_{\mathsf{b}}] \xrightarrow{\tau} m_{\mathbf{s}}[\mathsf{b}!\mathsf{log}.s_{\mathsf{b}}] \xrightarrow{\mathsf{b}!\mathsf{log}} m_{\mathbf{s}}[s_{\mathsf{b}}] \xrightarrow{\mathsf{a}?\mathsf{req}} m_{\mathbf{s}}[s'_{\mathsf{b}}] \xrightarrow{\tau} m_{\mathbf{s}}[\mathsf{b}!\mathsf{log}.s_{\mathsf{b}}] \ldots$$

Note that $m_{\mathbf{s}}$ does not specify a transformation behaviour for when the monitored system inputs a close request. The instrumentation handles this underspecification by defaulting to the identity transducer; in the case of s_{b} we get $m_{\mathbf{s}}[s_{\mathsf{b}}] \xrightarrow{\mathsf{b}?\mathsf{cls}} \mathsf{id}[\mathsf{nil}]$. Monitors $m_{\mathbf{t}}$ and $m_{\mathbf{et}}$ perform slightly more elaborate transformations. Concretely, for interactions on ports other than b, $m_{\mathbf{t}}$ suppresses the first consecutive answer that is output by the SuS following a serviced request (*i.e.,* a?req followed by a!ans) and reduces to sup which keeps on suppressing every subsequent action. Hence, for s_{b} we can observe the following behaviour:

$$\begin{aligned} m_{\mathbf{t}}[s_{\mathsf{b}}] &\xRightarrow{\mathsf{a}?\mathsf{req}\cdot\mathsf{a}!\mathsf{ans}} \big(\{\mathsf{a}!\mathsf{ans}, \mathsf{true}, \bullet\}.\mathsf{sup} + \{\mathsf{b}!\mathsf{log}, \mathsf{true}, \mathsf{b}!\mathsf{log}\}.m_{\mathbf{t}}\big)[\mathsf{a}!\mathsf{ans}.\mathsf{b}!\mathsf{log}.s_{\mathsf{g}}] \\ &\xrightarrow{\tau} \mathsf{sup}[\mathsf{b}!\mathsf{log}.s_{\mathsf{g}}] \xrightarrow{\tau} \mathsf{sup}[s_{\mathsf{g}}] \xrightarrow{\tau} \mathsf{sup}[\mathsf{a}!\mathsf{ans}.\mathsf{b}!\mathsf{log}.s_{\mathsf{g}}] \ldots \end{aligned}$$

By contrast, monitor $m_{\mathbf{et}}$ is more selective when applying its suppressions. In fact, it only suppresses the redundant consecutive answers that occur after a serviced request. Hence, when $m_{\mathbf{et}}$ is instrumented with s_{b}, the resulting composite system behaves as follows:

$$\begin{aligned} m_{\mathbf{et}}[s_{\mathsf{b}}] &\xRightarrow{\mathsf{a}?\mathsf{req}\cdot\mathsf{a}!\mathsf{ans}} \mathsf{rec}\, Y.\big(\{\mathsf{a}!\mathsf{ans}, \mathsf{true}, \bullet\}.Y + \{\mathsf{b}!\mathsf{log}, \mathsf{true}, \mathsf{b}!\mathsf{log}\}.m_{\mathbf{et}}\big)[\mathsf{a}!\mathsf{ans}.\mathsf{b}!\mathsf{log}.s_{\mathsf{g}}] \\ &\xrightarrow{\tau} \mathsf{rec}\, Y.\big(\{\mathsf{a}!\mathsf{ans}, \mathsf{true}, \bullet\}.Y + \{\mathsf{b}!\mathsf{log}, \mathsf{true}, \mathsf{b}!\mathsf{log}\}.m_{\mathbf{et}}\big)[\mathsf{b}!\mathsf{log}.s_{\mathsf{g}}] \\ &\xrightarrow{\mathsf{b}!\mathsf{log}} m_{\mathbf{et}}[s_{\mathsf{g}}] \xRightarrow{\mathsf{a}?\mathsf{req}\cdot\mathsf{a}!\mathsf{ans}\cdot\mathsf{b}!\mathsf{log}} m_{\mathbf{et}}[s_{\mathsf{g}}]. \end{aligned}$$

□

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.,* $\underline{(x)!(y)}$ denotes $x!y$. As shorthand notation, we elide the transformation pattern p' in a

$$\mathit{zip}(t,\kappa) = \begin{cases} \varepsilon & \text{if } t = \varepsilon \text{ and } \kappa = \varepsilon \\ \mathit{zip}(t',\kappa') & \text{if } t = \alpha t' \text{ and } \kappa = (\alpha \triangleright \bullet)\kappa' \\ \beta\mathit{zip}(t',\kappa') & \text{if } t = \alpha t' \text{ and } \kappa = (\alpha \triangleright \beta)\kappa' \\ \alpha\mathit{zip}(t,\kappa') & \text{if } \kappa = (\bullet \triangleright \alpha)\kappa' \end{cases}$$

Figure 3.3: The *zip* function.

transducer $\{p, c, p'\}.m$ whenever $p'{=}\underline{p}$ and simply write $\{p, c\}.m$. Similarly, we elide the filtering condition c whenever it is true. This allows us to express $m_{\mathbf{et}}$ from Example 3.2 as $\mathsf{rec}\, X.\big(\{(x)?\mathsf{req}, x \neq \mathsf{b}\}.\{x!\mathsf{ans}\}.\mathsf{rec}\, Y.\big(\{x!\mathsf{ans}, \bullet\}.Y + \{\mathsf{b}!\mathsf{log}\}.X\big)\big)$.

3.2 Zipping and unzipping

In our forthcoming work on unidirectional enforcement, we find it useful to have a way to decompose and recompose the behaviour of a composite system – this is commonly known as *unzipping* and *zipping* [7, 42]. Specifically, unzipping decomposes the composite behaviour $m[s] \stackrel{u}{\Longrightarrow} m'[s']$ into two separate computations, namely one for the monitor $m \stackrel{\kappa}{\Longrightarrow} m'$, and one for the system $s \stackrel{t}{\Longrightarrow} s'$. Zipping can then be used to reconstruct the composite behaviour from these two individual computations.

In order to prove that zipping and unzipping is supported by our enforcement model, we first define the *zip* (partial) function in Figure 3.3. The *zip* function analyses a system trace t and a monitor transformation trace κ in order to reconstruct a trace denoting the composite behaviour attained when t is scrutinised and transformed by a monitor that executes κ. In a sense, the *zip* function mimics the way the instrumentation rules interpret the transformations of the monitor in respect to the actions performed by the SuS. For instance, when the system trace is prefixed by an action α that is suppressed in the monitor's trace, *i.e.*, $t = \alpha t'$ and $\kappa = (\alpha \triangleright \bullet)\kappa'$, the function recurses (with the suffixes t' and κ') without adding α to the resulting composite trace. Similarly, if $t = \alpha t'$ and $\kappa = (\alpha \triangleright \beta)\kappa'$ it mimics the replacement of action α into β by adding the latter to the resulting trace before recursing. It also simulates action insertion by adding α to the composite trace when $\kappa = (\bullet \triangleright \alpha)\kappa'$. The *zip* function stops recursing when both t and κ are empty. Since *zip* is a *partial* function, it does not return a value for any other case of t and κ apart from the specified ones.

Using the *zip* function we can now show that the following results hold:

Proposition 3.1 (Unzipping)**.** For the monitored instances $m[s]$, $m'[s']$ and transformation trace κ, if $m[s] \stackrel{u}{\Longrightarrow} m'[s']$ then either

(a) $u = \textit{zip}(t, \kappa)$ and $m \stackrel{\kappa}{\Longrightarrow} m'$ and $s \stackrel{t}{\Longrightarrow} s'$; or

(b) $u = \textit{zip}(t, \kappa); \alpha t'$ and $m \stackrel{\kappa}{\Longrightarrow} m'' \stackrel{\alpha \blacktriangleright \gamma}{\not\longrightarrow}$ and $m'' \stackrel{\bullet}{\not\longrightarrow}$ and $s \stackrel{t;\alpha t'}{\Longrightarrow} s'$ and $m' = \mathsf{id}$. □

Proposition 3.2 (Zipping)**.** For any monitor m, m', system s, s', traces t, u, and transformation trace κ,

(a) if $m \stackrel{\kappa}{\Longrightarrow} m'$ and $s \stackrel{t}{\Longrightarrow} s'$ and $\textit{zip}(t, \kappa) = u$ then $m[s] \stackrel{u}{\Longrightarrow} m'[s']$.

(b) if $m \stackrel{\kappa}{\Longrightarrow} m' \stackrel{\alpha \blacktriangleright \gamma}{\not\longrightarrow}$ and $m' \stackrel{\bullet}{\not\longrightarrow}$ and $s \stackrel{t;\alpha t'}{\Longrightarrow} s'$ and $\textit{zip}(t, \kappa) = u$ then $m[s] \stackrel{u;\alpha t'}{\Longrightarrow} \mathsf{id}[s']$. □

On the one hand, Proposition 3.1 states that for a monitored execution $m[s] \stackrel{u}{\Longrightarrow} m'[s']$, there must exists a system trace t and a transformation trace κ so that $\textit{zip}(t, \kappa)$ produces either (a) u exactly, or (b) a prefix of u, *i.e.*, $u = \textit{zip}(t, \kappa); \alpha t'$. In the first case, (a), the composite behaviour can be elegantly decomposed into $m \stackrel{\kappa}{\Longrightarrow} m'$ and $s \stackrel{t}{\Longrightarrow} s'$. In the second case, (b), after executing κ, the monitor m reduces into a state m'' from which it can neither transform the system's visible action α, nor insert an action, *i.e.*, $m \stackrel{\kappa}{\Longrightarrow} m'' \stackrel{\alpha \blacktriangleright \gamma}{\not\longrightarrow}$ and $m'' \stackrel{\bullet}{\not\longrightarrow}$. This indicates that the monitor has defaulted to id *i.e.*, $m' = \mathsf{id}$, and hence the remaining composite behaviour $\alpha t'$ is identical to that of the SuS, *i.e.*, $s \stackrel{t;\alpha t'}{\Longrightarrow} s'$. On the other hand, Proposition 3.2 proves that the decomposed behaviours of the monitor and the SuS can be recomposed back together. The proofs for these propositions are given in Appendix B.1 on page 151.

3.3 Summary

In this chapter we have looked into unidirectional enforcement and sought to formalise this type of enforcement instrumentation. Particularly, we have presented:

(i) the transducer model $m \in \textsc{Trn}$ of Figure 3.2, defining how enforcement transducers behave at runtime,

(ii) the instrumentation model $m[s]$ of Figure 3.2, for achieving unidirectional enforcement, and

(iii) Propositions 3.1 and 3.2 (unzipping and zipping) proving that the behaviour of a composite system can be decomposed and recomposed as required.

4. Enforceability in a unidirectional context

In this chapter we investigate what it means for a logic to be enforceable. We start by introducing the notion of enforceability and define what it takes for a monitor to adequately enforce a logical formula. In this regard, we give three different definitions, namely, Definitions 4.4, 4.6 and 4.8, and show how they vary from one another. We then present the concept of optimal enforcement that assesses the level of intrusiveness of a monitor, and assists in finding the least intrusive one. Although these definitions are parametrisable with respect to any instrumentation relation, in this chapter we motivate them vis-a-vis the unidirectional enforcement framework of Figure 3.2 from Chapter 3.

Beforehand, however, we will present two μHML formulas that we will use as a running example throughout the first part of this thesis.

Example 4.1. Recall the request response server implementations s_{g} and s_{b} of Example 3.1 (restated below).

$$s_{\mathsf{g}} = \mathsf{rec}\, X.\big(\mathsf{a?req.(a!ans.b!log.}X\mathsf{)} + \mathsf{b?cls.nil}\big)$$

$$s_{\mathsf{b}} = \mathsf{rec}\, X.\big(\mathsf{a?req.(a!ans.b!log.}X \;+\; \mathsf{a!ans.a!ans.b!log.}s_{\mathsf{g}}\mathsf{)} + \mathsf{b?cls.nil}\big)$$

Using our logic μHML we can specify that a request on port a cannot be followed by two consecutive answers on that port.

$$\varphi_0 \stackrel{\text{def}}{=} \mathsf{max}\, X.[\{\mathsf{a?req}\}][\{\mathsf{a!ans}\}]([\{\mathsf{a!ans}\}]\mathsf{ff} \wedge [\{\mathsf{b!log}\}]X)$$

Formula φ_0 thus defines an invariant property ($\mathsf{max}\, X.\,(\cdots)$) requiring that whenever the system interacting on port a outputs an answer following a request, it cannot output a subsequent answer, *i.e.,* $[\{\mathsf{a!ans}\}]\mathsf{ff}$, unless it logs the response and inputs another request beforehand, in which case the formula recurses, *i.e.,* $[\{\mathsf{b!log}\}]X$.

Using symbolic actions, we can generalise φ_0 by requiring the property to hold for *any* interaction happening on *any* port *except* b (as this port is assumed to have

a special status).

$$\varphi_1 \stackrel{\text{def}}{=} \mathsf{max}\, X.[\{(x)?\mathsf{req}, x{\neq}\mathsf{b}\}][\{x!\mathsf{ans}\}]([\{x!\mathsf{ans}\}]\mathsf{ff}\wedge[\{\mathsf{b}!\mathsf{log}\}]X)$$

In φ_1, $(x)?\mathsf{req}$ binds the free occurrences of x found in $x{\neq}\mathsf{b}$ and in the continuation formula $[\{x!\mathsf{ans}\}]([\{x!\mathsf{ans}\}]\mathsf{ff}\wedge[\{\mathsf{b}!\mathsf{log}\}]X)$. The value of x is later rebound when the formula recurses on X. This means that subsequent references to x in the unfolded formula are replaced by the (potentially) new value bound to x. Using the semantics in Figure 2.2 of Chapter 2, one can check that $s_{\mathsf{g}}{\in}[\![\varphi_1]\!]$, whereas $s_{\mathsf{b}}{\notin}[\![\varphi_1]\!]$ since $s_{\mathsf{b}} \xrightarrow{\mathsf{a}?\mathsf{req}} \cdot \xrightarrow{\mathsf{a}!\mathsf{ans}} \cdot \xrightarrow{\mathsf{a}!\mathsf{ans}} s_{\mathsf{g}}$. □

4.1 Enforceability

The *enforceability* of a logical formula rests on the relationship between the semantic behaviour specified by the formula on the one hand, and the ability of the operational mechanism (*e.g.,* the transducers and instrumentation of Chapter 3) to enforce the specified behaviour on the other. We thus formally define the notion of enforceability as follows.

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

Definition 4.1 relies on the meaning of "m *adequately enforces* φ". Although several meanings can be given, it is reasonable to expect that a suitable definition should be applicable to *any* system that can be instrumented with monitor m. In particular, one should at least expect *soundness*, that is, if the property of interest φ is *satisfiable*, *i.e.,* $[\![\varphi]\!] \neq \varnothing$, then the composite system, $m[s]$, should satisfy φ for *every* possible LTS and system state s.

Definition 4.2 (Sound Enforcement)**.** Monitor m *soundly enforces* a satisfiable formula φ, denoted as $\mathsf{senf}(m, \varphi)$, iff $m[s] \in [\![\varphi]\!]$, for all LTSs $\langle \textsc{Sys}, \textsc{Act} \cup \{\tau\}, \rightarrow \rangle$ and system states $s \in \textsc{Sys}$. □

Example 4.2. In general, showing that a monitor m soundly enforces a formula φ requires showing that for *every* possible system state s, the instrumented system $m[s]$ satisfies φ. However, in this example we give an intuition based on systems s_{g} and s_{b} from Example 3.1. So recall the monitors presented in Example 3.2 from Chapter 3 (restated below), and φ_1 from Example 4.1 where $s_{\mathsf{g}} \in [\![\varphi_1]\!]$ (hence φ_1 is satisfiable) and $s_{\mathsf{b}} \notin [\![\varphi_1]\!]$.

$$m_{\mathbf{i}} \stackrel{\text{def}}{=} \{(x)?\mathsf{req}\}.\{\bullet, x!\mathsf{ans}\}.\mathsf{id}$$
$$m_{\mathbf{r}} \stackrel{\text{def}}{=} \mathsf{rec}\, X.\,(\{(x)?(y), \mathsf{b}?y\}.X + \{(x)!(y), \mathsf{b}!y\}.X\,)$$
$$m_{\mathbf{s}} \stackrel{\text{def}}{=} \mathsf{rec}\, X.\big(\{(x)?\mathsf{req}, x \neq \mathsf{b}\}.X + \{(x)!\mathsf{ans}, x \neq \mathsf{b}, \bullet\}.X + \{\mathsf{b}!\mathsf{log}\}.X\big)$$
$$m_{\mathbf{t}} \stackrel{\text{def}}{=} \mathsf{rec}\, X.\big(\{(x)?\mathsf{req}, x \neq \mathsf{b}\}.\{x!\mathsf{ans}\}.\big(\{x!\mathsf{ans}, \bullet\}.\mathsf{sup} + \{\mathsf{b}!\mathsf{log}\}.X\big)\big)$$
$$m_{\mathbf{et}} \stackrel{\text{def}}{=} \mathsf{rec}\, X.\big(\{(x)?\mathsf{req}, x \neq \mathsf{b}\}.\{x!\mathsf{ans}\}.\mathsf{rec}\, Y.\big(\{x!\mathsf{ans}, \bullet\}.Y + \{\mathsf{b}!\mathsf{log}\}.X\big)\big)$$

For monitors $m_{\mathbf{i}}$, $m_{\mathbf{r}}$, $m_{\mathbf{s}}$, $m_{\mathbf{t}}$ and $m_{\mathbf{et}}$ we have that:

- $m_{\mathbf{i}}[s_{\mathsf{b}}] \notin [\![\varphi_1]\!]$, since $m_{\mathbf{i}}[s_{\mathsf{b}}] \xrightarrow{\mathsf{a}?\mathsf{req}} (\{\bullet, \mathsf{a}!\mathsf{ans}\}.\mathsf{id})[s'_{\mathsf{b}}] \xrightarrow{\mathsf{a}!\mathsf{ans}} \mathsf{id}[s'_{\mathsf{b}}] \xrightarrow{\mathsf{a}!\mathsf{ans}} \mathsf{id}[s_{\mathsf{b}}]$. This counter-example implies that $\neg\mathsf{senf}(m_{\mathbf{i}}, \varphi_1)$.
- $m_{\mathbf{r}}[s_{\mathsf{g}}] \in [\![\varphi_1]\!]$ and $m_{\mathbf{r}}[s_{\mathsf{b}}] \in [\![\varphi_1]\!]$. 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 that monitor $m_{\mathbf{r}}$ is composed with, we can conclude that $\mathsf{senf}(m_{\mathbf{r}}, \varphi_1)$.
- $m_{\mathbf{s}}[s_{\mathsf{g}}] \in [\![\varphi_1]\!]$ and $m_{\mathbf{s}}[s_{\mathsf{b}}] \in [\![\varphi_1]\!]$ because the resulting instrumented systems never produce $x!\mathsf{ans}$ for any $x \neq \mathsf{b}$. We can thus conclude that $\mathsf{senf}(m_{\mathbf{s}}, \varphi_1)$.
- $m_{\mathbf{t}}[s_{\mathsf{g}}] \in [\![\varphi_1]\!]$ and $m_{\mathbf{t}}[s_{\mathsf{b}}] \in [\![\varphi_1]\!]$, since the monitor starts suppressing every system action from the point that a system performs a redundant answer. Hence, since it suppresses the invalid action along with every subsequent action, we can deduce that $\mathsf{senf}(m_{\mathbf{t}}, \varphi_1)$.
- $m_{\mathbf{et}}[s_{\mathsf{g}}] \in [\![\varphi_1]\!]$ and $m_{\mathbf{et}}[s_{\mathsf{b}}] \in [\![\varphi_1]\!]$. Since the resulting instrumentation suppresses consecutive answers (if any) after any number of serviced requests on any port other than b, we can conclude that $\mathsf{senf}(m_{\mathbf{et}}, \varphi_1)$. □

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 3.2. It manages to suppress the violating executions of system s_{b}, thereby bringing it in line with property φ_1. However, it also needlessly modifies the behaviour of s_{g}, even though it satisfies φ_1 (since it suppresses all of its answer actions $\mathsf{a}!\mathsf{ans}$). Thus, in addition to sound enforcement we require a *transparency* condition for adequate enforcement. This 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, if the SuS is correct then the behaviour of the enforced system should be (bisimulation) equivalent to that of the original system.

Definition 4.3 (Transparent Enforcement)**.** A monitor m is *transparent* when enforcing a formula φ, written as $\mathsf{tenf}(m, \varphi)$, iff for *all* LTSs $\langle \textsc{Sys}, \textsc{Act} \cup \{\tau\}, \rightarrow \rangle$ and

system states $s \in \text{Sys}$, if $s \in [\![\varphi]\!]$ then $m[s] \sim s$. □

Example 4.3. We have already argued—via the counter-example s_{g}—why $m_{\mathbf{s}}$ does *not* transparently enforce φ_1. Similarly, we can also argue easily that $\neg\mathsf{tenf}(m_{\mathbf{r}}, \varphi_1)$ as follows. Although the simple system a?req.a!ans.b!log.nil trivially satisfies φ_1, we clearly have the inequality $m_{\mathbf{r}}[\mathsf{a?req.a!ans.b!log.nil}] \not\sim \mathsf{a?req.a!ans.b!log.nil}$ since $m_{\mathbf{r}}[\mathsf{a?req.a!ans.b!log.nil}] \xrightarrow{\mathsf{b?req}} m_{\mathbf{r}}[\mathsf{b!log.nil}]$ and $\mathsf{a?req.a!ans.b!log.nil} \not\xrightarrow{\mathsf{b?req}}$.

It turns out, however, that for φ_1, monitors $m_{\mathbf{t}}$ and $m_{\mathbf{et}}$ are transparent, *i.e.,* both $\mathsf{tenf}(m_{\mathbf{t}}, \varphi_1)$ and $\mathsf{tenf}(m_{\mathbf{et}}, \varphi_1)$ hold. 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_{g}, since this system satisfies φ_1 and one can easily establish that $m_{\mathbf{t}}[s_{\mathsf{g}}] \sim s_{\mathsf{g}}$ and also that $m_{\mathbf{et}}[s_{\mathsf{g}}] \sim s_{\mathsf{g}}$. □

Having introduced Definitions 4.2 and 4.3 we can define our first definition for adequate enforcement.

Definition 4.4 (Enforcement)**.** A monitor m *adequately enforces* property φ whenever it is (i) *sound* and (ii) *transparent*. □

4.1.1 Weak Enforcement

The transparency requirement of Definition 4.3, however, only restricts transducers from modifying the behaviour of satisfying systems *i.e.,* when $s \in [\![\varphi]\!]$. It fails to specify any enforcement behaviour for the cases when the SuS violates the property.

Example 4.4. Recall φ_1 and s_{b} from Example 4.1, and also monitors $m_{\mathbf{t}}$ and $m_{\mathbf{et}}$ from Example 3.2. Even though $s_{\mathsf{b}} \notin [\![\varphi_1]\!]$, not all of its exhibited behaviours constitute violating traces. For instance, $s_{\mathsf{b}} \xRightarrow{\mathsf{a?req{\cdot}a!ans{\cdot}b!log{\cdot}b?cls}} \mathsf{nil}$ is not a violating trace, and so a system that only executes this trace satisfies φ_1 *e.g.,* $\mathsf{a?req.a!ans.b!log.b?cls.nil} \in [\![\varphi_1]\!]$. Correspondingly, we have $m_{\mathbf{t}}[s_{\mathsf{b}}] \xRightarrow{\mathsf{a?req{\cdot}a!ans{\cdot}b!log{\cdot}b?cls}} \mathsf{id}[\mathsf{nil}]$ and the same is attained for $m_{\mathbf{et}}$ *i.e.,* $m_{\mathbf{et}}[s_{\mathsf{b}}] \xRightarrow{\mathsf{a?req{\cdot}a!ans{\cdot}b!log{\cdot}b?cls}} \mathsf{id}[\mathsf{nil}]$. □

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 [25, 50, 78]. Monitors adhering to trace transparency must ensure that if an execution trace is correct, regardless of whether it originates from a valid or invalid system, the monitor should refrain from modifying it.

Definition 4.5 (Trace Transparent Enforcement)**.** A monitor m observes *trace transparency* when enforcing a formula φ, denoted as $\mathsf{ttenf}(m, \varphi)$ iff for every monitor state m' and all traces t, t' and t'', when $\mathsf{sys}(t) \in [\![\varphi]\!]$ and $m[\mathsf{sys}(t)] \stackrel{t'}{\Longrightarrow} m'[\mathsf{sys}(t'')]$ then $t = t'; t''$. □

Concretely, in Definition 4.5 we construct a trace system $\mathsf{sys}(t)$ (for every trace t) and check that if $\mathsf{sys}(t)$ satisfies φ and the instrumented system $m[\mathsf{sys}(t)]$ reduces over trace t' to some $m'[\mathsf{sys}(t'')]$, then t should be equal to the concatenation of t' and t''. Put differently, this criterion states that every trace t' executed by the instrumented system $m[\mathsf{sys}(t)]$ should be a prefix of trace t thereby signifying that monitor m is incapable of modifying trace t and any of its prefixes.

Going back to Example 4.4, a trace transparent monitor m must therefore ensure that although $s_{\mathsf{b}} \notin [\![\varphi_1]\!]$, its valid traces, such as $\mathsf{a?req.a!ans.b!log.b?cls}$, are not modified at runtime. More precisely, since $\mathsf{sys}(\mathsf{a?req.a!ans.b!log.b?cls}) \in [\![\varphi_1]\!]$, every trace u executed by the composite system *i.e.,* $m[\mathsf{sys}(\mathsf{a?req.a!ans.b!log.b?cls})] \stackrel{u}{\Longrightarrow}$, must be a prefix of $\mathsf{a?req.a!ans.b!log.b?cls}$. 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 4.5. 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 $m_1[\mathsf{sys}(t)] \stackrel{t'}{\Longrightarrow} m_1'[\mathsf{sys}(t'')]$ then $t = t'; t''$. We thus inspect the following cases for t.

(a) $t = ab; u$ (for any suffix u): This case holds vacuously since $\mathsf{sys}(ab; u) \notin [\![\varphi_2]\!]$.

(b) $t = t'; t''(\neq ab; u)$: This case also holds since monitor m_1 is unable to modify any trace that is not prefixed by ab, which means that when $m_1[\mathsf{sys}(t)] \stackrel{t'}{\Longrightarrow} m_1'[\mathsf{sys}(t'')]$ then $t = t'; t''$ as required.

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

Definition 4.3 (Transparency) and Definition 4.5 (Trace Transparency) provide two different ways of defining transparency, and so it is natural to wonder how they are related. The following result thus shows that trace transparency is in fact a *weaker* instance of Definition 4.3.

Theorem 4.1 (ttenf *vs.* tenf)**.** For every monitor m and μHML formula φ,

(i) $\mathsf{tenf}(m, \varphi)$ implies $\mathsf{ttenf}(m, \varphi)$; and

(ii) $\mathsf{ttenf}(m, \varphi)$ does not imply $\mathsf{tenf}(m, \varphi)$. □

Proof. The proof for (i) follows immediately from Definitions 4.3 and 4.5 since trace systems are a subset of all the possible system states of LTSs.

To prove (ii) it suffices to find a single monitor and formula that adhere to Definition 4.5 but not to Definition 4.3. Recall the result proven in Example 4.5 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. To this end, consider system $s_1 = a.b.\mathsf{nil} + a.c.\mathsf{nil}$. We have that $s_1 \in [\![\varphi_2]\!]$ and we also know that $m_1[s_1] \not\sim s_1$ since $s_1 \xrightarrow{a} \cdot \xrightarrow{b} \mathsf{nil}$ while $m_1[s_1] \overset{ab}{\Longrightarrow}$. 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 *adequately enforces* φ" than the one in Definition 4.4. This in turn gives us a new definition for enforceability for a logic, akin to Definition 4.1.

Definition 4.6 (Weak Enforcement)**.** A monitor m *adequately enforces* formula φ whenever it adheres to (i) *soundness*, Definition 4.2, and (ii) *trace transparency*, Definition 4.5. □

Remark 4.1. Since Definition 4.6 is defined it terms of Definition 4.2 (soundness) and Definition 4.5 (trace transparency), this definition resembles the most the classical definitions [25, 50, 78] for adequate enforcement. Therefore, Theorem 4.1 also suggests that in our setting, the classical definitions are relatively weak. □

Although Theorem 4.1 proves that Definition 4.6 is inherently weaker than Definition 4.4, both definitions may become *equally powerful* when restricted to particular subsets of μHML as per the case of the safety subset sHML. As both are defined in terms of Definition 4.2 (Soundness) and only vary with respect to the transparency definition, to ensure this result it suffices to prove that the Definitions 4.3 (Transparency) and 4.5 (Trace Transparency) coincide with respect to sHML formulas.

Theorem 4.2. For every monitor m and sHML formula φ, $\mathsf{tenf}(m, \varphi)$ iff $\mathsf{ttenf}(m, \varphi)$. □

Since the if-case of Theorem 4.2 has already been proven to hold for the full μHML (in Theorem 4.1) 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 Appendices B.2.1 and B.2.2 respectively starting on page 158.

Lemma 4.1. For every system state s, sHML formula φ and trace $t \in \mathit{traces}(s)$ when $s \in [\![\varphi]\!]$ then $\mathsf{sys}(t) \in [\![\varphi]\!]$.

Lemma 4.2. For every action α, sHML formula φ and trace t, if $\mathsf{sys}(t) \in [\![\mathit{after}(\varphi, \alpha)]\!]$ then $\mathsf{sys}(\alpha t) \in [\![\varphi]\!]$.

Upon first reading the reader may safely skip the content of this proof and proceed from 35.

Proof. We prove by coinduction that for every system state s, monitor m and sHML formula φ whenever $\mathsf{ttenf}(m, \varphi)$ and $s \in [\![\varphi]\!]$ then $m[s] \sim s$. We therefore show that relation $\mathcal{R} \stackrel{\text{def}}{=} \{ (m[s], s) \mid s \in [\![\varphi]\!] \text{ and } \mathsf{ttenf}(m, \varphi) \}$ 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

$$m[s] \xrightarrow{\mu} r' \tag{4.1}$$

$$s \in [\![\varphi]\!] \tag{4.2}$$

and that $\mathsf{ttenf}(m, \varphi)$ from which by Definition 4.5 we have that

$$\textit{if } \mathsf{sys}(t) \in [\![\varphi]\!] \textit{ and } m[\mathsf{sys}(t)] \stackrel{t'}{\Longrightarrow} m'[\mathsf{sys}(t'')] \textit{ then } t = t'; t''. \tag{4.3}$$

We now explore all the possible instrumentation rules by which the reduction in (4.1) can occur.

- iAsy: From (4.1) and rule iAsy we have that $\mu = \tau$ and that

  $$s \xrightarrow{\tau} s' \tag{4.4}$$

  $$r' = m[s']. \tag{4.5}$$

  Since by Proposition B.1 we know that sHML is agnostic of τ-actions, from (4.2) and (4.4) we also know that $s' \in [\![\varphi]\!]$ and so since from (4.5) we know that m remains unmodified by the transition, from (4.3) and the definition of $\mathcal{R}$ we conclude that

  $$(m'[s'], s') \in \mathcal{R} \tag{4.6}$$

  as required. Hence this case holds by (4.4) and (4.6).

- iDef: From (4.1) and rule iDef we have that $\mu = \alpha$ and that

  $$s \xrightarrow{\alpha} s' \tag{4.7}$$

  $$r' = \mathsf{id}[s']. \tag{4.8}$$

  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

  $$\mathsf{ttenf}(\mathsf{id}, \textit{after}(\varphi, \alpha)). \tag{4.9}$$

  Finally, by using the semantic justification of *after*, *i.e.,* Proposition 2.3, from (4.2) and (4.7) we deduce that $s' \in [\![\textit{after}(\varphi, \alpha)]\!]$. Therefore, knowing (4.9) and by the definition of $\mathcal{R}$ we conclude that

  $$(\mathsf{id}[s'], s') \in \mathcal{R} \tag{4.10}$$

  as required. Hence, this case holds by (4.7) and (4.10).

- ıTrn (identity): From (4.1) and rule ıTrn we have that

$$s \xrightarrow{\alpha} s' \tag{4.11}$$

$$m \xrightarrow{\alpha \blacktriangleright \alpha} m'' \tag{4.12}$$

$$r' = m''[s'] \tag{4.13}$$

and so by using the semantic justification of *after* Proposition 2.3, from (4.2) and (4.11) we can immediately deduce that

$$s' \in [\![\mathit{after}(\varphi, \alpha)]\!]. \tag{4.14}$$

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

$$\mathsf{sys}(u) \in [\![\mathit{after}(\varphi, \alpha)]\!] \tag{4.15}$$

$$m''[\mathsf{sys}(u)] \xRightarrow{u'} m'[\mathsf{sys}(u'')]. \tag{4.16}$$

Knowing (4.15), by Lemma 4.2 we have that

$$\mathsf{sys}(\alpha u) \in [\![\varphi]\!] \tag{4.17}$$

and so from (4.3) and (4.17) we can infer that

$$\mathit{if}\ m[\mathsf{sys}(\alpha u)] \xRightarrow{\alpha u'} m'[\mathsf{sys}(u'')]\ \mathit{then}\ \alpha u = \alpha u' u'' \tag{4.18}$$

and thus from (4.12), (4.16) and (4.18) we can conclude that

$$u = u' u''. \tag{4.19}$$

Hence, from assumptions (4.15), (4.16) and deduction (4.19) we can introduce an implication so that by Definition 4.5 we conclude that

$$\mathsf{ttenf}(m'', \mathit{after}(\varphi, \alpha)). \tag{4.20}$$

Finally, by (4.14), (4.20) and the definition of $\mathcal{R}$ we have that

$$(m''[s'], s') \in \mathcal{R} \tag{4.21}$$

as required, and so we are done by (4.11) and (4.21).

- iSup, iIns, ıTrn (replacement): Knowing (4.2), from Lemma 4.1 we infer that

$$\forall t \in \mathit{traces}(s) \cdot \mathsf{sys}(t) \in [\![\varphi]\!]. \tag{4.22}$$

and so from (4.3) and (4.22) we can conclude that monitor m does not modify any of the behaviours (traces) of s because

$$\forall t \in \mathit{traces}(s) \cdot m[\mathsf{sys}(t)] \xRightarrow{t} . \tag{4.23}$$

Therefore, these cases do not apply since these rules modify the trace actions executed by s, and so if (4.1) is the result of any these rules, it would contradict with (4.23).

These cases thus allow us to conclude that (a) holds. We now proceed to prove (b). So let's assume that

$$s \xrightarrow{\mu} s' \tag{4.24}$$

$$s \in [\![\varphi]\!] \tag{4.25}$$

$$\mathsf{ttenf}(m, \varphi) \tag{4.26}$$

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

- $\mu = \tau$: Since $s \xrightarrow{\tau} s'$, by (4.25) and since sHML is agnostic of τ-actions (Proposition B.1), we know that

  $$s' \in [\![\varphi]\!] \tag{4.27}$$

  and by rule ıAsy we can also deduce that

  $$m[s] \xrightarrow{\tau} m[s']. \tag{4.28}$$

  Hence by (4.26), (4.27) and the definition of $\mathcal{R}$ we can conclude that

  $$(m[s'], s') \in \mathcal{R} \tag{4.29}$$

  as required, and so this case holds by (4.28) and (4.29).

- $\mu = \alpha$: Since $s \xrightarrow{\alpha} s'$ and knowing (4.25) we can refer to the semantic justification of *after* and immediately deduce that

  $$s' \in [\![\mathit{after}(\varphi, \alpha)]\!]. \tag{4.30}$$

  From (4.26) and by Definition 4.5 we know that for every trace t

  $$\mathit{if}\ \mathsf{sys}(t) \in [\![\varphi]\!]\ \mathit{and}\ m[\mathsf{sys}(t)] \stackrel{t'}{\Longrightarrow} m'[\mathsf{sys}(t'')]\ \mathit{then}\ t = t'; t'' \tag{4.31}$$

  and by Lemma 4.1 from (4.25) 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 (4.31) we can infer that

  $$\mathit{if}\ m[\mathsf{sys}(\alpha u')] \stackrel{\alpha u''}{\Longrightarrow} m'[\mathsf{sys}(u''')]\ \mathit{then}\ \alpha u' = \alpha u'' u''' \tag{4.32}$$

  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

  $$\exists m'' \cdot m[s] \xrightarrow{\alpha} m''[s'] \tag{4.33}$$

  as required. Finally, lets assume that for every trace v,

  $$\mathsf{sys}(v) \in [\![\mathit{after}(\varphi, \alpha)]\!] \tag{4.34}$$

  $$m''[\mathsf{sys}(v)] \stackrel{v'}{\Longrightarrow} m'[\mathsf{sys}(v'')]. \tag{4.35}$$

  Since by (4.34) and Lemma 4.2 we have that $\mathsf{sys}(\alpha v) \in [\![\varphi]\!]$, from (4.31) we can infer that

  $$\mathit{if}\ m[\mathsf{sys}(\alpha v)] \stackrel{\alpha v'}{\Longrightarrow} m'[\mathsf{sys}(v'')]\ \mathit{then}\ \alpha v = \alpha v' v'' \tag{4.36}$$

  and thus from (4.35) and (4.36) we can conclude that

  $$v = v' v''. \tag{4.37}$$

  Hence, from assumptions (4.34), (4.35) and deduction (4.37) we can introduce an implication so that by Definition 4.5 we conclude that $\mathsf{ttenf}(m'', \mathit{after}(\varphi, \alpha))$

and so by (4.30) and the definition of $\mathcal{R}$ we have that

$$(m''[s'], s') \in \mathcal{R} \tag{4.38}$$

as required. Hence, this case holds by (4.33) and (4.38). □

4.1.2 Strong Enforcement

Theorem 4.1 shows that transparency as defined per Definition 4.3 is stronger than that of Definition 4.5. However, it turns out that Definition 4.3 is still a relatively weak constraint as it still disregards the extent of enforcement induced upon the erroneous systems. For instance, at runtime s_{b} can exhibit the following invalid behaviour: $s_{\mathsf{b}} \stackrel{t_1}{\Longrightarrow} \mathsf{b!log}.s_{\mathsf{g}}$ where $t_1 \stackrel{\text{def}}{=} \mathsf{a?req.a!ans.a!ans}$. In order to bring the invalid behaviour of s_{b} (shown in t_1) in line with our safety specification φ_1, it suffices to use some monitor m that omits only *one* of the answers, a!ans. After correcting t_1 into $t_1' \stackrel{\text{def}}{=} \mathsf{a?req.a!ans}$, no further modifications are required by m since the SuS reaches a valid point, that is, it reduces into the state $\mathsf{b!log}.s_{\mathsf{g}}$ that does not violate our property. However, when instrumented with $m_{\mathbf{t}}$, this monitor does not only suppress the invalid answer *i.e.,* $m_{\mathbf{t}}[s_{\mathsf{b}}] \xrightarrow{\mathsf{a?req.a!ans}.\tau} \mathsf{sup}[\mathsf{b!log}.s_{\mathsf{g}}]$, but keeps on suppressing every subsequent action as a result of reducing into sup *i.e.,* $\mathsf{sup}[\mathsf{b!log}.s_{\mathsf{g}}] \xrightarrow{\tau} \mathsf{sup}[s_{\mathsf{g}}] \ldots$.

It thus makes sense that transparency should also start applying whenever an invalid SuS reaches a valid point while instrumented with the monitor. Put differently, if a composite system, $m[s]$ (where $s \notin [\![\varphi]\!]$), reduces to some state $m'[s']$ over a trace t, where s' is in agreement with φ *after following* t (*i.e.,* $s' \in [\![\mathit{after}(\varphi, t)]\!]$), then the behaviour of $m'[s']$ should be *equivalent* to that of s'. Equipped with the *after* function of Definition 2.6 defined in Chapter 2, we can now introduce the *eventual transparency* constraint.

Definition 4.7 (Eventual Transparent Enforcement)**.** A monitor m is eventual transparent when enforcing φ, denoted as $\mathsf{evtenf}(m, \varphi)$, iff for all LTSs $\langle \textsc{Sys}, \textsc{Act} \cup \{\tau\}, \rightarrow \rangle$, system states s, s', traces t and monitor states m', $m[s] \stackrel{t}{\Longrightarrow} m'[s']$ and $s' \in [\![\mathit{after}(\varphi, t)]\!]$ imply that $m'[s'] \sim s'$ (when $\mathit{after}(\varphi, t)$ can be semantically justified). □

Example 4.6. We have already argued, via the counter example s_{b}, why $m_{\mathbf{t}}$ does not adhere to eventual transparency *i.e.,* $\neg\mathsf{evtenf}(m_{\mathbf{t}}, \varphi_1)$; this is not the case for $m_{\mathbf{et}}$ because $\mathsf{evtenf}(m_{\mathbf{et}}, \varphi_1)$. Although the universal quantification over all systems and traces make it hard to prove this property, we can get a good intuition as to why this is the case from s_{b}. Particularly, when $m_{\mathbf{et}}[s_{\mathsf{b}}] \xrightarrow{\mathsf{a?req.a!ans}.\tau} (\mathsf{rec}\, Y.\{\mathsf{a!ans}, \bullet\}.Y + \{\mathsf{b!log}\})[\mathsf{b!log}.s_{\mathsf{g}}]$ we know that at state $\mathsf{b!log}.s_{\mathsf{g}}$, the SuS no longer violates whatever remains of formula φ_1 *i.e.,* $\mathsf{b!log}.s_{\mathsf{g}} \in [\![\mathit{after}(\varphi_1, \mathsf{a?req.a!ans})]\!] = [\![([\mathsf{a!ans}]\mathsf{ff} \wedge [\{\mathsf{b!log}\}]\varphi_1)]\!]$. We also know that upon reducing to $\mathsf{b!log}.s_{\mathsf{g}}$, the monitor performs no further modifications to its behaviour *i.e.,* $\mathsf{b!log}.s_{\mathsf{g}} \sim (\mathsf{rec}\, Y.\{\mathsf{a!ans}, \bullet\}.Y + \{\mathsf{b!log}\}.m_{\mathbf{et}})[\mathsf{b!log}.s_{\mathsf{g}}]$. □

Although Definition 4.7 (eventual transparency) is yet another way of representing transparent monitoring, the following result shows that eventual transparency is in fact stronger than Definition 4.3 (transparency).

Theorem 4.3 (evtenf *vs.* tenf)**.** For every monitor m and μHML formula φ,

(i) $\mathsf{evtenf}(m, \varphi)$ implies $\mathsf{tenf}(m, \varphi)$; and that

(ii) $\mathsf{tenf}(m, \varphi)$ does not imply $\mathsf{evtenf}(m, \varphi)$. □

Proof. The proof for (i) follows immediately since Definition 4.3 is just an instance of Definition 4.7 *i.e.,* when *after*(φ, t) can be semantically justified and when t is the empty trace ε.

To prove (ii) we must show that there exist monitor and a formula that adhere to Definition 4.3 but not to Definition 4.7. This result therefore holds since in Example 4.3 we showed that $\mathsf{tenf}(m_{\mathbf{t}}, \varphi_1)$ but in Example 4.6 we also deduced that $\neg\mathsf{evtenf}(m_{\mathbf{t}}, \varphi_1)$. □

Note that by transitivity from Theorems 4.1 and 4.3 we can deduce that Definition 4.7 (eventual transparency) is also stronger than Definition 4.5 (trace transparency). Moreover, since Definition 4.3 (transparency) is just an instance of Definition 4.7 (eventual transparency), the latter requirement along with Definition 4.2 (soundness) suffice to provide yet another definition for "m adequately enforces φ".

Definition 4.8 (Strong Enforcement)**.** A monitor m *adequately enforces* property φ whenever it adheres to (i) *soundness*, Definition 4.2, and (ii) *eventual transparency*, Definition 4.7. □

Corollary 4.1. Since Definition 4.8 is defined in terms of eventual transparency (Definition 4.7) which we proved to be stronger than transparency (Definition 4.3) in Theorem 4.3, this new definition for "m adequately enforces φ" is also *stronger* than that of Definition 4.4. □

4.1.3 The limits of enforceability

For any reasonably expressive logic, it is usually the case that *not* every formula can be enforced – as supported by Theorem 4.4, our logic μHML is not an exception.

Theorem 4.4. There exists a μHML formula φ for which one cannot find a monitor m so that $\mathsf{enf}(m, \varphi)$ holds in the sense of Definitions 4.4, 4.6 or 4.8. □

In the following proof we thus present a counter example showing the existence of a particular formula for which there does not exist a monitor that enforces it adequately as stated by either one of our adequate enforcement definitions.

Proof. Consider the μHML property φ_{or}, together with systems s_2, s_3 and s_4:

$$\varphi_{\mathsf{or}} \stackrel{\text{def}}{=} [\{a\}]\mathsf{ff} \vee [\{b\}]\mathsf{ff} \qquad s_2 \stackrel{\text{def}}{=} a.\mathsf{nil} \qquad s_3 \stackrel{\text{def}}{=} b.\mathsf{nil} \qquad s_4 \stackrel{\text{def}}{=} s_2 + s_3$$

A system satisfies φ_{or} if *either* it cannot produce action a *or* it cannot produce action b. Clearly, s_4 violates this property as it can produce both, although s_2 and s_3 are both correct since $s_2, s_3 \in [\![\varphi_{\mathsf{or}}]\!]$. Enforcing this formula on s_4 requires suppressing (or replacing) either one of the actions, or the insertion of some prefixing action c. We now show that there does not exist an enforcement monitor m that enforces formula φ_{or}. Without loss of generality, assume that our monitors may insert, suppress or replace actions and consider the following 3 cases.

Case 1 (insertions): For starters, an insertion monitor such as $m_2 \stackrel{\text{def}}{=} \{\bullet, c\}.\mathsf{id}$ is able to bring in line invalid systems like s_4 by prefixing their execution with an action c thus causing the trivial satisfaction of φ_{or}. However, this insertion infringes the respective transparency criteria of Definitions 4.4, 4.6 and 4.8 when it is performed for valid systems *e.g.,* s_2 and s_3.

Case 2 (suppressions): Similarly, the suppression monitor $m_3 \stackrel{\text{def}}{=} \mathsf{rec}\, Y.\big(\{a, \bullet\}.Y + \{b, \bullet\}.Y\big)$ can suppress the offending actions produced by s_4, thus obtaining $m_3[s_4] \in [\![\varphi_{\mathsf{or}}]\!]$. However, it also suppresses the sole actions a and b produced by s_2 and s_3 respectively, even though they both satisfy φ_{or}. Once again, this infringes the transparency and eventual transparency criteria of Definitions 4.4 and 4.8 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 $m_3[s_2] \not\sim s_2$ and same for s_3. Note that a weaker version of m_3, such as $\mathsf{rec}\, Y.\{a, \bullet\}.Y$ (*resp.* $\mathsf{rec}\, Y.\{b, \bullet\}.Y$) still breaches these transparency constraints as it modifies s_2 (*resp.* s_3) unnecessarily. 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 execution trace. Similarly, m_3 also violates the weaker requirement of trace-transparency required by Definition 4.6, *i.e.,* we can deduce that $m_3[\mathsf{sys}(t)] \stackrel{t}{\not\rightarrow}$ despite that $\mathsf{sys}(t) \in [\![\varphi]\!]$ (for every trace executable by s_2, s_3 and s_4 *i.e.,* $t \in \{a, b\}$).

Case 3 (replacements): Proving this case entails following the same argument as that presented for case 2. □

4.2 Optimality

Definitions 4.4, 4.6 and 4.8 define what it means for a monitor to adequately enforce a formula, but fail to assess whether a monitor is (to some extent) the "*best*" that one can find to enforce a property. To define such a notion we must first be able to compare monitors to one another via some kind of *distance measurement*. One ideal measurement is to assess the monitor's level of *intrusion* when enforcing a property.

In Figure 4.1 we define function *mc* that inductively analyses a system run,

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

Figure 4.1: Modification Count (*mc*).

represented as an explicit trace t_τ, and counts the number of modifications applied by the monitor. In each case the function reconstructs a trace system $\mathsf{sys}(t_\tau)$ and instruments it with the monitor m in order to assess the type of transformations applied by m. Specifically, in the first case, *mc* increments the counter when the monitor replaces or suppresses the prefixing action μ *i.e.,* when it transforms μ into μ' where $\mu \neq \mu'$. It then recurses to keep on inspecting the continuation trace suffix t'_τ vis-a-vis the subsequent monitor state m'. In the second case, the counter is incremented when the monitor inserts an action μ'. Since the state of the SuS $\mathsf{sys}(t_\tau)$ remains unchanged after the insertion (*i.e.,* $m[\mathsf{sys}(t_\tau)] \xrightarrow{\mu'} m'[\mathsf{sys}(t_\tau)]$) function *mc* recurses vis-a-vis the unmodified continuation trace t_τ and the new monitor state m'.

The third case, specifies that the counter stays unmodified when the monitor applies an identity transformation *i.e.,* when $\mu = \alpha$, and when the system performs a τ-action independent of the monitor *i.e.,* when $\mu = \tau$. Finally, the last case returns the length of t_τ when $m[\mathsf{sys}(t_\tau)]$ is unable to execute further. This therefore specifies that if the termination of $m[\mathsf{sys}(t_\tau)]$ is caused due to the (normal) termination of $\mathsf{sys}(t_\tau)$ *i.e.,* when $t_\tau = \varepsilon$, then the number of modifications applied to trace t_τ is equal to 0. However, if the monitor m somehow manages to (abnormally) halt or block the execution of the SuS, then the modification count is equal to the number of actions that were supposed to execute *i.e.,* $|\, t_\tau \,|$.

Example 4.7. Recall monitors $m_{\mathbf{i}}$, $m_{\mathbf{r}}$, $m_{\mathbf{s}}$, $m_{\mathbf{t}}$ and $m_{\mathbf{et}}$ from Example 3.2 of Chapter 3 and consider the following system run t^0_τ=a?req.a!ans.a!ans.b!log. For $m_{\mathbf{i}}$ and $m_{\mathbf{r}}$, function *mc* respectively counts one inserted action, and three replaced actions (since b!log remains unmodified) *i.e.,* $mc(m_{\mathbf{i}}, t^0_\tau)$=1 and $mc(m_{\mathbf{r}}, t^0_\tau)$=3. Monitors $m_{\mathbf{s}}$ and $m_{\mathbf{t}}$ both score a count of two, since the former suppresses every answer, while the latter suppresses the first redundant answer along with the log action *i.e.,* b!log. Finally, $mc(m_{\mathbf{et}}, t^0_\tau)$=1 since $m_{\mathbf{et}}$ suppresses only the second answer action. □

We can now use function *mc* to compare monitors to each other in order to identify the least intrusive one, *i.e.,* the monitor that applies the least amount of transformations when enforcing a specific property. It is also desirable to be able to compare transducers based on their enforcement abilities. This would allow us to determine that a monitor is the least intrusive one that can be found, with a

$$ec(m) \stackrel{\text{def}}{=} \begin{cases} \varnothing & \text{if } m = X \\ \bigcup_{i \in I} ec(m_i) & \text{if } m = \sum_{i \in I} m_i \\ ec(m') & \text{if } m = \mathsf{rec}\, X.m' \text{ or } m = \{p, c, \underline{p}\}.m' \\ \{\mathsf{SUP}\} \cup ec(m') & \text{if } m = \{p, c, \bullet\}.m' \\ \{\mathsf{INS}\} \cup ec(m') & \text{if } m = \{\bullet, c, p\}.m' \\ \{\mathsf{REP}\} \cup ec(m') & \text{if } m = \{p, c, p'\}.m' \text{ and } p' \neq \underline{p} \neq \bullet \end{cases}$$

Figure 4.2: Enforcement Capabilities (*ec*).

restricted set of enforcement abilities. For instance, we can compare an adequate suppression monitor m to all other adequate suppression monitors in order to determine that m is the least intrusive monitor that enforces a property using only action suppressions. The restriction can then be relaxed to include other enforcement capabilities, such as insertions, in order to determine whether m is still the least intrusive monitor when compared to a wider range of monitors that can perform suppression or insertion transformations (or both).

We determine the enforcement capabilities of a monitor via function *ec* of Figure 4.2. It inductively analyses the structure of the monitor and deduces whether it can insert, suppress and replace actions based on the type of transformation triples it defines. For instance, if the monitor defines an suppression triple then *ec* determines that the monitor can suppress actions SUP, while if it defines an insertion or replacement transformation then it respectively concludes that the monitor can insert INS, and replace REP, actions. We represent an arbitrary set of enforcement capabilities using the metavariable χ.

Example 4.8. Recall the monitors of Example 3.2. With function *ec* we determine that $ec(m_{\mathbf{i}})$=$\{\mathsf{INS}\}$, $ec(m_{\mathbf{r}})$=$\{\mathsf{REP}\}$, $ec(m_{\mathbf{s}})$=$ec(m_{\mathbf{t}})$=$ec(m_{\mathbf{et}})$=$\{\mathsf{SUP}\}$. Monitors may also have multiple types of enforcement capabilities, for instance, $ec(m_{\mathbf{i}} + m_{\mathbf{r}} + m_{\mathbf{s}})$=$\{\mathsf{INS}, \mathsf{REP}, \mathsf{SUP}\}$. □

With these definitions we now define χ-*optimal enforcement* where χ restricts the search space to allow for finding an optimal monitor whose enforcement capabilities are restricted to those defined by χ.

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

Definition 4.9 states that an adequate monitor m is *optimal* for φ, if one cannot find another adequate monitor n, that has the same (or fewer) enforcement capabilities as those defined by χ, and that performs *fewer modifications* than m. We say that a monitor is the *most optimal* monitor that enforces φ when it is found to be the least intrusive when compared to every type of monitor, *i.e.,* it is $\{\mathsf{SUP}, \mathsf{REP}, \mathsf{INS}\}$-optimal.

Example 4.9. Recall formula φ_1 of Example 4.1 and monitor $m_{\mathbf{et}}$ of Example 3.2. Showing that $m_{\mathbf{et}}$ is the *most optimal* monitor that enforces φ_1 is inherently difficult as it requires comparing $m_{\mathbf{et}}$ to every kind of monitor. Similarly, showing that $m_{\mathbf{et}}$ is SUP-optimal *i.e.,* $\mathsf{oenf}_{\mathsf{SUP}}(m_{\mathbf{et}}, \varphi_1)$, is still quite difficult. However, from Example 4.7 we already get the intuition that it holds since $m_{\mathbf{et}}$ imposes the least amount of modifications compared to the other suppression monitors $m_{\mathbf{s}}$ and $m_{\mathbf{t}}$.

Specifically, $\mathsf{oenf}_{\mathsf{SUP}}(m_{\mathbf{et}}, \varphi_1)$ holds regardless of the chosen adequacy definition. If we use our strongest definition (Definition 4.8), then only $m_{\mathbf{et}}$ is considered adequate and so it is optimal by default, whereas if we use Definitions 4.4 or 4.6 it still holds since $m_{\mathbf{t}}$ is more intrusive, while $m_{\mathbf{s}}$ is not considered in the comparison as it is inadequate *i.e.,* $\neg\mathsf{enf}(m_{\mathbf{s}}, \varphi_1)$. □

4.3 Summary

In this chapter we have formalised the meaning of adequate and optimal enforcement, which we motivated in the context of the unidirectional enforcement setting of Figure 3.2 defined in Chapter 3. More specifically, we have presented the following contributions:

(i) Three different definitions for adequate enforcement, namely, Definitions 4.4, 4.6 and 4.8. They are parametrisable with respect to any instrumentation relation (an instance of which is given by the unidirectional framework of Figure 3.2 in Chapter 3). We showed that Definition 4.6 is the weakest while Definition 4.8 is the strongest, Theorems 4.1 and 4.3.

(ii) The distance function of Figure 4.1 that measures the level of intrusiveness of a monitor by counting the number of modifications it applies to the SuS at runtime.

(iii) The novel definition for optimal enforcement, Definition 4.9, that uses the distance measurement mentioned in (ii) to assess a monitor's level of intrusion and guides the search for the least intrusive one.

5. Synthesising suppression monitors

Despite their merits, the enforcement definitions introduced in Chapter 4 are not easy to work with. Particularly, the universal quantifications over all possible systems (of which there could be an infinite amount), make it hard to establish that a monitor correctly enforces a property. Moreover, as stated by Definition 4.1 (Enforceability), 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. 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*. Second, for *every* formula φ in this enforceable subset, we provide an *automated procedure* to *synthesise* a monitor m from it. We then ensure that when instrumented with an arbitrary system, the synthesised monitor m enforces φ, *adequately* as defined by Definitions 4.4, 4.6 and 4.8, and *optimally* as per Definition 4.9. This procedure can then be used as a basis for constructing tools that automate property enforcement.

Specifically, in this first attempt to study the enforceability of μHML formulas in the unidirectional setting of Chapter 3, we restrict ourselves to *suppression monitors i.e.,* transducers that are only allowed to intervene by dropping system actions. Despite being more constrained, suppression monitors are relatively easier to work with since they side-step problems associated with what data to use in a payload-carrying action generated by the monitor, as in the case of insertion and replacement monitors. Moreover, suppression monitors are particularly useful for enforcing *safety* properties, as shown in [23, 50, 78]. 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 refer to the result from [56], which identified the safety fragment sHML (presented in Figure 2.4 of Chapter 2) as being the maximally-expressive logical subset of μHML that can be handled by violation-detecting (runtime verification) monitors. Using this result as a guideline, we limit our enforceability study to sHML, given that an *eventual transparent* suppression monitor cannot judiciously suppress actions without first detecting a (potential) violation. In Chapter 6 we will then show that sHML is also the maximally-expressive subset of μHML that can be enforced by suppression monitors.

One way of achieving our aims would be to (i) define a (total) synthesis function $(\!|-|\!) : \text{sHML} \mapsto \textsc{Trn}$ from sHML formulas to suppression monitors and (ii) then show that for *any* $\varphi \in \text{sHML}$, the synthesised monitor $(\!|\varphi|\!)$ adequately and optimally enforces φ according to Definition 4.8 — which implicitly implies adherence to Definitions 4.4 and 4.6 — and Definition 4.9 respectively. 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 criteria mentioned in (ii). However, a naive approach to such a scheme is bound to fail, as discussed in the next example.

Example 5.1. Consider an equivalent reformulation of φ_1 from Example 4.1.

$$\varphi_4 \stackrel{\text{def}}{=} \max X.[\{(x)?\mathsf{req}, x{\neq}\mathsf{b}\}]([\{x!\mathsf{ans}\}][\{x!\mathsf{ans}\}]\mathsf{ff} \wedge [\{x!\mathsf{ans}\}][\{\mathsf{b}!\mathsf{log}\}]X)$$

At an intuitive level, the monitor that one expects to obtain for the subformula $\varphi_4' \stackrel{\text{def}}{=} [\{x!\mathsf{ans}\}][\{x!\mathsf{ans}\}]\mathsf{ff}$ is $\{x!\mathsf{ans}\}.\mathsf{rec}\, Y.\{x!\mathsf{ans}, \bullet\}.Y$ (*i.e.,* a transducer that repeatedly suppresses every output ans that follows a serviced request on the same port), whereas the monitor for subformula $\varphi_4'' \stackrel{\text{def}}{=} [\{x!\mathsf{ans}\}][\{\mathsf{b}!\mathsf{log}\}]X$ is $\{x!\mathsf{ans}\}.\{\mathsf{b}!\mathsf{log}\}.X$. These monitors would then be combined in the synthesis for φ_2 as:

$$m_{\mathbf{b}} \stackrel{\text{def}}{=} \mathsf{rec}\, X.\{(x)?\mathsf{req}, x{\neq}\mathsf{b}\}.\big(\{x!\mathsf{ans}\}.\mathsf{rec}\, Y.\{x!\mathsf{ans}, \bullet\}.Y + \{x!\mathsf{ans}\}.\{\mathsf{b}!\mathsf{log}\}.X\big).$$

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

Instead of complicating our synthesis function to cater for anomalies such as those presented in Example 5.1—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 def-

inition that is compositional. This also facilitates the proofs for the criteria that our synthesised monitors are required to appease in order to be adequate and optimal. Second, we show that every sHML formula 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 sHML.

5.1 The synthesis function

The following grammar presents the normalised syntactic subset of sHML, which we refer to as $\text{sHML}_{\mathbf{nf}}$. It combines the modal necessity operators and conjunctions into one construct $\bigwedge_{i\in I}[\{p_i,c_i\}]\varphi_i$, which is written as $[\{p_0,c_0\}]\varphi_0 \wedge \ldots \wedge [\{p_n,c_n\}]\varphi_n$ when $I=\{0,\ldots,n\}$ and simply as $[\{p_0,c_0\}]\varphi$ when $|\,I\,|=1$.

Definition 5.1 (sHML normal form)**.** The set of normalised sHML formulas is defined as follows:

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

In addition, normalised sHML formulas must satisfy the following conditions:

1. Every branch in $\bigwedge_{i\in I}[\{p_i,c_i\}]\varphi_i$, must be *disjoint*, *i.e.,* $\#_{i\in I}\{p_i,c_i\}$ which entails that for every $i,j\in I$, $i\neq j$ and substitution environment σ, if both $\{p_i,c_i\}\sigma$ and $\{p_j,c_i\}\sigma$ are *closed*, then $[\![\{p_i,c_i\}\sigma]\!]\cap[\![\{p_j,c_j\}\sigma]\!]=\varnothing$.
2. For every $\max X.\varphi$ we have $X\in\mathbf{fv}(\varphi)$. □

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 $\max X_0.\ldots.\max X_n.\mathsf{ff}$). 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. Same as in μHML, fixpoint variables, X, must be guarded by a modal necessity.

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

Definition 5.2. Our synthesis $(\!|-|\!):\text{sHML}_{\mathbf{nf}}\to\text{TRN}$ is defined inductively as:

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

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

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, $\max X.\varphi$, are converted into the corresponding recursive monitor. The synthesis also converts truth tt into the identity monitor id. Intuitively, since every system satisfies tt, the passiveness of monitor id helps

preserve the transparency requirements of Definitions 4.3, 4.5 and 4.7. Similarly, ff is synthesised into id because a system cannot be altered in any way to satisfy ff.

Normalized conjunctions, $\bigwedge_{i\in I}[\{p_i, c_i\}]\varphi_i$, are synthesised into a *recursive summation* of monitors, *i.e.,* $\mathsf{rec}\, Y.\sum_{i\in I} m_i$, where Y is fresh, and every branch m_i can be 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 $\{p_i, c_i, \bullet\}.Y$.

Once again, (i) helps preserve the transparency requirements by ensuring that the synthesised monitor does not suppress actions that do not violate the property, *i.e.,* those that satisfy $[\{p_i, c_i\}]\varphi_i$ where $\varphi_i \neq \mathsf{ff}$. In comparison, (ii) employs a recursive suppression loop to completely incapacitate the composite system from producing violating actions at runtime, *i.e.,* those that satisfy $[\{p_i, c_i\}]\mathsf{ff}$.

Example 5.2. Recall formula φ_1 from Example 4.1:

$$\varphi_1 \stackrel{\text{def}}{=} \mathsf{max}\, X.[\{(x)?\mathsf{req}, x \neq \mathsf{b}\}][\{x!\mathsf{ans}\}]([\{x!\mathsf{ans}\}]\mathsf{ff} \wedge [\{\mathsf{b}!\mathsf{log}\}]X).$$

Using the synthesis function defined in Definition 5.2, we generate monitor

$$(\!|\, \varphi_1 \,|\!) = \mathsf{rec}\, X.\{(x)?\mathsf{req}, x \neq \mathsf{b}\}.\mathsf{rec}\, Z.\big(\{x!\mathsf{ans}\}.\mathsf{rec}\, Y.\big(\{x!\mathsf{ans}, \bullet\}.Y + \{\mathsf{b}!\mathsf{log}\}.X\big)\big)$$

which can be optimized by removing redundant recursive constructs (*e.g.,* $\mathsf{rec}\, Z._-$), thus obtaining:

$$\mathsf{rec}\, X.\{(x)?\mathsf{req}, x \neq \mathsf{b}\}.\big(\{x!\mathsf{ans}\}.\mathsf{rec}\, Y.\big(\{x!\mathsf{ans}, \bullet\}.Y + \{\mathsf{b}!\mathsf{log}\}.X\big)\big) \;=\; m_{\mathbf{et}}. \qquad \square$$

To evaluate the quality of our synthesis, we prove the following results.

Theorem 5.1 (Enforceability)**.** The (sub)logic $\mathrm{sHML}_{\mathbf{nf}}$ is suppression enforceable with respect to Definitions 4.4, 4.6 and 4.8.

Proof. By Definition 4.1, the result follows if we show that for all $\varphi \in \mathrm{sHML}_{\mathbf{nf}}$, $(\!|\, \varphi \,|\!)$ *adequately enforces* φ in the sense of our strictest definition Definition 4.8. Hence, by Definition 4.8, this is a corollary of Propositions 5.1 and 5.2 stated below. $\square$

Proposition 5.1 (Soundness)**.** For every system state $s \in \mathrm{Sys}$ and $\varphi \in \mathrm{sHML}_{\mathbf{nf}}$ then $[\![\varphi]\!] \neq \varnothing$ *implies* $(\!|\, \varphi \,|\!)[s] \in [\![\varphi]\!]$. $\square$

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

Figure 5.1: A satisfaction relation for sHML formulas

Proposition 5.2 (Eventual Transparency)**.** For every system state $s \in \text{Sys}$ and formula $\varphi \in \text{sHML}_{\mathbf{nf}}$, if $(\!|\, \varphi \,|\!)[s] \stackrel{t}{\Longrightarrow} m'[s']$ and $s' \in [\![\textit{after}(\varphi, t)]\!]$ then $m'[s'] \sim s'$. □

To facilitate the proofs for Propositions 5.1 and 5.2 we use the satisfaction semantics for sHML from [10] which are defined in terms of the *satisfaction relation*, $\models$. When restricted to sHML, $\models$ is the *largest relation* $\mathcal{R}$ satisfying the implications defined in Figure 5.1. Since in [10] this satisfaction semantics was shown to agree with the sHML semantics of Figure 2.2, we use $s \models \varphi$ in lieu of $s \in [\![\varphi]\!]$. We also assume the classic notion of *strong similarity*, $s \precsim r$ as our touchstone system preorder for LTSs [85, 97].

Definition 5.3 (Strong Similarity)**.** A relation $\mathcal{R}$ over a set of system states is a *strong simulation* iff whenever $(s, r) \in \mathcal{R}$ for every action μ:

- 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*. □

The reader may safely skip these proofs upon first reading and continue on page 49.

Proof for Proposition 5.1. We prove a stronger result stating that for every system r that can be simulated by $(\!|\, \varphi \,|\!)[s]$, *i.e.,* $r \precsim (\!|\, \varphi \,|\!)[s]$, if $[\![\varphi]\!] \neq \varnothing$ then $r \models \varphi$. We prove this result by showing that relation $\mathcal{R} \stackrel{\text{def}}{=} \{ (r, \varphi) \mid [\![\varphi]\!] \neq \varnothing \textit{ and } r \precsim (\!|\, \varphi \,|\!)[s] \}$ is a *satisfaction relation* ($\models$) as defined by the rules in Figure 5.1. 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 $[\![\varphi]\!] = \varnothing$.

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

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

$$[\![\mathsf{max}\, X.\varphi]\!] \neq \varnothing \tag{5.1}$$

$$r \precsim (\!|\, \mathsf{max}\, X.\varphi \,|\!)[s]. \tag{5.2}$$

To prove that $\mathcal{R}$ is a satisfaction relation we show that $(r, \varphi\{\max X.\varphi/X\}) \in \mathcal{R}$ as well. Hence, since $(\!|\, \varphi\{\max X.\varphi/X\} \,|\!)$ produces monitor $(\!|\, \varphi \,|\!)\{^{\mathsf{rec}\, X.(\!|\, \varphi \,|\!)}/_{X}\}$ that is the *unfolded equivalent* of $\mathsf{rec}\, X.(\!|\, \varphi \,|\!)$ produced by $(\!|\, \max X.\varphi \,|\!)$, we can conclude that $(\!|\, \max X.\varphi \,|\!) \sim (\!|\, \varphi\{\max X.\varphi/X\} \,|\!)$ and so from (5.2) we have that

$$r \sqsubseteq_{\approx} (\!|\, \varphi\{\max X.\varphi/X\} \,|\!)[s]. \tag{5.3}$$

Since from (5.1) and $[\![\max X.\varphi]\!] = [\![\varphi\{\max X.\varphi/X\}]\!]$ we have that $[\![\varphi\{\max X.\varphi/X\}]\!] \neq \varnothing$, by (5.3) and the definition of $\mathcal{R}$ we can finally 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* $\#_{h\in I}\{p_h, c_h\}$. Start by assuming that $(r, \bigwedge_{i\in I} [\{p_i, c_i\}]\varphi_i) \in \mathcal{R}$ and so we have that

$$[\![\textstyle\bigwedge_{i\in I} [\{p_i, c_i\}]\varphi_i]\!] \neq \varnothing \tag{5.4}$$

$$r \sqsubseteq_{\approx} (\!|\, \textstyle\bigwedge_{i\in I} [\{p_i, c_i\}]\varphi_i \,|\!)[s]. \tag{5.5}$$

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

$$(\!|\, \textstyle\bigwedge_{i\in I} [\{p_i, c_i\}]\varphi_i \,|\!) = \Big(\sum_{i\in I} \begin{cases} \{p_i, c_i, \bullet\}.m & \text{if } \varphi_i{=}\mathsf{ff} \\ \{p_i, c_i\}.(\!|\, \varphi_i \,|\!) & \text{otherwise} \end{cases}\Big) = m_{\wedge}. \tag{5.6}$$

In order to prove that $\mathcal{R}$ is a satisfaction relation, 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 $\mathrm{sHML}_{\mathbf{nf}}$ 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 the following criteria hold: (a) $[\![[\{p_j, c_j\}]\mathsf{ff}]\!] \neq \varnothing$, (b) $r \sqsubseteq_{\approx} (\!|\, [\{p_j, c_j\}]\mathsf{ff} \,|\!)[s]$, and (c) that for every action α, when $\mathsf{mtch}(p_j, \alpha) = \sigma$ and $c_j\sigma \Downarrow \mathsf{true}$, then there does not exist a system r' such that $r \xRightarrow{\alpha} r'$. From (5.4) and the definition of $[\![-]\!]$ we can immediately infer that (a) holds, and so we have that

$$[\![[\{p_j, c_j\}]\mathsf{ff}]\!] \neq \varnothing. \tag{5.7}$$

We now note that since from (5.6) 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 (5.5) we can deduce that (b) holds since

$$r \sqsubseteq \approx (\!|\, \textstyle\bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i \,|\!)[s] \sqsubseteq \approx (\!|\, [\{p_j, c_j\}]\mathsf{ff} \,|\!)[s] \tag{5.8}$$

as required. Finally, from (5.6) we know that monitor m was synthesised from a normalized conjunction which is *disjoint* ($\#_{h \in I}\{p_h, c_h\}$) from which we conclude that whenever the system performs action α such that $\mathsf{mtch}(p_j, \alpha) = \sigma$ and $c_j\sigma \Downarrow \mathsf{true}$, only the suppression branch $\{p_j, c_j, \bullet\}.m$ (which is a single branch of m in (5.6)) 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. This therefore means that $m[s] \stackrel{\alpha}{\Longrightarrow}\!\!\!\!\!/$, and so from (5.5) we can deduce that (c) also holds since

$$\nexists r' \cdot r \stackrel{\alpha}{\Longrightarrow} r' \tag{5.9}$$

which means that any modal necessity that precedes ff can never be satisfied by r, as required. This case thus holds by (5.7), (5.8) and (5.9).

(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) $[\![[\{p_j, c_j\}]\varphi_j]\!] \neq \varnothing$, (b) $r \sqsubseteq \approx (\!|\, [\{p_j, c_j\}]\varphi_j \,|\!)[s]$ and then that (c) for every action α and derivative r', when $\mathsf{mtch}(p_j, \alpha) = \sigma$, $c_j\sigma \Downarrow \mathsf{true}$ and $r \stackrel{\alpha}{\Longrightarrow} r'$ then $(r', \varphi_j\sigma) \in \mathcal{R}$. From (5.4) and by the definition of $[\![-]\!]$ we can immediately determine that (a) holds, and so that

$$[\![[\{p_j, c_j\}]\varphi_j]\!] \neq \varnothing. \tag{5.10}$$

Since $(\!|\, [\{p_j, c_j\}]\varphi_j \,|\!) = \mathsf{rec}\, Y.\{p_j, c_j\}.(\!|\, \varphi \,|\!) = m'$, from (5.6) we deduce that both monitors m' and $m_\wedge$ refrain from modifying actions matching $\{p_j, c_j\}$ but m may suppress more actions. We can thus infer that for all s, $m[s] \sqsubseteq \approx (\!|\, [\{p_j, c_j\}]\varphi_j \,|\!)[s]$ and so from (5.5) we can deduce that (b) holds since

$$r \sqsubseteq \approx m[s] \sqsubseteq \approx (\!|\, [\{p_j, c_j\}]\varphi_j \,|\!)[s] \tag{5.11}$$

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

$$\mathsf{mtch}(p_j, \alpha) = \sigma \textit{ and } c_j\sigma \Downarrow \mathsf{true} \tag{5.12}$$

$$r \stackrel{\alpha}{\Longrightarrow} r' \tag{5.13}$$

and so from (5.5) and (5.13) we can deduce that

$$m[s] \stackrel{\alpha}{\Longrightarrow} q \text{ for some } q \text{ where } r' \sqsubseteq_{\approx} q. \tag{5.14}$$

By the definition of $\stackrel{\alpha}{\Longrightarrow}$, we know that the delayed transition in (5.14) is composed from zero or more τ-transitions followed by the α-transition *i.e.,*

$$m[s] \xrightarrow{\tau} {}^{*} q' \xrightarrow{\alpha} q. \tag{5.15}$$

By the rules in our model we know that the τ-reductions in (5.15) could have been the result of either one of these instrumentation rules, namely iSup or iAsy. From (5.6) 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 happens when rule iAsy is applied. Hence, we know that $q' = m[s']$ (for some derivative s' of s), and so from (5.15) we have that

$$m[s'] \xrightarrow{\alpha} q. \tag{5.16}$$

From (5.12) we also know that the reduction in (5.16) can only be the result of rule iTrn, and so we can infer that $s' \xrightarrow{\alpha} s''$ and that

$$q = m'[s''] \text{ where} \tag{5.17}$$

$$m \xrightarrow{\alpha \blacktriangleright \alpha} m'. \tag{5.18}$$

Since we know that $[\{p_j, c_j\}]\varphi_j$ and $\varphi_j \neq \mathsf{ff}$, from (5.6) 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 normalized conjunction in which $\#_{i \in I}\{p_i, c_i\}$. Hence, from (5.6), (5.12) and (5.18) we can deduce that

$$m' = (\!|\, \varphi_j \sigma \,|\!). \tag{5.19}$$

Since from (5.10) and by the definition of $[\![-]\!]$ we know that $[\![\varphi_j\sigma]\!] \neq \varnothing$ and from (5.14), (5.17) and (5.19) we have that $r' \sqsubseteq_{\approx} (\!|\, \varphi_j\sigma \,|\!)[s'']$, 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

$$\forall \alpha, r' \cdot \ \textit{if } \mathsf{mtch}(p_j, \alpha) = \sigma, c_j\sigma \Downarrow \mathsf{true} \textit{ and } r \stackrel{\alpha}{\Longrightarrow} r' \text{ then } (r', \varphi_j\sigma) \in \mathcal{R}. \tag{5.20}$$

This case is therefore done by (5.10), (5.11) and (5.20). □

Proof for Proposition 5.2. We must prove that for every formula $\varphi \in \mathrm{sHML}_{\mathbf{nf}}$ if $(\!|\, \varphi \,|\!) = m$

then $\mathsf{evtenf}(m, \varphi)$. We then prove that for every $\varphi \in \text{sHML}_{\mathbf{nf}}$, if $(\!|\varphi|\!)[s] \stackrel{t}{\Longrightarrow} m'[s']$ and $s' \in [\![\mathit{after}(\varphi, t)]\!]$ then $m'[s'] \sim s'$. We also refer to Proposition 5.3 and lemma 5.1 whose proofs are provided in Appendices B.3.1 and B.3.2 respectively starting on page 160.

Proposition 5.3 (Transparency)**.** For every system state $s \in \text{Sys}$ and $\varphi \in \text{sHML}_{\mathbf{nf}}$, if $s \in [\![\varphi]\!]$ then $(\!|\varphi|\!)[s] \sim s$. □

Lemma 5.1. For every formula $\varphi \in \text{sHML}_{\mathbf{nf}}$, system state s and trace t, if $(\!|\varphi|\!)[s] \stackrel{t}{\Longrightarrow} m'[s']$ then $\exists \psi \in \text{sHML}_{\mathbf{nf}} \cdot \psi = \mathit{after}(\varphi, t)$ and $(\!|\psi|\!) = m'$. □

Now, assume that

$$(\!|\varphi|\!)[s] \stackrel{t}{\Longrightarrow} m'[s'] \tag{5.21}$$

$$s' \in [\![\mathit{after}(\varphi, t)]\!] \tag{5.22}$$

and so from (5.21) and Lemma 5.1 we have that

$$\exists \psi \in \text{sHML}_{\mathbf{nf}} \cdot \psi = \mathit{after}(\varphi, t) \tag{5.23}$$

$$(\!|\mathit{after}(\varphi, t)|\!) = m' = (\!|\psi|\!). \tag{5.24}$$

Hence, knowing (5.22) and (5.23), by Proposition 5.3 (Transparency) we deduce that

$$(\!|\mathit{after}(\varphi, t)|\!)[s'] \sim s'. \tag{5.25}$$

We are therefore done since from (5.24) and (5.25) we can conclude that $m'[s'] \sim s'$ as required. □

We now proceed to show that the synthesised monitor $(\!|\varphi|\!)$ is also guaranteed to be —in some sense— optimal when enforcing φ. Recall from Definition 4.9 that in order to determine that a synthesised monitor $(\!|\varphi|\!)$ is the *most optimal* monitor enforcing φ, we must compare $(\!|\varphi|\!)$ to all adequate monitors, including those that perform insertions and replacements. Since we have limited ourselves to studying the enforceability of unidirectional transducers vis-a-vis action suppressions, we have not yet explored what properties we can enforce using action insertions and replacements.

In this first attempt at determining a level of optimality for our synthesised unidirectional transducers, we will limit our comparison to other suppression transducers only and thus aim to establish that the synthesised monitors are *SUP-optimal*. Having already established that $(\!|\varphi|\!)$ adequately enforces φ in Theorem 5.1, to prove that it is also SUP-optimal it suffices to prove the following result.

Theorem 5.2 (SUP-Optimal Enforcement)**.** For every system state s, system run t_τ and monitor m, if $\mathit{ec}(m) \subseteq \{\mathsf{SUP}\}$, $\mathsf{enf}(m, \varphi)$ and $s \stackrel{t_\tau}{\Longrightarrow}$ then $\mathit{mc}((\!|\varphi|\!), t_\tau) \leq \mathit{mc}(m, t_\tau)$. □

Once again the reader may safely skip this proof and proceed on page 54.

Proof. To simplify this proof we assume that SUPTRN represents the set of all suppression monitors, *i.e.,* $\text{SUPTRN} \stackrel{\text{def}}{=} \{\, m \mid \mathit{ec}(m)=\{\mathsf{SUP}\} \,\}$, and refer to the following lemmas proven in Appendices B.3.3 to B.3.5 starting on page 174.

Lemma 5.2. For every monitor $m \in \text{SUPTRN}$ and system run t_τ there exists a number N so that $\mathit{mc}(m, t_\tau) = N$. □

Lemma 5.3. For every action α and $m \in \text{SUPTRN}$, if $\mathsf{enf}(m, \bigwedge_{i\in I} [\{p_i, c_i\}]\varphi_i)$, $m \xrightarrow{\alpha\blacktriangleright\alpha} m'$, $\mathsf{mtch}(p_j, \alpha) = \sigma$ and $c_j\sigma \Downarrow \mathsf{true}$ for some $j \in I$ then $\mathsf{enf}(m', \varphi_j\sigma)$. □

Lemma 5.4. For every action α and $m \in \text{SUPTRN}$, if $\mathsf{enf}(m, \bigwedge_{i\in I} [\{p_i, c_i\}]\varphi_i)$ and $m \xrightarrow{\alpha\blacktriangleright\bullet} m'$ then $\mathsf{enf}(m', \bigwedge_{i\in I} [\{p_i, c_i\}]\varphi_i)$. □

We proceed to prove that for every system state s, system run t_τ and monitor m if $\mathit{ec}(m) \subseteq \{\mathsf{SUP}\}$, $\mathsf{enf}(m, \varphi)$ and $s \stackrel{t_\tau}{\Longrightarrow}$ then $\mathit{mc}(\langle\!|\, \varphi \,|\!\rangle, t_\tau) \leq \mathit{mc}(m, t_\tau)$. Since we limit our comparison to suppression monitors only, and since by Lemma 5.2 we know that for every suppression monitor m, $\mathit{mc}(m, t_\tau) = N$, we can instead prove that, for every monitor $m \in \text{SUPTRN}$, system run t_τ and state s, if $\mathsf{enf}(m, \varphi)$, $s \stackrel{t_\tau}{\Longrightarrow}$ and $\mathit{mc}(\langle\!|\, \varphi \,|\!\rangle, t_\tau) = N$ then $N \leq \mathit{mc}(m, t_\tau)$. We now proceed by rule induction on the transition step derivations and thus consider every case for $\mathit{mc}(\langle\!|\, \varphi \,|\!\rangle, t_\tau)$.

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

$$\mathit{mc}(\langle\!|\, \varphi \,|\!\rangle, \mu t'_\tau) = \mathit{mc}(m'_\varphi, t'_\tau) = N \tag{5.26}$$

which implies that

$$\langle\!|\, \varphi \,|\!\rangle[\mathsf{sys}(\mu t'_\tau)] \xrightarrow{\mu} m'_\varphi[\mathsf{sys}(t'_\tau)] \tag{5.27}$$

and also assume that

$$\mathsf{enf}(m, \varphi) \tag{5.28}$$

and that $s \stackrel{\mu t'_\tau}{\Longrightarrow}$. By the rules in our model we can infer that the reduction in (5.27) can result from either IASY when $\mu = \tau$ or IDEF and ITRN when $\mu = \alpha$. We consider each case individually.

- IASY: By rule IASY from (5.27) we know that $\mu = \tau$ and that

  $$m'_\varphi = \langle\!|\, \varphi \,|\!\rangle. \tag{5.29}$$

  Since $\mu = \tau$, by rule IASY any monitor m is bound to allow the prefixing τ action of $\mathsf{sys}(\tau t'_\tau)$ to execute, and so we have that

  $$m[\mathsf{sys}(\tau t'_\tau)] \xrightarrow{\tau} m[\mathsf{sys}(t'_\tau)]. \tag{5.30}$$

Hence, by (5.26), (5.28) and since $s \xRightarrow{\tau t'_\tau}$ entails $s \xrightarrow{\tau} s'$ and $s' \xRightarrow{t'_\tau}$ for some s', we can apply the *inductive hypothesis* and deduce that $N \leq mc(m, t'_\tau)$ so that by (5.30) and the definition of *mc*, we conclude that $N \leq mc(m, \tau t'_\tau)$ as required.

- iDef: From (5.27) and rule iDef we know that $\mu = \alpha$, $(\!|\, \varphi \,|\!) \xrightarrow{\alpha}\!\!\!\!\!/\;\;$ and that $m'_\varphi = \mathsf{id}$. Since id does not modify system actions, we can deduce that $mc(m'_\varphi, t'_\tau) = 0$ and so by the definition of *mc* we can infer that $mc((\!|\, \varphi \,|\!), \alpha t'_\tau) = 0$ as well. This means that we cannot find a monitor that performs fewer transformations, and so we conclude that $0 \leq mc(m, \alpha t'_\tau)$ as required.
- iTrn: From (5.27) and rule iTrn we know that $\mu = \alpha$ and that

$$(\!|\, \varphi \,|\!) \xrightarrow{\alpha \blacktriangleright \alpha} m'_\varphi. \tag{5.31}$$

We now inspect the cases for φ.

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

$$\begin{array}{rcl} (\!|\, \bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i \,|\!) & = & \mathsf{rec}\, Y. \sum\limits_{i \in I} \begin{cases} \{p_i, c_i, \bullet\}.Y & \textit{if } \varphi_i = \mathsf{ff} \\ \{p_i, c_i\}.(\!|\, \varphi_i \,|\!) & \textit{otherwise} \end{cases} \\ & = & \sum\limits_{i \in I} \begin{cases} \{p_i, c_i, \bullet\}.(\!|\, \bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i \,|\!) & \textit{if } \varphi_i = \mathsf{ff} \\ \{p_i, c_i\}.(\!|\, \varphi_i \,|\!) & \textit{otherwise} \end{cases} \end{array} \tag{5.32}$$

Since normalized conjunctions are disjoint, *i.e.,* $\#_{i \in I} \{p_i, c_i\}$, from (5.32) we can infer that the identity reduction in (5.31) can only happen when α matches an identity branch, $\{p_j, c_j\}.(\!|\, \varphi_j \,|\!)$ (for some $j \in I$), that is

$$\mathsf{mtch}(p_j, \alpha) = \sigma \textit{ and } c_j\sigma \Downarrow \mathsf{true}. \tag{5.33}$$

Hence, knowing (5.31) and (5.33), by rule eTrn we can infer that $m'_\varphi = (\!|\, \varphi_j\sigma \,|\!)$ and so from (5.26) we can infer that

$$mc(m'_\varphi, t'_\tau) = N \;\; \textit{where } m'_\varphi = (\!|\, \varphi_j\sigma \,|\!). \tag{5.34}$$

Since from (5.34) we also know that the monitor branch $\{p_j, c_j\}.(\!|\, \varphi_j \,|\!)$ is derived from a non-violating modal necessity, *i.e.,* $[\{p_j, c_j\}]\varphi_j$ where $\varphi_j \neq \mathsf{ff}$, we can infer that α does not violate the property and so it should not be modified by any other monitor m, as otherwise it would infringe the eventual transparency constraint of assumption (5.28) (for every $s \in [\![\varphi]\!]$).

Therefore, we can deduce that

$$m \xrightarrow{\alpha \blacktriangleright \alpha} m' \quad \text{(for some } m'\text{)} \tag{5.35}$$

and subsequently, knowing (5.35) and that $t_\tau = \alpha t'_\tau$ and $\mathsf{sys}(\alpha t'_\tau) \xrightarrow{\alpha} \mathsf{sys}(t'_\tau)$, by rule ıTRN and the definition of *mc* we infer that

$$\mathit{mc}(m, \alpha t'_\tau) = \mathit{mc}(m', t'_\tau). \tag{5.36}$$

Since from (5.28), (5.31), (5.33) and Lemma 5.3 we know that $\mathsf{enf}(m', \varphi_j\sigma)$, by (5.34) and since $s \xRightarrow{\alpha t'_\tau}$ entails that $s \xrightarrow{\alpha} s'$ and $s' \xRightarrow{t'_\tau}$ for some s', we can apply the *inductive hypothesis* and deduce that $N \leq \mathit{mc}(m', t'_\tau)$ and so from (5.36) we conclude that $N \leq \mathit{mc}(m, \alpha t'_\tau)$ as required.

– $\varphi = \max X.\varphi'$ and $X \in \mathbf{fv}(\varphi')$: Since $\varphi = \max X.\varphi'$, by the syntactic rules of $\text{sHML}_{\mathbf{nf}}$ we can determine that φ' cannot be ff or tt since $X \notin \mathbf{fv}(\varphi')$ otherwise, and it cannot be X since every logical variable must be guarded. Hence, φ' can only have the following form: $\max Y_1 \ldots Y_n. \bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i$. Therefore, by unfolding every fixpoint in $\max X.\varphi'$ we reduce our formula to $\varphi \overset{\text{def}}{=} \bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i\{{}^{\max X.\varphi'}/_X, \ldots\}$. The remainder of this proof is analogous to that of the subcase when $\varphi = \bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i$.

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

$$\mathit{mc}(\langle\!|\, \varphi \,|\!\rangle, \mu t'_\tau) = 1 + M \tag{5.37}$$

$$\textit{where } M = \mathit{mc}(m'_\varphi, t'_\tau) \tag{5.38}$$

which implies that

$$\langle\!|\, \varphi \,|\!\rangle[\mathsf{sys}(\mu t'_\tau)] \xrightarrow{\mu'} m'_\varphi[\mathsf{sys}(t'_\tau)] \;\; \textit{where } \mu' \neq \mu. \tag{5.39}$$

Also, assume that

$$\mathsf{enf}(m, \varphi) \tag{5.40}$$

and that $s \xRightarrow{\mu t'_\tau}$. Since we only consider suppression monitors, rule ıSUP is the only rule that can transition over $\mu' \neq \mu$ in (5.39), and so we have that $\mu = \alpha$ and $\mu' = \tau$ and that

$$\langle\!|\, \varphi \,|\!\rangle \xrightarrow{\alpha \blacktriangleright \bullet} m'_\varphi. \tag{5.41}$$

We now inspect the cases for φ.

- $\varphi \in \{\mathsf{ff}, \mathsf{tt}, X\}$: The case for X does not apply since $(\!|\, X \,|\!)$ does not yield a valid monitor. The cases for tt and ff do not apply as well since $(\!|\, \mathsf{ff} \,|\!) = (\!|\, \mathsf{tt} \,|\!) = \mathsf{id}$ which does not perform the reduction in (5.41).
- $\varphi = \bigwedge_{i\in I} [\{p_i, c_i\}]\varphi_i$ where $\#_{i\in I}\{p_i, c_i\}$: Since $\varphi = \bigwedge_{i\in I} [\{p_i, c_i\}]\varphi_i$, by the definition of $(\!|-|\!)$ we have that

$$\begin{array}{rcl} (\!|\, \bigwedge_{i\in I} [\{p_i, c_i\}]\varphi_i \,|\!) & = & \mathsf{rec}\, Y. \sum\limits_{i\in I} \begin{cases} \{p_i, c_i, \bullet\}.Y & \textit{if } \varphi_i = \mathsf{ff} \\ \{p_i, c_i\}.(\!|\, \varphi_i \,|\!) & \textit{otherwise} \end{cases} \\ & = & \sum\limits_{i\in I} \begin{cases} \{p_i, c_i, \bullet\}.(\!|\, \bigwedge_{i\in I} [\{p_i, c_i\}]\varphi_i \,|\!) & \textit{if } \varphi_i = \mathsf{ff} \\ \{p_i, c_i\}.(\!|\, \varphi_i \,|\!) & \textit{otherwise} \end{cases} \end{array} \tag{5.42}$$

  Since normalized conjunctions are disjoint *i.e.*, $\#_{i\in I}\{p_i, c_i\}$, from (5.42) we can infer that the suppression reduction in (5.41) is only possible when α matches a suppression branch, $\{p_j, c_j, \bullet\}.(\!|\, \bigwedge_{i\in I} [\{p_i, c_i\}]\varphi_i \,|\!)$ (for some $j \in I$), that was derived from a violating modal necessity $[\{p_j, c_j\}]\mathsf{ff}$, and so we can infer that

$$m'_\varphi = (\!|\, \bigwedge_{i\in I} [\{p_i, c_i\}]\varphi_i \,|\!) \tag{5.43}$$

$$\mathsf{mtch}(p_j, \alpha) = \sigma \textit{ and } c_j\sigma \Downarrow \mathsf{true} = \sigma. \tag{5.44}$$

  Knowing (5.44) and that $[\{p_j, c_j\}]\mathsf{ff}$ we can deduce that α violates the property and so for the soundness constraint of assumption (5.40) to hold, any other monitor m is obliged to somehow modify α. Since we only consider suppression monitors we can infer that

$$m \xrightarrow{\alpha \blacktriangleright \bullet} m' \quad (\text{for some } m') \tag{5.45}$$

  and so knowing (5.45) and that $\mathsf{sys}(\alpha t'_\tau) \xrightarrow{\alpha} \mathsf{sys}(t'_\tau)$, by rule ISUP and the definition of *mc* we have that

$$\mathit{mc}(m, \alpha t'_\tau) = 1 + \mathit{mc}(m', t'_\tau) \tag{5.46}$$

  Since from (5.40), (5.41) and Lemma 5.4 we know that $\mathsf{enf}(m', \bigwedge_{i\in I} [\{p_i, c_i\}]\varphi_i)$, by (5.38), (5.43), (5.46) and since $s \xrightarrow{\mu t'_\tau}$ entails that $s \xrightarrow{\mu} s'$ and $s' \xrightarrow{t'_\tau}$ (for some s'), we can apply the *inductive hypothesis* and deduce that $M \leq \mathit{mc}(m', t'_\tau)$. Hence, from (5.37), (5.38) and (5.46) we conclude that $1 + M \leq \mathit{mc}(m, \alpha t'_\tau)$ as required.
- $\varphi = \mathsf{max}\, X.\varphi'$ and $X \in \mathbf{fv}(\varphi')$: Since $\varphi = \mathsf{max}\, X.\varphi'$, by the syntactic rules of $\mathrm{sHML}_{\mathbf{nf}}$ we can determine that φ' cannot be ff or tt since $X \notin \mathbf{fv}(\varphi')$ otherwise, and it cannot be X since every logical variable must be guarded. Hence, φ' can only have the following form, *i.e.*, $\varphi \stackrel{\text{def}}{=} \bigwedge_{i\in I} [\{p_i, c_i\}]\varphi_i\{{}^{\mathsf{max}\, X.\varphi'}/_X, \ldots\}$. Hence by unfolding every fixpoint in $\mathsf{max}\, X.\varphi'$ we reduce our formula to $\varphi \stackrel{\text{def}}{=} \varphi\{\mathsf{max}\, X.\varphi/X\}$.

We thus omit the remainder of this proof as it becomes identical to that of the subcase when $\varphi = \bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i$.

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

$$mc(\langle\!|\, \varphi \,|\!\rangle, t_\tau) = |\, t_\tau \,| \quad (\text{where } t_\tau \in \{\mu t'_\tau, \varepsilon\}) \tag{5.47}$$

$$\langle\!|\, \varphi \,|\!\rangle[\mathsf{sys}(\mu t'_\tau)] \overset{\mu'}{\not\longrightarrow}. \tag{5.48}$$

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

- $t_\tau = \varepsilon$: This case holds trivially since by (5.47), (5.48) and the definition of *mc*, $mc(\langle\!|\, \varphi \,|\!\rangle, \varepsilon) = |\, \varepsilon \,| = 0$.
- $t_\tau = \mu t'_\tau$: This case does not apply since rule iDef prevents (5.48) from happening.

Case $mc(\langle\!|\, \varphi \,|\!\rangle, t_\tau)$ when $t_\tau \in \{\mu t'_\tau, \varepsilon\}$ and $\langle\!|\, \varphi \,|\!\rangle[\mathit{sys}(t_\tau)] \xrightarrow{\mu'} m'_\varphi[\mathit{sys}(t_\tau)]$. As we only consider suppression monitors, this case does not apply since $\langle\!|\, \varphi \,|\!\rangle[\mathsf{sys}(t_\tau)] \xrightarrow{\mu'} m'_\varphi[\mathsf{sys}(t_\tau)]$ can only be achieved via an insertion monitor and rule iIns. □

In light of Theorems 5.1 and 5.2, 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 $\psi \in \text{sHML}_{\mathbf{nf}}$ 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$\to$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 $\langle\!|\, \langle\!\langle \varphi \rangle\!\rangle \,|\!\rangle$, which is useful for tool construction.

5.2 The normalisation algorithm

Our normalisation algorithm, $\langle\!\langle - \rangle\!\rangle$, converts *closed* sHML formulas to sHML$_{\mathbf{nf}}$ formulas. It is based on Aceto *et al.*'s work [4] for determinising (possibly open) sHML formulas defining concrete actions, and on Rabinovich's work [91] for determinising systems of equations, both of which rely on the standard powerset construction for converting NFAs into DFAs. More specifically, the work by Aceto *et al.* does not explicitly provide an algorithm, but rather the algorithm is intertwined within the given proofs. It is also defined with respect to a version of sHML that only allows for specifying formulas about concrete system actions rather than symbolic ones.

Hence our improvements to Aceto *et al.*'s work [4] is two fold. First, we alleviate the process of having to go through the proofs in order to understand their determinisation algorithm. We extract the necessary steps from their proofs and present

them as a stand-alone algorithm that is separate from their respective proofs but which works for formulas defining concrete system actions (represented as singleton symbolic actions). Second, we define additional normalisation steps to allow for normalising sHML formulas that specify symbolic actions, which we evaluate by adding the necessary semantic preservation proofs.

In the end, our resulting algorithm does not only provide a procedural way for normalising sHML formulas with symbolic actions, but also allows us to prove another major result, this time attesting that the normalised sHML subset is equally expressive to sHML.

Theorem 5.3 (Normalisation Equivalence)**.** For every closed sHML formula φ there exists a formula $\psi \in \text{sHML}_{\mathbf{nf}}$ such that $[\![\varphi]\!]=[\![\psi]\!]$. □

5.2.1 Reconstructing sHML into sHML_{nf} with respect to singleton symbolic actions

We first define the normalization algorithm for *sHML* formulas that only specify *singleton symbolic actions*. Since these type of symbolic actions can only match a *single* visible action, they can be easily *distinguished statically* based on their syntactic form.

Example 5.3. Consider the following singleton symbolic actions $\{(x)!(y), x{=}\mathsf{a} \wedge y{=}3\}$ and $\{(x)!(y), x{=}\mathsf{a} \wedge y{=}4\}$, or for short $\{\mathsf{a}!3\}$ and $\{\mathsf{a}!4\}$. From their appearance one can immediately tell that these symbolic actions differ both syntactically and semantically, *i.e.,* $\{\mathsf{a}!3\} \neq \{\mathsf{a}!4\}$ implies $[\![\mathsf{a}!3]\!] \cap [\![\mathsf{a}!4]\!] = \varnothing$. By contrast, despite differing syntactically, non-singleton symbolic actions such as $\{(x)!3\}$ and $\{\mathsf{a}!(y)\}$ might not necessarily be disjoint, *i.e.,* although $\{(x)!3\} \neq \{\mathsf{a}!(y)\}$ we have that $[\![\{(x)!3\}]\!] \cap [\![\{\mathsf{a}!(y)\}]\!] = \{\mathsf{a}!\mathsf{ans}\}$. □

We define the algorithm in terms of the *four constructions* given below. As the first three constructions are derived directly from [4], we rely on the proofs provided in [4] to ensure their semantic preservation *i.e.,* that the result of each construction is semantically equivalent to its input. However, the last construction was not included in [4], and so we include an additional proof that ensures semantic preservation. The construction sequence is as follows:

§1. Equation construction: the formula is reformulated into a system of equations to enable easier manipulation in later stages (Section 5.2.1.1).

§2. Powerset construction: the resulting system of equations is restructured into an equivalent system of equations that defines syntactically disjoint conjunctions (Section 5.2.1.2).

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

Figure 5.2: A syntactic restriction for equated formulas.

§3. Formula reconstruction: the system of equations is converted back into an sHML formula with disjoint conjunctions which may define redundant fixpoints (Section 5.2.1.3).

§4. Redundant fixpoint removal: redundant fixpoint variable declarations, $\max X.\varphi$, are finally removed whenever variable X is not used in φ – this produces the required $\text{sHML}_{\textbf{nf}}$ formula (Section 5.2.1.4).

For conciseness, in the following subsections we refer to symbolic actions by the abbreviation *SAs*, and we use notation η to refer to an arbitrary symbolic action $\{p, c\}$. We also use notation $p[x_0 \dots x_n]$ to denote an arbitrary pattern that *binds* variables $x_0 \dots x_n$, and $c[x_0 \dots x_n]$ for a condition whose evaluation depends on the values of variables $x_0 \dots x_n$.

5.2.1.1 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 5.4 (System of Equations)**.** A system of equations is defined as a triple $(\mathit{Eq}, X, \mathcal{Y})$, 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.*, $\{X_1 = \psi_1, X_2 = \psi_2, \dots, X_n = \varphi_n\}$, where for $1 \leq i < j \leq n$, X_i is different from X_j, and each φ_i is a $\text{sHML}_{\textbf{eq}}$ expression as defined in Figure 5.2. □

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 $\mathit{Eq}(X_i) = \varphi_i$ when $X_i = \varphi_i \in \mathit{Eq}$. A maximal fixpoint $\max X.\varphi$ 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 5.4. A recursive formula such as $\max X_0.[\{\mathsf{a}?3\}]([\{\mathsf{a}!4\}]X_0 \wedge [\{\mathsf{a}!5\}]\mathsf{ff})$ can be represented as a system of four equations $(\mathit{Eq}, X_0, \mathcal{Y})$ where X_0 is the principal variable, $\mathit{Eq} \stackrel{\text{def}}{=} \{X_0 = [\{\mathsf{a}?3\}]X_2, X_2 = [\{\mathsf{a}!4\}]X_3 \wedge [\{\mathsf{a}!5\}]X_4, X_3 = [\{\mathsf{a}?3\}]X_2, X_4 = \mathsf{ff}\}$[1], and $\mathcal{Y} = \varnothing$ since all the logical variables defined in the system are *bound*, *i.e.*, equated to some

[1]Variable X_1 was left out on purpose. The reason why is explained in Example 5.5.

$$\langle\!\langle \varphi \rangle\!\rangle_1 \stackrel{\text{def}}{=} \begin{cases} \left(\{X_j = \mathsf{tt}\}, X_j, \varnothing\right) & \text{if } \varphi = \mathsf{tt} \\ \left(\{X_j = \mathsf{ff}\}, X_j, \varnothing\right) & \text{if } \varphi = \mathsf{ff} \\ \left(\{X_j = Y\}, X_j, \{Y\}\right) & \text{if } \varphi = Y \\ \left(\bigcup_{i\in I} \mathit{Eq}_i \cup \{X_j = \bigwedge_{i\in I} \mathit{Eq}_i(X_i)\}, X_j, \bigcup_{i\in I} \mathcal{Y}_i\right) & \text{if } \varphi = \bigwedge_{i\in I} \varphi_i \text{ and } \\ & \forall i \in I \cdot \langle\!\langle \varphi_i \rangle\!\rangle_1 = (\mathit{Eq}_i, X_i, \mathcal{Y}_i) \\ \left(\mathit{Eq} \cup \{X_j = [\eta]X_k\}, X_j, \mathcal{Y}\right) & \text{if } \varphi = [\eta]\psi \text{ and } \\ & \langle\!\langle \psi \rangle\!\rangle_1 = (\mathit{Eq}, X_k, \mathcal{Y}) \\ \left(\{Y{=}\mathit{Eq}(X_i)\} \cup \begin{Bmatrix} X_j{=}\mathit{Eq}(X_i) & \text{if } X_j{=}Y \in \mathit{Eq} \\ X_k{=}\varphi_k & \text{if } X_k{=}\varphi_k \in \mathit{Eq} \end{Bmatrix}, Y, \mathcal{Y}\backslash\{Y\}\right) & \text{if } \varphi = \max Y.\varphi' \text{ and } \\ & \langle\!\langle \varphi' \rangle\!\rangle_1 = (\mathit{Eq}, X_i, \mathcal{Y}) \end{cases}$$

where variable X_j is *fresh in all cases.*

Figure 5.3: The conversion algorithm from an sHML formula to a *SoE*.

sHML$_{\mathbf{eq}}$ formula. Notice how recursion is represented by referring to X_2 in the penultimate equation. □

Function $\langle\!\langle - \rangle\!\rangle_1 : \text{sHML} \to (\mathit{Eq} \times \text{VAR} \times \mathcal{P}(\text{VAR}))$ in Figure 5.3 compositionally analyses a given closed sHML 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 resulting *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 sHML$_{\mathbf{eq}}$ (and is thus invalid), since we assume closed formulas, this equation gets fixed when $\langle\!\langle - \rangle\!\rangle_1$ recurses back to the binding fixpoint, *i.e.,* in the case when $\varphi{=}\max X.\varphi'$.

In fact, fixpoints, $\max Y.\varphi$, are converted into equation $Y = \mathit{Eq}(X_i)$, where X_i is the principal variable of the *SoE* obtained from the recursive application on the continuation φ' *i.e.,* $\langle\!\langle \varphi' \rangle\!\rangle_1 = (\mathit{Eq}, X_i, \mathcal{Y})$. 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 when $\varphi = \max Y.\varphi'$. Equations of the sort $X_j = Y$ in *Eq* are reformulated into valid equations as $X_j = \mathit{Eq}(X_i)$ where X_i points to the same equation as Y; this ensures that every logical variable remains *guarded* by a modal necessity.

Example 5.5. Recall formula $\max X_0.[\{\mathsf{a}?3\}]([\{\mathsf{a}!4\}]X_0 \wedge [\{\mathsf{a}!5\}]\mathsf{ff})$ from Example 5.4. Using function $\langle\!\langle - \rangle\!\rangle_1$ we start constructing a *SoE* by analysing the fixpoint $\max X_0$. However, in order to add the required equation, it must first recursively analyse the rest of the formula. Hence, after analysing $[\{\mathsf{a}?3\}]([\{\mathsf{a}!4\}]X_0 \wedge [\{\mathsf{a}!5\}]\mathsf{ff})$, the function constructs the *SoE* $(\mathit{Eq}', X_1, \{X_0\})$, where X_1 is the principal variable, X_0 is open, and $\mathit{Eq}' \stackrel{\text{def}}{=} \{X_1{=}[\{\mathsf{a}?3\}]X_2, X_2{=}[\{\mathsf{a}!4\}]X_3 \wedge [\{\mathsf{a}!5\}]X_4, X_3{=}X_0, X_4{=}\mathsf{ff}\}$.

The function then returns to the initial case and continues analysing the fixpoint, where it uses the returned equation set Eq' to add equation $X_0 = \mathit{Eq}'(X_1)$ where

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

Figure 5.4: The $\text{sHML}^{\#}_{\textbf{eq}}$ syntax.

$Eq'(X_1) = [\{\mathsf{a}?3\}]X_2$. It also fixes the invalid equation $X_3 = X_0$ by changing it into $X_3 = Eq'(X_1)$. As a result we obtain the *SoE* $(Eq, X_0, \varnothing)$ where the equation set Eq is defined as $\{X_0{=}[\{\mathsf{a}?3\}]X_2, X_1{=}[\{\mathsf{a}?3\}]X_2, X_2{=}[\{\mathsf{a}!4\}]X_3 \wedge [\{\mathsf{a}!5\}]X_4, X_3{=}[\{\mathsf{a}?3\}]X_2, X_4{=}\mathsf{ff}\}$. Since variable X_1 is not reachable from the principal equation $X_0{=}[\{\mathsf{a}?3\}]X_2$, it becomes redundant and can thus be removed. Hence, we end up with the *SoE* presented in Example 5.4. □

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 $\langle\!\langle \varphi \rangle\!\rangle_1$ 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 $\langle\!\langle \varphi_i \rangle\!\rangle_1$ (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 5.6. Consider a reformulated version of φ_0 from Example 4.1 of Chapter 4 given by $\varphi_3 \stackrel{\text{def}}{=} \max X_0.([\{\mathsf{a}?\mathsf{req}\}]([\{\mathsf{a}!\mathsf{ans}\}][\{\mathsf{a}!\mathsf{ans}\}]\mathsf{ff} \wedge [\{\mathsf{a}!\mathsf{ans}\}][\{\mathsf{b}!\mathsf{log}\}]X_0)$. From $\langle\!\langle \varphi_3 \rangle\!\rangle_1$ we obtain $(Eq, X_0, \varnothing)$ where

$$Eq = \left\{ \begin{array}{l} X_0 = Eq(X_1) = [\{\mathsf{a}?\mathsf{req}\}]X_2, \quad X_1 = [\{\mathsf{a}?\mathsf{req}\}]X_2\,, \quad X_3 = [\{\mathsf{a}!\mathsf{ans}\}]X_5\,, \\ X_2 = Eq(X_3) \wedge Eq(X_4) = [\{\mathsf{a}!\mathsf{ans}\}]X_5 \wedge [\{\mathsf{a}!\mathsf{ans}\}]X_6, \quad X_4 = [\{\mathsf{a}!\mathsf{ans}\}]X_6\,, \\ X_5 = [\{\mathsf{a}!\mathsf{ans}\}]X_7, \quad X_6 = [\{\mathsf{b}!\mathsf{log}\}]X_0, \quad X_7 = \mathsf{ff} \end{array} \right\}.$$

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

Lemma 5.5. For every closed sHML formula φ, the *SoE* obtained from $\langle\!\langle \varphi \rangle\!\rangle_1$ has the same meaning as φ.

Proof. The proof follows from *Lemma 10* given in [4]. This lemma was originally proven in relation to formulas that define concrete visible actions $\alpha \in \text{Act}$. However, it still applies to formulas defining symbolic actions since the construction is independent of the type of action (*i.e.,* symbolic or concrete) described in the modal necessities. □

5.2.1.2 Powerset construction

In this step we convert a *SoE* into an equivalent *SoE* in which every equated formula meets the restrictions of $\text{sHML}^{\#}_{\textbf{eq}}$ defined in Figure 5.4. Conjunctions in the equated

$$\langle\!\langle (\mathit{Eq}, X_i, \mathcal{Y}) \rangle\!\rangle_2 \stackrel{\text{def}}{=} (\mathit{Eq}_{\#}, X_{\{i\}}, \mathcal{Y})$$

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

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

Figure 5.5: The powerset construction for systems of equations.

$\text{sHML}^{\#}_{\mathbf{eq}}$ formulas are now required to be guarded by *disjoint* modal necessities. Figure 5.5 presents $\langle\!\langle - \rangle\!\rangle_2 : (\mathit{Eq} \times \textsc{Var} \times \mathcal{P}(\textsc{Var})) \to (\mathit{Eq}_{\#} \times \textsc{Var} \times \mathcal{P}(\textsc{Var}))$ where for every logical variable X, $\mathit{Eq}_{\#}(X) \in \text{sHML}^{\#}_{\mathbf{eq}}$. 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 with a conjunction. This technique mimics the classic powerset construction for determinising automata in automata theory [91].

Specifically, $\langle\!\langle - \rangle\!\rangle_2$ 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 $j \in I$ such 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 resulting 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 defined in Figure 5.5.

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

Remark 5.1. Function $\langle\!\langle - \rangle\!\rangle_2$ 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 Section 5.2.2 we will see how additional transformations are required to ensure this for non-singleton *SAs*. □

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

Figure 5.6: The sHML$^{\#}$ syntax.

Example 5.7. Recall the *SoE* obtained in Example 5.6, *i.e.,* $(Eq, X_0, \varnothing)$ where

$$Eq = \left\{ \begin{array}{llll} X_0 = [\{\mathsf{a?req}\}]X_2, & X_2 = [\{\mathsf{a!ans}\}]X_5 \wedge [\{\mathsf{a!ans}\}]X_6, \\ X_5 = [\{\mathsf{a!ans}\}]X_7, & X_6 = [\{\mathsf{b!log}\}]X_0, & X_7 = \mathsf{ff} \end{array} \right\}.$$

Function $\langle\!\langle - \rangle\!\rangle_2$ generates every possible combination and merges the modal necessities where necessary. From $\langle\!\langle (Eq, X_0, \varnothing) \rangle\!\rangle_2$ we therefore obtain $(Eq_{\#}, X_{\{0\}}, \varnothing)$ where $Eq_{\#} = \{X_{\{0\}} = [\{\mathsf{a?req}\}]X_{\{2\}}, X_{\{2\}} = [\{\mathsf{a!ans}\}]X_{\{5,6\}}\} \cup Eq'_{\#}$. Notice how continuations X_5 and X_6 in $X_2 = [\{\mathsf{a!ans}\}]X_5 \wedge [\{\mathsf{a!ans}\}]X_6$ were combined into a single continuation in $X_{\{2\}} = [\{\mathsf{a!ans}\}]X_{\{5,6\}}$. As the algorithm constructs all combinations, it also includes those for $X_{\{5,6\}}$ as per $Eq'_{\#} = \{ X_{\{5,6\}} = [\{\mathsf{a!ans}\}]X_{\{7\}} \wedge [\{\mathsf{b!log}\}]X_{\{0\}}, X_{\{7\}} = \mathsf{ff}, \ldots \}$. From the resulting equation set, we omit the redundant combinations that are *not reachable* from the new principal variable $X_{\{0\}}$. □

Lemma 5.6. For every *SoE* $(Eq, X_0, \mathcal{Y})$, if $\langle\!\langle (Eq, X_0, \mathcal{Y}) \rangle\!\rangle_2 = (Eq', X_{\{0\}}, \mathcal{Y})$ then $(Eq, X_0, \mathcal{Y}) \equiv (Eq', X_{\{0\}}, \mathcal{Y})$ and for every $(X_i = \varphi_i) \in Eq'$, $\varphi_i \in \text{sHML}^{\#}_{\mathbf{eq}}$.

Proof. In [4] the authors present a version of $\langle\!\langle - \rangle\!\rangle_2$ which processes equations that equate formulas which only specify visible actions *i.e.,* $\alpha \in \text{Act}$. By definition syntactically different visible 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 [4]. In Section 5.2.2 we will present the necessary steps for ensuring that this criterion holds for every *SA*. □

5.2.1.3 Formula Reconstruction

With this step we convert the *SoE* back to a formula that adheres to the restrictions imposed by sHML$^{\#}$ defined in Figure 5.6. sHML$^{\#}$ requires conjunctions to be guarded by disjoint modal necessities, but allows for defining redundant fixpoint declarations.

Figure 5.7 presents $\langle\!\langle - \rangle\!\rangle_3 : (Eq_{\#}, \text{Var}, \mathcal{P}(\text{Var})) \to \text{sHML}^{\#}$, which internally uses the function $\sigma_{\mathsf{shml}} : (\text{sHML}^{\#} \times Eq) \to \text{sHML}^{\#}$ to construct the corresponding sHML$^{\#}$ formula. Internally, this function uses fvars to determine the set of free logical variables of a formula φ. Intuitively, fvars adds to the set the logical variable X when $\varphi = X$, but removes it if it is bound by a maximal fixpoint *i.e.,* when $\varphi = \max X.\psi$. It reapplies recursively for every branch φ_i of conjunction $\bigwedge_{i\in I} \varphi_i$ and for every continuation ψ when $\varphi = [p, c]\psi$. Truth and falsehood do not define free variables and so the empty set is returned.

$$\langle\!\langle (\mathit{Eq}, X_i, \mathcal{Y}) \rangle\!\rangle_3 \stackrel{\text{def}}{=} \sigma_{\mathsf{shml}}(X_i, \mathit{Eq})$$

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

$$\mathsf{fvars}(\varphi) \stackrel{\text{def}}{=} \begin{cases} \varnothing & \text{if } \varphi \in \{\mathsf{tt}, \mathsf{ff}\} \\ \{X\} & \text{if } \varphi = X \\ \mathsf{fvars}(\psi) \setminus \{X\} & \text{if } \varphi = \max X.\psi \\ \bigcup_{i\in I} \mathsf{fvars}(\varphi_i) & \text{if } \varphi = \bigwedge_{i\in I} \varphi_i \\ \mathsf{fvars}(\psi) & \text{if } \varphi = [\{p, c\}]\psi \end{cases}$$

Figure 5.7: Constructing a sHML$^{\#}$ formula from a *SoE*.

Equipped with the fvars function, the σ_{shml} function takes as input the principal variable X_i along with the equation set *Eq*. Since X_i is an open term, $\mathsf{fvars}(X_i) = \{X_i\}$, the function searches for equation $X_i{=}\varphi_i$ in *Eq* and converts it into a substitution environment which substitutes variable X_i with $\max X_i.\varphi_i$, *i.e.,* $\{\max X_i.\varphi_i/X_i\}$. This substitution is then applied to X_i and the function recurses with the substituted value, $\sigma_{\mathsf{shml}}(\max X_i.\varphi_i, \mathit{Eq})$. Recursion stops when the resulting formula φ becomes closed, *i.e.,* $\mathsf{fvars}(\varphi) = \varnothing$, in which case φ is returned as a result.

Example 5.8. Recall the *SoE* $(\mathit{Eq}_{\#}, X_{\{0\}}, \varnothing)$ obtained in Example 5.7, where

$$\mathit{Eq}_{\#} = \left\{ \begin{array}{l} X_{\{0\}} = [\{\mathsf{a?req}\}]X_{\{2\}}, \;\; X_{\{2\}} = [\{\mathsf{a!ans}\}]X_{\{5,6\}}, \\ X_{\{5,6\}} = [\{\mathsf{a!ans}\}]X_{\{7\}} \wedge [\{\mathsf{b!log}\}]X_{\{0\}}, \;\; X_{\{7\}} = \mathsf{ff} \end{array} \right\}.$$

By applying $\langle\!\langle - \rangle\!\rangle_3$ we obtain $\psi_4 \in$ sHML$^{\#}$ where $\psi_4 = \sigma_{\mathsf{shml}}(X_{\{0\}}, \mathit{Eq}_{\#})$ which yields $\max X_{\{0\}}.[\{\mathsf{a?req}\}]\max X_{\{2\}}.([\{\mathsf{a!ans}\}]\max X_{\{5,6\}}.([\{\mathsf{a!ans}\}]\max X_{\{7\}}.\mathsf{ff} \wedge [\{\mathsf{b!log}\}]X_{\{0\}}))$. □

Lemma 5.7. For every *SoE* $(\mathit{Eq}, X_{\{0\}}, \mathcal{Y})$, if $\langle\!\langle (\mathit{Eq}, X_{\{0\}}, \mathcal{Y}) \rangle\!\rangle_3 = \varphi$ then φ conveys the same meaning as $(\mathit{Eq}, X_{\{0\}}, \mathcal{Y})$ and that $\varphi \in$ sHML$^{\#}$.

Proof. Since construction $\langle\!\langle - \rangle\!\rangle_3$ is independent of the type of actions defined in the modal necessities of the given *SoE*, we refer to *Lemma 12* from [4] as proof that $\langle\!\langle - \rangle\!\rangle_3$ always produces a semantically equivalent formula $\varphi \in$ sHML$^{\#}$. □

5.2.1.4 Removing Redundant Fixpoints

The final construction produces a sHML$_{\mathbf{nf}}$ formula in which every logical variable X defined by a fixpoint $\max X.\varphi$, is free in the continuation formula φ (*i.e.,* $X \in \mathsf{fvars}(\varphi)$), meaning that X is used at least once in φ. We formalize this construction as function $\langle\!\langle - \rangle\!\rangle_4 : \text{sHML}^{\#} \to \text{sHML}_{\mathbf{nf}}$ in Figure 5.8. 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.

$$\langle\!\langle \varphi \rangle\!\rangle_4 \stackrel{\text{def}}{=} \begin{cases} \varphi & \text{if } \varphi \in \{\mathsf{ff}, \mathsf{tt}\} \\ \langle\!\langle \varphi' \rangle\!\rangle_4 & \text{if } \varphi{=}\max X.\varphi' \text{ and } X \notin \mathsf{fvars}(\varphi') \\ \max X.\langle\!\langle \varphi' \rangle\!\rangle_4 & \text{if } \varphi{=}\max X.\varphi' \text{ and } X \in \mathsf{fvars}(\varphi') \\ \bigwedge_{i\in I} [\eta_i]\langle\!\langle \varphi_i \rangle\!\rangle_4 & \text{if } \varphi{=}\bigwedge_{i\in I} [\eta_i]\varphi_i \end{cases}$$

Figure 5.8: Converting sHML$^{\#}$ formulas into sHML$_{\mathbf{nf}}$.

Example 5.9. The redundant fixpoints in ψ_4 from Example 5.8, can be removed via function $\langle\!\langle - \rangle\!\rangle_4$, thus obtaining the following sHML$_{\mathbf{nf}}$ formula:

$$\psi_5 \stackrel{\text{def}}{=} \max X_{\{0\}}.[\{\mathsf{a?req}\}][\{\mathsf{a!ans}\}]([\{\mathsf{a!ans}\}]\mathsf{ff} \wedge [\{\mathsf{b!log}\}]X_{\{0\}}).$$

Notice that the obtained formula ψ_5 is identical to φ_0 (modulo α-renaming) from Example 4.1 of Chapter 4, and are both definable via the sHML$_{\mathbf{nf}}$ syntax, and thus in *normal form*. □

Lemma 5.8. For every formula $\varphi \in \text{sHML}^{\#}$, $[\![\langle\!\langle \varphi \rangle\!\rangle_4]\!] = [\![\varphi]\!]$.

The proof is provided in Appendix B.3.6 on page 177.

We have presented a sequence of constructions that transform sHML formulas defining singleton *SAs* into their normalized equivalent in sHML$_{\mathbf{nf}}$. We thus conclude that when we *only* consider *singleton SAs*, Theorem 5.3 holds as a result of Lemmas 5.5 to 5.8.

5.2.2 Reconstructing sHML into sHML$_{\mathbf{nf}}$ with respect to *any* Symbolic Action

Up until now we have only considered normalizing 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 **§2**. However, as we previously argued in Example 5.3, modal necessities in general can also describe non-singleton *SAs* for which syntactic difference does not necessarily reflect their disjointness. Due to this issue, normalizing a non-singleton symbolic formula using the algorithm presented in Section 5.2.1, may sometimes fail to produce a semantically equivalent formula that is in normal form. The following example shows a specific instance where our algorithm fails.

Example 5.10. Consider φ_4 a variant of φ_2 from Example 5.1.

$$\varphi_4 \stackrel{\text{def}}{=} \max X_0.\left([\{(x_1)\mathsf{?req}, \mathsf{true}\}]\left(\begin{array}{l}[\{(x_2)\mathsf{!ans}, x_2 = x_1\}][\{(x_4)\mathsf{!ans}, x_4{=}x_2\}]\mathsf{ff} \wedge \\ [\{(x_3)\mathsf{!ans}, x_3{\neq}\mathsf{b} \wedge x_3{=}x_1\}][\{\mathsf{b!log}\}]X_0\end{array}\right)\right).$$

By applying **§1** we obtain $(Eq_4, X_0, \varnothing)$ where

$$Eq_4 = \left\{ \begin{array}{c} X_0 = [\{(x_1)?\mathsf{req}, \mathsf{true}\}]X_1, \\ X_1 = [\{(x_2)!\mathsf{ans}, x_2 = x_1\}]X_2 \wedge [\{(x_3)!\mathsf{ans}, x_3 \neq \mathsf{b} \wedge x_3 = x_1\}]X_3, \quad \ldots \end{array} \right\}.$$

However, when we apply **§2** the algorithm fails to combine *SAs* $\{(x_2)!\mathsf{ans}, x_2{=}x_1\}$ and $\{(x_3)!\mathsf{ans}, x_3{\neq}\mathsf{b} \wedge x_3{=}x_1\}$ as despite not being disjoint, they still differ syntactically. Hence, the equations defining these actions remain unmerged and we thus end up with $\left(Eq^4_{\#}, X_{\{0\}}, \varnothing\right)$ where

$$Eq^4_{\#} = \left\{ \begin{array}{c} X_{\{0\}} = [\{(x_1)?\mathsf{req}, \mathsf{true}\}]X_{\{1\}}, \\ X_{\{1\}} = [\{(x_2)!\mathsf{ans}, x_2{=}x_1\}]X_{\{2\}} \wedge [\{(x_3)!\mathsf{ans}, x_3{\neq}\mathsf{b} \wedge x_3{=}x_1\}]X_{\{3\}}, \quad \ldots \end{array} \right\}.$$

This error propagates through to steps **§3** and **§4** which produce a formula that despite being semantically equivalent to the original formula φ_4, 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 **§2** correctly merges the conjunctions within a formula.

Example 5.11. To give some intuition of the necessary steps, consider again actions $\{(x_1)!(y_1), y_1 = 5\}$ and $\{(x_2)!(y_2), x_2 = \mathsf{a}\}$. Despite being syntactically different, these *SAs* are not disjoint as both can match $\mathsf{a}!5$. The information they convey can however be encoded into 4 disjoint *SAs* as follows:

- $\{(x_1)!(y_1), y_1{=}5\}$ becomes $\{(x)!(y), y{=}5 \wedge x{=}\mathsf{a}\}$ and $\{(x)!(y), y{=}5 \wedge x{\neq}\mathsf{a}\}$, while
- $\{(x_2)!(y_2), x_2{=}\mathsf{a}\}$ becomes $\{(x)!(y), y{=}5 \wedge x{=}\mathsf{a}\}$ and $\{(x)!(y), y{\neq}5 \wedge x{=}\mathsf{a}\}$

where x and y 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, $\{(x)?(y), y{=}5 \wedge x{=}\mathsf{a}\}$ and $\{(x)?(y), y{\neq}5 \wedge x{=}\mathsf{a}\}$ are not only syntactically different, but their contradicting conditions, $y{=}5$ and $y{\neq}5$, also guarantee their disjointness. □

5.2.2.1 Additional steps for normalizing formulas defining any symbolic action

We formally define two additional constructions that must be applied between steps **§1** and **§2**. They convert conjunctions that are guarded by necessities defining non-disjoint *SAs*, into equivalent conjunctions guarded by *syntactically disjoint* modal necessities, *i.e.,* necessities describing *SAs* that are both syntactically and semantically different. The additional steps include:

§i. Conversion to uniform *SAs*: we inspect modal necessities defined at the *same modal depth* within a conjunction and substitute their data variables

Traversal Function.

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

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

$$\mathit{Eq}_{/\!/I} \stackrel{\text{def}}{=} \{\, X_i{=}\varphi_i \mid (X_i{=}\varphi_i) \in \mathit{Eq} \text{ and } i{\in}I \,\}$$

Figure 5.9: The breath first traversal algorithm.

with the *same* fresh variable whenever they define *pattern equivalent SAs* (Section 5.2.2.2).

§ii. Condition reformulation of conjunct *SAs*: once uniformed, the conjunctions are recomposed to define branches that are prefixed by modal necessities specifying syntactically disjoint *SAs* (Section 5.2.2.3).

Example 5.12. Recall $\{(x_1)?(y_1), y_1 = 5\}$ and $\{(x_2)?(y_2), x_2 = \mathsf{a}\}$ from Example 5.11. Construction **§i** uniforms the *SAs* by assigning the same fresh variables to both *SAs*, and so they become $\{(x)?(y), y = 5\}$ and $\{(x)?(y), x = \mathsf{a}\}$. Construction **§ii** then reformulates the conditions of the resulting *SAs* to obtain $\{(x)?(y), y{=}5 \wedge x{=}\mathsf{a}\}$ and $\{(x)?(y), y{\neq}5 \wedge x{=}\mathsf{a}\}$ which are disjoint. □

Internally, constructions **§i** and **§ii** both use the traverse function defined in Figure 5.9 to process the given set of equations in a *tree-like* manner. The higher order function $\mathsf{traverse} : (\mathit{Eq} \times \mathcal{P}(\textsc{Index}) \times \textsc{Fun} \times \textsc{Acc}) \to \textsc{Acc}$ 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 *breath first* traversal on the equation set, starting from the equation of the principal variable as the root of the tree traversal. For instance, in Figure 5.10 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 $(\mathit{Eq}, X_0, \mathcal{Y})$.

The children of the root are calculated via the $\mathsf{child}{:}(\mathit{Eq} \times \textsc{Index}) \to \mathcal{P}(\textsc{Index})$ function defined in Figure 5.9. 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 $\mathit{Eq}(X_i){=}\bigwedge_{j\in I} [\eta_j]X_j \wedge \varphi$ (where i is the parent's index) and returns the set containing the indices j of every prefixed branch $[\eta_j]X_j$, defined in the equated formula. For example, the children of $X_0{=}[\eta_1]X_1 \wedge [\eta_2]X_2 \wedge [\eta_3]X_3$ in Figure 5.10 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.

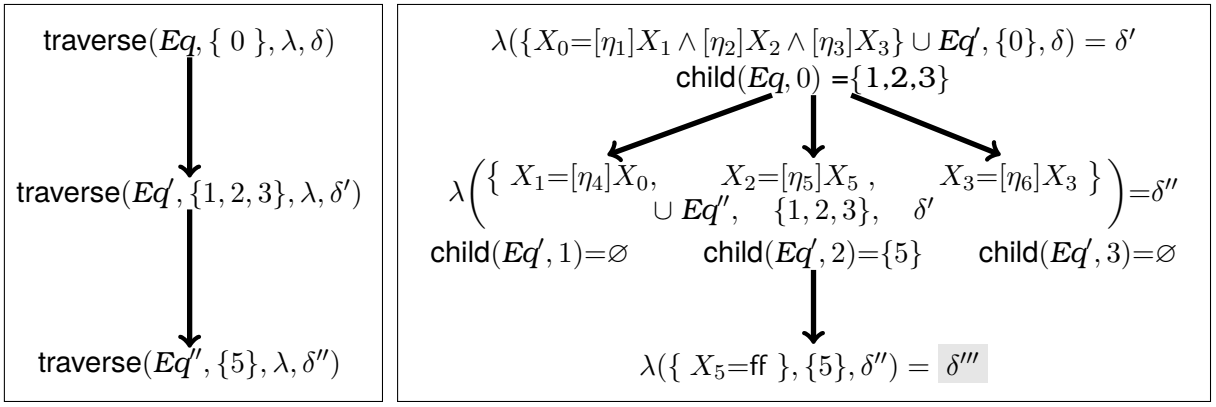

Figure 5.10: A pictorial view of an example equation set traversal.

The child function, however, omits adding the index j of logical variables X_j that are not part of the domain of the equation set *i.e.,* when $X_j \notin \mathbf{dom}(Eq)$. In this way, cycles in the traversal are avoided since the child function is always executed with respect to a smaller set of equations $Eq'{=}Eq\backslash Eq_{//I}$, *i.e.,* one which *does not* include the parent equations $Eq_{//I}$ (where I is the set of all parent indices). Moreover, cycles to the (immediate) parent are also avoided since the child function removes the parent's index from the returned set of child indices via the condition that $j \neq i$ (where i is the index of the parent equation). This guarantees termination.

Example 5.13. While analysing equation $X_1{=}[\eta_4]X_0$ in Figure 5.10, traverse is evaluated with respect to 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 child$(Eq', 1)$) index 0 is not added to the resulting set of child indices, since $X_0 \notin \mathbf{dom}(Eq')$'; this avoids cycling back to some (grand) parent equation. Moreover, when evaluating 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, *i.e.,* when $Eq{=}\varnothing$, or whenever no further children can be visited *i.e.,* for every branch i, child$(Eq, i){=}\varnothing$. The latter is an optimization which omits the redundant processing of equations that are not reachable from the principal equation.

$$\langle\!\langle (\mathit{Eq}, X_0, \mathcal{Y}) \rangle\!\rangle_{\text{(i)}} \stackrel{\text{def}}{=} (\mathsf{uni}(\mathit{Eq}, \zeta), X_0, \mathcal{Y}) \quad \text{where } \zeta = \mathsf{traverse}(\mathit{Eq}, \{0\}, \mathsf{partition}, \varnothing)$$

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

$$\mathsf{partition}(\mathit{Eq}, I, \zeta) \stackrel{\text{def}}{=} \left\{ \begin{array}{l|l} j \mapsto \zeta(i) \dot\cup \{^{z_0}/_{x_0} \ldots {}^{z_n}/_{x_n}\} & \forall i, l \in I \cdot \textit{if } \mathit{Eq}(i) = \bigwedge_{j \in I'} [\{p_j[x_0 \ldots x_n], c_j\}] X_j \wedge \varphi', \\ k \mapsto \zeta(l) \dot\cup \{^{z_0}/_{y_0} \ldots {}^{z_n}/_{y_n}\} & \mathit{Eq}(l) = \bigwedge_{k \in I''} [\{p_k[y_0 \ldots y_n], c_k\}] X_k \wedge \varphi'' \textit{ and} \\ & p_j \textit{ and } p_k \textit{are pattern equiv., (when } j \neq k\textit{) then} \\ & \textit{we assign the } \text{same} \textit{ fresh variables } z_0 \ldots z_n. \end{array} \right\} \cup \zeta$$

where $\sigma \dot\cup \{^z/_y\} = \sigma \cup \{^z/_y\}$ *iff* $y \notin \mathbf{dom}(\sigma)$.

Figure 5.11: The uniformity algorithm for symbolic actions.

With this mechanism in place, we now define steps **§i** and **§ii** in Sections 5.2.2.2 and 5.2.2.3 respectively.

5.2.2.2 Uniformity of Symbolic Actions.

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

Definition 5.5 (Uniform System of Equations)**.** An equation is *uniform* when the *SAs* of *pattern equivalent sibling* necessities define the exact *same* variable names. A system of equations is uniform when all of its equations are uniform. □

Example 5.14. Consider $X_0 = [\{(x_1)?(x_2), c_1[x_1, x_2]\}] X_1 \wedge [\{(y_1)?(y_2), c_2[y_1, y_2]\}] X_2$. Even though the sibling modal necessities in the equation define pattern equivalent *SAs*, they are *not uniform* as they *do not* use the same variable names. Uniformity can be attained by renaming both x_1 and y_1 to the same z_1 and similarly x_2 and y_2 to a fresh variable z_2. This therefore produces a uniform version of the equation, *i.e.*, $X_0 = [\{(z_1)?(z_2), c_1[z_1, z_2]\}] X_1 \wedge [\{(z_1)?(z_2), c_2[z_1, z_2]\}] X_2$. □

To automate this process, we thus use the construction defined in Figure 5.11, namely, $\langle\!\langle - \rangle\!\rangle_{\text{(i)}} : (\mathit{Eq}, \textsc{Var}, \mathcal{P}(\textsc{Var})) \to (\mathit{Eq}^{\mathsf{uni}}, \textsc{Var}, \mathcal{P}(\textsc{Var}))$. This construction internally uses the uni function to create the required uniform set of equations $\mathit{Eq}^{\mathsf{uni}}$ from a given equation set. Specifically, uni reconstructs the equation set by performing a linear scan during which it converts equations of the form $X_i = \bigwedge_{j \in I} [\eta_j] X_j \wedge \varphi$ into $X_i = \bigwedge_{j \in I} [\eta_j \zeta(j)] X_j \wedge \varphi$ (where $\zeta : \textsc{Index} \to \sigma$ 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 5.6 (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

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

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

Example 5.15. Consider the following system of equations $(Eq, X_0, \varnothing)$ where

$$Eq = \left\{ \begin{array}{c} X_0{=}[\{(x_1)?(x_2), x_1{\neq}\mathsf{a}\}]X_1 \wedge [\{(x_3)?(x_4), x_4{\neq}3\}]X_2, \quad X_3{=}\mathsf{ff}, \\ X_1{=}[\{(x_5)!(x_6), \mathsf{true}\}]X_3, \quad X_2{=}[\{(x_7)!(x_8), x_7{=}x_3\}]X_4, \quad X_4{=}\mathsf{ff} \end{array} \right\}.$$

For convenience, in Figure 5.12 we also represent these equations as a tree starting with the principal equation $X_0{=}[\{(x_1)?(x_2), x_1{\neq}\mathsf{a}\}]X_1 \wedge [\{(x_3)?(x_4), x_4{\neq}3\}]X_2$ as the root. We also assume the knowledge of a *well-formed* ζ map:

$$\zeta = \left\{ \begin{array}{c} 0 \mapsto \{\varnothing\}, \quad 1 \mapsto \zeta(0) \,\dot\cup\, \{{}^{x_1}/_{y_1}, {}^{x_2}/_{y_2}\}, \quad 2 \mapsto \zeta(0) \,\dot\cup\, \{{}^{x_3}/_{y_1}, {}^{x_4}/_{y_2}\}, \\ 3 \mapsto \zeta(1) \,\dot\cup\, \{{}^{x_5}/_{y_3}, {}^{x_6}/_{y_4}\}, \quad 4 \mapsto \zeta(2) \,\dot\cup\, \{{}^{x_7}/_{y_3}, {}^{x_8}/_{y_4}\} \end{array} \right\}.$$

As shown by the tree representation in Figure 5.12, actions $\{(x_1)?(x_2), x_1{\neq}\mathsf{a}\}$ and $\{(x_3)?(x_4), x_4{\neq}3\}$ 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 $\{{}^{x_1}/_{y_1}, {}^{x_2}/_{y_2}\}$ and $\{{}^{x_3}/_{y_1}, {}^{x_4}/_{y_2}\}$ respectively. Once the substitution is applied to both *SAs* we obtain $\{(y_1)?(y_2), y_1{\neq}\mathsf{a}\}$ and $\{(y_1)?(y_2), y_2{\neq}3\}$. Notice how the patterns in both of the necessities are now *syntactically equal*, meaning that the resulting equation $X_0{=}[\{(y_1)?(y_2), y_1{\neq}\mathsf{a}\}]X_1 \wedge [\{(y_1)?(y_2), y_2{\neq}3\}]X_2$ is now *uniform*.

Since $\{(x_5)!(x_6), \mathsf{true}\}$ and $\{(x_7)!(x_8), x_7{=}x_3\}$ are pattern equivalent siblings in X_0, to achieve uniformity ζ provides mappings $3 \mapsto \zeta(1) \dot\cup \{{}^{x_5}/_{y_3}, {}^{x_6}/_{y_4}\}$ and $4 \mapsto \zeta(2) \dot\cup \{{}^{x_7}/_{y_3}, {}^{x_8}/_{y_4}\}$ that rename these *SAs* to $\{(y_3)!(y_4), \mathsf{true}\}$ and $\{(y_3)!(y_4), y_3{=}y_1\}$. Notice how condition $x_7{=}x_3$ in $\{(x_7)!(x_8), x_7{=}x_3\}$ was also renamed to $y_3{=}y_1$ as variable x_3 was substituted by y_1 when its binding *SA* $\{(x_3)?(x_4), x_4{\neq}3\}$ was uniformed into $\{(y_1)?(y_2), y_2{\neq}3\}$. This substitution was possible since mapping $\zeta(4)$ also includes the substitutions returned by the parent's index *i.e.,* $\zeta(2)$. It therefore 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 by performing a breath first traversal on the given equation set, via the traverse function, using the partition function (defined in Figure 5.11) as the projection function λ. The function partition:$(Eq \times \mathcal{P}(\textsc{Index}) \times \textsc{Acc}) \to \textsc{Acc}$ 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.

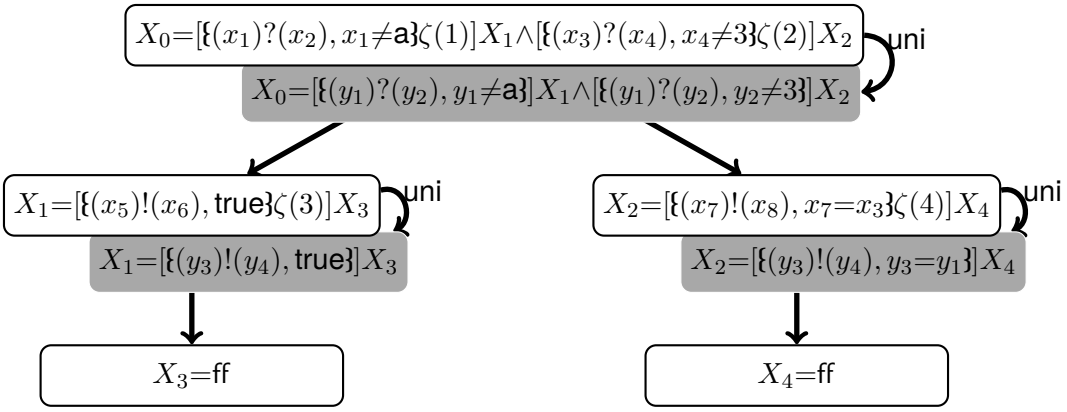

Figure 5.12: A Tree representation of the uni traversal performed on Eq.

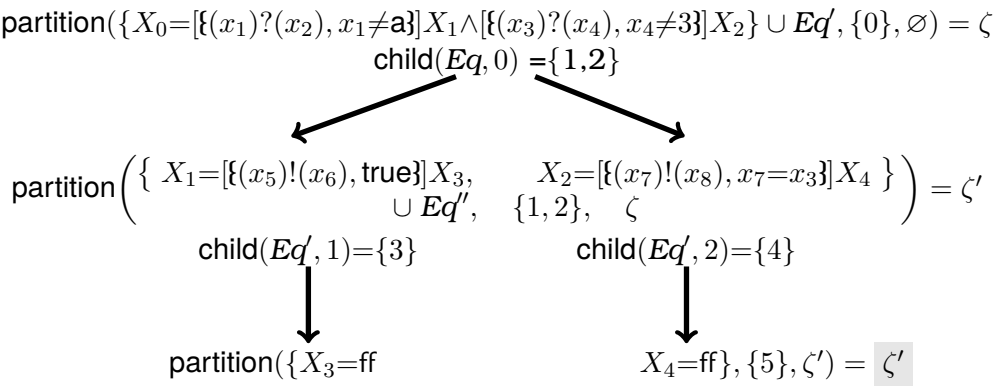

Figure 5.13: A breath first traversal using partition to obtain ζ.

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 5.16. Recall $(Eq, X_0, \varnothing)$ from Example 5.15 where

$$Eq = \left\{ \begin{array}{ccc} X_0{=}[\{(x_1)?(x_2), x_1{\neq}\text{a}\}]X_1 \wedge [\{(x_3)?(x_4), x_4{\neq}3\}]X_2, & X_3{=}\text{ff}, \\ X_1{=}[\{(x_5)!(x_6), \text{true}\}]X_3, & X_2{=}[\{(x_7)!(x_8), x_7{=}x_3\}]X_4, \quad X_4{=}\text{ff} \end{array} \right\}.$$

Figure 5.13 depicts the breath 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 $\zeta{=}\varnothing \cup \{1 \mapsto \varnothing \,\dot\cup\, \{{}^{y_1}/_{x_1}, {}^{y_2}/_{x_2}\}, 2 \mapsto \varnothing \,\dot\cup\, \{{}^{y_1}/_{x_3}, {}^{y_2}/_{x_4}\}\}$. As shown in Example 5.15, this allows for the sibling necessities defined in X_0 to be uniformed. The ζ map is further extended into $\zeta'{=}\zeta \cup \{3 \mapsto \zeta(1) \dot\cup \{{}^{y_3}/_{x_5}, {}^{y_4}/_{x_6}\}, 4 \mapsto \zeta(2) \dot\cup \{{}^{y_3}/_{x_7}, {}^{y_4}/_{x_8}\}\}$, since the partition function recognises that sibling SAs {$(x_5)!(x_6)$, true} and {$(x_7)!(x_8), x_7{=}x_3$} are also pattern equivalent. It therefore maps variables x_5, x_7 to the same fresh variable y_3, and x_6, x_8 to y_4. □

Lemma 5.9. For every *SoE* $(Eq, X_0, \mathcal{Y})$ if $\langle\!\langle (Eq, X_0, \mathcal{Y}) \rangle\!\rangle_{\text{(i)}} = (Eq', X_0', \mathcal{Y}')$ then $(Eq, X_0, \mathcal{Y}) \equiv (Eq', X_0', \mathcal{Y}')$ and $(Eq', X_0', \mathcal{Y}')$ is *uniform*.

Proof. To prove this statement we assume knowledge of Lemmas 5.10 and 5.11 which are proven in Appendices B.3.7 and B.3.8 respectively starting on page 183.

Lemma 5.10. For every equation set *Eq* if $\mathsf{traverse}(Eq, \{0\}, \mathsf{partition}, \varnothing)$=$\zeta$ then ζ is a *well-formed* map for *Eq*.

Lemma 5.11. 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 $\langle\!\langle (Eq, X_0, \mathcal{Y}) \rangle\!\rangle_{\text{(i)}} = (Eq', X_0', \mathcal{Y}')$ and so by the definition of $\langle\!\langle - \rangle\!\rangle_{\text{(i)}}$ we have that $X_0' = X_0$, $\mathcal{Y}' = \mathcal{Y}$ and $Eq' = \mathsf{uni}(Eq, \zeta)$ where $\zeta = \mathsf{traverse}(Eq, \{0\}, \mathsf{partition}, \varnothing)$ from which by Lemma 5.10 we can deduce that ζ is a *well-formed* map for *Eq*. This means that from Lemma 5.11 we can infer that

$$\mathsf{uni}(Eq, \zeta) \equiv Eq \tag{5.49}$$

$$\text{every equation } (X_k = \psi_k) \in \mathsf{uni}(Eq, \zeta) \text{ is } \textit{uniform}. \tag{5.50}$$

Hence, since from (5.49) we know that the uniformed equation set conveys the same meaning as *Eq*, and since from (5.50) we know that every equation is uniform, we conclude that

$$(Eq, X_0, \mathcal{Y}) \equiv (Eq', X_0', \mathcal{Y}') \text{ and that } (Eq', X_0', \mathcal{Y}') \text{ is } \textit{uniform} \tag{5.51}$$

as required, and so we are done. □

5.2.2.3 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 5.7 (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 5.17. As per Definition 5.7, we can thus infer that equation

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

is *equi-disjoint* since there does not exist a system action that can satisfy both $\{(x)?(y), y{>}5\}$ and $\{(x)?(y), y{\leq}5\}$. The only two branches that are satisfied by common actions are $[\{(x)?(y), y{>}5\}]X_1$ and $[\{(x)?(y), y{>}5\}]X_2$ but they are both prefixed by *syntactically equal* modal necessities. However, we can immediately conclude that the modal necessities in equation $X_1 = [\{(x_1)?(y_1), \mathsf{true}\}]X_4 \wedge [\{(x_1)?(y_1), y_1 \neq 5\}]X_5$ are *not* equi-disjoint. □

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

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

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

Figure 5.14: The Conjunction Reformulation Algorithm.

Figure 5.14 presents function $\langle\!\langle - \rangle\!\rangle_{\text{(ii)}} : (\mathit{Eq}^{\mathsf{uni}}, \textsc{Var}, \mathcal{P}(\textsc{Var})) \to \mathit{Eq}^{\mathsf{ed}}$ for recomposing uniform *SoEs* into equi-disjoint ones. Internally, this function uses the traverse function to perform a breath first traversal on the given uniform equation set, $\mathit{Eq}^{\mathsf{uni}}$, starting from the principal equation, *i.e.,* with I=$\{0\}$. While conducting the traversal, it applies the cond_comb function to reconstruct the uniform conjunctions, defined in $(X_i{=}\varphi_i) \in \mathit{Eq}^{\mathsf{uni}}$, into equi-disjoint ones, thereby producing an *equi-disjoint* equation set $\mathit{Eq}^{\mathsf{ed}}$ at the end of the traversal.

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

To compute these truth combinations, the cond_comb function starts by computing the child indices of the current sibling equations – denoted by I – by using the child function, *i.e.,* $I'{=}\bigcup_{l \in I} \mathsf{child}(\mathit{Eq}, l)$. 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 conditions that are defined by the modal necessities that: guard the branches identified by the indices in I', and that specify *syntactically equal* patterns. 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 condition c_j of the

branch that is currently being reconstructed *i.e.,* $[\{p_j, c_j\}]X_j$, is *true*. For instance, $\mathbb{C}(1, \{1,2,3\})$ issues combinations $\{(c_1 \wedge c_2 \wedge c_3),\ (c_1 \wedge c_2 \wedge \neg c_3),\ (c_1 \wedge \neg c_2 \wedge c_3),\ (c_1 \wedge \neg c_2 \wedge \neg c_3)\}$ 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 resulting equations are thus *equi-disjoint* as the reconstructed conditions ensure that a visible action α can *never* satisfy multiple symbolic necessities in the reconstructed branches, unless they 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 5.18. 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 reconstruct branches $[\{p, c_2\}]X_2$ and $[\{p, c_3\}]X_3$ in the same way such that the resulting equation becomes:

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

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. This means that continuation X_1 is only reachable when c_1 is true, and respectively X_2 and X_3 when c_2 and c_3 are true. Hence as argued earlier, the (underlined) conditions in the reconstructed equation are *never negated* when prefixing the respective logical variable. □

Lemma 5.12. For every system of equations, $(Eq, X_0, \mathcal{Y})$, if $(Eq, X_0, \mathcal{Y})$ is *uniform* then $\langle\!\langle (Eq, X_0, \mathcal{Y}) \rangle\!\rangle_{\text{(ii)}} \equiv (Eq, X_0, \mathcal{Y})$ and $\langle\!\langle (Eq, X_0, \mathcal{Y}) \rangle\!\rangle_{\text{(ii)}}$ is *equi-disjoint*. □

Proof. To prove this lemma we assume the knowledge of Lemma 5.13 that is proven in Appendix B.3.9 on page 71.

Lemma 5.13. If every equation $(X_j = \varphi_j) \in Eq$ is *uniform* then all equations $(X_k = \psi_k) \in \mathsf{traverse}(Eq, \{0\}, \mathsf{cond\_comb}, \varnothing)$ are *equi-disjoint* and $Eq \equiv \mathsf{traverse}(Eq, \{0\}, \mathsf{cond\_comb}, \varnothing)$.

Now, lets assume that $(Eq, X_0, \mathcal{Y})$ is *uniform* which means that every equation

$(X_j{=}\varphi_j) \in \mathit{Eq}$ is uniform, and so by Lemma 5.13 we deduce that

$$\mathit{Eq} \equiv \mathsf{traverse}(\mathit{Eq}, \{0\}, \mathsf{cond\_comb}, \varnothing), \text{ and that} \tag{5.52}$$

$$\forall (X_k{=}\psi_k) \in \mathsf{traverse}(\mathit{Eq}, \{0\}, \mathsf{cond\_comb}, \varnothing) \cdot \mathsf{eqn}\ (X_k{=}\psi_k) \text{ is } \textit{equi-disjoint}. \tag{5.53}$$

Now since $\langle\!\langle (\mathit{Eq}, X_0, \mathcal{Y}) \rangle\!\rangle_{\text{(ii)}} = (\mathsf{traverse}(\mathit{Eq}, \{0\}, \mathsf{cond\_comb}, \varnothing), X_0, \mathcal{Y})$, by (5.52) we can conclude that $\langle\!\langle (\mathit{Eq}, X_0, \mathcal{Y}) \rangle\!\rangle_{\text{(ii)}} \equiv (\mathit{Eq}, X_0, \mathcal{Y})$, and by (5.53) that the resulting *SoE* $\langle\!\langle (\mathit{Eq}, X_0, \mathcal{Y}) \rangle\!\rangle_{\text{(ii)}}$ is also *equi-disjoint* as required, and so we are done. □

In Example 5.10 we had shown that the algorithm presented in Section 5.2.1 fails when dealing with non-singleton *SA*s. This can now be resolved by applying steps **§i** and **§ii** prior to applying **§2** – we leave this as an exercise to the reader.

With the extended normalization algorithm we can finally conclude that Theorem 5.3 (Normalisation Equivalence) also holds for any sHML formula (defining any kind of *SA*s) as a result of Lemma 5.5 followed by Lemmas 5.9 and 5.12, and then by Lemmas 5.7 and 5.8.

5.3 Summary

In this chapter we have studied the enforceability of μHML formulas in the context of the unidirectional enforcement model introduced in Figure 3.2 of Chapter 3. In particular, we have identified the safety subset sHML to be enforceable by suppression monitors. As a result of this investigation we have produced the following contributions:

(i) A synthesis function that constructs action suppression monitors from normalised sHML formulas, Definition 5.2.

(ii) The proofs for Theorems 5.1 and 5.2 ascertaining that the synthesised suppression monitors enforce their respective formula both adequately and SUP-optimally as defined by Definitions 4.8 and 4.9. Since Definition 4.8 is the strictest definition for adequate enforcement, the synthesised monitors are also guaranteed to be adequate as stated by our weaker definitions 4.4 and 4.6.

(iii) A normalisation procedure that converts sHML formulas into a normalised $\text{sHML}_{\mathbf{nf}}$ formula.

(iv) Semantic preservation proofs for steps **§4**, **§i** and **§ii** of the normalisation procedure. Along with the proofs presented in [4] for **§1** - **§3**, they allow us to conclude that sHML and $\text{sHML}_{\mathbf{nf}}$ are equally expressive even though $\text{sHML}_{\mathbf{nf}}$ is a syntactic subset of sHML.

6. Maximal expressiveness

So far the results of Chapter 5 determine that sHML formulas can be adequately enforced by optimal suppression monitors. However, they do not answer whether sHML is *maximally expressive*, that is, whether it is the largest μHML subset that can be enforced by these type of monitors. Knowing this result would entail that if a μHML formula φ is enforceable via action suppressions but is not specified using the sHML syntax, then there must be some sHML formula ψ that conveys the same meaning as φ, *i.e.*, $[\![\varphi]\!] = [\![\psi]\!]$

Investigating maximal expressiveness requires establishing a result of *expression-completeness*, by showing that, in some sense, every suppression monitor corresponds to a formula in sHML as stated by Definition 4.8 (Strong enforcement). This approach, however, becomes problematic as for some suppression monitors there might not exist a μHML formula that they can adequately enforce in the sense of Definition 4.8.

In this chapter we thus identify a structural criterion that a suppression monitor must adhere to in order to be able to adequately enforce some μHML formula in the sense of Definition 4.8. Suppression monitors that do not follow this criterion are therefore deemed unsound. We then limit ourselves to sound suppression monitors and proceed to investigate the expression-completeness problem of sHML which allows us to establish whether sHML is the maximally expressive subset that can be enforced using suppression monitors.

6.1 Sound suppression monitors

As with any reasonably expressive model, the high expressiveness of the unidirectional enforcement model presented in Figure 3.2 of Chapter 3 allows for defining suppression monitors that make little sense in view of Definitions 4.4, 4.6 and 4.8. In particular, we have reason to believe that the ability to *persistently suppress*

invalid system actions is one major requirement for a suppression monitor to be *sound*.

Definition 6.1 (Persistent Suppression Monitors)**.** A monitor m performs *persistent suppressions*, if for every reachable state m' that performs a suppression transformation, $\alpha_{\blacktriangleright}\bullet$, it can then suppress α *infinitely often* from then onwards. Formally, we say that m is *persistent* iff for every identity transformation trace κ_{ID}, monitor states m' and m'' and action α,

$$\textit{if } m \xRightarrow{\kappa_{\text{ID}}} m' \textit{ and } m' \xrightarrow{\alpha_{\blacktriangleright}\bullet} m'' \textit{ then } \textit{inf}(m'', \alpha)$$

where $\kappa_{\text{ID}} ::= \varepsilon \mid (\alpha_{\blacktriangleright}\alpha).\kappa_{\text{ID}}$ (for every $\alpha \in \text{Act}$) and $\textit{inf}(m'', \alpha) \stackrel{\text{def}}{=} \forall N \in \text{Nat}.\ \exists m''' \cdot m'' \xrightarrow{\alpha_{\blacktriangleright}\bullet}{}^{N} m'''$. We also define PSupTrn as the set defining all persistent suppression monitors, that is, $\text{PSupTrn} \stackrel{\text{def}}{=} \{\ m \mid m \text{ is persistent}\ \}$. □

Example 6.1. To get an intuition for why this criterion is important, consider the following suppression monitor $m_{\text{np}} \stackrel{\text{def}}{=} \{\alpha, \mathsf{true}, \bullet\}.\mathsf{id}$ and formula $\varphi_5 = [\{\alpha\}]\mathsf{ff}$. When instrumented with system $\alpha.\mathsf{nil}$, monitor m_{np} successfully prevents the invalid action α from executing at runtime and so $m_{\text{np}}[\alpha.\mathsf{nil}] \in [\![\varphi_5]\!]$. However, it does not work for system $\alpha.\alpha.\mathsf{nil}$ since m_{np} does *not* persist in suppressing α and so $m_{\text{np}}[\alpha.\alpha.\mathsf{nil}] \xRightarrow{\alpha} \mathsf{id}[\alpha.\mathsf{nil}]$ which means that $m_{\text{np}}[\alpha.\alpha.\mathsf{nil}] \notin [\![\varphi_5]\!]$ thus breaching the soundness constraint of our enforcement adequacy definitions. On the other hand, monitor $m_{\text{p}} \stackrel{\text{def}}{=} \mathsf{rec}\, X.\{\alpha, \mathsf{true}, \bullet\}.X$ adequately enforces φ_5 since it keeps on suppressing the invalid action α infinitely often. □

This example thus demonstrates that suppression monitors that fail to suppress actions in a *persistent* manner might be problematic and unsound. For this reason, we now prove that these monitors are necessarily *unsound* and can therefore be ignored during our investigation of the maximal expressiveness of sHML. Put differently, in Theorem 6.1 we prove that when a monitor reaches a point whereby it suppresses an action α but does not persist in suppressing it, cannot adequately enforce any formula expressible via μHML.

Theorem 6.1 (Unsound Non-Persistent Suppression Monitors)**.** For all suppression monitors $m \in \text{SupTrn}$, $\neg(\forall \kappa_{\text{ID}}, m', m'', \alpha \cdot\ \textit{if } m \xRightarrow{\kappa_{\text{ID}}} m' \textit{ and } m' \xrightarrow{\alpha_{\blacktriangleright}\bullet} m'' \textit{ then } \textit{inf}(m'', \alpha))$ implies $\nexists \varphi \in \mu\text{HML} \cdot \mathsf{enf}(m, \varphi)$. □

To prove Theorem 6.1 we use function $\textit{systr}(-)$ to extract a system trace from the transformation trace of a suppression monitor.

Definition 6.2 (System Trace Extraction)**.**

$$\textit{systr}(\kappa) \stackrel{\text{def}}{=} \begin{cases} \alpha.\textit{systr}(\kappa') & \textit{if } \kappa = (\alpha_{\blacktriangleright}\gamma)\kappa' \\ \varepsilon & \textit{if } \kappa = \varepsilon \end{cases}$$

We also make use of Lemma 6.1 which we prove in Appendix B.4.1 on page 194.

Lemma 6.1. For every formula φ, monitors m and m', transformation trace κ_{ID}, and trace u, if $m \xRightarrow{\kappa_{\text{ID}}} m'$ and $\mathsf{sys}(u) \notin [\![\varphi]\!]$ and $\mathit{zip}(u, \kappa_{\text{ID}}) = u$ then $\neg\mathsf{enf}(m, \varphi)$.

Proof for Theorem 6.1. We proceed towards a contradiction. Assume that

$$\neg(\forall \kappa_{\text{ID}}, m', m'', \alpha \cdot \mathit{if}\ m \xRightarrow{\kappa_{\text{ID}}} m'\ \mathit{and}\ m' \xrightarrow{\alpha_{\blacktriangleright\bullet}} m''\ \mathit{then}\ \mathit{inf}(m'', \alpha)) \tag{6.1}$$

$$\equiv \quad \exists \kappa_{\text{ID}}, m', m''', \alpha, N \in \textsc{Nat} \cdot m \xRightarrow{\kappa_{\text{ID}}} m'\ \mathit{and}\ m' \xrightarrow{\alpha_{\blacktriangleright\bullet}}{}^{N+1}\ m'''\ \mathit{and}\ m''' \xrightarrow{\alpha_{\blacktriangleright\bullet}}\!\!\!\!\!\!/\ \rightarrow$$

(for some $N \in \textsc{Nat}$), and also that

$$\exists \varphi \in \mu\text{HML} \cdot \mathsf{enf}(m, \varphi) \tag{6.2}$$

which means that for *every* system s

$$\mathsf{senf}(m, \varphi) \stackrel{\text{def}}{=} m[s] \in [\![\varphi]\!] \tag{6.3}$$

and that $\mathsf{evtenf}(m, \varphi)$ which means that we also know that $\mathsf{tenf}(m, \varphi)$ and that

$$\mathsf{ttenf}(m, \varphi) \stackrel{\text{def}}{=} \forall t, t', t'' \cdot \mathit{if}\ \mathsf{sys}(t) \in [\![\varphi]\!]\ \mathit{and}\ m[\mathsf{sys}(t)] \xRightarrow{t'} m'[\mathsf{sys}(t'')]\ \mathit{then}\ t = t'; t''. \tag{6.4}$$

Let us now extract a trace $u; \alpha^{N+1}$ from the transformation trace $\kappa_{\text{ID}}(\alpha_{\blacktriangleright\bullet})^{N+1}$ of monitor m, via the function *systr* (defined in Definition 6.2). We then use an extended version of this trace that is suffixed by an additional α action, $u; \alpha^{N+2}$ to construct a trace system. The resultant system, $\mathsf{sys}(u; \alpha^{N+2})$, can thus exhibit the following behaviour, *i.e.,* that

$$\mathsf{sys}(u; \alpha^{N+2}) \xRightarrow{u} \mathsf{sys}(\alpha^{N+2}) \xrightarrow{\alpha}{}^{N+1}\ \mathsf{sys}(\alpha) \xrightarrow{\alpha} \mathsf{nil}. \tag{6.5}$$

So far we do not know whether the attained trace system, $\mathsf{sys}(u; \alpha^{N+2})$, actually satisfies or violates φ, we thus inspect both cases.

$\mathsf{sys}(u; \alpha^{N+2}) \in [\![\varphi]\!]$: In this case, from (6.4) we can immediately deduce that

$$\forall t', t'' \cdot \mathit{if}\ m[u; \alpha^{N+2}] \xRightarrow{t'} m'[\mathsf{sys}(t'')]\ \mathit{then}\ u; \alpha^{N+2} = t'; t''. \tag{6.6}$$

However, since we know that $u = \mathit{systr}(\kappa_{\text{ID}})$ and that κ_{ID} is a trace of identity transformations, we infer that $\mathit{zip}(u, \kappa_{\text{ID}}) = u$ and so by (6.1), (6.5) and Proposition 3.2 (zipping) we have that

$$m[\mathsf{sys}(u; \alpha^{N+2})] \xRightarrow{u} m'[\mathsf{sys}(\alpha^{N+2})] \xrightarrow{\tau}{}^{N+1}\ m'''[\mathsf{sys}(\alpha)] \xrightarrow{\alpha} m''''[\mathsf{nil}] \tag{6.7}$$

(for some monitor m''''). Hence, the monitored execution of (6.7) generates the run $u; \tau^{N+1}\alpha$ which is not equal to the original system run of $\mathsf{sys}(u; \alpha^{N+2})$ *i.e.,* $u; \tau^{N+1}\alpha \neq u; \alpha^{N+2}$. The monitored execution of (6.7) thus infringes the trace transparency criterion of Equation (6.4) (along with the stronger constraints of transparency and eventual transparency), and as a result (6.7) contradicts with (6.6).

$\mathsf{sys}(u; \alpha^{N+2}) \notin [\![\varphi]\!]$: For (6.3) to hold, we can infer that when $s = \mathsf{sys}(u; \alpha^{N+2})$, monitor m must somehow modify the execution of the system. In fact, when we zip (6.5)

to (6.1) using Proposition 3.2, we get that

$$m[\mathsf{sys}(u;\alpha^{N+2})] \stackrel{u}{\Longrightarrow} m'[\mathsf{sys}(\alpha^{N+2})] \stackrel{\tau}{\longrightarrow}^{N+1} m'''[\mathsf{sys}(\alpha)] \stackrel{\alpha}{\longrightarrow} m''''[\mathsf{nil}] \tag{6.8}$$

(for some monitor m''''). As from (6.1) we know that $m \stackrel{\kappa_{\text{ID}}}{\Longrightarrow} m'$ where $\mathit{zip}(u, \kappa_{\text{ID}}) = u$, by Lemma 6.1 we know that if the trace system for u violates φ, *i.e.,* $\mathsf{sys}(u) \notin [\![\varphi]\!]$, then $\neg\mathsf{enf}(m, \varphi)$ which clearly contradicts with (6.2), and so we can safely assume that

$$\mathsf{sys}(u) \in [\![\varphi]\!]. \tag{6.9}$$

Since in this case we assume that $\mathsf{sys}(u;\alpha^{N+2}) \notin [\![\varphi]\!]$, and since from (6.1) we know that m suppresses at most $N+1$ occurrences of α that happen after u, we can deduce that if for any arbitrary number $M \leq N+1$ of α's we have that $\mathsf{sys}(u;\alpha^{M}) \in [\![\varphi]\!]$, then the applied suppressions would contradict with the transparency requirement of (6.4). Hence, knowing (6.9) we are forced to conclude that an occurrence of an α action after u *immediately* violates φ, and so every occurrence of α must be suppressed infinitely often in order for m to enforce φ. However, from (6.8) we can also infer that m fails to do so since action α is still permitted to execute by the instrumented system, $m[\mathsf{sys}(u;\alpha^{N+2})]$ since m reduces to m''' (after $N+1$ suppressions of α), which is unable to suppress a subsequent occurrence of α. This implies that $m[\mathsf{sys}(u;\alpha^{N+2})] \notin [\![\varphi]\!]$ which thus contradicts with (6.3).

Since both of the above cases lead to a contradiction, we can thus conclude that assumption (6.2) is false, as required. □

Theorem 6.1 is an important result since it enables us to conclude that if a property can be enforced via a suppression monitor, then this can only be achieved using a persistent suppression transducer.

6.2 Expression-completeness and Maximal expressiveness

Thanks to Theorem 6.1, when answering the question of expression-completeness it thus suffices to focus on *persistent suppression monitors* that are capable of suppressing an action infinitely often until some other action is performed. In fact, one can easily verify that the suppression monitors synthesised by the algorithm of Definition 5.2 (defined in Chapter 5) apply a persistent form of suppression enforcement.

Definition 6.3 (Suppression Expression-Completeness)**.** A subset $\mathcal{L} \subseteq \mu\text{HML}$ is *expressive-complete* with respect to persistent suppression monitors iff for every $m \in \text{PSupTrn}$, there exists some $\varphi \in \mathcal{L}$ such that $\mathsf{enf}(m, \varphi)$. □

We show that the sHML subset is suppression expressive-complete with the aid of function $\langle\!\langle - \rangle\!\rangle$ (Definition 6.4) that maps suppression monitors to a corresponding sHML formula in a very straight forward manner. Same as per the synthesis in Definition 5.2, it assumes a bijective mapping between the denumerable sets of logical variables LVar and the monitor's recursion variables Var.

Definition 6.4 (Suppression Monitors to sHML Formulas)**.**

$$\langle\!\langle \{p, c, \underline{p}\}.m' \rangle\!\rangle \stackrel{\text{def}}{=} [\{p, c\}]\langle\!\langle m' \rangle\!\rangle \qquad \langle\!\langle \{p, c, \bullet\}.m' \rangle\!\rangle \stackrel{\text{def}}{=} [\{p, c\}]\mathsf{ff} \qquad \langle\!\langle X \rangle\!\rangle \stackrel{\text{def}}{=} X$$

$$\langle\!\langle \mathsf{rec}\, X.m' \rangle\!\rangle \stackrel{\text{def}}{=} \mathsf{max}\, X.\langle\!\langle m' \rangle\!\rangle \qquad \langle\!\langle \textstyle\sum_{i\in I} m_i \rangle\!\rangle \stackrel{\text{def}}{=} \bigwedge_{i\in I} \langle\!\langle m_i \rangle\!\rangle \qquad \square$$

Proposition 6.1. sHML is Suppression Expressive-Complete.

The proof for Proposition 6.1 follows from Lemmas 6.2 and 6.3, and the fact that the co-domain of $\langle\!\langle - \rangle\!\rangle$ is that of sHML formulas.

Lemma 6.2. For all $m \in$ PSupTrn, $m[s] \in [\![\langle\!\langle m \rangle\!\rangle]\!]$.

Lemma 6.3. For all $m \in$ PSupTrn, if $m[s] \stackrel{t}{\Longrightarrow} m'[s']$ and $s' \in [\![\mathit{after}(\langle\!\langle m \rangle\!\rangle, t)]\!]$ then $m'[s'] \sim s'$.

Proof for Lemma 6.2. We prove that for every $m \in$ PSupTrn then $m[s] \in [\![\langle\!\langle m \rangle\!\rangle]\!]$. We conduct this proof by coinduction. Since the codomain of $\langle\!\langle - \rangle\!\rangle$ is sHML we assume a relation $\mathcal{R} \stackrel{\text{def}}{=} \{\, (r, \psi) \mid \forall s, r \cdot r \sqsubseteq_{\approx} m[s] \textit{ and } \langle\!\langle m \rangle\!\rangle = \psi \,\}$ and show that it is a satisfaction relation, *i.e.,* it adheres to the satisfaction semantics of Figure 5.1. The proof proceeds by case analysis on m.

Case $m = X$. This case does not apply since $X[s]$ does not constitute a valid system and so $\nexists \varphi \cdot \mathsf{enf}(X, \varphi)$.

Case $m = \sum_{i\in I} m_i$. Assume that

$$r \sqsubseteq_{\approx} \sum_{i\in I} m_i[s] \tag{6.10}$$

and that $\langle\!\langle \sum_{i\in I} m_i \rangle\!\rangle = \bigwedge_{i\in I} \langle\!\langle m_i \rangle\!\rangle$ so that we can deduce that

$$\forall j \in I, \exists \psi \cdot \langle\!\langle m_j \rangle\!\rangle = \psi_j. \tag{6.11}$$

Since we assume that the summed monitors in $\sum_{i\in I} m_i$ are only capable of suppression enforcement, we know that the instrumented system $m_j[s]$ (for any $j \in I$) can simulate $\sum_{i\in I} m_i[s]$ since m_j can suppress (at most) the same behaviours as per $\sum_{i\in I} m_i$, and so from (6.10) we can deduce that

$$\forall j \in I \cdot r \sqsubseteq_{\approx} \sum_{i\in I} m_i[s] \sqsubseteq_{\approx} m_j[s]. \tag{6.12}$$

Therefore, by (6.11), (6.12) and the definition of $\mathcal{R}$ we can conclude that $\forall j \in I \cdot (r, \psi_j) \in \mathcal{R}$ as required.

Case $m = \{p, c, \underline{p}\}.m'$. Assume that

$$\langle\!\langle\, \{p, c, \underline{p}\}.m' \,\rangle\!\rangle = [\{p, c\}]\langle\!\langle\, m_i \,\rangle\!\rangle \tag{6.13}$$

$$r \sqsubseteq_{\approx} \{p, c, \underline{p}\}.m'[s]. \tag{6.14}$$

Now, lets assume that

$$r \stackrel{\alpha}{\Longrightarrow} r' \tag{6.15}$$

$$\mathsf{mtch}(p, \alpha) = \sigma \textit{ and } c\sigma \Downarrow \mathsf{true} \tag{6.16}$$

and so from (6.14), (6.15) and (6.16) we can deduce that

$$\{p, c, \underline{p}\}.m'[s] \stackrel{\tau}{\longrightarrow}{}^* q' \tag{6.17}$$

$$q' \stackrel{\alpha}{\longrightarrow} q \tag{6.18}$$

$$r' \sqsubseteq_{\approx} q. \tag{6.19}$$

Since the reductions in (6.17) could have only been made via rule IASY we have that $s \stackrel{\tau}{\longrightarrow}{}^* s''$ and $q' = \{p, c, \underline{p}\}.m'[s'']$, and so from (6.16) and (6.18) we can infer that $s'' \stackrel{\alpha}{\longrightarrow} s'$ and that $q = m'\sigma[s']$. Hence, from (6.19) we deduce that

$$r' \sqsubseteq_{\approx} m'\sigma[s']. \tag{6.20}$$

Finally, from (6.13), (6.16) and by the definition of $\langle\!\langle\, - \,\rangle\!\rangle$ we have that $\exists \psi \cdot \langle\!\langle\, m'\sigma \,\rangle\!\rangle = \psi$ and so by (6.20) and the definition of $\mathcal{R}$ we can conclude that

$$(r', \psi) \in \mathcal{R}. \tag{6.21}$$

Hence, knowing assumption (6.15) and deduction (6.21) we can introduce an implication so that we know that if $r \stackrel{\alpha}{\Longrightarrow} r'$, $\mathsf{mtch}(p, \alpha) = \sigma$ and $c\sigma \Downarrow \mathsf{true}$ then $(r', \psi) \in \mathcal{R}$ as required.

Case $m = \{p, c, \bullet\}.m'$. Assume that

$$\langle\!\langle\, \{p, c, \bullet\}.m' \,\rangle\!\rangle = [\{p, c\}]\mathsf{ff} \tag{6.22}$$

$$r \sqsubseteq_{\approx} \{p, c, \bullet\}.m'[s]. \tag{6.23}$$

As we only consider persistent suppression monitors we can immediately deduce that $\{p, c, \bullet\}.m'$ suppresses any action α (where $\mathsf{mtch}(p, \alpha) = \sigma$ and $c\sigma \Downarrow \mathsf{true}$) *infinitely often* (until some action β, where $\mathsf{mtch}(p, \beta) = \mathsf{undef}$ or $\exists \sigma \cdot c\sigma \Downarrow \mathsf{false}$, is executed

instead). This means that we can conclude that

$$\{p, c, \bullet\}.m'[s] \stackrel{\alpha}{\not\Longrightarrow} \quad \text{(where } \mathsf{mtch}(p, \alpha) = \sigma \text{ and } c\sigma \Downarrow \mathsf{true}\text{)}. \tag{6.24}$$

Hence, knowing (6.23) we can infer that any behaviour that cannot be performed by the simulating system $\{p, c, \bullet\}.m'[s]$, such as (6.24), cannot be performed by the simulated system r as well, and so we conclude that $r \stackrel{\alpha}{\not\Longrightarrow}$ (for all α when $\mathsf{mtch}(p, \alpha) = \sigma$ and $c\sigma \Downarrow \mathsf{true}$) as required.

Case $m = \mathsf{rec}\, X.m'$. Assume that

$$\langle\!\langle\, \mathsf{rec}\, X.m' \,\rangle\!\rangle = \mathsf{max}\, X.\langle\!\langle\, m' \,\rangle\!\rangle \tag{6.25}$$

$$r \sqsubseteq_{\approx} \mathsf{rec}\, X.m'[s] \tag{6.26}$$

and so since $[\![\mathsf{max}\, X.\langle\!\langle\, m' \,\rangle\!\rangle]\!] = [\![\langle\!\langle\, m' \,\rangle\!\rangle\{^{\mathsf{max}\, X.\langle\!\langle m' \rangle\!\rangle}/_X\}]\!]$, from (6.25) we can deduce that $\langle\!\langle\, \mathsf{rec}\, X.m' \,\rangle\!\rangle = \langle\!\langle\, m' \,\rangle\!\rangle\{^{\mathsf{max}\, X.\langle\!\langle m' \rangle\!\rangle}/_X\}$ so that by the definition of $\mathcal{R}$ we can conclude that $(r, \langle\!\langle\, m' \,\rangle\!\rangle\{^{\mathsf{max}\, X.\langle\!\langle m' \rangle\!\rangle}/_X\}) \in \mathcal{R}$ as required, and so we are done. □

Proof for Lemma 6.3. We proceed to prove that for every $m \in$ PSupTrn, if $m[s] \stackrel{t}{\Longrightarrow} m'[s']$ and $s' \in [\![\mathit{after}(\langle\!\langle\, m \,\rangle\!\rangle, t)]\!]$ then $m'[s'] \sim s'$. To simplify our proof we consider t as an explicit trace t_τ and rely on Lemmas 6.4 and 6.5.

Lemma 6.4. For all $m \in$ PSupTrn, if $s \in [\![\langle\!\langle\, m \,\rangle\!\rangle]\!]$ then $m[s] \sim s$.

Lemma 6.5. For every $m, m' \in$ SupTrn, system s and trace t, if $m \xrightarrow{\alpha \blacktriangleright \bullet} m'$ and $s \in [\![\mathit{after}(\langle\!\langle\, m \,\rangle\!\rangle, t)]\!]$ then $s \in [\![\mathit{after}(\langle\!\langle\, m' \,\rangle\!\rangle, t)]\!]$. □

We give the proof for these lemmas in Appendices B.4.2 and B.4.3 starting on page 195. We now proceed by induction on the length of t_τ.

Case $t_\tau = \varepsilon$. Assume that

$$m[s] \xrightarrow{\varepsilon} m'[s'] \tag{6.27}$$

$$s' \in [\![\mathit{after}(\langle\!\langle\, m \,\rangle\!\rangle, \varepsilon)]\!] = [\![\langle\!\langle\, m \,\rangle\!\rangle]\!]. \tag{6.28}$$

Since ε is an explicit trace, from (6.27) we can deduce that $m' = m$ and $s' = s$. Therefore, from (6.28) and by Lemma 6.4 (transparency) we conclude that $m'[s'] \sim s'$ as required.

Case $t_\tau = \mu t'_\tau$. Assume that

$$m[s] \xrightarrow{\mu} m''[s''] \tag{6.29}$$

$$m''[s''] \xrightarrow{t'_\tau} m'[s'] \tag{6.30}$$

$$s' \in [\![\mathit{after}(\langle\!\langle\, m \,\rangle\!\rangle, \mu t'_\tau)]\!]. \tag{6.31}$$

We now proceed by analysing the instrumentation rules that permit for (6.29) to occur, namely, iDef, iTrn, iSup and iAsy.

- iDef: From (6.29) and by rule iDef we know that $\mu = \alpha$, $m \overset{\alpha}{\not\longrightarrow}$ and that $m'' = \mathsf{id}$. Hence, since $m'' = \mathsf{id}$ and from (6.30) we can infer that $m' = m'' = \mathsf{id}$ and so this case holds trivially since $\mathsf{id}[s'] \sim s'$ as required.
- iTrn: Since m is a suppression monitor we know that it cannot perform action replacements, and so from (6.29) and by rule iTrn we infer that $\mu = \alpha$ and that

$$s \xrightarrow{\alpha} s'' \tag{6.32}$$

$$m \xrightarrow{\alpha \blacktriangleright \alpha} m''. \tag{6.33}$$

From (6.33) we can infer that m must at least define an identity branch that matches action α, that is,

$$m = \mathsf{rec}\, Y_0 \ldots Y_n . \textstyle\sum_{i \in I} m_i + \{p, c, \underline{p}\}.m''' \quad (\text{where } \mathsf{mtch}(p, \alpha) = \sigma \text{ and } c\sigma \Downarrow \mathsf{true}) \tag{6.34}$$

$$m'' = m'''\{{}^{m}/_{Y_0}, \ldots, {}^{\mathsf{rec}\, Y_n . \sum_{i \in I} m_i + \{p,c,\underline{p}\}.m'''}/_{Y_n}\}\sigma \tag{6.35}$$

and therefore from (6.31), (6.34) and via the definition of $\langle\!\langle - \rangle\!\rangle$ we can infer that $s' \in [\![\mathit{after}(\mathsf{max}\, X_0 \ldots X_n . \bigwedge_{i \in I} \langle\!\langle\, m_i \,\rangle\!\rangle \wedge [\{p, c\}] \langle\!\langle\, m''' \,\rangle\!\rangle, \alpha t'_\tau)]\!]$. Since we know (6.35) we apply the definition of *after* and deduce that $s' \in \bigcap_{i \in I} [\![\mathit{after}(\langle\!\langle\, m_i\{\ldots\}\sigma \,\rangle\!\rangle, \alpha t'_\tau)]\!] \cap [\![\mathit{after}(\langle\!\langle\, m'' \,\rangle\!\rangle, t'_\tau)]\!]$ which means that

$$s' \in [\![\mathit{after}(\langle\!\langle\, m'' \,\rangle\!\rangle, t'_\tau)]\!]. \tag{6.36}$$

Hence from (6.30), (6.36) and by the *inductive hypothesis* we can conclude that $m'[s'] \sim s'$ as required.

- iSup: From (6.29) and by rule iSup we know that $\mu = \tau$ and that

$$s \xrightarrow{\alpha} s'' \tag{6.37}$$

$$m \xrightarrow{\alpha \blacktriangleright \bullet} m''. \tag{6.38}$$

Since $\mu = \tau$, from (6.31) and the definition of *after* we have that $s' \in [\![\mathit{after}(\langle\!\langle\, m \,\rangle\!\rangle, t'_\tau)]\!]$ and so knowing (6.38) we can apply Lemma 6.5 and deduce that

$$s' \in [\![\mathit{after}(\langle\!\langle\, m'' \,\rangle\!\rangle, t'_\tau)]\!]. \tag{6.39}$$

Finally, by (6.30), (6.39) and by the *inductive hypothesis* we conclude that $m'[s'] \sim s'$ as required.

- ıAsy: From (6.29) and by rule ıAsy we know that $\mu = \tau$, $s \xrightarrow{\tau} s''$ and that

$$m'' = m. \tag{6.40}$$

  Since from (6.31) and by the definition of *after* we know that $s' \in [\![\mathit{after}(\langle\!\langle\, m \,\rangle\!\rangle, \tau t'_\tau)]\!]$ $= [\![\mathit{after}(\langle\!\langle\, m'' \,\rangle\!\rangle, t'_\tau)]\!]$, knowing (6.30) and (6.40) we can apply the *inductive hypothesis* and conclude that $m'[s'] \sim s'$ as required, and we are done. □

Having established that sHML is suppression expressive-complete, we are now in a position to prove *maximal expressiveness*, namely that sHML is the largest μHML subset that can be enforced by suppression monitors, up to logical equivalence *i.e.,* Theorem 6.2.

Theorem 6.2 (Maximal Expressiveness for sHML)**.** If a language $\mathcal{L} \subseteq \mu$HML is *suppression-enforceable* then $\mathcal{L}$ cannot (semantically) express more properties than sHML, *i.e.,* for every formula $\varphi \in \mathcal{L}$, there exists $\psi \in$ sHML such that $[\![\varphi]\!]$=$[\![\psi]\!]$.

Proof. Assume that every formula $\varphi \in \mathcal{L}$ is *suppression-enforceable* and so we know that there exists a suppression monitor m that adequately enforces φ in the sense of Definition 4.8, *i.e.,* $\mathsf{enf}(m, \varphi)$. Hence, since Definition 4.8 is stronger than Definition 4.4 we can deduce that

$$\exists m \in \textsc{SupTrn}, \forall s \cdot m[s] \in [\![\varphi]\!] \textit{ and } (\text{if } s \in [\![\varphi]\!] \text{ then } m[s] \sim s). \tag{6.41}$$

By Proposition 6.1 for the monitor m used in (6.41) we also know that there exists an sHML formula ψ such that $\mathsf{enf}(m, \varphi)$ (in the sense of Definition 4.8), and so we can infer that

$$\exists \psi \in \text{sHML}, \forall s \cdot m[s] \in [\![\psi]\!] \textit{ and } (\text{if } s \in [\![\psi]\!] \text{ then } m[s] \sim s). \tag{6.42}$$

Now assume an arbitrary $s \in [\![\varphi]\!]$ so that from (6.41) we have that $m[s] \sim s$ and so since from (6.42) we know that $m[s] \in [\![\psi]\!]$, by Theorem 2.1 (*i.e.,* the Hennessy-Milner Theorem) we can infer that $s \in [\![\psi]\!]$ and subsequently that $[\![\varphi]\!] \subseteq [\![\psi]\!]$. Dually, we can also conclude that $[\![\psi]\!] \subseteq [\![\varphi]\!]$. □

Theorem 6.2 represents a significant contribution towards understanding the enforceability of the μHML branching-time logic. Specifically, it ensures that limiting the set of suppression-enforceable formulas to the syntactic subset sHML, still allows for expressing all suppression-enforceable properties. Hence, this result can be used to reduce the problem of checking whether a μHML formula can be enforced via action suppression, to the problem of finding whether there exists a semantically equivalent sHML formula. This could be utilised by verification frameworks to decide whether to enforce a property at runtime or adopt more expressive, yet expensive, techniques.

6.3 Summary

In this chapter we have investigated whether sHML is the maximally expressive subset of μHML that is enforcable by the unidirectional suppression monitors of Chapter 3. As a result, we have produced the following contributions:

(i) The formulation of a criterion denoting what it means for a suppression monitor to apply persistent suppressions, *i.e.,* Definition 6.1.

(ii) A proof for Theorem 6.1 denoting that if a μHML formula can be enforced by a suppression monitor then the monitor must perform persistent suppressions.

(iii) The creation of a mapping function $\langle\!\langle - \rangle\!\rangle$ (Definition 6.4) that maps persistent suppression monitors to sHML formulas. This function played a crucial role in proving that sHML is suppression expressive-complete, Proposition 6.1.

(iv) A proof for Theorem 6.2 affirming that sHML is the maximally expressive subset of μHML that is suppression enforceable.

7. A static counterpart to suppression enforcement

A great deal of effort [5, 43, 58, 59, 65] has been made to study the interplay between static and dynamic techniques, particularly to understand how the two can be used in unison to minimise their respective weaknesses. However, the work conducted in this regard has mainly focussed on verification rather than enforcement. It is therefore unclear whether the maximally expressive fragment sHML (identified in Chapters 5 and 6), can also be enforced statically. In this chapter we thus aim to identify a technique that can be considered as being the *static counterpart* to suppression enforcement, *i.e.,* a technique that can statically achieve the same (or equivalent) results as per suppression enforcement.

Definition 7.1 (Static Counterpart)**.** A static verification technique $\mathcal{S}$ is the *static counterpart* to suppression enforcement when, for every safety formula φ and system s, there exists a transducer m so that $m[s] \in [\![\varphi]\!]$ iff $\mathcal{S}(s) \in [\![\varphi]\!]$ (where $\mathcal{S}(s)$ is a statically reformulated version of s obtained from applying $\mathcal{S}$). □

Identifying such a technique is quite desirable as it would allow for properties to be enforced statically when dynamic verification is not ideal, *e.g.,* when the monitor's runtime overheads are infeasible, and vice versa *e.g.,* to avoid state explosions during static analysis.

One promising static technique that has several things in common with runtime enforcement is *controlled system synthesis* (CSS) [14, 45, 88, 101]. This approach analyses the state space of the SuS and reformulates it pre-deployment to *remove* the system's ability of executing erroneous behaviour. As a result, a restricted (yet valid) version of the SuS is produced; this is known as a *controlled system*. Similar to runtime enforcement, the primary aim of CSS is that the resulting controlled system adheres to the respective property – this is known as *validity* and corresponds

$$\varphi, \psi \in \text{sHML}_{\mathbf{inv}} ::= \mathsf{tt} \mid \mathsf{ff} \mid \varphi \wedge \psi \mid [\alpha]\varphi \mid \Box\varphi$$

Figure 7.1: The syntax for $\text{sHML}_{\mathbf{inv}}$.

to the notion of soundness (Definition 4.2). In addition, CSS also requires further guarantees to ensure minimal disruption to valid systems – this is ensured by *maximal permissiveness* in CSS which is conceptually similar to our transparency constraint (Definition 4.3). Moreover, since CSS aims to correct a SuS by omitting its invalid behaviours, it is ideal for ensuring *safety*. These observations, along with other commonalities, hint at the existence of a relation between suppression enforcement and controlled system synthesis.

In order to investigate the interplay between CSS and suppression enforcement, we choose the recent work on CSS by van Hulst *et al.* [101], and compare it to our work on suppression enforcement presented in Chapters 5 and 6. Specifically, we chose the work of [101] since it has several things in common with our work, including the chosen specification language, modelling of systems, *etc.* To further simplify our comparison, we identify a common core setting between our work and that of [101], and show that in spite of their subtle differences, CSS is in fact a static counter part to suppression enforcement in the context of safety properties. Working in respect to a common core entails:

(i) Working with respect to the *intersection* of the logics used in both works.

(ii) Removing constructs and aspects that are supported by one technique and not by the other, and by taking into account the assumptions considered in both bodies of work.

In the case of (i), we standardise the logics used in both works and work with respect to Safe Hennessy Milner Logic with invariance ($\text{sHML}_{\mathbf{inv}}$), defined in Figure 7.1. $\text{sHML}_{\mathbf{inv}}$ is a strict subset of sHML which results from the intersection of sHML, used in our work on suppression enforcement, and Hennessy Milner Logic with invariance and reachability ($\text{HML}^{\mathbf{reach}}_{\mathbf{inv}}$), used for controlled system synthesis in [101]. Since the $\text{HML}^{\mathbf{reach}}_{\mathbf{inv}}$ variant used in [101] only allows for defining visible actions $\alpha \in \text{Act}$ in its modal operators (instead of symbolic actions), we also include this restriction throughout our comparison.

The primary difference between sHML and $\text{sHML}_{\mathbf{inv}}$ is that the latter restricts recursion to only allow for defining *invariance*. An invariant property $\Box\varphi$ requires every reachable system state to satisfy φ. To avoid having to define new semantics for this operator, we encode it in terms of the sHML syntax as the recursive property $\max X.\varphi \wedge \bigwedge_{\beta \in \text{Act}} [\beta]X$ where Act is now assumed to be *finite*.

We address (ii) in a number of ways. First, we acknowledge that unlike our work,

the work on CSS [101] assumes that a SuS does not perform internal τ actions and that it can have output labels associated to its states. We thus equalise the system models by working with respect to LTSs that do not associate labels to states, and do not perform τ actions. Since we do not focus on state-based properties, the removal of state labels is not a major limitation as we are only forgoing additional state information from the SuS. Although the removal of τ actions requires the SuS to be fully observable, this does not impose significant drawbacks as the work on CSS can easily be extended to allow such actions. We however assume that the resulting monitored and controlled systems may still perform τ actions.

Second, despite that controlled system synthesis differentiates between system actions that can be removed (controllable) and those which cannot (uncontrollable), our work on enforcement does not. This is also not a major limitation since enforcement models can easily be adapted to make such a distinction. However, in our first attempt at a comparison, we opt to simplify the models as much as possible, and so to enable our comparison we assume that every system action is controllable and can be removed and suppressed by the respective techniques. Finally, since controlled systems are not required to satisfy a correctness criterion similar to our notion of eventual transparency (Definition 4.7), in this part we work with respect monitors that adequately enforce formulas in the sense of Definition 4.4 (rather than Definition 4.8).

7.1 Controlled System Synthesis

In Figure 7.2 we present our simplified (more restricted) version of the synthesis function presented in [101]. It takes a LTS $\langle \textsc{Sys}, \textsc{Act}, \rightarrow \rangle$ representing a SuS and a formula φ in order to statically construct a controlled version of the system that satisfies φ. The new system is synthesised in two stages. In the first stage, a new transition relation $\mapsto \subseteq (\textsc{Sys} \times \text{sHML}) \times \textsc{Act} \times (\textsc{Sys} \times \text{sHML})$ is constructed over the state-formula product space, $(\textsc{Sys} \times \text{sHML})$. Intuitively, this transition relation associates a sHML formula to the initial system state and defines how this changes when the system transitions to other subsequent states. The composite behaviour of the formula and the system is statically computed using the first five rules of Figure 7.2.

Rule cBool always adds a transition when the formula is $b \in \{\mathsf{tt}, \mathsf{ff}\}$. Rules cNec1 and cNec2 add a transition from $[\alpha]\varphi$ to φ when s has a transition over α, and to tt if it reduces over $\beta \neq \alpha$. cAnd adds a transition for conjunct formulas, $\varphi \wedge \psi$, when both formulas can reduce independently to some φ' and ψ', with the formula of the end state of the new transition being $\mathit{mini}(\varphi' \wedge \psi')$. Finally, cMax adds a greatest

Static Composition

$$\textsc{cBool}\ \frac{s \xrightarrow{\alpha} s' \qquad b \in \{\mathsf{tt}, \mathsf{ff}\}}{(s, b) \overset{\alpha}{\longmapsto} (s', b)} \qquad \textsc{cNec1}\ \frac{s \xrightarrow{\alpha} s'}{(s, [\alpha]\varphi) \overset{\alpha}{\longmapsto} (s', \varphi)}$$

$$\textsc{cNec2}\ \frac{s \xrightarrow{\beta} s' \qquad \beta \neq \alpha}{(s, [\alpha]\varphi) \overset{\beta}{\longmapsto} (s', \mathsf{tt})} \qquad \textsc{cAnd}\ \frac{(s, \varphi) \overset{\alpha}{\longmapsto} (s', \varphi') \qquad (s, \psi) \overset{\alpha}{\longmapsto} (s', \psi')}{(s, \varphi \wedge \psi) \overset{\alpha}{\longmapsto} (s', \mathit{mini}(\varphi' \wedge \psi'))}$$

$$\textsc{cMax}\ \frac{(s, \varphi\{\max X.\varphi / X\}) \overset{\alpha}{\longmapsto} (s', \psi)}{(s, \max X.\varphi) \overset{\alpha}{\longmapsto} (s', \mathit{mini}(\psi))}$$

Synthesizability Test

$$\frac{\psi \in \{\mathsf{tt}, X, [\alpha]\varphi\}}{(s, \varphi) \downarrow \psi} \qquad \frac{(s, \varphi) \downarrow \psi_1 \qquad (s, \varphi) \downarrow \psi_2}{(s, \varphi) \downarrow (\psi_1 \wedge \psi_2)} \qquad \frac{(s, \varphi) \downarrow \psi}{(s, \varphi) \downarrow \max X.\psi}$$

Invalid Transition Removal

$$\textsc{cTr}\ \frac{(s, \varphi) \overset{\alpha}{\longmapsto} (s', \varphi') \qquad (s', \varphi') \downarrow \varphi'}{(s, \varphi) \xrightarrow{\alpha} (s', \varphi')}$$

Figure 7.2: The Controlled System Synthesis.

fixpoint $\max X.\varphi$ transition to *mini*(ψ), when its unfolding can reduce to ψ. In both cAnd and cMax, *mini*(φ) stands for a *minimal* logically equivalent formula of φ. This is an oversimplification of the minimisation techniques used in [101] to avoid synthesising an infinite LTS due to invariant formulas and conjunctions, see [101] for more details.

Example 7.1. Formulas $\varphi' \wedge \mathsf{tt}$, $\varphi' \wedge \mathsf{ff}$ and $\varphi \wedge \psi \wedge \psi$ are *logically equivalent* to (and can thus be minimized into) φ', ff and $\varphi \wedge \psi$ respectively. □

Instead of defining a rule for greatest fixpoint operators, the authors of [101] define a synthesis rule directly for invariance stating that when $(s, \varphi) \overset{\alpha}{\longmapsto} (s', \varphi')$, then $(s, \Box\varphi) \overset{\alpha}{\longmapsto} (s', \mathit{mini}(\Box\varphi \wedge \varphi'))$. We, however, opted to generalize this rule to fixpoints to simplify our comparison, while still limiting ourselves to sHML$_{\mathbf{inv}}$ formulas. This is possible since by encoding $\Box\varphi$ as $\max X.\varphi \wedge \bigwedge_{\beta \in \mathrm{Act}} [\beta]X$, we get that $(s, \max X.\varphi \wedge \bigwedge_{\beta \in \mathrm{Act}} [\beta]X) \overset{\alpha}{\longmapsto} (s', \mathit{mini}((\max X.\varphi \wedge \bigwedge_{\beta \in \mathrm{Act}} [\beta]X) \wedge \varphi'))$ when $(s, \varphi) \overset{\alpha}{\longmapsto} (s', \varphi')$ where $\mathit{mini}((\max X.\varphi \wedge \bigwedge_{\beta \in \mathrm{Act}} [\beta]X) \wedge \varphi')$ is the encoded version of $\mathit{mini}(\Box\varphi \wedge \varphi')$.

The second stage of the synthesis involves using rule cTr to remove invalid transitions that lead to violating states; this yields the required transition function for the controlled system. This rule relies on the synthesizability test rules to tell whether a controlled state, (s,φ), is valid or not. Intuitively, the test rules fail whenever the current formula φ is semantically equivalent to ff, *e.g.,* formulas $\max X.([\alpha]X \wedge \mathsf{ff})$ and $\varphi \wedge \mathsf{ff}$ both fail the synthesizability test rules as they are equiva-

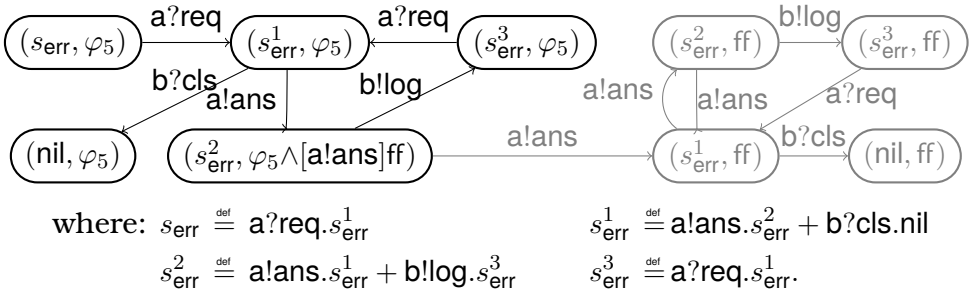

Figure 7.3: The LTS obtained from controlling s_{b} via φ_5.

lent to ff. Concretely, the test is vacuously satisfied by truth, tt, logical variables, X, and guarded formulas, $[\alpha]\varphi$, as none of them are logically equivalent to ff. Conjunct formulas, $\psi_1 \wedge \psi_2$, pass the test when both ψ_1 and ψ_2 pass independently. A fixpoint, $\max X.\varphi'$, is synthesisable if φ' passes the test.

Transitions leading to a state that fails the test are therefore removed, and transitions outgoing from failing states become redundant as they are unreachable. The resulting transition function is then used to construct the controlled LTS $\langle (\textsc{Sys} \times \text{sHML}_{\mathbf{inv}}), \textsc{Act}, \rightarrow \rangle$.

Remark 7.1. Since we do not liberally introduce constructs that are not present in the original models of [101], the simplified model is just a *restricted version* of the original one. Hence, the results proven with respect to this simplified model should either apply to the original one or extend easily to the more general setting.

Example 7.2. Recall the request-response server setting of Example 3.1 from Chapter 3. Now consider the following $\text{sHML}_{\mathbf{inv}}$ formula stating that at every system state, two consecutive answers on port a always indicate invalid behaviour.

$$\varphi_5 \stackrel{\text{def}}{=} \Box\,[\mathsf{a!ans}][\mathsf{a!ans}]\mathsf{ff}$$

Now consider the following erroneous server implementation that similar to s_{b} from Example 3.1 of Chapter 3, it may occasionally produce multiple answers for a given request (see the underlined branch in the description of s_{err} below).

$$s_{\mathsf{err}} \stackrel{\text{def}}{=} \mathsf{a?req.rec}\,X.(\mathsf{a!ans}.(\underline{\mathsf{a!ans}.X} + \mathsf{b!log.a?req}.X) + \mathsf{b?cls.nil})$$

Using the synthesis function of Figure 7.2 we can synthesise a controlled version of s_{err} that satisfies φ_5. The synthesis is conducted in two stages. In the first stage we compose them together and generate the LTS of Figure 7.3. The (grey) a!ans transition leading to the test failing state $(s^1_{\mathsf{err}}, \mathsf{ff})$, is then removed in the second stage along with the unreachable system states and transitions. Hence, this produces the required (black) controlled system. □

7.2 Establishing a static counterpart to enforcement

Establishing that CSS is the static counterpart to suppression enforcement (as stated by Definition 7.1) is inherently difficult as it requires showing that every $\text{sHML}_{\textbf{inv}}$ formula (of which there is an infinite amount) can be enforced using both techniques. It is, however, a well known fact that *trace equivalent* systems satisfy the same set of *safety properties*. As the (recursion-free) subset of sHML characterises safety properties [56], this means that systems sharing the same traces also satisfy the same sHML formulas.

Theorem 7.1. Let s and r be system states in an LTS. Then $\mathit{traces}(s) = \mathit{traces}(r)$ iff s and r satisfy exactly the same sHML formulas. □

A proof for this result is provided in Appendix B.5 on page 203. Although we prove this for the recursion free subset it still applies to the full sHML (see [99, 100]).

Hence, since trace equivalent systems satisfy the same set of safety properties (Theorem 7.1), it suffices to conclude that the controlled LTS can produce the same set of traces as that generated by a monitored one at runtime.

Theorem 7.2 (Trace Equivalence)**.** For every LTS $\langle \text{Sys}, \text{Act}, \rightarrow \rangle$, formula $\varphi \in \text{sHML}_{\textbf{inv}}$ and $s \in \text{Sys}$, there exists a monitor m such that $\mathit{traces}(m[s]) = \mathit{traces}((s, \varphi))$. □

The existential quantification on the monitor m in Theorem 7.2 entails the need of using some sort of mapping that maps $\text{sHML}_{\textbf{inv}}$ formulas to suppression monitors. One possible candidate is to use the synthesis function of Definition 5.2 introduced in Chapter 5. Although our synthesis function is defined for $\text{sHML}_{\textbf{nf}}$ formulas, it is still valid since $\text{sHML}_{\textbf{inv}}$ is a (strict) subset of sHML and so Theorem 5.3 ensures that for every $\text{sHML}_{\textbf{inv}}$ formula we can always find a logically equivalent $\text{sHML}_{\textbf{nf}}$ formula. However, as shown in the following example, the composite system obtained from instrumenting the synthesised monitor with the SuS might not always be trace equivalent to the controlled system obtained using the rules of Figure 7.2.

Example 7.3. Recall formula φ_5 from Example 7.2 which can be normalised as:
$\varphi_5 \stackrel{\text{def}}{=} \max X.([\mathsf{a!ans}]([\mathsf{a!ans}]\mathsf{ff} \wedge [\mathsf{a?req}]X \wedge [\mathsf{b!log}]X \wedge [\mathsf{b?cls}]X)) \wedge [\mathsf{a?req}]X \wedge [\mathsf{b!log}]X \wedge [\mathsf{b?cls}]X.$
Using the synthesis function of Definition 5.2 from Chapter 5 we produce the following monitor:

$$(\!|\, \varphi_5 \,|\!) = \mathsf{rec}\, X.(\{\mathsf{a!ans}\}.m_{\varphi 5}) + \{\mathsf{a?req}\}.X + \{\mathsf{b?cls}\}.X + \{\mathsf{b!log}\}.X$$

$$\text{where } m_{\varphi 5} = (\mathsf{rec}\, Y.\{\mathsf{a!ans}, \bullet\}.Y + \{\mathsf{a?req}\}.X + \{\mathsf{b?cls}\}.X + \{\mathsf{b!log}\}.X).$$

When instrumented with s_{err} from Example 7.2 we obtain the composite system

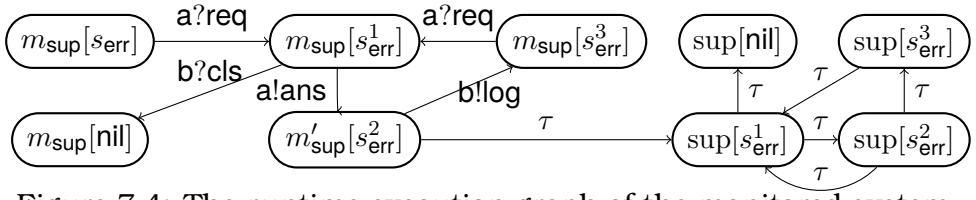

Figure 7.4: The runtime execution graph of the monitored system.

$(\!|\, \varphi_5 \,|\!)[s_{\mathsf{err}}]$ that can execute the trace a?req.a!ans.b?cls, since

$$\begin{aligned}(\!|\, \varphi_5 \,|\!)[s_{\mathsf{err}}] &\xrightarrow{\text{a?req.a!ans}} \mathsf{rec}\, Y.\{\mathsf{a!ans}, \bullet\}.Y + \{\mathsf{a?req}\}.(\!|\, \varphi_5 \,|\!) + \{\mathsf{b?cls}\}.(\!|\, \varphi_5 \,|\!)[s^2_{\mathsf{err}}] \\ &\xrightarrow{\ \tau\ } \mathsf{rec}\, Y.\{\mathsf{a!ans}, \bullet\}.Y + \{\mathsf{a?req}\}.(\!|\, \varphi_5 \,|\!) + \{\mathsf{b?cls}\}.(\!|\, \varphi_5 \,|\!)[s^1_{\mathsf{err}}] \\ &\xrightarrow{\text{b?cls}} (\!|\, \varphi_5 \,|\!)[\mathsf{nil}].\end{aligned}$$

However, this trace cannot be generated by the controlled system $(s_{\mathsf{err}}, \varphi_5)$ of Figure 7.3.

On the other hand, instrumenting s_{err} with monitor m_{sup} (defined below) produces the composite system $m_{\mathsf{sup}}[s_{\mathsf{err}}]$ that is trace equivalent to the controlled system $(s_{\mathsf{err}}, \varphi_5)$. This can easily be verified by comparing the composite behaviour of $m_{\mathsf{sup}}[s_{\mathsf{err}}]$ shown in Figure 7.4, to that of the controlled LTS illustrated in Figure 7.3.

$$\begin{aligned} m_{\mathsf{sup}} &\stackrel{\text{def}}{=} \mathsf{rec}\, X.(\{\mathsf{a!ans}\}.m'_{\mathsf{sup}} + \{\mathsf{a?req}\}.X + \{\mathsf{b?cls}\}.X + \{\mathsf{b!log}\}.X \\ m'_{\mathsf{sup}} &\stackrel{\text{def}}{=} \{\mathsf{a!ans}, \bullet\}.\mathsf{sup} + \{\mathsf{a?req}\}.X + \{\mathsf{b?cls}\}.X + \{\mathsf{b!log}\}.X \end{aligned}$$

where $\mathsf{sup} \stackrel{\text{def}}{=} \mathsf{rec}\, Y.\sum_{\beta \in \text{Act}} \{\beta, \mathsf{true}, \bullet\}.Y$ □

Intuitively, the composite system $m_{\mathsf{sup}}[s_{\mathsf{err}}]$ is trace equivalent to $(s_{\mathsf{err}}, \varphi_5)$ since, unlike $(\!|\, \varphi_5 \,|\!)$, monitor m_{sup} is — in some sense — mimicking dynamically the static modifications applied by the rules of Figure 7.2. Specifically, by suppressing the first redundant (invalid) a!ans action, it mimics the case where the CSS rules remove this invalid action. As a consequence of this removal, the CSS rules make every subsequent state and transition unreachable. This is also simulated at runtime by $m_{\mathbf{t}}$ since it persists in suppressing *every* action that the SuS executes after the invalid a!ans action.

To be able to prove Theorem 7.2, we thus define a new synthesis function that maps $\text{sHML}_{\mathbf{inv}}$ formulas to enforcement transducers. Once again, we reduce the complexity of this mapping by defining it over the normalised sHML formulas as follows.

Definition 7.2. We define our mapping $(\!|\, - \,|\!) : \text{sHML}_{\mathbf{nf}} \mapsto \text{TRN}$ inductively as:

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

$$(\!|\textstyle\bigwedge_{i \in I} [\alpha_i]\varphi_i|\!) \stackrel{\text{def}}{=} \sum_{i \in I} m_i \quad \textit{where } m_i \stackrel{\text{def}}{=} \begin{cases} \{\alpha_i, \alpha_i\}.(\!|\varphi_i|\!) & \text{if } \varphi_i \neq \mathsf{ff} \\ \{\alpha_i, \bullet\}.\mathsf{sup} & \text{otherwise} \end{cases}$$ □

The new synthesis function is almost identical to that of Definition 5.2 from

Chapter 5 as it only differs in the way it handles the normalized conjunctions, $\bigwedge_{i \in I} [\alpha_i]\varphi_i$. They are mapped into a *summation* of monitors, $\sum_{i \in I} m_i$, where every branch m_i can either be prefixed by an identity transformation when $\varphi_i \neq \mathsf{ff}$, or otherwise by the suppression branch $\{\alpha_i, \bullet\}.\mathsf{sup}$ that suppresses the invalid action α_i along with every subsequent action. Notice that the requirement that, $\varphi_i \neq \mathsf{ff}$, is in some sense analogous to the synthesisability test applied by the CSS rule cTr of Figure 7.2 to retain the valid transitions only. In this mapping function, this requirement is essential to ensure that only the valid actions remain unsuppressed by the resulting monitor.

Example 7.4. Recall the normalised version of formula φ_5 from Example 7.3.

$\varphi_5 \stackrel{\text{def}}{=} \max X.([\mathsf{a!ans}]([\mathsf{a!ans}]\mathsf{ff} \wedge [\mathsf{a?req}]X \wedge [\mathsf{b!log}]X \wedge [\mathsf{b?cls}]X)) \wedge [\mathsf{a?req}]X \wedge [\mathsf{b!log}]X \wedge [\mathsf{b?cls}]X.$

Using the mapping function specified in Definition 7.2, we generate monitor

$$(\!|\varphi_5|\!) = \mathsf{rec}\, X.(\{\mathsf{a!ans}\}.m'_{\mathsf{sup}} + \{\mathsf{a?req}\}.X + \{\mathsf{b?cls}\}.X + \{\mathsf{b!log}\}.X$$

$$\text{where } m'_{\mathsf{sup}} \stackrel{\text{def}}{=} \{\mathsf{a!ans}, \bullet\}.\mathsf{sup} + \{\mathsf{a?req}\}.X + \{\mathsf{b?cls}\}.X + \{\mathsf{b!log}\}.X$$

and so $(\!|\varphi_5|\!)$ is identical to m_{sup} from Example 7.3. □

Corollary 7.1 (Enforcement adequacy)**.** The new synthesis function of Definition 7.2 applies adequate enforcement in the sense of Definition 4.4 from Chapter 4 since it produces monitors that adhere to soundness (Definition 4.2) and transparency (Definition 4.3). This claim can be easily justified using Proposition 5.1 and Proposition 5.3 (respectively proven in Chapter 5 on page 41 and Appendix B.3 given on page 160). Although these results are proven for the synthesis function of Definition 5.2 from Chapter 5, the proofs remain identical when recast to the new synthesis function of Definition 7.2.

However, the new function sometimes fails to produce monitors that satisfy the eventual transparency constraint (Definition 4.7). For instance, when given formula φ_0 (from Example 4.1), the new synthesis produces monitor $m_{\mathbf{t}}$ (from Example 3.2) whose failure to adhere to Definition 4.7 was shown in Example 4.6 of Chapter 4. □

With this new mapping function in hand, we are now able to prove Theorem 7.2 as a corollary of Proposition 7.1. The proof for this proposition is given in Appendix B.5.2 on page 204.

Proposition 7.1. For every $\text{sHML}_{\mathbf{nf}}$ formula φ, system s and trace t, when $(\!|\varphi|\!) = m$ then $t \in \mathit{traces}(m[s])$ iff $t \in \mathit{traces}((s, \varphi))$. □

Having concluded the proof of Theorem 7.2 and knowing Theorem 7.1, we can finally obtain our main result with respect to Definition 7.1.

Theorem 7.3. Controlled system synthesis is the *static counterpart* of suppression enforcement in the context of safety properties. □

7.3 Distinguishing between Suppression Enforcement and CSS

Despite concluding that CSS is the static counterpart to suppression enforcement, there are still a number of subtle differences between these two techniques. For one, since suppression enforcement is a dynamic technique, the monitor and the system still remain two separate entities, and the instrumentation between them is merely a way for the monitor to interact with the SuS. In general, the monitor does affect the execution of the SuS itself, but rather modifies its observable trace of actions, such as its inputs and outputs [51]. By contrast, when a controlled system is synthesised, an existing system is paired up with a formula and statically reconstructed into a *new* (correct) system that is incapable of executing the erroneous behaviour.

By removing invalid transitions entirely, controlled system synthesis is more ideal to guarantee the property compliance of the *internal* (less observable) behaviour of a system. For example, this can be useful to ensure that the system does not use a shared resource before locking it. By contrast, the invalid actions are still executed by the system in suppression enforcement, but their effect is rendered invisible to any external observer. This makes suppression enforcement more suitable to ensure that the *external* (observable) behaviour of the system complies with a desired property. For instance, one can ensure that the system does not perform an output that is innocuous to the system itself, but may be providing harmful information to the external environment.

Moreover, it turns out that although both techniques produce composite systems that are trace equivalent to each other, an external observer may still be able to tell them apart by merely observing them. One way of formally assessing this is to use observational equivalence (characterised by delay bisimilarity) as a yardstick, thus:

$$\forall \varphi \in \text{sHML}, s \in \textsc{Sys}, \exists m \in \textsc{Trn} \cdot m[s] \approx (s, \varphi). \tag{7.1}$$

We show by means of a counter example that (7.1) is in fact *false* and as a result prove Theorem 7.4.

Theorem 7.4 (Non Observational Equivalence)**.** There exist an $\text{sHML}_{\textbf{inv}}$ formula φ, an LTS $\langle \textsc{Sys}, \textsc{Act}, \rightarrow \rangle$ and a system state $s \in \textsc{Sys}$ such that for every monitor $m \in \textsc{Trn}$, $m[s] \not\approx (s, \varphi)$. □

Proof sketch. Recall the controlled LTS with initial state $(s_{\mathsf{err}}, \varphi_5)$ obtained in Example 7.2. To prove Theorem 7.4 we must show that *for every action suppression monitor* m, one *cannot* find a delay bisimulation relation $\mathcal{R}$ so that $(m[s_{\mathsf{err}}], (s_{\mathsf{err}}, \varphi_5)) \in \mathcal{R}$. An elegant way of showing this claim, is by using *bisimulation game characterisation* [11] starting from the pair $(m[s_{\mathsf{err}}], (s_{\mathsf{err}}, \varphi_5))$, for every possible m. The game is played

between two players, namely, the attacker and the defender. The attacker wins the game by finding a sequence of moves from the monitored state $m[s_{\mathsf{err}}]$ (or the controlled state $(s_{\mathsf{err}}, \varphi_5)$), which the defender cannot counter, *i.e.,* the move sequence cannot be performed by the controlled state $(s_{\mathsf{err}}, \varphi_5)$ (*resp.* monitored state $m[s_{\mathsf{err}}]$). Note that the attacker is allowed to play a transition from either the current monitored state or the controlled state at each round of the game. A winning strategy for the attacker entails that the composite systems are *not* observationally equivalent.

We start playing the game from the initial pair $(m[s_{\mathsf{err}}], (s_{\mathsf{err}}, \varphi_5))$ for every monitor m. Pick any monitor that suppresses any action other than a second consecutive $\mathsf{a!ans}$, such as $m_0 \stackrel{\text{def}}{=} \{\mathsf{a?req}, \bullet\}.m_0'$. In this case, it is easy to deduce that the defender always loses the game, that is, if the attacker attacks with $(s_{\mathsf{err}}, \varphi_5) \xrightarrow{\mathsf{a?req}} (s^1_{\mathsf{err}}, \varphi_5)$ the defender is defenceless since $m_0[s_{\mathsf{err}}] \stackrel{\mathsf{a?req}}{\Longrightarrow\!\!\!\!\!\!/}$. This remains true regardless of the "depth" at which the suppression of $\mathsf{a?req}$ transitions occur.

On the one hand, using the same game characterisation one can also deduce that by picking a monitor that *fails to suppress* the second consecutive $\mathsf{a!ans}$ action, such as $m_1 \stackrel{\text{def}}{=} \{\mathsf{a?req}\}.\{\mathsf{a!ans}\}.\{\mathsf{a!ans}\}.m_1'$, also prevents the defender from winning. If the attacker plays with $m_1[s_{\mathsf{err}}] \stackrel{\mathsf{a?req.a!ans.a!ans}}{\Longrightarrow} m_1'[s^1_{\mathsf{err}}]$, the defender loses since it can only counter the first two transitions, *i.e.,* $(s_{\mathsf{err}}, \varphi_5) \stackrel{\mathsf{a?req.a!ans}}{\Longrightarrow} \stackrel{\mathsf{a!ans}}{\Longrightarrow\!\!\!\!\!\!/}$. Again, this holds regardless of the "depth" of such a failed suppression.

On the other hand, any monitor that actually suppresses the second consecutive $\mathsf{a!ans}$ action, such as $m_2 \stackrel{\text{def}}{=} \{\mathsf{a?req}\}.\{\mathsf{a!ans}\}.\{\mathsf{a!ans}, \bullet\}.m_2'$, still negates a win for the defender. In this case, the attacker can play $(s_{\mathsf{err}}, \varphi_5) \stackrel{\mathsf{a?req.a!ans}}{\Longrightarrow} (s^2_{\mathsf{err}}, \varphi_5 \wedge [\mathsf{a!ans}]\mathsf{ff})$ to which the defender must reply with $m_2[s_{\mathsf{err}}] \stackrel{\mathsf{a?req.a!ans}}{\Longrightarrow} \{\mathsf{a!ans}, \bullet\}.m_2'[s^2_{\mathsf{err}}]$. The attacker can subsequently play $\{\mathsf{a!ans}, \bullet\}.m_2'[s^2_{\mathsf{err}}] \xrightarrow{\tau} m_2'[s^1_{\mathsf{err}}]$, which can only be countered by an inaction on behalf of the defender, *i.e.,* the controlled system remains in state $(s^2_{\mathsf{err}}, \varphi_5 \wedge [\mathsf{a!ans}]\mathsf{ff})$.

Since we do not know the form of m_2', we consider the following two cases, namely, when m_2' suppresses $\mathsf{b?cls}$, and the case when it does not. In the first case, the attacker can attack with $m_2'[s^1_{\mathsf{err}}] \xrightarrow{\tau} m_2''[\mathsf{nil}]$ (for some m_2'') where τ represents the suppression of the $\mathsf{b?cls}$ action. Once again the defender can only counter with an inaction and stay in state $(s^2_{\mathsf{err}}, \varphi_5 \wedge [\mathsf{a!ans}]\mathsf{ff})$. At this point the attacker wins the play by attacking with $(s^2_{\mathsf{err}}, \varphi_5 \wedge [\mathsf{a!ans}]\mathsf{ff}) \xrightarrow{\mathsf{b!log}} (s^3_{\mathsf{err}}, \varphi_5)$ since $m_2''[\mathsf{nil}] \stackrel{\mathsf{b!log}}{\Longrightarrow\!\!\!\!\!\!/}$. In the second case, the attacker also wins by attacking with $m_2'[s^1_{\mathsf{err}}] \xrightarrow{\mathsf{b?cls}} m_2'''[\mathsf{nil}]$ (for some m_2''') since $(s^2_{\mathsf{err}}, \varphi_5 \wedge [\mathsf{a!ans}]\mathsf{ff}) \stackrel{\mathsf{b?cls}}{\Longrightarrow\!\!\!\!\!\!/}$.

These cases therefore suffice to deduce that for every possible monitor the attacker always manages to win the game, and hence we conclude that Theorem 7.4 holds as required. □

This result is important since it proves that powerful external observers, such as the ones presented by Abramsky in [1], can still distinguish between the resulting monitored and controlled systems.

7.4 Summary

In this chapter we have looked into a static technique called Controlled System Synthesis (CSS), and compared it to suppression enforcement. As a result of this comparison we identified CSS as being the static counterpart to suppression enforcement in the context of safety properties, as defined by Definition 7.1. In spite of this relationship, we also studied the intricate differences between the two techniques and concluded that an Abramsky-type external observer [1] can tell the difference between a monitored and controlled system resulting from the same formula and SuS. As a result of this work, we have produced the following contributions:

(i) A new synthesis function that produces sound and transparent monitors, Definition 7.2. This allowed us to prove that for every controlled system, there exists a *trace equivalent* monitored system, Theorem 7.2.

(ii) A proof confirming that when restricted to safety properties, controlled system synthesis is the *static counterpart* (Definition 7.1) to suppression enforcement, Theorem 7.3.

(iii) Another proof that ensures that the monitored system obtained from instrumenting a synthesised suppression monitor, and the controlled version of the same system, might *not* always be observationally equivalent, Theorem 7.4.

To our knowledge this is the first formal comparison to be made between these two techniques.

8. End of Part I

In the first part of this thesis we have presented a preliminary investigation of the enforceability of μHML properties in a unidirectional context. Specifically, we have focussed on studying the ability to enforce μHML properties via suppression-based enforcement monitors. We concluded that the safety subset sHML is the maximally expressive fragment of μHML that is enforceable via these kind of monitors. To show this, we adopted a trace-based view of the SuS and defined the unidirectional enforcement framework of Chapter 3. In Chapter 4 we then defined enforceability for logics and system descriptions interpreted over labelled transition systems which we motivated vis-a-vis our enforcement framework. Although enforceability builds upon soundness and transparency requirements that have been considered in other work, we developed more stringent definitions for these requirements, and also introduced new constraints, namely, eventual transparency and optimality. We also contend that the definitions that we develop for our 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 our notion of enforcement, in Chapter 5 we thus devised a two-phase procedure to synthesise correct suppression monitors from safety properties expressed via the safety fragment of μHML *i.e.,* sHML. We first identified a syntactic subset of our target sublogic sHML that affords certain structural properties and permits a compositional definition of the synthesis function. We then showed that, by augmenting existing rewriting techniques to our setting, we can convert any sHML formula into this syntactic subset. In Chapter 6 we then showed that the identified enforceable fragment sHML is also the maximally expressive subset of μHML that can be enforced via suppression monitors. To achieve this we devised a mapping that maps sound suppression monitors to sHML formulas which allowed us to prove that sHML is suppression expressive complete, *i.e.,* that every sound

suppression monitor can enforce at least one sHML formula. This in turn allowed us to determine that sHML is the maximally expressive subset of μHML that is suppression enforceable.

In Chapter 7 we finally presented a novel comparison between suppression enforcement and controlled system synthesis – two verification techniques that automate system correction for erroneous systems. As a catalyst for conducting this comparison we chose the work by van Hulst *et al.* [101] on controlled system synthesis, and compared it to our work on suppression enforcement. This investigation allowed us to conclude that controlled system synthesis is the static counterpart to suppression enforcement in the context of safety, which means that safety formulas can be enforced both statically and dynamically. To achieve this, we developed a function that maps logic formulas to suppression monitors and proved inductively that for every system and formula, one can obtain a monitored and a controlled system that execute the same set of traces at runtime. As trace equivalent systems satisfy the same safety properties, this result was enough to reach our conclusion. However, using a counter-example we also determined that these two techniques are different modulo observational equivalence. This means that an external observer can still tell the difference between a monitored and controlled system in spite of being trace equivalent.

As stated earlier, our unidirectional enforcement approach assumes a trace-based view of the SuS and that every action can be freely modified via the monitor's transformations. When we lift this assumption and start differentiating between certain kinds of actions (such as inputs and outputs), the results obtained so far might, however, not always apply. As several works (such as [69, 87, 103]) have used similar unidirectional approaches to enforce properties about the system's output behaviour, we are confident that our results will still apply for these properties after our assumption is lifted. This is further explored (and confirmed) in the second part of this thesis when we explore bi-directional enforcement.

8.1 Related Work

In his seminal work [98], Schneider adopts a trace-based view of the SuS and regards a property to be enforceable if its *violation* can be *detected* and subsequently *prevented* by a *truncation automaton* which terminates the system. By preventing misbehaviour, these automata can only enforce safety properties. Ligatti *et al.* [78] 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 [23, 78]. As a means to assess the correctness of these automata, the authors introduced *soundness* and *transparency* where the latter corresponds to our notion of trace transparency, Definition 4.5, which we showed to be weaker than Definitions 4.3 and 4.7. Moreover, in both of Ligatti's and Schneider's settings, there is no clear separation between the specification and the enforcement mechanism, and properties are encoded in terms of the languages accepted by the enforcement model itself, *i.e.,* as edit/truncation automata. By contrast, we keep the specification and verification aspects of the logic separate.

Bielova and Massacci [23, 25] remark that, on their own, soundness and transparency fail to specify the extent in which a transducer should modify invalid runtime behaviour and thus introduce a *predictability* criterion. A transducer is *predictable* if one can predict the edit-distance between an invalid execution and a valid one. With this criterion, adequate monitors are further restricted by setting an upper bound on the number of transformations that a monitor can apply to correct invalid traces. Although this is similar to our notion of optimality, we however use it to compare an adequate (sound and eventual transparent) monitor to *all* the other adequate monitors and determine whether it is the least intrusive monitor that can enforce the property of interest.

Könighofer *et al.* [69] present a synthesis algorithm that produces unidirectional action replacement transducers called *shields* from safety properties encoded as automata-based specifications. By definition, shields should adhere to two desired properties, namely correctness and minimum deviation which are, in some sense, analogous to soundness and transparency respectively. Although shields analyse both the inputs and outputs of a reactive system, they can only enforce properties by modifying the system's output actions whenever it deviates from the specified behaviour. This is similar to the enforcement approach that we adopt in Example 3.2 of Chapter 3, *i.e.,* although our suppression monitors analyse both the (input) requests and the (output) answers, they only modify the latter.

Falcone *et al.* [47, 50, 52], also propose synthesis procedures to translate properties – expressed as Streett automata – into the respective enforcement automata. The authors show that most of the property classes defined within the *safety-progress hierarchy* [90] are enforceable, as they can be encoded as Streett automata and subsequently converted into enforcement automata. As opposed to Ligatti *et al.*, both Könighofer *et al.* and Falcone *et al.* separate the specification of the property from the enforcement mechanism, but unlike our work they do not study the enforceability of a logic.

Lanotte *et al.* [72, 73] adopt a formal approach to model and enforce security-oriented properties on industrial control systems (ICSs). ICSs are made of a set of distributed programmable logic controllers (PLCs) that perform a specific sequence of actions, called the scan cycle, to control a physical device. The authors represent PLCs as processes expressed in a variant of the TPL calculus [63] and assume that malware may execute alongside them to compromise their behaviour. To counter security attacks, the authors instrument monitors that suppress the injected malicious behaviour and define two functions that synthesise suppression monitors. In their first function, the monitors are synthesised from uncompromised PLCs, while in their second one they are synthesised from a subset of regular properties that are defined using linear-time semantics. The synthesised monitors suppress any behaviour that is not specified by the PLC or regular property it was derived from, as this is assumed to be introduced by a malware. The monitors are also proven to satisfy several criteria including soundness and transparency. Contrary to this setting, in our work we take a more general stance and do not make any assumptions about the system's behaviour, in fact, we consider any system that can be expressed as an LTS. Our primary aim also differs from that of [72, 73] since in our work we strived to understand the enforceability of a highly expressive branching-time logic, and as a result we identified a maximally expressive fragment that is enforceable via suppression transducers.

More similar to our work, Beauquier *et al.* [21] identify a subclass of regular expressions and prove that it is the largest subclass that is enforceable using Ligatti's edit automata [78]. Due to the similarities with our work it would be interesting to investigate whether sHML is strictly contained in their identified class of regular expressions.

Francalanza *et al.* [56] study the monitorability of μHML, but in the sense of runtime verification. They identify the maximally expressive subset of μHML that syntactically characterises the μHML formulas that are verifiable via detection (RV) monitors. By contrast, in our work we identify sHML as being the maximal expressive fragment of μHML in the context of suppression-based enforcement monitoring.

To the best of our knowledge, the only other work that tackles enforceability for the modal μ-calculus [70] (a reformulation of μHML) is that of Martinelli and Matteucci [83, 84]. Their approach is, however, different from ours. Instead of defining a compositional synthesis function that constructs a monitor from a formula, they reduce the synthesis problem to satisfiability. Specifically, their synthesis requires finding a system that satisfies the formula under certain conditions that vary according to whether the formula is enforced using suppressions, insertions or a mixture of both. The existence of a satisfying system can (at worst) be decided in

exponential time according to the length of the formula. By contrast our synthesis produces a monitor in linear time (unless normalisation is required). If a satisfying system exists, it is used as a monitor to transform the actions of the SuS as required. Their work therefore dictates that a μ-calculus formula is enforceable if there exists a system that satisfies the formula under the conditions we alluded to above. Put differently, they do not explicitly identify a maximally expressive logic subset that is enforceable via their enforcement mechanisms. Moreover, they only focus on synthesising sound monitors and do not assess their transparency and optimality.

Bocchi *et al.* [26] 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 $\text{sHML}_{\mathbf{nf}}$), whereas we can synthesise enforcers 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).

Castellani *et al.* [36] 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. Although their work is pitched towards adaptation [30, 55], rather than enforcement, in certain instances they adapt the network by suppressing messages or by replacing messages with messages carrying a default nonce value. As our enforcement mechanism provides the necessary suppression and replacement mechanisms, it is possible to investi-

gate whether our monitor correctness criteria could be adapted or extended to this information-flow setting.

Similar to our work in Chapter 7, several works can be found comparing formal verification techniques. Van Hulst *et al.* [101] explore the relationship between their work on controlled system synthesis and the synthesis problem in Ramadge and Wonham's Supervisory Control Theory (SCT) [92]. The aim in SCT is to generate a *supervisor controller* from the SuS and its specification (*e.g.,* a formal property). If successfully generated, the synchronous product of the SuS and the controller is computed to obtain a supervised system. To enable the investigation, van Hulst *et al.* developed language-based notations akin to that used in [92], and proved that Ramadge and Wonham's work can be expressed using their theory.

Ehlers *et al.* [45] establish a connection between SCT and reactive synthesis – a formal method that attempts to automatically derive a valid reactive system from a given specification. To form this connection, the authors first equalise both fields by using a simplified version of the standard supervisory control problem and focus on a class of reactive synthesis problems that adhere to the requirements imposed by SCT. They then show that the supervisory control synthesis problem can be reduced to a reactive synthesis problem. In this work Ehlers *et al.* focussed on properties expressed as CTL formulas [41] and modelled their systems as descrete event systems.

Basile *et al.* [17] explore the gap between SCT and coordination of services, which describe how control and data exchanges are coordinated in distributed systems. This was achieved via a new notion of controllability that allows one to reduce the classical SCT synthesis algorithms to produce orchestrations and choreographies describing the coordination of services as contract automata. Falcone *et al.* [51] also made a brief, comparison between runtime enforcement and SCT in the context of K-step opacity, but established no formal results that relate these two techniques.

8.2 Future Work

We plan to extend the work of this part in several ways. For one, we aim to extend the enforceable fragment of μHML. Since we have already determined that the suppression enforceable subset sHML is maximally expressive, enlarging the enforceable fragment requires studying more expressive enforcement mechanisms, such as action insertions and replacements.

The co-safety fragment cHML, is generally regarded as being the dual of sHML, namely because, cHML formulas specify the (valid) behaviour that a SuS must adhere to, whereas sHML formulas define the (invalid) behaviour that the SuS must

not exhibit. In fact, several work [4, 5, 28, 55–57] conducted vis-a-vis sHML claims to be easily adapted to apply for cHML (and vice versa). For instance in [56], Francalanza *et al.* elucidate this duality when exploring the monitorability of μHML properties in the sense of runtime verification. Specifically, they determine that sHML formulas can be verified at runtime by monitors that detect their violation, and dually cHML formulas are verifiable by monitors that detect their satisfaction.

We are, however, uncertain if this duality exists in the case of runtime enforcement. Recall that when dealing with safety properties, our suppression monitors adopt a late enforcement approach that allows a system to execute unhindered until it attempts to perform an action α that leads to a violation ff *e.g.,* $[\alpha]\mathsf{ff}$, in which case α gets suppressed. However, this late enforcement approach might fail when applied to cHML properties, specifically since, any system that is unable to execute the exact behaviour specified by the property is by default erroneous. This means that a (replacement or insertion) monitor must constantly steer the execution of the SuS to ensure that every action it executes, conforms to the action sequence specified by the co-safety formula. It is also unclear whether cHML formulas can be adequately enforced in the sense of Definitions 4.4, 4.6 and 4.8 as shown by the following example.

Example 8.1. Consider the cHML formula φ_{cs} stating that each server request must always be followed by a single answer until the server terminates with a close action.

$$\varphi_{\mathsf{cs}} \stackrel{\text{def}}{=} \min X.\langle \mathsf{a?req}\rangle\langle \mathsf{a!ans}\rangle(\langle \mathsf{b?cls}\rangle \mathsf{tt} \vee X)$$

To *soundly* enforce φ_{cs} (as defined by Definition 4.2) it suffices to ensure that the composite system executes the specified behaviour. This can be easily achieved using a monitor that immediately inserts the required behaviour, such as:

$$\{\bullet, \mathsf{a?req}\}.\{\bullet, \mathsf{a!ans}\}.\{\bullet, \mathsf{b?cls}\}.\mathsf{id}.$$

In fact, when instrumented with an invalid system such as $\mathsf{a?req.nil} \notin [\![\varphi_{\mathsf{cs}}]\!]$ yields a valid system that satisfies φ_{cs} and similarly, when composed to a well-behaved system such as $s_{\mathsf{g}} \in [\![\varphi_{\mathsf{cs}}]\!]$ from Example 4.1 of Chapter 3. It, however, fails to adequately enforce φ_{cs} since it breaches transparency (as stated by Definitions 4.3 and 4.5) when it unnecessarily modifies well-behaved systems such as s_{g}. Action replacement also runs into similar issues. □

Extending the enforceable subset to include cHML thus requires a thorough investigation of a completely different enforcement approach. This might either require developing less stringent definitions for enforceability, or else studying more elaborate instrumentation setups to enforce these type of properties. Such setups may include the ones explored in [2], that can reveal refusals in addition to the actions performed by the system.

Second, having established a connection between suppression enforcement and control system synthesis with respect to safety properties, it is worth exploring how runtime enforcement and controlled system synthesis are related with respect to properties other than those representing safety. This would first require expanding our work on enforceability and then investigate whether controlled system synthesis is still able to statically achieve the same results vis-a-vis the wider set of enforceable properties. We also aim to study how runtime enforcement relates to other verification techniques such as supervisory control theory [92], reactive synthesis [45], *etc.* The connection established by van Hulst *et al.* in [101] between control system synthesis and supervisory control theory is a plausible starting point for conducting this future investigation.

Finally, we also plan to study the implementability and feasibility of our unidirectional enforcement 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 [57]). 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. This work can then be used to guide tool construction. To further ease the construction of a tool, we have also developed in advanced an aspect oriented programming framework called eAOP[1] [31, 34]. This framework has already been used successfully by the tools detectEr [15, 28, 35] and adapterEr [29, 30] to instrument runtime verification and adaptation monitors in Erlang programs.

[1]The eAOP framework is open-source and available from: `https://github.com/casian/eaop`.

Part II

Bidirectional Enforcement

9. A bidirectional enforcement model

In the second part of this thesis we start investigating bidirectional enforcement. We thus lift the assumption of Part I and instead of viewing the SuS as a trace of actions that can be freely modified, we adopt a branching time view of the SuS and start differentiating between its input and output actions. Therefore, the set of system actions (Act) now consists of the union of its input (iAct) and output (oAct) actions, *i.e.,* $\text{Act} = \text{iAct} \cup \text{oAct}$.

Recall that unlike outputs, input actions are instigated by the environment and not the SuS itself. The system's control over its inputs thus depends only on whether it provides an input port on which the environment can supply a payload value. As systems have *no control* over the data values supplied in its inputs, we make the following updates to our logic specifications and system representations:

- In our examples, we concisely represent LTSs using the regular fragment of *value-passing CCS* [60] (instead of CCS). Unlike CCS, value-passing CCS does not permit for defining systems that accept specific input values. For instance, the CCS system $\mathsf{a}?2.\mathsf{a}!3.\mathsf{nil}$ cannot be defined in its value-passing variant since it defines the input action $\mathsf{a}?2$ which entails that the input is only made when the payload value is 2. Instead, value-passing CCS requires input values to be symbolic, this is achieved by introducing variables that bind the input data and process it accordingly, *e.g.,* $\mathsf{a}?x.\mathsf{a}!(x{+}1).\mathsf{nil}$ where x binds any value provided by the environment at runtime. For more details about value passing CCS we refer the reader to consult [60].
- When specifying logic formulas, we assume that the condition c of a symbolic action that defines an input pattern $\{(x)?(y), c\}$ may *not* restrict the values of the payload binder y, *i.e.,* $y \notin \mathbf{fv}(c)$. Put differently, for a closed input symbolic action $\{(x)?(y), c\}$, if σ and σ' are substitutions that agree on x, then $c\sigma \Downarrow \mathsf{true}$

iff $c\sigma' \Downarrow \mathsf{true}$. With this restriction, our version of μHML becomes a variant of the *value-passing* μHML introduced in [61, 93].

This lack of control over inputs means that the unidirectional enforcement approach developed in Part I is not powerful enough to transform input actions. This therefore restricts the type of properties that the enforcement framework can enforce. For instance, not every safety property is enforceable, particularly, those that are violated when the system inputs an invalid payload. This happens because it may be too late for the monitor to prevent the violation if it allows the SuS to input a value that then turns out to be invalid. Hence, it is questionable whether the monitor can intercept and suppress (or replace) an invalid input that has already been provided by the environment. The monitor must therefore exploit the system's limited control over its inputs and resort to unconventional methods when enforcing properties that require transforming input actions.

In this chapter we thus develop a formal model for bidirectional enforcement that adopts a different enforcement approach then the ones conventionally used for enforcing behaviour in unidirectional settings. We introduce the proposed bidirectional enforcement approach in Section 9.1 and then formalise the approach as a new enforcement instrumentation model in Section 9.2. Beforehand, however, we will first present the running example that we will be referring to throughout the second part of this dissertation.

Example 9.1. Consider a property stating that for *every* input request that is made on a specific port, the server should not input another request in succession. It may, however, output a *single* answer on the same port in response, and then log the serviced request by outputting a notification on a dedicated port b. Due to the special status of port b, this property does not apply to requests that are input from this port. Using our logic we can formalise this property as the sHML formula φ_1 (recall from Chapter 2 that $(\_)$ represents a *"don't care"* binder).

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

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

This formula defines an invariant property $\mathsf{max}\, X.(..)$ and uses binder (x) to bind the port on which the request is input, and binders (y_1), (y_2) and (y_3) to bind the input and output payloads. The values bound to y_1 and y_2 are later referenced in condition $y_3 = (\mathsf{log}, y_1, y_2)$. The formula is violated by two consecutive inputs on the same port x, and when a request is serviced with multiple answers. An answer output followed by a log action on port b is normal, and thus the formula recurses.

Now consider systems $s_{\mathbf{a}}$, $s_{\mathbf{b}}$ and $s_{\mathbf{c}}$ (where $s_{\mathbf{cls}} \stackrel{\text{def}}{=} (\mathsf{b}?z.\mathsf{if}\ z = \mathsf{cls}\ \mathsf{then}\ \mathsf{nil}\ \mathsf{else}\ X)$).

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

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

$s_{\mathbf{a}}$ implements a request-response server that repeatedly inputs values (for some domain VAL) on port a, $\mathsf{a}?x$, for which it internally computes an answer and assigns it to the data variable y, $y := \mathsf{ans}(x)$. It then outputs the answer on port a in response to each request, $\mathsf{a}!y$, and finally logs the serviced request by outputting the triple (log, x, y) on port b, $\mathsf{b}!(\mathsf{log}, x, y)$. It terminates whenever it inputs a close request cls from port b, *i.e.,* $\mathsf{b}?z$ when $z = \mathsf{cls}$.

Systems $s_{\mathbf{b}}$ and $s_{\mathbf{c}}$ are similar to $s_{\mathbf{a}}$ but define additional behaviour. In fact, $s_{\mathbf{c}}$ is initialised in a suspended state that requires an extra (unused) input, $\mathsf{a}?y$, to start working as $s_{\mathbf{a}}$, whereas $s_{\mathbf{b}}$ may occasionally provide a redundant (underlined) answer prior to logging the serviced request. Using the semantics of Figure 2.2, one can check that $s_{\mathbf{a}} \in [\![\varphi_1]\!]$ whereas $s_{\mathbf{c}} \notin [\![\varphi_1]\!]$ since $s_{\mathbf{c}} \xRightarrow{\mathsf{a}?v_1.\mathsf{a}?v_2}$, and $s_{\mathbf{b}} \notin [\![\varphi_1]\!]$ since $s_{\mathbf{b}} \xRightarrow{\mathsf{a}?v_1.\mathsf{a}!\mathsf{ans}(v_1).\mathsf{a}!\mathsf{ans}(v_1)}$ (for some input values v_1 and v_2). □

9.1 The proposed approach

In bidirectional enforcement we seek to transform the entire (input and output) behaviour of the SuS; this contrasts with unidirectional approaches that only support for transforming its output behaviour. When modifying the entire behaviour of a system (and not just a single trace) it makes sense to distinguish between the transformations performed by the monitor (*i.e.,* insertions, suppressions and replacements), and the way they can be used to affect the resulting behaviour of the composite system. In particular, we say that an action that can be performed by the SuS has been *disabled* when it is no longer visible in the resulting composite system. Similarly, a visible action is *enabled* when the composite system can execute it unlike the SuS. Actions are *adapted* when either the payload of an action in the SuS differs from that of the composite system, or when the action is rerouted through a different port. However, since inputs and outputs are fundamentally different, the type of the action itself cannot be adapted, that is, an input cannot become an output, and vice versa.

Implementing action enabling, disabling and adaptation differs according to whether the action is an input or an output. In Figure 9.1 we illustrate our proposed instrumentation setup that implements them by using the monitor's existing transformations. As the instrumented monitor can only fully control the actions instigated by the SuS, enforcing its outputs is more intuitive. In fact, (a), (b) and (c)

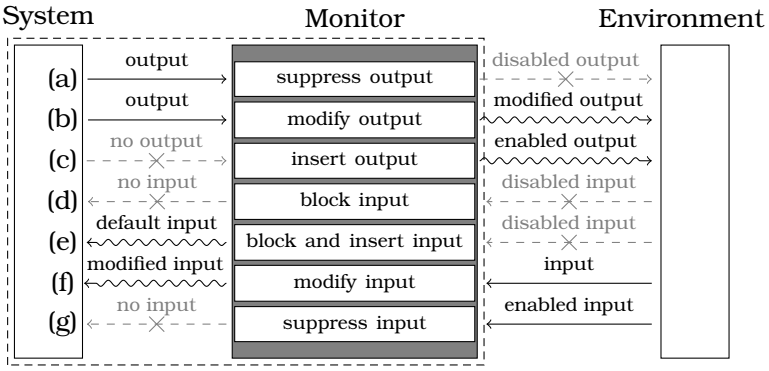

Figure 9.1: Our bi-directional enforcement setup.

in Figure 9.1 respectively show that to disable an output it suffices to suppress it, to adapt it the monitor may replace the output data and forward it to the environment on a (potentially) different port, while to enable an output it suffices to produce the required data via an insertion transformation. In essence, this is the exact *same* approach that we used for the unidirectional enforcement setting of Part I.

Since the SuS enjoys limited control over its inputs, working with these actions is less straightforward. In our setup, we propose that to *disable* an input, item (d) in Figure 9.1, it suffices that the monitor conceals the system's input port so to prevent the environment from forwarding a value as input to the system. As this technique may block the system's execution from progressing, the instrumented monitor may additionally *insert* a default input that unblocks the system[1], item (e) in Figure 9.1. Input *adaptation*, item (f) in Figure 9.1, is also attained via a *replacement* transformation, but unlike in the case of outputs, it is applied in the reverse direction. In fact, it modifies the data received by the monitor over some port, and forwards it to the SuS over the same (or a different) port. Inputs can also be *enabled*, item (g), by allowing the monitor to accept the required input on a desired port and then *suppress* it. To an external viewer, the input has been made, yet discarded internally.

9.2 The model

In Figure 9.2 we now formalise the novel bidirectional instrumentation setup of Figure 9.1 using the *same* (symbolic) transducers $m, n \in \textsc{Trn}$ that we introduced priorly in Chapter 3. Recall that transducers are monitors that define *symbolic transformation triples*, $\{p, c, p'\}$, where pattern p and condition c determine the range of system (input or output) actions upon which the transformation dictated by p' should be applied. Also, recall that the transformation patterns p and p' may also

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

Transducer syntax

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

Transducer dynamics

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

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

Bidirectional enforcement instrumentation

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

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

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

$$\textsc{biDef}\ \frac{s \xrightarrow{\ \mathsf{a}!v\ } s' \qquad m \xrightarrow{\ \mathsf{a}!v\ }\!\!\!\!\!\!\rightarrow \qquad \forall\, \mathsf{b}\in\textsc{Port}, w\in\textsc{Val}\cdot m \xrightarrow{\ \bullet\blacktriangleright\mathsf{b}!w\ }\!\!\!\!\!\!\rightarrow}{m[s] \xrightarrow{\ \mathsf{a}!v\ } \mathsf{id}[s']}$$

where id is shorthand for $\mathsf{rec}\, Y.\{(x)!(y), \mathsf{true}, x!y\}.Y + \{(x)?(y), \mathsf{true}, x?y\}.Y$ and $m \xrightarrow{\ \gamma\ }\!\!\!\!\!\!\rightarrow$ means $\nexists \gamma', n \cdot m \xrightarrow{\ \gamma\blacktriangleright\gamma'\ } n$.

Figure 9.2: A bi-directional instrumentation model for enforcement monitors.

specify $p = \bullet$, to specify the insertion of the action defined by p', and $p' = \bullet$ to represent the suppression of the action specified by p. In addition to the well-formedness constraint stating that for every $\{p, c, p'\}.m$, either p or p' is $\bullet$ (but not both), we now also require that if neither p nor p' is $\bullet$, then both patterns must be of the *same* type *i.e.,* both must be input or output patterns. For instance, symbolic transformations $\{\bullet, \mathsf{true}, \mathsf{a}?v\}$ and $\{(x)!(y), \mathsf{true}, \bullet\}$ are valid since only one of their patterns is $\bullet$, and so is $\{(x)!(y), \mathsf{true}, \mathsf{a}!v\}$ since both patterns are output patterns. This new constraint ensures that input actions cannot be adapted into outputs and vice versa. It is crucial since inputs and outputs are instigated by different entities, namely, the environment and the SuS respectively.

The monitor transition rules in Figure 9.2 are the same as those presented in Figure 3.2 of Chapter 3. Recall that they assume closed terms and that each transducer m yields an LTS that transitions with labels of the form $\gamma\blacktriangleright\gamma'$, *i.e.,* $m \xrightarrow{\ \gamma\blacktriangleright\gamma'\ } n$ where $\gamma, \gamma' \in (\textsc{Act} \cup \{\bullet\})$. This denotes the *transformation* of γ into γ' whilst reduc-

ing into state n, and allows for transducers to replace, suppress and insert actions, *e.g.,* $(\mathsf{a}?3)_{\blacktriangleright}(\mathsf{b}?4)$, $(\mathsf{a}?3)_{\blacktriangleright}\bullet$ and $\bullet_{\blacktriangleright}(\mathsf{a}?3)$. These transitions are made possible by the key transition rule eTrn restated in Figure 3.2. It states that the transformation-prefix transducer $\{p, c, p'\}.m$ transforms action γ into γ' and (where $\gamma'{=}p\sigma$) and reduces to state $m\sigma$, whenever γ matches pattern p, *i.e.,* $\mathsf{mtch}(p, \gamma){=}\sigma$, and satisfies condition c, *i.e.,* $c\sigma \Downarrow \mathsf{true}$. The rules for recursion (eRec) and selection (eSel) are standard.

The primary contribution of this enforcement model lies in the new bidirectional enforcement *instrumentation relation* of Figure 9.2. This relation links the behaviour of the SuS s with the transformations of a monitor m. The term $m[s]$ thus denotes the resulting *monitored system* whose behaviour is defined in terms of $\text{Act} \cup \{\tau\}$. Concretely, rule biTrnO states that if the SuS s transitions with an output $\mathsf{b}!w$ to s' and the transducer m can *replace* $\mathsf{b}!w$ with $\mathsf{a}!v$ and reduce to n, the *adapted* output can be externalised so that the composite system $m[s]$ transitions over $\mathsf{a}!v$ to $n[s']$. Rule biDisO states that if s performs an output $\mathsf{a}!v$ that can be *suppressed* into $\bullet$, the instrumentation withholds this output and so the composite system transitions silently over τ thereby *disabling* it. Dually, rule biEnO *enables* and augments the composite system $m[s]$ with an output $\mathsf{a}!v$ whenever m is able to *insert* $\mathsf{a}!v$ independently of the behaviour of s. Rules biDisO, biTrnO and biEnO therefore correspond to items (a), (b) and (c) in Figure 9.1 respectively.

Rule biDef is analogous to standard rules for premature monitor termination [2, 53, 54, 56], and accounts for underspecification of transformations. We, however, restrict defaulting (termination) exclusively to output actions performed by the SuS. A monitor therefore defaults to id when it cannot react to or enable a system output. By forbidding the monitor from defaulting upon unspecified inputs, the monitor is able to *block* them from becoming part of the composite system's behaviour. Hence, any input that the monitor is unable to react to *i.e.,* $m \xrightarrow{\mathsf{a}?v}\!\!\!\!\!\!/\;\;$, is by default considered as being invalid and blocked. This technique is thus used to implement item (d) of Figure 9.1. To avoid disabling valid inputs unnecessarily, the monitor must explicitly define symbolic transformations that specify *all* the valid inputs of the SuS. For instance, the symbolic transformation $\{\mathsf{a}?(x), \mathsf{true}, \mathsf{a}?x\}$ allows values to be input on port a only, while $\{(y)?(\_), y{\neq}\mathsf{b}, \bullet\}$ allows inputs on any port except b; any other input is invalid and thus blocked. By including such symbolic transformations, rules biTrnI and biEnI can be applied.

With rule biTrnI any value v that is input on some port a by the instrumented system is *adapted* into a (potentially) different value w and forwarded to the SuS over port b, provided the SuS is willing to accept that input. As far as the environment is concerned, the SuS accepted the input provided by the environment on port a. Similarly, rule biEnI *enables* an input on a port a by allowing the composite system

to accept a value v which is then suppressed by the monitor and concealed from the SuS. Although unspecified inputs on a port a are implicitly *disabled* (since the monitor cannot react to them, *i.e.*, $m \xrightarrow{\mathsf{a}?v}\!\!\!\!\!/\;$), rule BIDISI prevents the monitor from blocking systems that require the blocked input in order to progress. Specifically, this rule allows the monitor to generate a default input value v and forward it to the SuS on a port a, thereby unblocking it by allowing the composite system to silently move on to the next state. Hence, rules BIDISI, BITRNI and BIENI respectively implement items (e), (f) and (g) of Figure 9.1. Finally, rule BIASY allows the SuS s to internally transition with a silent action τ to some state s' independent of m.

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

Example 9.2. Consider the following action disabling transducer $m_{\mathbf{d}}$:

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

It is a recursive transducer, $\mathsf{rec}\, Y.\_$, that repeatedly disables every output performed by the system via the branch $\{(x)!(y), \bullet\}.Y$. Moreover, by only defining the input branch $\{\mathsf{b}?(\_)\}.Y$ it also restricts the composite system by allowing it to only input values from port b. Concretely, inputs from other ports are disabled since none of the instrumentation rules in Figure 9.2 can be applied to allow the composite system to transition over these input actions. For instance, when instrumented with $s_{\mathbf{c}}$ from Example 9.1, $m_{\mathbf{d}}$ blocks its initial input so that for every action α, $m_{\mathbf{d}}[s_{\mathbf{c}}] \xrightarrow{\alpha}\!\!\!\!\!/\;$. For $s_{\mathbf{b}}$, the composite system $m_{\mathbf{d}}[s_{\mathbf{b}}]$ can only input termination requests on port b, *i.e.*, $m_{\mathbf{d}}[s_{\mathbf{b}}] \xrightarrow{\mathsf{b?cls}} m_{\mathbf{d}}[\mathsf{nil}]$.

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

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

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

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

blocking further inputs that occur immediately after the first response.

When instrumented with $s_{\mathbf{c}}$ from Example 9.1, $m_{\mathbf{dt}}$ allows the composite system to perform the first input but then blocks the second one which means that it can only input termination requests, *i.e.,* $m_{\mathbf{dt}}[s_{\mathbf{c}}] \xrightarrow{\mathsf{a}?v} \cdot \xrightarrow{\mathsf{b?cls}} \mathsf{id}[\mathsf{nil}]$. It also disables the first redundant response of $s_{\mathbf{b}}$, and as a result, it reduces to $m_{\mathbf{d}}$ which carries on to suppress every subsequent output (even log actions) and blocks every port except b, *i.e.,* $m_{\mathbf{dt}}[s_{\mathbf{b}}] \xrightarrow{\mathsf{a}?v} \cdot \stackrel{\mathsf{a}!w}{\Longrightarrow} \cdot \xrightarrow{\tau} m_{\mathbf{d}}[\mathsf{b}!(\mathsf{log}, v, w).s_{\mathbf{b}}] \xrightarrow{\tau} m_{\mathbf{d}}[s_{\mathbf{b}}] \not\xrightarrow{\mathsf{a}?v}$ (for every port a where a≠b and value v). Moreover, it resorts to defaulting to handle unspecified outputs *e.g.,* for system $\mathsf{b}!(\mathsf{log}, v, w).s_{\mathbf{a}}$ although $m_{\mathbf{dt}} \not\xrightarrow{\mathsf{b}!(\mathsf{log},v,w)}$, using rule iDef the composite system can still perform the output, *i.e.,* $m_{\mathbf{dt}}[\mathsf{b}!(\mathsf{log}, v, w).s_{\mathbf{a}}] \xrightarrow{\mathsf{b}!(\mathsf{log},v,w)} \mathsf{id}[s_{\mathbf{a}}]$.

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

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

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

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

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

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

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

Although in the rest of this dissertation we will mainly focus on action disabling monitors, using our model one can also define action enabling and adaptation monitors.

$$\mathit{zip}_{bi}(t,\kappa) = \begin{cases} \varepsilon & \text{if } t = \varepsilon \text{ and } \kappa = \varepsilon \\ \mathit{zip}_{bi}(t',\kappa') & \text{if } t = (\mathsf{a}!v)t' \text{ and } \kappa = ((\mathsf{a}!v)_{\blacktriangleright}\bullet)\kappa', \text{ or} \\ & \text{if } t = (\mathsf{a}?v)t' \text{ and } \kappa = (\bullet_{\blacktriangleright}(\mathsf{a}?v))\kappa' \\ (\mathsf{a}!v)\mathit{zip}_{bi}(t',\kappa') & \text{if } t = (\mathsf{b}!w)t' \text{ and } \kappa = ((\mathsf{b}!w)_{\blacktriangleright}(\mathsf{a}!v))\kappa' \\ (\mathsf{a}?v)\mathit{zip}_{bi}(t',\kappa') & \text{if } t = (\mathsf{b}?w)t' \text{ and } \kappa = ((\mathsf{a}?v)_{\blacktriangleright}(\mathsf{b}?w))\kappa' \\ (\mathsf{a}!v)\mathit{zip}_{bi}(t,\kappa') & \text{if } \kappa = (\bullet_{\blacktriangleright}(\mathsf{a}!v))\kappa' \\ (\mathsf{a}?v)\mathit{zip}_{bi}(t,\kappa') & \text{if } \kappa = ((\mathsf{a}?v)_{\blacktriangleright}\bullet)\kappa' \end{cases}$$

Figure 9.3: The zip_{bi} function.

Example 9.3. Consider now transducers $m_{\mathbf{e}}$ and $m_{\mathbf{a}}$ below:

$$m_{\mathbf{e}} \stackrel{\text{def}}{=} \{(x)?(y), x{\neq}\mathsf{b}, \bullet\}.\{\bullet, x!\mathsf{ans}(y)\}.\{\bullet, \mathsf{b}!(\mathsf{log}, y, \mathsf{ans}(y))\}.\mathsf{id}$$

$$m_{\mathbf{a}} \stackrel{\text{def}}{=} \mathsf{rec}\, X.\{\mathsf{b}?(y), \mathsf{a}?y\}.X + \{(x)!(y), \mathsf{b}!y\}.X.$$

Once instrumented with a system, $m_{\mathbf{e}}$ first uses a suppression transformation to enable an input that may come from any port $x \neq \mathsf{b}$ (but then gets discarded), *e.g.,* port c. It then automates a response by inserting an answer followed by a log action. Concretely, when composed with $r \in \{s_{\mathbf{b}}, s_{\mathbf{c}}\}$ from Example 9.1, the execution of the composite system can only start as follows:

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

for some values v and $w = \mathsf{ans}(v)$. By contrast, $m_{\mathbf{a}}$ uses action adaptation to redirect the inputs and outputs of the SuS through port b. Specifically, the monitor allows the composite system to input values only from port b and forwards them to the SuS on its input port a. Similarly, outputs from the SuS on port a are rerouted to port b. As a result, the composite system is *only* able to interact on port b. For instance, $m_{\mathbf{a}}[s_{\mathbf{c}}] \xrightarrow{\mathsf{b}?v_1} m_{\mathbf{a}}[s_{\mathbf{a}}] \xRightarrow{\mathsf{b}?v_2.\mathsf{b}!w_2.\mathsf{b}!(\mathsf{log},v_2,w_2)} m_{\mathbf{a}}[s_{\mathbf{a}}]$ and $m_{\mathbf{a}}[s_{\mathbf{b}}] \xRightarrow{\mathsf{b}?v_1.\mathsf{b}!w_1.\mathsf{b}!(\mathsf{log},v_1,w_1)} m_{\mathbf{a}}[s_{\mathbf{b}}]$. □

9.3 Zipping and Unzipping

Once again we introduce the notions of zipping and unzipping, this time for our bidirectional enforcement model. Recall that *unzipping* permits us to decompose the composite system's behaviour, $m[s] \stackrel{u}{\Longrightarrow} m'[s']$ into the monitor's computation $m \stackrel{\kappa}{\Longrightarrow} m'$ and the computation of the SuS, $s \stackrel{t}{\longrightarrow} s'$. The original composite behaviour can then be reconstructed by *zipping* back the computations of the monitor and the SuS. Proving that our new bidirectional enforcement model supports zipping and unzipping requires defining a relation that relates the composite behaviour to the behaviours of the monitor and the SuS. We define this relation as the zip_{bi} function that satisfies the rules of Figure 9.3.

The zip_{bi} function mimics the way the bidirectional instrumentation rules of Figure 9.2 interpret the transformations of the monitor in respect to the actions

performed by the SuS. In this way, when analysing the system trace t and the monitor transformation trace κ, the zip_{bi} function reconstructs a trace representing the composite behaviour that one would obtain if the SuS and the monitor were to execute traces t and κ respectively, while instrumented via the rules of Figure 9.2. Similar to the instrumentation rules of bidirectional enforcement, when defining the zip_{bi} function we differentiate between inputs and outputs and define different cases to handle them accordingly.

More specifically, if the system trace t is prefixed by an output action that gets suppressed in the monitor's transformation trace u, *i.e.*, $t = (\mathsf{a}!v)t'$ and $\kappa = ((\mathsf{a}!v)_{\blacktriangleright}\bullet)\kappa'$, the zip_{bi} function denotes the *disabling* of this output by recursing without adding it to the resulting composite trace. Similarly, if the prefixing action is an input *i.e.*, $t = (\mathsf{a}?v)t'$, the same result is achieved if the transformation trace is prefixed by the insertion of that input, *i.e.*, $\kappa = (\bullet_{\blacktriangleright}(\mathsf{a}?v))\kappa'$. The output action $(\mathsf{b}!w)$ is adapted into $(\mathsf{a}!v)$ and added to the resulting composite trace, if it prefixes the system's trace t, *i.e.*, $t = (\mathsf{b}!w)t'$ and if the monitor's transformation trace κ replaces the former into the latter, *i.e.*, $\kappa = ((\mathsf{b}!w)_{\blacktriangleright}(\mathsf{a}!v))\kappa'$. Likewise, if the system's prefixing action is an input $(\mathsf{b}?w)$, this gets adapted into $(\mathsf{a}?v)$ whenever the latter is replaced by the former, *i.e.*, $\kappa = ((\mathsf{a}?v)_{\blacktriangleright}(\mathsf{b}?w))\kappa'$. This implies that although the composite system has input value v from port a, this has internally been adapted into w and forwarded to the underlying SuS on port b. Finally, the zip_{bi} function adds the output action $(\mathsf{a}!v)$ to the composite trace if the monitor's trace κ inserts it, *i.e.*, $\kappa = (\bullet_{\blacktriangleright}(\mathsf{a}!v))\kappa'$, and it similarly adds $(\mathsf{a}?v)$ when $\kappa = ((\mathsf{a}?v)_{\blacktriangleright}\bullet)\kappa'$. The zip_{bi} function stops recursing when t and κ are both empty.

With the zip_{bi} function we can now show that our bidirectional enforcement model also supports for zipping and unzipping the behaviour of a composite system.

Proposition 9.1 (Unzipping)**.** For the monitored instances $m[s]$, $m'[s']$ and transformation trace κ, if $m[s] \stackrel{u}{\Longrightarrow} m'[s']$ then either

(a) $u = zip_{bi}(t, \kappa)$ and $m \stackrel{\kappa}{\Longrightarrow} m'$ and $s \stackrel{t}{\Longrightarrow} s'$; or

(b) $u = zip_{bi}(t, \kappa)$ and $m \stackrel{\kappa}{\Longrightarrow} m'' \stackrel{(\mathsf{a}!v)}{\not\longrightarrow}$ and $m'' \stackrel{\bullet}{\not\longrightarrow}$ and $s \stackrel{t;(\mathsf{a}!v)t'}{\Longrightarrow} s'$ and $m' = \mathsf{id}$. □

Proposition 9.2 (Zipping)**.** For any monitor m, m', system s, s', traces t, u, and transformation trace κ,

(a) if $m \stackrel{\kappa}{\Longrightarrow} m'$ and $s \stackrel{t}{\Longrightarrow} s'$ and $zip_{bi}(t, \kappa) = u$ then $m[s] \stackrel{u}{\Longrightarrow} m'[s']$.

(b) if $m \stackrel{\kappa}{\Longrightarrow} m' \stackrel{\mathsf{a}!v}{\not\longrightarrow}$ and $m' \stackrel{\bullet}{\not\longrightarrow}$ and $s \stackrel{t;(\mathsf{a}!v)t'}{\Longrightarrow} s'$ and $zip_{bi}(t, \kappa) = u$
then $m[s] \stackrel{u;(\mathsf{a}!v)t'}{\Longrightarrow} \mathsf{id}[s']$. □

Propositions 9.1 and 9.2 are conceptually similar to Propositions 3.1 and 3.2 from Chapter 3 respectively, but recast vis-a-vis our bidirectional setting. Specifically, Proposition 9.1 states that for a monitored execution $m[s] \stackrel{u}{\Longrightarrow} m'[s']$, there must exists a system trace t and a transformation trace κ so that $\mathit{zip}_{bi}(t, \kappa)$ produces either (a) u exactly, or (b) a prefix of u, *i.e.,* $u = \mathit{zip}_{bi}(t, \kappa); (\mathsf{a}!v)t'$ (for some output action $(\mathsf{a}!v)$). In the first case, (a), the composite behaviour can be elegantly decomposed into $m \stackrel{\kappa}{\Longrightarrow} m'$ and $s \stackrel{t}{\Longrightarrow} s'$. In the second case, (b), after executing κ, monitor m reaches a point m'' from which it can neither transform the system's current output action $\mathsf{a}!v$, nor insert an action, *i.e.,* $m \stackrel{\kappa}{\Longrightarrow} m'' \stackrel{(\mathsf{a}!v)}{\not\longrightarrow}$ and $m'' \stackrel{\bullet}{\not\rightarrow}$. This indicates that the monitor has defaulted to id *i.e.,* $m' = \mathsf{id}$, and hence the remaining composite behaviour $(\mathsf{a}!v)t'$ is identical to that of the SuS, *i.e.,* $s \stackrel{t;(\mathsf{a}!v)t'}{\Longrightarrow} s'$. With Proposition 9.2 we then prove that the decomposed behaviours of the monitor and the SuS can be recomposed back together to obtain the composite behaviour. We provide the necessary proofs for these propositions in Appendix C.1 given on page 212.

9.4 Summary

In this chapter we have investigated bidirectional enforcement and formalised a novel instrumentation model for this enforcement setting. Specifically, we have presented the following:

(i) the instrumentation model rules of Figure 9.2 that use the same transducers that we originally introduce in Figure 3.2 of Chapter 3, to achieve bidirectional enforcement, and

(ii) the unzipping and zipping propositions, *i.e.,* Propositions 9.1 and 9.2 respectively, for decomposing and recomposing the behaviour of a monitored system that was composed using the bidirectional enforcement rules of Figure 9.2.

10. Enforceability in a bidirectional context

In this chapter we show that the notion of enforceability and the parametrisable definitions for adequate and optimal enforcement introduced in Chapter 3, are still relevant for the bidirectional enforcement setting of Figure 9.2 from Chapter 9.

10.1 Enforceability

Recall Definition 4.1 from Chapter 4 (restated below as Definition 10.1). This definition states that the *enforceability* of a logic depends on the existence of a relationship between the meaning of a logic formula and the transducer's ability to adequately enforce the behaviour specified by the formula.

Definition 10.1 (Enforceability)**.** A formula φ is *enforceable* iff there *exists* a transducer m such that m *adequately enforces* φ. A logic $\mathcal{L}$ is enforceable iff *every* formula $\varphi \in \mathcal{L}$ is *enforceable*. □

In Chapter 4 we gave several meanings for the requirement that "m *adequately enforces* φ", namely, Definitions 4.4, 4.6 and 4.8. The strongest meaning is defined by Definition 4.8 (restated below as Definition 10.2).

Definition 10.2 (Strong Enforcement)**.** A monitor m *adequately enforces* property φ whenever it adheres to *soundness*, *transparency* and *eventual transparency*. □

It states that a monitor m adequately enforces φ when it is *sound*, *transparent* and *eventual transparent* as defined by Definitions 4.2, 4.3 and 4.7 respectively (all of which are restated below as Definitions 10.3 to 10.5 respectively).

Definition 10.3 (Sound Enforcement)**.** Monitor m *soundly enforces* a satisfiable formula φ, denoted as $\mathsf{senf}(m, \varphi)$, iff $m[s] \in [\![\varphi]\!]$, for every state $s \in \textsc{Sys}$. □

Definition 10.4 (Transparent Enforcement)**.** A monitor m is *transparent* when enforcing a formula φ, written as $\mathsf{tenf}(m,\varphi)$, iff for *all* system states $s \in \textsc{Sys}$, if $s \in [\![\varphi]\!]$ then $m[s] \sim s$. □

Definition 10.5 (Eventual Transparent Enforcement)**.** A monitor m adheres to eventual transparency when enforcing φ, denoted as $\mathsf{evtenf}(m,\varphi)$, iff for all system states s, s', trace t and monitor m', $m[s] \stackrel{t}{\Longrightarrow} m'[s']$ and $s' \in [\![\mathit{after}(\varphi,t)]\!]$ imply that $m'[s'] \sim s'$. □

In what follows, we use the bidirectional enforcement transducers of Example 9.2 from Chapter 9 to motivate why these definitions are also relevant in our bidirectional enforcement setting. We therefore start by showing which of the bidirectional enforcement transducers of Example 9.2 are considered to be *sound*.

Example 10.1. Showing that a monitor soundly enforces a formula requires showing this for *every* possible system. However, we give an intuition based on systems $s_{\mathbf{a}}$, $s_{\mathbf{b}}$, $s_{\mathbf{c}}$ and formula φ_1 (restated below) from Example 9.1.

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

Recall that $s_{\mathbf{a}} \in [\![\varphi_1]\!]$ (hence $[\![\varphi_1]\!] \neq \varnothing$) and $s_{\mathbf{b}}, s_{\mathbf{c}} \notin [\![\varphi_1]\!]$ along with the transducers of Example 9.2 (restated below):

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

where the continuations $m'_{\mathbf{dt}}$, $m'_{\mathbf{det}}$ and $m''_{\mathbf{det}}$ are defined as follows:

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

When assessing the soundness of these transducers in relation to φ_1, we have that:

- $m_{\mathbf{e}}$ is *unsound* as it allows invalid behaviour such as $m_{\mathbf{e}}[s_{\mathbf{b}}] \stackrel{t_{\mathsf{e}}^1}{\Longrightarrow} \mathsf{id}[s_{\mathbf{b}}]$ where $t_{\mathsf{e}}^1 \stackrel{\text{def}}{=} \mathsf{c}?v_1.\mathsf{c}!\mathsf{ans}(v_1).\mathsf{b}!(\mathsf{log}, v_1, \mathsf{ans}(v_1)).\mathsf{a}?v_2.\underline{\mathsf{a}!w_2.\mathsf{a}!w_2}$. This shows that the composite system $m_{\mathbf{e}}[s_{\mathbf{b}}]$ can still make two consecutive output replies (underlined), and so $m_{\mathbf{e}}[s_{\mathbf{b}}] \notin [\![\varphi_1]\!]$. Similarly, $m_{\mathbf{e}}[s_{\mathbf{c}}] \notin [\![\varphi_1]\!]$ since the $m_{\mathbf{e}}[s_{\mathbf{c}}]$ executes the erroneous trace $\mathsf{c}?v_1.\mathsf{c}!\mathsf{ans}(v_1).\mathsf{b}!(\mathsf{log}, v_1, \mathsf{ans}(v_1)).\underline{\mathsf{a}?v_2.\mathsf{a}?v_3}$. This demonstrates that $m_{\mathbf{e}}[s_{\mathbf{c}}]$ can still input two consecutive requests on port a (underlined). Either one of these counter examples suffice to prove that $\neg\mathsf{senf}(m_{\mathbf{e}}, \varphi_1)$.

- $m_{\mathbf{a}}$ is *sound* because once instrumented, the resulting composite system is adapted to only interact on port b, and so its actions are not of concern to φ_1. As $m_{\mathbf{a}}$ applies this enforcement strategy to any SuS, we can safely conclude that $\mathsf{senf}(m_{\mathbf{a}}, \varphi_1)$, in fact $m_{\mathbf{a}}[s_{\mathbf{a}}], m_{\mathbf{a}}[s_{\mathbf{b}}], m_{\mathbf{a}}[s_{\mathbf{c}}] \in [\![\varphi_1]\!]$. Monitors $m_{\mathbf{d}}$, $m_{\mathbf{dt}}$ and $m_{\mathbf{det}}$ are also *sound*. Intuitively, $m_{\mathbf{d}}$ prevents the violation of φ_1 by blocking all input ports except b, whereas $m_{\mathbf{dt}}$ and $m_{\mathbf{det}}$ achieve the same goal by disabling the invalid consecutive requests and answers that occur on any port (except b). □

Although sound enforcement is a fundamental aspect of enforceability, in Chapter 4 we had found it to be relatively weak to solely define adequate enforcement because it does not regulate the *extent* of the applied enforcement. We thus restrict ourselves to the sound transducers identified in Example 10.1 and assess their adherence to Definition 10.4 (transparency) which dictates that, whenever a system s already satisfies the property φ, the assigned monitor m should not alter the behaviour of s.

Example 10.2. Consider $m_{\mathbf{d}}$ from Example 9.2. Although it successfully prevents the violating behaviour of $s_{\mathbf{b}}$ and $s_{\mathbf{c}}$, it needlessly modifies the behaviour of $s_{\mathbf{a}}$ even though $s_{\mathbf{a}}$ satisfies φ_1. By blocking the initial input of $s_{\mathbf{a}}$, $m_{\mathbf{d}}$ causes it to block indefinitely. This counter-example thus suffices to prove that $\neg\mathsf{tenf}(m_{\mathbf{d}}, \varphi_1)$. Monitor $m_{\mathbf{a}}$ from Example 9.3 also fails to meet this requirement: although $s_{\mathbf{a}}$ satisfies φ_1, we have that $m_{\mathbf{a}}[s_{\mathbf{a}}] \not\sim s_{\mathbf{a}}$ since for any value v and w, $m_{\mathbf{a}}[s_{\mathbf{a}}] \xrightarrow{\mathsf{b}?v} \cdot \xrightarrow{\mathsf{b}!w}$ but $s_{\mathbf{a}} \xrightarrow{\mathsf{b}?v} \cdot \not\xrightarrow{\mathsf{b}!w}$. By contrast, monitors $m_{\mathbf{dt}}$ and $m_{\mathbf{det}}$ follow this criterion as they only intervene when it becomes apparent that a violation will occur. For instance, they only disable inputs on a specific port, as a precaution, following an unanswered request on the same port, and they only disable the redundant responses that are produced after the first response to a request. The universal quantification over all systems makes it difficult to show that $\mathsf{tenf}(m_{\mathbf{dt}}, \varphi_1)$ and $\mathsf{tenf}(m_{\mathbf{det}}, \varphi_1)$. However, since both monitors do not modify valid systems such as $s_{\mathbf{a}}$, *i.e.*, $s_{\mathbf{a}} \in [\![\varphi_1]\!]$ and $m_{\mathbf{dt}}[s_{\mathbf{a}}] \sim s_{\mathbf{a}} \sim m_{\mathbf{det}}[s_{\mathbf{a}}]$, and only modify invalid ones, such as $s_{\mathbf{b}}$ and $s_{\mathbf{c}}$ (see Example 10.1), we get a good intuition for why this is the case. □

Now recall that the addition of the transparency constraint still yields an enforcement adequacy definition that is relatively weak. More specifically, it only restricts the enforcement applied to well-behaved systems, and disregards the extent of enforcement induced upon the erroneous ones. Transparency should therefore apply in cases when an invalid SuS eventually reaches a valid point while instrumented with the monitor. Showing adherence to eventual transparency is thus also required to ensure the strong adequate enforcement imposed by Definition 10.2. We therefore assess which of the transparent transducers identified in Example 10.2

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

Figure 10.1: Modification Count (*mc*).

also adhere to Definition 10.5 (eventual transparency).

Example 10.3. Consider monitor $m_{\mathbf{dt}}$ from Example 9.2 and system $s_{\mathbf{b}}$ from Example 9.1. At runtime $s_{\mathbf{b}}$ can exhibit the following invalid behaviour: $s_{\mathbf{b}} \stackrel{t_1}{\Longrightarrow} \mathsf{b}!(\mathsf{log}, v, w).s_{\mathbf{a}}$ where $t_1 \stackrel{\text{def}}{=} \mathsf{a}?v.\mathsf{a}!w.\mathsf{a}!w$. In order to bring the invalid behaviour of $s_{\mathbf{b}}$ (shown in t_1) in line with our specification φ_1, it suffices to use some monitor m that only disables *one* of its responses, $\mathsf{a}!w$. After correcting t_1 into $t'_1 \stackrel{\text{def}}{=} \mathsf{a}?v.\mathsf{a}!w$, no further modifications are required by m since the SuS reaches a valid point, that is, it reduces into a state that does not violate the rest of the property. In this case, the SuS reduces into $\mathsf{b}!(\mathsf{log}, v, w).s_{\mathbf{a}}$ where $\mathsf{b}!(\mathsf{log}, v, w).s_{\mathbf{a}} \in [\![\mathit{after}(\varphi_1, t'_1)]\!]$. However, when instrumented with $m_{\mathbf{dt}}$, this monitor does not only disable the invalid response, *i.e.,* $m_{\mathbf{dt}}[s_{\mathbf{b}}] \stackrel{\mathsf{a}?v.\mathsf{a}!w}{\Longrightarrow} m_{\mathbf{d}}[\mathsf{b}!(\mathsf{log}, v, w).s_{\mathbf{a}}]$, but keeps on disabling every subsequent action as a result of reducing into $m_{\mathbf{d}}$, $m_{\mathbf{d}}[\mathsf{b}!(\mathsf{log}, v, w).s_{\mathbf{a}}] \xrightarrow{\tau} m_{\mathbf{d}}[s_{\mathbf{a}}]$.

Hence, this counter example suffices to deduce that $\neg\mathsf{evtenf}(m_{\mathbf{dt}}, \varphi_1)$; this is not the case for $m_{\mathbf{det}}$ because $\mathsf{evtenf}(m_{\mathbf{det}}, \varphi_1)$. Although the universal quantification over all systems and traces make it hard to prove this property, we can get a good intuition of why this is the case from $s_{\mathbf{b}}$, as when $m_{\mathbf{det}}[s_{\mathbf{b}}] \stackrel{\mathsf{a}?v_1.\mathsf{a}!w_1}{\Longrightarrow} \cdot \xrightarrow{\tau} m''_{\mathbf{det}}[\mathsf{b}!(\mathsf{log}, v_1, w_1).s_{\mathbf{a}}]$ we have that $\mathsf{b}!(\mathsf{log}, v_1, w_1).s_{\mathbf{a}} \in [\![\mathit{after}(\varphi_1, \mathsf{a}?v_1.\mathsf{a}!w_1)]\!]$ and that $m''_{\mathbf{det}}[\mathsf{b}!(\mathsf{log}, v_1, w_1).s_{\mathbf{a}}] \sim \mathsf{b}!(\mathsf{log}, v_1, w_1).s_{\mathbf{a}}$. □

In Examples 10.1 to 10.3 we used the constraints of Definitions 10.3 to 10.5 as guidance to identify $m_{\mathbf{det}}$ as being an adequate transducer that enforces formula φ_1 from Example 9.1 of Chapter 9. Since Definition 10.2 helped us filter out the non-adequate transducers from those that are, confirms that this definition is still relevant in our current bidirectional setting.

10.2 Optimality

Recall that optimal enforcement aims to assess whether an adequate monitor is (to some extent) the "*best*" that one can find. This notion thus requires the use of a *distance measurement* to tell whether one monitor is —in some sense— better than another one. In Chapter 4 we posited that function *mc* (restated in Figure 10.1) is one ideal measurement that assesses the monitor's level of *intrusiveness* when it

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

Figure 10.2: Enforcement Capabilities (ec_{bi}).

enforces a property. In fact this function inductively analyses a system run t_τ, and counts the number of modifications applied by the monitor while instrumented with the SuS— further details about this function are given in Section 4.2 of Chapter 4.

Although the transformations of the instrumented transducer are now applied in a bidirectional context, the following example demonstrates that this function still correctly counts the applied modifications.

Example 10.4. Recall the monitors of Example 9.2 and consider the following system run t^0_τ=$\mathsf{a}?v_1.\mathsf{a}?v_2.\tau.\mathsf{a}!w_2.\mathsf{a}!w_2.\mathsf{b}!(\mathsf{log}, v_2, w_2)$. For $m_{\mathbf{e}}$ and $m_{\mathbf{a}}$, function *mc* respectively counts three enabled actions, *i.e.,* $mc(m_{\mathbf{e}}, t^0_\tau)$=3, and four adapted actions, *i.e.,* $mc(m_{\mathbf{a}}, t^0_\tau)$=4 (since $\mathsf{b}!(\mathsf{log}, v_2, w_2)$ remains unmodified). The maximum count of 5 is attained by $m_{\mathbf{d}}$ as it immediately blocks the first input $\mathsf{a}?v_1$, and so none of the actions in t^0_τ can be executed by the composite system *i.e.,* $\forall \mu \cdot m_{\mathbf{d}}[\mathsf{sys}(t^0_\tau)] \stackrel{\mu}{\not\longrightarrow}$ hence $mc(m_{\mathbf{d}}, t^0_\tau)$=5. Similarly, $mc(m_{\mathbf{dt}}, t^0_\tau)$=4 since $m_{\mathbf{dt}}$ allows the first request to be made, but blocks the second erroneous one, and as a result it also forbids the execution of the subsequent actions, *i.e.,* $\forall \mu \cdot m_{\mathbf{dt}}[\mathsf{sys}(t^0_\tau)] \xrightarrow{\mathsf{a}?v_1} \cdot \stackrel{\mu}{\not\longrightarrow}$. Finally, $m_{\mathbf{det}}$ performs the least number of modifications, namely $mc(m_{\mathbf{det}}, t^0_\tau)$=2. The first modification is caused when the monitor blocks the second erroneous input and internally *inserts* a default input value that allows the composite system to proceed over a τ-action. This contrasts with $m_{\mathbf{d}}$ and $m_{\mathbf{dt}}$ which fail to perform this insertion step thereby contributing to their high intrusiveness score. The second modification is attained when $m_{\mathbf{det}}$ suppresses the redundant response. □

Although the *mc* function allows us to compare monitors in order to identify the least intrusive one, in Chapter 4 we had determined that for this comparison to be fair, we must also compare like with like. It is reasonable to expect that monitors with more enforcement capabilities are likely to be better than those with fewer capabilities. For instance, a monitor that can enforce a property by replacing and disabling actions might be less intrusive than one that can only disable actions.

As in our bidirectional setting we reason in terms of action enabling, disabling and adaptation, the function *ec* of Figure 4.2 (given in Chapter 4) is no longer ideal for determining the enforcement capabilities of bidirectional enforcement monitors. We therefore redefine this function into the variant ec_{bi} presented in Fig-

ure 10.2. Function ec_{bi} inductively analyses the structure of a monitor and deduces whether it can *enable*, *disable* and *adapt* actions based on the type of transformation triples it defines. For instance, if the monitor defines an output suppression triple, $\{(x)!(y), c, \bullet\}.m'$, or an input insertion branch, $\{\bullet, c, \mathsf{a}?v\}.m'$, then ec_{bi} determines that the monitor can disable actions DIS, while if it defines an input suppression, $\{(x)?(y), c, \bullet\}.m'$, or an output insertion branch, $\{\bullet, c, \mathsf{a}!v\}.m'$, then it concludes that the monitor can enable actions, EN. Similarly, if a monitor defines a replacement transformation, it infers that the monitor can adapt actions, ADPT. Once again we use the metavariable χ to denote an arbitrary set of enforcement capabilities.

Example 10.5. Recall the monitors of Example 9.2. With function ec_{bi} we determine that $ec_{bi}(m_{\mathbf{e}})$=$\{\mathsf{EN}\}$, $ec_{bi}(m_{\mathbf{a}})$=$\{\mathsf{ADPT}\}$, $ec_{bi}(m_{\mathbf{d}})$=$ec_{bi}(m_{\mathbf{dt}})$=$ec_{bi}(m_{\mathbf{det}})$=$\{\mathsf{DIS}\}$. Monitors may also have multiple types of enforcement capabilities, for instance, $ec_{bi}(\mathsf{rec}\, X.\{(x)?(y), \bullet\}.X + \{(x)!(y), \bullet\}.X)$=$\{\mathsf{EN}, \mathsf{DIS}\}$. □

With these definitions we now show how Definition 4.9 from Chapter 4 (restated below as Definition 10.6) can guide in identifying an adequate monitor that performs the least number of modifications and is thus the least intrusive.

Definition 10.6 (Optimal Enforcement)**.** A monitor m is χ-*optimal* when enforcing φ, denoted as $\mathsf{oenf}_{\chi}(m, \varphi)$, iff it *adequately enforces* φ and when for every system run $s \xrightarrow{t_\tau}$ and monitor n, if $ec_{bi}(n) \subseteq \chi$ and $\mathsf{enf}(n, \varphi)$ then $mc(m, t_\tau) \leq mc(n, t_\tau)$. □

Once again we refer to a monitor m as being the *most optimal* for enforcing φ when it is found to be the least intrusive after being compared to all types of monitors *i.e.,* it is found to be $\{$DIS,EN,ADPT $\}$-optimal.

Example 10.6. Recall formula φ_1 of Example 9.1 and monitor $m_{\mathbf{det}}$ of Example 10.3. Showing that $m_{\mathbf{det}}$ is the most optimal monitor for enforcing φ_1 is inherently difficult. However, if we limit our comparison to other action disabling monitors only, from Example 10.4 we can get the intuition that $\mathsf{oenf}_{\mathsf{DIS}}(m_{\mathbf{det}}, \varphi_1)$ holds since $m_{\mathbf{det}}$ imposes the least amount of modifications compared to the other action disabling monitors of Examples 9.2 and 9.3. We further reaffirm this intuition using systems $s_{\mathbf{b}}$ and $s_{\mathbf{c}}$ from Example 9.1. In fact, when considering the following invalid system runs $t^1_\tau \stackrel{\text{def}}{=} \mathsf{a}?v_1.\tau.\mathsf{a}!w_1.\mathsf{a}!w_1.\mathsf{b}!(\mathsf{log}, v_1, w_1)$ of $s_{\mathbf{b}}$, and $t^2_\tau \stackrel{\text{def}}{=} \mathsf{a}?v_1.\mathsf{a}?v_2.\tau.\mathsf{a}!w_2.\mathsf{b}!(\mathsf{log}, v_2, w_2)$ of $s_{\mathbf{c}}$, one can easily deduce that no other adequate action disabling monitor can enforce φ_1 with fewer modifications than those imposed by $m_{\mathbf{det}}$ *i.e.,* $mc(m_{\mathbf{det}}, t^1_\tau)$=$mc(m_{\mathbf{det}}, t^2_\tau)$=1. Furthermore, consider the invalid traces $t^1_\tau\{^{\mathsf{c}}/_{\mathsf{a}}\}$ and $t^2_\tau\{^{\mathsf{c}}/_{\mathsf{a}}\}$ that are respectively produced by versions of $s_{\mathbf{b}}$ and $s_{\mathbf{c}}$ that interact on some port c instead of a (for any port $\mathsf{c} \neq \mathsf{a}$). Since $m_{\mathbf{det}}$ binds the port c to its data binder x and uses this information in its insertion branch, $\{\bullet, x?v_{\text{def}}\}.Y$, the *same* modification count is achieved for these traces, as well *i.e.,* $mc(m_{\mathbf{det}}, t^1_\tau\{^{\mathsf{c}}/_{\mathsf{a}}\}) = mc(m_{\mathbf{det}}, t^2_\tau\{^{\mathsf{c}}/_{\mathsf{a}}\}) = 1$. □

Example 10.6 describes the case where formula φ is DIS-optimally enforced by a *finite-state* and *finitely-branching* monitor *i.e.,* $m_{\mathbf{det}}$. However, this is not always possible in the general case.

Example 10.7. Consider formula φ_2 stating that an initial input on port a followed by another input from some other port $x_2{\neq}\mathsf{a}$ constitutes invalid system behaviour. Also consider monitor m_1 where $\mathsf{enf}(m_1, \varphi_2)$.

$$\varphi_2 \stackrel{\text{def}}{=} [\{\mathsf{a}?(\_)\}][\{(x_2)?(\_), x_2{\neq}\mathsf{a}\}]\mathsf{ff}$$

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

When enforcing a system that generates the run $t^3_\tau \stackrel{\text{def}}{=} \mathsf{a}?v_1.\mathsf{b}?v_2.\mathsf{a}!w_1.u^3_\tau$, monitor m_1 modifies the trace only *once*. Although it disables the input $\mathsf{b}?v_2$, it subsequently unblocks the SuS by inserting $\mathsf{b}?v_{\text{def}}$ and so trace t^3_τ is transformed into $\mathsf{a}?v_1.\tau.\mathsf{a}!w_1.u^3_\tau$. However, for a slightly modified version of t^3_τ, *e.g.,* $t^3_\tau\{^{\mathsf{c}}/_{\mathsf{b}}\}$, m_1 scores a modification count of $2 + |\, u^3_\tau\,|$, as despite blocking the invalid input on port c, it fails to insert the default value that unblocks the SuS. A more expressive version of m_1, such as $m_2 \stackrel{\text{def}}{=} \{\mathsf{a}?(\_)\}.\mathsf{rec}\, Y.(\{\bullet, \mathsf{b}?v_{\text{def}}\}.Y + \underline{\{\bullet, \mathsf{c}?v_{\text{def}}\}.Y} + \{\mathsf{a}?(\_)\}.\mathsf{id})$, circumvents this problem by defining an extra insertion branch (underlined), but still fails to be DIS-optimal in the case of $t^3_\tau\{^{\mathsf{d}}/_{\mathsf{b}}\}$. In this case, there does not exist a way to finitely define a monitor that can insert a default value on every possible input port $x_2 \neq \mathsf{a}$. Hence, it means that the optimal monitor m_{opt} for φ_1 would be an *infinite branching* one, *i.e.,* it requires a countably infinite summation that is not expressible in TRN, such as $\{\mathsf{a}?(\_)\}.(\mathsf{rec}\, Y.\sum_{\mathsf{b} \in \text{PORT } \textit{and}\ \mathsf{a}\neq\mathsf{b}} \{\bullet, \mathsf{b}?v_{\text{def}}\}.Y + \{\mathsf{a}?(\_)\}.\mathsf{id})$. Alternatively, the same monitor can also be expressed $\{\mathsf{a}?(\_)\}.(\mathsf{rec}\, Y.\sum_{\mathsf{b} \in \text{PORT}} \{\bullet, \mathsf{a}{\neq}\mathsf{b}, \mathsf{b}?v_{\text{def}}\}.Y + \{\mathsf{a}?(\_)\}.\mathsf{id})$ where the condition $\mathsf{a}{\neq}\mathsf{b}$ is evaluated at runtime instead. □

Unlike Example 10.6, Example 10.7 presents a case where a level of optimality (specifically DIS-optimality) can only be attained by a monitor that defines an *infinite number of branches*; this is problematic since monitors are required to be *finitely* described. As it is not always possible to find a finite monitor that enforces a formula using the least amount of transformation for every possible system, this indicates that Definition 10.6 is too strict. We thus mitigate this issue by weakening Definition 10.6 and redefine it in terms of the set of system states SYS_Π, *i.e.,* the set of states that can only perform inputs using the ports specified in a finite $\Pi \subset \text{PORT}$. Although this weaker version does *not* guarantee that the monitor m χ-optimally enforces φ on *all* possible systems, it ensures optimal enforcement for all the systems that input values via the ports specified in Π.

Definition 10.7 (Weak Optimal Enforcement)**.** A monitor m is *weakly χ-optimal* when enforcing φ, denoted as $\mathsf{oenf}_\chi(m, \varphi, \Pi)$, iff it *adequately enforces* φ and when

for every state $s \in \underline{\mathrm{Sys}_{\Pi}}$, explicit trace t_τ and monitor n, if $ec_{bi}(n) \subseteq \chi$, $\mathsf{enf}(n, \varphi)$ and $s \xrightarrow{t_\tau}$ then $mc(m, t_\tau) \leq mc(n, t_\tau)$. □

Example 10.8. Monitor m_1 from Example 10.7 ensures that φ_2 is DIS-optimally enforced on systems that interact on ports a and b, *i.e.,* when $\Pi = \{\mathsf{a}, \mathsf{b}\}$, while monitor m_2 guarantees it when $\Pi = \{\mathsf{a}, \mathsf{b}, \mathsf{c}\}$. □

10.3 Summary

In this chapter we have shown via a number of examples that the definitions of enforceability, adequate enforcement and optimality that we had introduced in Chapter 4 and motivated vis-a-vis the unidirectional setting of Chapter 3, are still relevant when applied to the bidirectional setting of Chapter 9. More specifically, we have shown that:

(i) The constraints imposed by Definition 10.2 are still sufficient to assess the adequacy of enforcement monitors even when these are instrumented using the bidirectional enforcement rules of Chapter 9.

(ii) We showed in certain cases, Definition 4.9 (restated as Definition 10.6) can be too strict when assessing the optimality of transducers in a bidirectional enforcement setting. However, we also proposed Definition 10.7 as a weaker version that guarantees optimal enforcement under certain assumptions about the SuS. In this way, instead of finding a monitor that is the least intrusive possible (which may be unattainable), Definition 10.7 guides in finding the a monitor that is the least intrusive under the stated assumptions.

11. Synthesising action disabling monitors

The investigation carried out in Part I allowed us to look into the enforceability of our logic with respect to the unidirectional setting of Chapter 3. As a result, we determined that the omission of actions (via suppressions) from the resulting behaviour of a composite system can be used to adequately and optimally enforce safety properties. Intuitively, safety is ensured when actions are omitted as soon as it becomes apparent that a violation is about to be committed by the SuS. Even though the work relating action omission to safety explored in Chapter 5 was set in a unidirectional enforcement context, we are confident that similar results can be attained in the case of a bi-directional one.

In this chapter we thus focus on studying the enforceability of safety properties in a bidirectional setting. We explore how action disabling monitors can adequately and optimally enforce safety properties expressed as sHML formulas. Recall, however, that the universal quantifications in Definitions 10.2 and 10.7 (*i.e.*, strong enforcement and weak optimality from Chapter 4) make it difficult to ensure that bidirectional transducers can adequately and optimally enforce sHML properties. Particularly, establishing that a formula is enforceable, in the sense of Definition 10.2, requires finding a monitor that satisfies the constraints imposed by soundness (Definition 10.3), transparency (Definition 10.4) and eventual transparency (Definition 10.5). This monitor must then be scrutinised vis-a-vis Definition 10.7 to determine whether it is (weak) optimal under certain assumptions about the system's ports. Establishing the enforceability of a logic, Definition 10.1, is even harder, as it entails yet another universal quantification on all the formulas in the logic.

To address these problems we follow a similar approach to that of Chapter 5 and develop an *automated synthesis procedure* that produces action disabling monitors.

We then prove that for *every* sHML formula φ, our synthesis can produce a monitor m that adequately enforces φ as stated by Definition 10.2, and is (weak) optimal in the sense of Definition 10.7. As learnt from our prior work in Chapter 3 of Part I, it is imperative for the synthesis function to be *compositional*. This is desirable, as it simplifies our analysis of the produced monitors, and allows us to use standard inductive proof techniques to prove properties about the synthesis function, such as proving adherence to Definitions 10.2 and 10.7. However, recall from Chapter 5 that a naive approach to such a scheme is bound to fail, and that *normalisation* provides an effective way for preserving the compositionality and simplicity of our synthesis while circumventing the problems of a naive approach. We thus work with respect to the normalised syntactic subset of sHML known as $\text{sHML}_{\mathbf{nf}}$ (restated in Definition 11.1 below) which we showed to be as expressive as sHML and even presented an algorithmic translation procedure in Section 5.2 of Chapter 5.

Definition 11.1 (sHML normal form)**.** The set of $\text{sHML}_{\mathbf{nf}}$ formulas is defined by the following grammar:

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

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

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

In a (closed) $\text{sHML}_{\mathbf{nf}}$ formula, the basic terms tt and ff can never appear unguarded unless they are at the top level. Modal operators are combined with conjunctions into one construct $\bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i$ where the conjunct modal guards must be *disjoint* so that *at most one* necessity guard can satisfy any particular visible action. Along with these restrictions, $\text{sHML}_{\mathbf{nf}}$ also inherits the assumptions that fixpoint variables are guarded, and that for every input symbolic action $\{(x)?(y), c\}$, the condition c does not restrict the input payload data bound to y, *i.e.*, $y \notin \mathbf{fv}(c)$.

Example 11.1. Consider the following formula φ_3. It defines a recursive property that is violated when a system outputs a value of 4 on any port, after inputting any value from port a. It also recurses if the output is made on port a with a value that is not equal to 3.

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

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

presented in Section 5.2 of Chapter 5, we can reformulate φ_3 into $\varphi_3' \in \text{sHML}_{\mathbf{nf}}$:

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

where x_4 and y_4 are fresh variables. □

Moreover, recall from Example 10.7 that certain formulas, such as φ_2 (restated below in Example 11.2), one cannot find an image finite monitor (*i.e.,* a monitor with a finite number of insertion branches) that enforces them optimally in the sense of Definition 10.6. However, in Example 10.8 we determined that one can still find a monitor that ensures (weak) optimal enforcement under certain assumptions about the ports used by the SuS, *i.e.,* as defined by Definition 10.7. Therefore, for our synthesis to produce weak optimal enforcement monitors it must also be aware of these assumptions.

Example 11.2. Recall formula φ_2 and monitors m_1 and m_2 from Example 10.7 of Chapter 10 (restated below).

$$\varphi_2 \stackrel{\text{def}}{=} [\{\mathsf{a}?(\_)\}][\{(x_2)?(\_), x_2{\neq}\mathsf{a}\}]\mathsf{ff}$$

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

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

Synthesising m_1 from φ_2 the SuS can be unblocked when a value is inserted on port b. This information can be supplied to the synthesis via the set of ports $\Pi = \{\mathsf{b}\}$ which in turn uses this information to add the insertion branch $\{\bullet, \mathsf{b}?v_{\text{def}}\}.Y$ in m_1. If $\Pi = \{\mathsf{b}, \mathsf{c}\}$ is supplied instead, monitor m_2 is synthesised. □

11.1 The synthesis function

We now proceed to define the synthesis function in Definition 11.2. It defines a *compositional* mapping that converts an $\text{sHML}_{\mathbf{nf}}$ formula φ into a transducer m. As motivated by Example 11.2, this new synthesis also requires information regarding the input ports of the SuS, as this is used to add the necessary insertion branches that silently unblock the SuS at runtime. This information must be supplied in the form of a *finite* set of input ports $\Pi \subset \text{Port}$. The synthesis then relays this information to the resulting monitor.

Definition 11.2. The synthesis function $(\!|-|\!) : \text{sHML}_{\mathbf{nf}} \times \mathcal{P}_{\text{fin}}(\text{Port}) \mapsto \text{Trn}$ is defined inductively as follows:

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

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

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

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

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

The above definition shares some similarities to Definition 5.2 of Chapter 5 that synthesises unidirectional suppression monitors. For one, it assumes a bijective mapping between formula variables and monitor recursion variables which it uses to convert logical variables X and maximal fixpoints, $\mathsf{max}\, X.\varphi$, accordingly. It also converts truth, tt, and falsehood, ff, formulas into the identity monitor id. The main difference lies in how it handles normalized conjunctions, $\bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i$. They are synthesised into a *recursive summation* of monitors, *i.e.,* $\mathsf{rec}\, Y. \sum_{i \in I} m_i$, where Y is fresh, and every branch m_i can be one of the following:

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

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

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

This blocking mechanism can, however, block *unspecified* inputs, *i.e.,* those that do not satisfy any modal necessity in the normalised conjunction. This is undesirable since unspecified actions do not contribute towards a safety violation and, on the contrary, lead to its trivial satisfaction. To prevent this, the *default monitor* $\mathsf{def}(\varphi)$ is also added to the resulting summation. The def function produces a 'catch-all' identity monitor that forwards an input to the SuS whenever it satisfies the nega-

tion of *all* the conditions associated to modal necessities defining an input pattern in the normalized conjunction. Put differently, the default monitor allows inputs to reach the system whenever they satisfy the condition $\bigwedge_{i\in I} \neg c_i$. This condition is constructed when the normalised conjunction is $\bigwedge_{i\in I} [\{(x_i)?(y_i), c_i\}]\varphi_i \wedge \psi$ (assuming that ψ does not include further input modalities). Otherwise, if none of the conjunct modalities define an input pattern, every input is allowed, *i.e.,* the default monitor becomes $\{(\_)?(\_)\}.\mathsf{id}$. Upon matching a system action, the default monitor transitions to id after forwarding the input to the SuS.

Example 11.3. Consider the following unshortened version of formula φ_1 from Example 9.1 of Chapter 9.

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

For any set of ports Π, the synthesis function of Definition 11.2 produces the following monitor.

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

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

Notice that monitor m_{φ_1} (synthesised via $(\!|\varphi_1, \Pi|\!)$ in Example 11.3) has essentially the same structure as $m_{\mathbf{det}}$ of Example 9.2 in Chapter 9. It mainly varies in how it defines its insertion branches for unblocking the SuS. For instance if $\Pi = \{\mathsf{b}, \mathsf{c}\}$, $(\!|\varphi_1, \Pi|\!)$ would synthesise two insertion branches, namely, $\{\bullet, \mathsf{b} = x, \mathsf{b}?v_{\text{def}}\}$ and $\{\bullet, \mathsf{c} = x, \mathsf{c}?v_{\text{def}}\}$, whereas if Π also includes d, it would add another branch. By contrast, the manually defined $m_{\mathbf{det}}$ attains the same result more elegantly via the single insertion branch $\{\bullet, x?v_{\text{def}}\}$. As argued in Example 10.7 of Chapter 10, it is not always possible to define a monitor like $m_{\mathbf{det}}$, especially when the formula defines complex conditions in its violating modal necessities, such as in the case of φ_2. Despite this, the following results, namely, Theorems 11.1 and 11.2, show that our synthesis still guarantees enforceability and weak optimality.

Theorem 11.1 (Enforceability)**.** The logic sHML is enforceable in a bi-directional setting. □

Proof. Since sHML is logically equivalent to $\text{sHML}_{\mathbf{nf}}$, by Definition 10.1 the result follows from showing that for every $\varphi \in \text{sHML}_{\mathbf{nf}}$ and $\Pi \subseteq \textsc{Port}$, $(\!|\varphi, \Pi|\!)$ adequately enforces φ (for every Π). By Definition 10.2, this claim comes as a result of the below

stated Propositions 11.1 and 11.2 which entail that the synthesised monitors *enforce* their respective sHML$_{\mathbf{nf}}$ formula and are correct by construction. Recall from Corollary 4.1 (of Chapter 4) that Definition 10.4 (transparency) is a special instance of Definition 10.5 (eventual transparency). Hence, adherence to the latter implicitly entails adherence to the former. □

Proposition 11.1 (Soundness)**.** For every input port set Π, system state $s \in \text{Sys}$ and $\varphi \in \text{sHML}_{\mathbf{nf}}$, if $[\![\varphi]\!] \neq \varnothing$ then $(\!|\varphi, \Pi|\!)[s] \in [\![\varphi]\!]$. □

Proposition 11.2 (Eventual Transparency)**.** For every input port set Π, sHML formula φ, system states $s, s' \in \text{Sys}$, action disabling monitor m' and trace t, if $(\!|\varphi, \Pi|\!)[s] \stackrel{t}{\Longrightarrow} m'[s']$ and $s' \in [\![\mathit{after}(\varphi, t)]\!]$ then $m'[s'] \sim s'$. □

The proofs of these propositions are given in Appendices C.2.1 and C.2.2 on pages 216 and 222 respectively.

We now proceed to show that the synthesised monitor $(\!|\varphi, \Pi|\!)$ also ensures a degree of optimality when enforcing formula φ. Recall from Example 10.7 of Chapter 10 that optimal enforcement as stated by Definition 10.6 can at times be too strict for our bidirectional setting. For this reason, we aim to prove that our synthesis function $(\!|\varphi, \Pi|\!)$ at least produces monitors that are weak optimal (as stated by Definition 10.7) when enforcing φ on a set of system states s whose input ports are specified by Π, *i.e.,* $s \in \text{Sys}_{\Pi}$.

Moreover, recall that in order to determine that a synthesised monitor is the most optimal one, we must compare it to *all* adequate monitors *i.e.,* including those that disable, enable and adapt actions. As so far we have focussed on exploring the enforceability of action disabling monitors, we opt to restrict our comparison to other action disabling transducers. We thus aim to prove that the synthesised monitors are weakly DIS-optimal.

Theorem 11.2 (Weak DIS-Optimal Enforcement)**.** For every sHML$_{\mathbf{nf}}$ formula φ, system state $s \in \text{Sys}_{\Pi}$, explicit trace t_τ and monitor m, if $\mathit{ec}_{bi}(m) \subseteq \{\mathsf{DIS}\}$, $\mathsf{enf}(m, \varphi)$ and $s \xrightarrow{t_\tau}$ then $\mathit{mc}((\!|\varphi, \Pi|\!), t_\tau) \leq \mathit{mc}(m, t_\tau)$. □

This result therefore ensures that one cannot find a *less intrusive* action disabling (finite) monitor that: has the same information portrayed by Π and has less impact on the original behaviour of the SuS, than the ones we synthesise. However, it does not exclude that a monitor with different, or additional, transformation capabilities and port information might be more optimal when enforcing φ. The full proof for this theorem is given in Appendix C.2.3 on page 237.

11.2 Summary

In this chapter we have explored the enforceability of μHML formulas in a bidirectional setting. Particularly, we have investigated how sHML formulas can be adequately and (weak) optimally enforced using action disabling monitors defined via the bidirectional enforcement model of Figure 9.2 from Chapter 9. The outcome of this study led to the following contributions:

(i) The synthesis function of Definition 11.2 which converts normalized sHML formulas into bidirectional action disabling monitors.

(ii) The proofs for Theorems 11.1 and 11.2 which ensure that the synthesised monitors adequately and optimally enforce the formula they were derived from, as stated by Definitions 10.2 and 10.7 respectively.

These results thus suffice to conclude that the sHML fragment is also enforceable in a bidirectional enforcement setting via action disabling transducers.

12. End of Part II

In the second part of this thesis we have studied the enforceability of the μHML branching time logic in a bidirectional enforcement setting. Unlike in Part I we have adopted a branching time view of the SuS rather than a trace based view, and lifted the assumption that every action can be freely modified by the monitor's transformations in the same way. To enable this study, it was therefore required to differentiate between the system's input and output behaviour. As a result, we conceptualised a novel distinction between the transformations (suppression, insertions and replacements) performed by the monitor, and the way the instrumentation interprets these transformations to enable, disable or adapt the actions of the SuS at runtime. Based on this distinction, in Chapter 9 we thus developed a bidirectional enforcement instrumentation model.

In order to study enforceability vis-a-vis this new model, in Chapter 10 we have explored whether the enforceability definitions of adequate and optimal enforcement, introduced in Chapter 4 of Part I, are still relevant for our new setting. Specifically, we showed how the criteria of soundness, transparency and eventual transparency can still be used to filter out inadequate transducers, and to guide into finding adequate ones. However, we also showed that the optimality criterion of Chapter 4 can at times be too strict. For this reason, we introduced a weaker version of this constraint that only guarantees optimal enforcement when the formula is enforced upon systems that use a specific set of communication ports.

Based on these notions of enforcement and using the work of Part I as guidance, in Chapter 11 we showed that sHML properties can also be enforced by omitting violating (input and output) actions using action disabling monitors. We thus devised a synthesis function that produces action disabling monitors that are correct and optimal by construction. Similar to the approach used in Part I, we defined the new synthesis function in a compositional manner by working with respect to a normalised sHML syntactic subset, given that it was shown to be equally expressive

to sHML. To ensure adherence to the weak optimality criterion, the new synthesis was developed with the capability of accepting the necessary port information and which it uses to produce weak optimal monitors.

As a result, this synthesis function allowed us to conclude that sHML is enforceable via action disabling in a bidirectional setting.

12.1 Related Work

Most work in runtime enforcement [25, 50, 69, 78, 82, 98] assume that *every* system action can be freely modified regardless of its type, *e.g.,* input, output, function call, *etc.* However, in our work we found out that some actions must be treated differently especially in a bidirectional setting. Similarly, Khoury and Hallé [67] remarked that for certain types of actions, the monitor might fail to suppress or insert them, and that certain actions are only observable, as no transformations are applicable in practice. As a result, they introduce a lattice of different classes of actions starting with the class of observable actions at the bottom. Side-by-side on top of this class lie the (non-intersecting) classes of suppressable and insertable actions, and at the top of the lattice lies the class of controllable actions, *i.e.,* actions that can be both suppressed or inserted. The work by Khoury and Hallé extends a prior investigation by Basin *et al.* [18], where the authors generalised Schneider's work on security automata [98] and proposed a dichotomy of observable and controllable actions. In the work by Basin *et al.* [18], actions are classified as controllable when their execution can be prevented by aborting the system at runtime, otherwise they are classified as being observable.

Ligatti *et al.* [44, 80] proposed an extension to their work on edit automata [78] by introducing an enforcement mechanism called Mandatory Results Automata (MRAs). MRAs were specifically designed to enforce properties about the interactions between *two* parties. The MRA-based monitor is placed in between the two entities in order to scrutinise their interactions by transforming their actions as necessary. Specifically, the monitor can only transform the *output* behaviour of each of the two entities and, unlike our work, it does not deal directly with the problem of enforcing properties about their input behaviour by transforming the input actions themselves. Moreover, in their work Ligatti *et al.* do not explore how MRAs can be used to enforce properties expressed using a logic, and so they do not address the problem of enforceability.

Pinisetty *et al.* [86, 87] conduct a preliminary investigation of RE in a bidirectional setting. They, however, model the behaviour of the SuS as a trace of input and output pairs, *a.k.a. reactions*, and focus on enforcing properties by modifying

the payloads exchanged by these reactions. This way of modelling system behaviour is, however, quite restrictive as it only applies to synchronous reactive systems that output a value in reaction to an input. This differs substantially from the way we model systems as LTSs, particularly since we can model more complex systems that may opt to collect data from multiple inputs, or supply multiple outputs in response to an input. The enforcement abilities studied by Pinisetty *et al.* [86, 87], are also confined to action replacement that only allows the monitor to modify the data exchanged by the system in its reactions. Therefore, their monitors are unable to disable and enable an individual (input or output) action itself. Due to their trace based view of the SuS, their correctness specifications do not allow for defining correct system behaviour in view of its different execution branches. This is particularly useful when considering systems whose inputs may lead them into taking erroneous computation branches that produce invalid outputs. Moreover, since their systems do not model communication ports, their monitors cannot influence directly the control structure of the SuS, *e.g.,* by opening, closing or rerouting data through different ports.

12.2 Future Work

We plan to expand on our current work on bidirectional enforcement along different avenues. First, we intend to study the maximality results for action disabling enforcement along the lines of the work by Aceto *et al.* [3, 56]. We are confident that sHML is the maximally expressive fragment of μHML that can be enforced by action disabling monitors, because in Part I we have already established that sHML is maximally expressive when enforced by suppression monitors in a unidirectional setting. We thus conjecture that the work conducted in Part I can be used as a guide during this future investigation.

Second, we intend to identify a static analysis technique that achieves the same result as our bidirectional action disabling monitors. Although in Part I we showed that controlled system synthesis is the static counterpart to suppression enforcement, it is unclear whether this is the case for bidirectional monitors. The main point of concern revolves around input actions. In controlled system synthesis, inputs are generally regarded as being uncontrollable, which means that they should remain untouched. By contrast, our disabling monitors can at times block certain inputs as necessary. Understanding how this technique relates to bidirectional enforcement thus merits an in depth investigation.

Third, we want to expand our exploration into bidirectional enforcement by studying the enforceability of a μHML fragment that is larger than sHML. This requires

understanding how the capabilities of the other enforcement instrumentation mechanisms (action enabling and adaptation) can be fully harnessed to enforce properties that cannot be expressed via sHML. As explained in Section 8.2 at the end of Part I, expanding the set of enforceable properties is not a trivial task. For instance, it is unclear whether our work on enforcing sHML properties can guide in identifying a set of monitors that enforce the co-safety dual of sHML *i.e.,* cHML.

Finally, we would also like to explore the implementability and feasibility of our instrumentation model for bidirectional enforcement. This would require looking into channel-based programming languages (such as GO) that can be used to implement bidirectional monitors that can block inputs by closing channels, thereby preventing invalid input interactions with the environment.

13. Concluding Remarks

Throughout the course of this thesis, we investigated the enforceability of the logic μHML. Put simply, our main aim was to answer the following question:

Q: Which μHML formulas can be adequately enforced by a monitor at runtime?

Our investigation gave rise to several other questions. The following five questions thus became our fundamental objectives throughout this thesis.

Q1 (modelling): What is an enforcement monitor, and how should we instrument it with the SuS?

Q2 (correctness): What are the criteria for adequate enforcement?

Q3 (expressiveness): Can all μHML formulas be adequately enforced? If no, then which μHML fragment is actually enforceable?

Q4 (maximality): Is the identified μHML fragment the largest fragment that one can enforce?

Q5 (static counterpart): Is the identified fragment only dynamically enforceable? Are there static alternatives that achieve an equivalent result?

As different enforcement instrumentation setups exist, answering objective Q1 required us to choose a specific instrumentation setup. In this thesis, we considered two instrumentation setups, namely, a unidirectional and a bidirectional enforcement setup. For this reason, we subdivided this thesis into two parts.

In Part I we aimed to answer objectives Q1 - Q5 in a unidirectional context. To answer Q1, in Chapter 3 we adopted a trace-based view of the SuS and defined an instrumentation model for unidirectional enforcement. As a result, we now have a formal model that defines how enforcement monitors behave at runtime and how they should be instrumented with the SuS to attain unidirectional enforcement. Having a way to define monitor behaviour, in Chapter 4 we addressed question Q2 by introducing three different definitions for adequate enforcement (Definitions 4.4,

4.6 and 4.8) where Definition 4.8 is the strictest and Definition 4.6 is the weakest. Moreover, since in certain cases a variety of monitors may be considered adequate for enforcing a single formula, we also introduced the notion of optimal enforcement which guides the search for the least intrusive one. These definitions are a crucial contribution as they define what it takes for a monitor to bring a system in line with a property while being minimally intrusive.

Based on these notions, in Chapter 5 we focussed on studying objective Q3. We thus devised a synthesis function that allowed us to determine that every sHML formula can be adequately and optimally enforced in a unidirectional setting via suppression monitors. This therefore implies that every safety property is enforceable in a unidirectional setting. In Chapter 6 we then focussed on answering question Q4 and showed that sHML is also the maximally expressive subset of μHML that can be enforced via suppression monitors. This is an important result since we now know that in a unidirectional setting, suppression monitors can *only* adequately enforce safety properties.

In Chapter 7 we finally addressed Q5 where we identified the static analysis technique called Controlled System Synthesis as being the static counterpart to suppression enforcement in the context of safety. This result thus entails that safety formulas can be enforced both dynamically via suppression monitors or statically by synthesising a controlled system. We also determined that in certain cases an external observer may still be able to tell the difference between a monitored and a controlled version of a system even though they enforce the same behaviour.

In Part II we addressed once again objectives Q1 - Q3, but this time, in a bidirectional enforcement setting. Unlike in Part I we have: adopted a branching time view of the SuS rather than a trace based one, differentiated between the system's input and output behaviour, and lifted the assumption that every action can be freely modified by the monitor's transformations. We also introduced a novel distinction between the transformations (suppression, insertions and replacements) performed by the monitor, and the instrumentation's resulting effects (action enabling, disabling and adaptation) on the composite system's behaviour. Using this distinction as a foundation, in Chapter 9 we developed a new instrumentation model for bidirectional enforcement. This work thus served to address question Q1 from a bidirectional enforcement perspective. As a result, we now have another formal model that defines how monitors must be instrumented to achieve bidirectional enforcement.

To address objective Q2 from a bidirectional perspective, in Chapter 10 we motivated why the enforcement definitions introduced in Part I, are still relevant for the bidirectional setting of Chapter 9. We also showed that the optimality criterion in-

troduced in Part I can sometimes be too strict, and so we defined a weaker version. This implies that our enforcement definitions are general enough to remain relevant for different enforcement settings.

Having shown the relevance of our enforcement definitions, we then addressed objective Q3 in Chapter 11. In particular, we showed that in a bidirectional setting, sHML properties can still be enforced by disabling violating (input and output) actions. This result strengthens that of Chapter 5 as we now know that safety properties are enforceable in a bidirectional setting via action disabling monitors. To show this, we defined another synthesis function that produces action disabling monitors that are adequate and weak optimal by construction. Due to time restrictions, the study of objectives Q4 and Q5 in a bidirectional context were left to future investigations.

Although the work in this thesis was conducted in respect to the logic μHML, our findings can be easily applied to other (more popular) logics such as LTL and CTL. This is permitted since μHML embeds a wide range of logics due to its high expressiveness. In fact, to apply our results to other logics one can simply encode the syntax of that logic in terms of the μHML syntax *e.g.*, an encoding of this sort was given in [5]. Moreover, despite that we describe systems as CCS (and value-passing CCS) processes, our transducers can enforce properties on *any* system that can be described as a LTS. Hence, to apply our work to systems other than CCS processes such as cyber-physical systems [76], one only requires providing LTS semantics for this kind of systems; in the case of cyber-physical systems, this has already been achieved [74, 94].

13.1 Overview of published work

We now give an overview of the research papers that the author of this thesis has contributed to during the course of his PhD.

Our core study about the enforceability of μHML in a unidirectional context was published in the 29th international conference on Concurrency Theory (CONCUR 2018), in the paper cited below. It contained most of the core results and contributions presented in Chapters 3 to 5 of Part I.

- L. Aceto, I. Cassar, A. Francalanza, and A. Ingólfsdóttir. On runtime enforcement via suppressions. In *29th International Conference on Concurrency Theory, CONCUR 2018*, pages 34:1–34:17, 2018.

The author formulated the formal models and proofs presented in the paper under the supervision of the co-authors *i.e.*, his PhD advisers. He was also extensively

involved in writing up the paper; this included writing up an initial draft which was then revised and edited based on the co-authors' feedback. An extended journal version of this conference paper is currently under review.

Another conference paper presenting the work of Chapter 7 has also been published in the 19th international conference on Runtime Verification (RV 2019), as cited below.

- L. Aceto, I. Cassar, A. Francalanza, and A. Ingólfsdóttir. Comparing controlled system synthesis and suppression enforcement. In *Runtime Verification: 19th International Conference, RV 2019*, pages 148–164. Springer, 2019.

Once again, the author formulated the presented mathematics and wrote the content of the paper with the assistance of the co-authors who provided continuous reviews to improve the quality of the content. At the time of writing, an extended journal version of this paper is being finalised to be submitted for review. In addition, most of the work presented in Part II has also been adapted into a research paper and submitted to a conference for review.

Besides publishing papers related to the work described in this thesis, the author of this thesis contributed to several other research results during his PhD studies. For starters, the following publications were made in relation to runtime verification and the Erlang RV tool detectEr.

- I. Cassar, A. Francalanza, D. Attard, L. Aceto, and A. Ingolfsdottir. A suite of monitoring tools for Erlang. In *RV-CuBES 2017. An International Workshop on Competitions, Usability, Benchmarks, Evaluation, and Standardisation for Runtime Verification Tools*, volume 3 of *Kalpa Publications in Computing*, pages 41–47. EasyChair, 2017.
- A. Francalanza, L. Aceto, A. Achilleos, D. P. Attard, I. Cassar, D. Della Monica, and A. Ingólfsdóttir. A Foundation for Runtime Monitoring. In *Runtime Verification: 17th International Conference, RV 2017*, pages 8–29. Springer, 2017.
- I. Cassar, A. Francalanza, L. Aceto, and A. Ingólfsdóttir. A survey of runtime monitoring instrumentation techniques. In *PrePost2017*, pages 15–28, 2017.
- D. P. Attard, I. Cassar, A. Francalanza, L. Aceto, and A. Ingolfsdottir. *A Runtime Monitoring Tool for Actor-Based Systems.*, chapter 3, pages 49–74. River Publishers, 2017.

To facilitate the development of monitoring tools for Erlang systems, the author also developed an aspect oriented programming framework for Erlang called eAOP[1]. As a result, the following papers were published containing implementation details and

[1]The eAOP framework is open-source and available from: `https://github.com/casian/eaop`.

use cases of how this framework has been used for developing runtime verification and adaptation tools.

- I. Cassar, A. Francalanza, L. Aceto, and A. Ingólfsdóttir. eAOP: An Aspect Oriented Programming Framework for Erlang. In *Erlang*, ACM SIGPLAN, 2017.
- I. Cassar, A. Francalanza, D. P. Attard, L. Aceto, and A. Ingolfsdottir. A generic instrumentation tool for Erlang. In *RV-CuBES 2017. An International Workshop on Competitions, Usability, Benchmarks, Evaluation, and Standardisation for Runtime Verification Tools*, volume 3 of *Kalpa Publications in Computing*, pages 48–54. EasyChair, 2017.

Bibliography

[1] S. Abramsky. Observation equivalence as a testing equivalence. *Theoretical Computer Science*, 53:225–241, 1987.

[2] L. Aceto, A. Achilleos, A. Francalanza, and A. Ingólfsdóttir. A framework for parameterized monitorability. In *Foundations of Software Science and Computation Structures*, pages 203–220. Springer, 2018.

[3] L. Aceto, A. Achilleos, A. Francalanza, and A. Ingólfsdóttir. Monitoring for silent actions. In *FSTTCS 2017: Foundations of Software Technology and Theoretical Computer Science*, volume 93 of *LIPIcs*, pages 7:1–7:14, 2018.

[4] L. Aceto, A. Achilleos, A. Francalanza, A. Ingólfsdóttir, and S. Ö. Kjartansson. Determinizing Monitors for HML with Recursion. *Journal of Logical and Algebraic Methods in Programming*, 111, 2016.

[5] L. Aceto, A. Achilleos, A. Francalanza, A. Ingólfsdóttir, and K. Lehtinen. Adventures in Monitorability: From Branching to Linear Time and Back Again. *Proceedings of the ACM Programing Languages*, pages 52:1–52:29, 2019.

[6] L. Aceto, A. Achilleos, A. Francalanza, A. Ingólfsdóttir, and K. Lehtinen. An Operational Guide to Monitorability. In *Software Engineering and Formal Methods*, pages 433–453. Springer, 2019.

[7] L. Aceto, A. Achilleos, A. Francalanza, A. Ingólfsdóttir, and K. Lehtinen. Testing equivalence vs. runtime monitoring. In *Models, Languages, and Tools for Concurrent and Distributed Programming: Essays Dedicated to Rocco De Nicola on the Occasion of His 65th Birthday*, pages 28–44. Springer, 2019.

[8] L. Aceto, I. Cassar, A. Francalanza, and A. Ingólfsdóttir. On runtime enforcement via suppressions. In *29th International Conference on Concurrency Theory, CONCUR 2018*, pages 34:1–34:17, 2018.

[9] L. Aceto, I. Cassar, A. Francalanza, and A. Ingólfsdóttir. Comparing controlled

system synthesis and suppression enforcement. In *Runtime Verification: 19th International Conference, RV 2019*, pages 148–164. Springer, 2019.

[10] L. Aceto and A. Ingólfsdóttir. Testing Hennessy-Milner logic with recursion. In *Foundations of Software Science and Computation Structures*, pages 41–55. Springer, 1999.

[11] L. Aceto, A. Ingólfsdóttir, K. G. Larsen, and J. Srba. *Reactive Systems: Modelling, Specification and Verification*. Cambridge University Press, 2007.

[12] R. Alur and P. Černý. 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*, pages 599–610. ACM, 2011.

[13] H. R. Andersen. Partial model checking. In *Proceedings of Tenth Annual IEEE Symposium on Logic in Computer Science*, pages 398–407. IEEE, 1995.

[14] A. Arnold and I. Walukiewicz. Nondeterministic controllers of nondeterministic processes. In *Logic and Automata*, volume 2 of *Texts in Logic and Games*, pages 29–52. Amsterdam University Press, 2008.

[15] D. P. Attard, I. Cassar, A. Francalanza, L. Aceto, and A. Ingolfsdottir. *A Runtime Monitoring Tool for Actor-Based Systems.*, chapter 3, pages 49–74. River Publishers, 2017.

[16] B. Banieqbal and H. Barringer. Temporal logic with fixed points. In *Temporal Logic in Specification*, pages 62–74, 1989.

[17] D. Basile, M. H. ter Beek, and R. Pugliese. Bridging the gap between supervisory control and coordination of services: Synthesis of orchestrations and choreographies. In *COORDINATION 2019 - 21st International Conference on Coordination Models and Languages*, 2019.

[18] D. Basin, V. Jugé, F. Klaedtke, and E. Zălinescu. Enforceable security policies revisited. In *Principles of Security and Trust*, pages 309–328. Springer, 2012.

[19] A. Bauer, M. Leucker, and C. Schallhart. Comparing LTL semantics for runtime verification. *Journal of Logic and Computation*, 20(3):651–674, 2010.

[20] A. Bauer, M. Leucker, and C. Schallhart. Runtime verification for LTL and TLTL. *ACM Transactions on Software Engineering and Methodology*, 20(4):14:1–14:64, 2011.

[21] D. Beauquier, J. Cohen, and R. Lanotte. Security policies enforcement using finite edit automata. *Electronic Notes in Theoretical Computer Science*, 229(3):19–35, 2009.

[22] J. Berstel and L. Boasson. Transductions and context-free languages. *Ed. Teubner*, pages 1–278, 1979.

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

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

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

[26] L. Bocchi, T.-C. Chen, R. Demangeon, K. Honda, and N. Yoshida. Monitoring networks through multiparty session types. *Theoretical Computer Science*, 669:33 – 58, 2017.

[27] J. A. Brzozowski. Derivatives of regular expressions. *Journal of the ACM*, 11(4):481–494, 1964.

[28] I. Cassar and A. Francalanza. On synchronous and asynchronous monitor instrumentation for actor-based systems. In *Proceedings 13th International Workshop on Foundations of Coordination Languages and Self-Adaptive Systems, FOCLASA 2014*, pages 54–68, 2014.

[29] I. Cassar and A. Francalanza. Runtime adaptation for actor systems. In *Runtime Verification - 6th International Conference, RV 2015*, volume 9333 of *Lecture Notes in Computer Science*, pages 38–54. Springer, 2015.

[30] I. Cassar and A. Francalanza. On implementing a monitor-oriented programming framework for actor systems. In *International Conference on Integrated Formal Methods*, pages 176–192. Springer, 2016.

[31] I. Cassar, A. Francalanza, L. Aceto, and A. Ingólfsdóttir. eAOP: An Aspect Oriented Programming Framework for Erlang. In *Erlang*, ACM SIGPLAN, 2017.

[32] I. Cassar, A. Francalanza, L. Aceto, and A. Ingólfsdóttir. A survey of runtime monitoring instrumentation techniques. In *PrePost2017*, pages 15–28, 2017.

[33] I. Cassar, A. Francalanza, D. Attard, L. Aceto, and A. Ingolfsdottir. A suite of monitoring tools for Erlang. In *RV-CuBES 2017. An International Workshop on Competitions, Usability, Benchmarks, Evaluation, and Standardisation for Runtime Verification Tools*, volume 3 of *Kalpa Publications in Computing*, pages 41–47. EasyChair, 2017.

[34] I. Cassar, A. Francalanza, D. P. Attard, L. Aceto, and A. Ingolfsdottir. A generic instrumentation tool for Erlang. In *RV-CuBES 2017. An International Workshop on Competitions, Usability, Benchmarks, Evaluation, and Standardisation for Runtime Verification Tools*, volume 3 of *Kalpa Publications in Computing*, pages 48–54. EasyChair, 2017.

[35] I. Cassar, A. Francalanza, and S. Said. Improving runtime overheads for detecter. In *Proceedings 12th International Workshop on Formal Engineering approaches to Software Components and Architectures, FESCA 2015.*, pages 1–8, 2015.

[36] I. Castellani, M. Dezani-Ciancaglini, and J. A. Pérez. Self-adaptation and secure information flow in multiparty communications. *Formal Aspects of Computing*, 28(4):669–696, 2016.

[37] E. Chang, Z. Manna, and A. Pnueli. The safety-progress classification. In *Logic and Algebra of Specification*, pages 143–202. Springer, 1993.

[38] T.-C. Chen, L. Bocchi, P.-M. Deniélou, K. Honda, and N. Yoshida. Asynchronous Distributed Monitoring for Multiparty Session Enforcement. In *Trustworthy Global Computing*, pages 25–45. Springer, 2012.

[39] C. Cini and A. Francalanza. An LTL proof system for runtime verification. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 581–595. Springer, 2015.

[40] E. M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *25 Years of Model Checking*, pages 196–215. Springer, 2008.

[41] E. M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT press, 1999.

[42] E. de Vries, V. Koutavas, and M. Hennessy. Communicating transactions. In *CONCUR 2010 - Concurrency Theory*, pages 569–583. Springer, 2010.

[43] A. Desai, T. Dreossi, and S. A. Seshia. Combining Model Checking and Runtime Verification for Safe Robotics. In *Runtime Verification: 17th International Conference, RV 2017*, LNCS, pages 172–189. Springer, 2017.

[44] E. Dolzhenko, J. Ligatti, and S. Reddy. Modeling runtime enforcement with mandatory results automata. *International Journal of Information Security*, 14(1):47–60, 2015.

[45] R. Ehlers, S. Lafortune, S. Tripakis, and M. Y. Vardi. Bridging the gap between supervisory control and reactive synthesis: Case of full observation and centralized control. In *WODES*, pages 222–227. International Federation of Automatic Control, 2014.

[46] U. Erlingsson and F. B. Schneider. SASI enforcement of security policies: A retrospective. In *Proceedings of the 1999 Workshop on New Security Paradigms*, NSPW '99, pages 87–95. ACM, 1999.

[47] Y. Falcone. You Should Better Enforce Than Verify. In *Runtime Verification: First International Conference, RV 2010*, pages 89–105. Springer, 2010.

[48] Y. Falcone, J.-C. Fernandez, and L. Mounier. Synthesizing enforcement monitors wrt. the safety-progress classification of properties. In *Information Systems Security*. Springer, 2008.

[49] Y. Falcone, J.-C. Fernandez, and L. Mounier. Enforcement monitoring wrt. the safety-progress classification of properties. In *Proceedings of the 2009 ACM Symposium on Applied Computing*, SAC '09, pages 593–600. ACM, 2009.

[50] Y. Falcone, J.-C. Fernandez, and L. Mounier. What can you verify and enforce at runtime? *International Journal on Software Tools for Technology Transfer*, 14(3):349, 2012.

[51] Y. Falcone and H. Marchand. Runtime enforcement of k-step opacity. In *52nd IEEE Conference on Decision and Control*, pages 7271–7278, 2013.

[52] Y. Falcone, L. Mounier, J.-C. Fernandez, and J.-L. Richier. Runtime enforcement monitors: composition, synthesis, and enforcement abilities. *Formal Methods in System Design*, 38(3):223–262, 2011.

[53] A. Francalanza. A Theory of Monitors. In *International Conference on Foundations of Software Science and Computation Structures*, pages 145–161. Springer, 2016.

[54] A. Francalanza. Consistently-Detecting Monitors. In *28th International Conference on Concurrency Theory (CONCUR 2017)*, volume 85 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 8:1–8:19. Schloss Dagstuhl, 2017.

[55] A. Francalanza, L. Aceto, A. Achilleos, D. P. Attard, I. Cassar, D. Della Monica, and A. Ingólfsdóttir. A Foundation for Runtime Monitoring. In *Runtime Verification: 17th International Conference, RV 2017*, pages 8–29. Springer, 2017.

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

[57] A. Francalanza and A. Seychell. Synthesising correct concurrent runtime monitors. *Formal Methods in System Design*, 46(3):226–261, 2015.

[58] K. Havelund and T. Pressburger. Model checking java programs using java pathfinder. *International Journal on Software Tools for Technology Transfer*, 2(4):366–381, 2000.

[59] K. Havelund and G. Roşu. An overview of the runtime verification tool java pathexplorer. *Formal methods in system design*, 24(2):189–215, 2004.

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

[61] M. Hennessy and X. Liu. A modal logic for message passing processes. *Acta Informatica*, 32(4):375–393, 1995.

[62] M. Hennessy and R. Milner. Algebraic laws for nondeterminism and concurrency. *Journal of the ACM*, 32(1):137–161, 1985.

[63] M. Hennessy and T. Regan. A process algebra for timed systems. *Information and Computation*, 117(2):221 – 239, 1995.

[64] L. Jia, H. Gommerstadt, and F. Pfenning. Monitors and blame assignment for higher-order session types. In *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 582–594, 2016.

[65] K. Kejstová, P. Ročkai, and J. Barnat. From Model Checking to Runtime Verification and Back. In *Runtime Verification: 17th International Conference, RV 2017*. Springer, 2017.

[66] R. M. Keller. Formal Verification of Parallel Programs. *Communications of the ACM*, 19(7):371–384, 1976.

[67] R. Khoury and S. Hallé. Runtime enforcement with partial control. *Foundations and Practice of Security*, 2016.

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

[69] B. Könighofer, M. Alshiekh, R. Bloem, L. Humphrey, R. Könighofer, U. Topcu, and C. Wang. Shield synthesis. *Formal Methods in System Design*, 51(2):332–361, 2017.

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

[71] F. Lang and R. Mateescu. Partial model checking using networks of labelled transition systems and boolean equation systems. In *18th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2012*, pages 141–156. Springer, 2012.

[72] R. Lanotte, M. Merro, and A. Munteanu. A process calculus approach to correctness enforcement of plcs. In *Proceedings of the 21st Italian Conference on Theoretical Computer Science (ICTCS 2020)*, pages 81–94, 2020.

[73] R. Lanotte, M. Merro, and A. Munteanu. Runtime enforcement for control system security. In *Proceedings of the 33rd IEEE Computer Security Foundations Symposium, CSF 2020*, pages 246–261, 2020.

[74] R. Lanotte, M. Merro, A. Munteanu, and L. Viganò. A formal approach to physics-based attacks in cyber-physical systems (extended version). *CoRR*, abs/1902.04572, 2019.

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

[76] E. A. Lee and S. A. Seshia. *Introduction to Embedded Systems: A Cyber-Physical Systems Approach*. The MIT Press, 2nd edition, 2016.

[77] M. Leucker and C. Schallhart. A brief account of runtime verification. *The Journal of Logic and Algebraic Programming*, 78(5):293–303, 2009.

[78] J. Ligatti, L. Bauer, and D. Walker. Edit automata: enforcement mechanisms for run-time security policies. *International Journal of Information Security*, 4(1):2–16, 2005.

[79] J. Ligatti, L. Bauer, and D. Walker. Run-time enforcement of nonsafety policies. *ACM Transactions on Information and System Security.*, 12(3):19:1–19:41, 2009.

[80] J. Ligatti and S. Reddy. A theory of runtime enforcement, with results. In *Computer Secutity - European Symposium on Research in Computer Security, CESORICS 2010*, pages 87–100. Springer, 2010.

[81] Z. Manna and A. Pnueli. Completing the temporal picture. *Theoretical Computer Science*, 83(1):91–130, 1991.

[82] F. Martinelli and I. Matteucci. Partial model checking, process algebra operators and satisfiability procedures for (automatically) enforcing security properties. In *Foundations of Computer Security*, pages 133–144, 2005.

[83] F. Martinelli and I. Matteucci. Through modeling to synthesis of security automata. *Electronic Notes in Theoretical Computer Science*, 179:31–46, 2006.

[84] F. Martinelli and I. Matteucci. An approach for the specification, verification and synthesis of secure systems. *Electronic Notes in Theoretical Computer Science*, 168:29–43, 2007.

[85] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, I. *Information and computation*, 100(1):1–40, 1992.

[86] S. Pinisetty, P. S. Roop, S. Smyth, N. Allen, S. Tripakis, and R. V. Hanxleden. Runtime enforcement of cyber-physical systems. *ACM Transactions on Embedded Computing Systems*, 16(5):178:1–178:25, 2017.

[87] S. Pinisetty, P. S. Roop, S. Smyth, S. Tripakis, and R. von Hanxleden. Runtime enforcement of reactive systems using synchronous enforcers. *CoRR*, abs/1612.05030, 2016.

[88] A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Proceedings of the 16th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '89, pages 179–190. ACM, 1989.

[89] A. Pnueli and A. Zaks. PSL model checking and run-time verification via testers. In *International Symposium on Formal Methods*, pages 573–586. Springer, 2006.

[90] Z. M. A. Pnueli. A hierarchy of temporal properties. *Proceedings of the 2th symph. ACM of principle of distributed computer*, 1990.

[91] A. M. Rabinovich. A complete axiomatisation for trace congruence of finite state behaviors. In *Proceedings of the 9th International Conference on Mathematical Foundations of Programming Semantics*, pages 530–543. Springer-Verlag, 1994.

[92] P. J. Ramadge and W. M. Wonham. Supervisory control of a class of discrete event processes. *SIAM Journal on Control and Optimization*, 25(1):206–230, 1987.

[93] J. Rathke and M. Hennessy. Local model checking for value-passing processes (extended abstract). In *Theoretical Aspects of Computer Software*, pages 250–266. Springer, 1997.

[94] W. C. Rounds and H. Song. The ö-calculus: A language for distributed control of reconfigurable embedded systems. In *Hybrid Systems: Computation and Control*, pages 435–449. Springer, 2003.

[95] J. Sakarovitch. *Elements of Automata Theory*. Cambridge University Press, 2009.

[96] D. Sangiorgi. A theory of bisimulation for the pi-calculus. *Acta Informatica*, 33(1):69–97, 1996.

[97] D. Sangiorgi. *Introduction to Bisimulation and Coinduction*. Cambridge University Press, 2011.

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

[99] C. Stirling. Handbook of logic in computer science (vol. 2). chapter Modal and Temporal Logics, pages 477–563. Oxford University Press, Inc., 1992.

[100] C. Stirling. Model checking and other games. In *Notes for Mathfit Workshop on finite model theory, University of Wales, Swansea*, 1996.

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

[102] I. Walukiewicz. Completeness of Kozen's axiomatisation of the propositional μ-calculus. *Information and Computation*, 157(1):142–182, 2000.

[103] M. Wu, H. Zeng, C. Wang, and H. Yu. Invited: Safety guard: Runtime enforcement for safety-critical cyber-physical systems. In *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6, 2017.

A. Missing Proofs from Chapter 2

In this brief chapter we provide the proofs for Propositions 2.3 and 2.4 which justify the *after* function for the safety and cosafety fragments of μHML.

A.1 Proving Proposition 2.3

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

Proof. As we assume that logical variables are guarded (refer to Section 2.2 on Page 11), we can proceed by rule induction on *after*.

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

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

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

$$s \xRightarrow{\alpha} s' \tag{A.1}$$

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

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

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

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

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

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

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

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

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

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

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

Now consider the following two cases:

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

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

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

□

A.2 Proving Proposition 2.4

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

Proof. As we assume that logical variables are guarded, we can once again by rule induction on *after*.

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

Case after(tt, α). This case holds trivially since $s \in [\![\mathsf{tt}]\!]$.

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

$$s \stackrel{\alpha}{\Longrightarrow} s' \tag{A.8}$$

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

$$s' \in [\![\bigvee_{i \in I} \mathit{after}(\varphi_i, \alpha)]\!] \;\equiv\; \exists j \in I \cdot s' \in [\![\mathit{after}(\varphi_j, \alpha)]\!]. \tag{A.9}$$

Hence, by (A.8) and (A.9) we can apply the inductive hypothesis and deduce that there exists a $j \in I$ such that $s \in [\![\varphi_j]\!]$ which means that $s \in \bigcup_{i \in I} [\![\varphi_i]\!] = [\![\bigvee_{i \in I} \varphi_i]\!]$ as required.

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

$$s \stackrel{\alpha}{\Longrightarrow} s' \tag{A.10}$$

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

$$s' \in [\![\mathit{after}(\varphi\{\mathsf{min}\, X.\varphi / X\}, \alpha)]\!]. \tag{A.11}$$

By (A.10), (A.11) and the inductive hypothesis we have that $s \in [\![\varphi\{\mathsf{min}\, X.\varphi / X\}]\!]$. Since $[\![\varphi\{\mathsf{min}\, X.\varphi / X\}]\!] = [\![\mathsf{min}\, X.\varphi]\!]$, we can conclude that $s \in [\![\mathsf{min}\, X.\varphi]\!]$ as required.

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

$$s \stackrel{\alpha}{\Longrightarrow} s' \quad \textit{and} \tag{A.12}$$

$$s' \in [\![\textit{after}(\langle\{p, c\}\varphi\rangle, \alpha)]\!]. \tag{A.13}$$

Now consider the following two cases:

- $\mathsf{mtch}(p, \alpha) = \sigma$ and $c\sigma \Downarrow \mathsf{true}$ (for some σ): By (A.13) and the definition of *after* we know that

  $$s' \in [\![\varphi\sigma]\!] \tag{A.14}$$

  and so from (A.12), (A.14) and by the definition of $[\![-]\!]$ we can infer that $s \in [\![\langle\{p, c\}\varphi\rangle]\!]$ since there exists a transition, *i.e.,* (A.12), that leads to a satisfaction, *i.e.,* (A.14).

- Otherwise: This case does not apply since *after*$(\langle\{p, c\}\varphi\rangle, \alpha) = \mathsf{ff}$ which contradicts assumption (A.13).

□

B. Missing Proofs from Part I

In this appendix chapter we provide the proofs for the theorems, propositions and lemmas that were omitted from the chapters that form Part I of the main text.

B.1 Missing proofs from Chapter 3

This section provides the proofs for Propositions 3.1 and 3.2 that were omitted from the main text of Chapter 3.

B.1.1 Proving Proposition 3.1 (Unzipping)

We prove that for every instrumented transition $m[s] \stackrel{u}{\Longrightarrow} m'[s']$ we can conclude:

(a) $u = \mathit{zip}(t, \kappa)$ and $m \stackrel{\kappa}{\Longrightarrow} m'$ and $s \stackrel{t}{\Longrightarrow} s'$; or

(b) $u = \mathit{zip}(t, \kappa); \alpha t'$ and $m \stackrel{\kappa}{\Longrightarrow} m'' \stackrel{\alpha \blacktriangleright \gamma}{\not\longrightarrow}$ and $m'' \stackrel{\bullet}{\not\rightarrow}$ and $s \stackrel{t;\alpha t'}{\Longrightarrow} s'$ and $m' = \mathsf{id}$.

Proof. We proceed by induction on the number of μ reductions in $m[s] \stackrel{u}{\Longrightarrow} m'[s']$.

Case 0 reductions. Since we assume 0 reductions we know that $m[s] \stackrel{u}{\Longrightarrow} m'[s']$ is equivalent to $m[s] \stackrel{\mu}{\rightarrow}^0 m'[s']$ and so this means that $m' = m$, $s' = s$ and $u = \varepsilon$ so that by the definition of $\stackrel{\varepsilon}{\Longrightarrow}$ we can conclude that

$$s \stackrel{\varepsilon}{\Longrightarrow} s' \tag{B.1}$$

$$m \stackrel{\varepsilon}{\Longrightarrow} m' \tag{B.2}$$

as required. Hence, this case holds by (B.1) and (B.2) since $u = \varepsilon = \mathit{zip}(\varepsilon, \varepsilon)$.

Case $k+1$ reductions. As we now assume $k+1$ reductions we have that

$$m[s] \xrightarrow{\mu} m''[s''] \tag{B.3}$$

$$m''[s''] \xrightarrow{\mu}{}^{k} m'[s'] \quad (\equiv m''[s''] \stackrel{u}{\Longrightarrow} m'[s']) \tag{B.4}$$

and so by (B.4) and the *inductive hypothesis* we can immediately deduce either that

$$\begin{gathered} u = \mathit{zip}(t,\kappa) \text{ and } m'' \stackrel{\kappa}{\Longrightarrow} m' \text{ and } s'' \stackrel{t}{\Longrightarrow} s'; \qquad \text{or that} \\ \left(\begin{array}{c} u = \mathit{zip}(t,\kappa);\alpha t' \text{ and } m'' \stackrel{\kappa}{\Longrightarrow} m''' \xrightarrow{\alpha \blacktriangleright \gamma} \not\to \text{ and } m''' \xrightarrow{\bullet} \not\to \\ \text{and } s'' \stackrel{t;\alpha t'}{\Longrightarrow} s' \text{ and } m' = \mathsf{id} \end{array} \right). \end{gathered} \tag{B.5}$$

Since the reduction in (B.3) can be the result of any instrumentation rule, we must consider each eventuality.

- iSup: In this case from (B.3) we can deduce that $\mu = \tau$ and that

$$m \xrightarrow{\beta \blacktriangleright \bullet} m'' \tag{B.6}$$

$$s \xrightarrow{\beta} s'' \tag{B.7}$$

  and so since $\mathit{zip}(\beta t, (\beta \blacktriangleright \bullet)\kappa) = u = \mathit{zip}(t,\kappa)$, if we combine (B.6) and (B.7) to the respective reductions in (B.5) we can conclude that

$$\begin{gathered} u{=}\mathit{zip}(\beta t, (\beta \blacktriangleright \bullet)\kappa) \text{ and } m \stackrel{(\beta \blacktriangleright \bullet)\kappa}{\Longrightarrow} m' \text{ and } s \stackrel{\beta t}{\Longrightarrow} s'; \text{ or that} \\ \left(\begin{array}{c} u = \mathit{zip}(\beta t, (\beta \blacktriangleright \bullet)\kappa);\alpha t' \text{ and } m \stackrel{(\beta \blacktriangleright \bullet)\kappa}{\Longrightarrow} m''' \xrightarrow{\alpha \blacktriangleright \gamma} \not\to \text{ and} \\ m''' \xrightarrow{\bullet} \not\to \text{ and } s \stackrel{\beta t;\alpha t'}{\Longrightarrow} s' \text{ and } m' = \mathsf{id} \end{array} \right) \end{gathered}$$

  as required.

- iDef: As we now assume that (B.3) results from rule iDef we thus have that $\mu = \alpha$ and that

$$s \xrightarrow{\alpha} s'' \tag{B.8}$$

$$m \xrightarrow{\alpha \blacktriangleright \gamma} \not\to \text{ and } m \xrightarrow{\bullet} \not\to \tag{B.9}$$

$$m'' = \mathsf{id}. \tag{B.10}$$

  Since we know (B.10) we can deduce that during the remaining reductions (shown in (B.4)) the identity monitor m'' can only follow the actions of the

system without modifying them or transitioning to some state other than id, and so we have that

$$s'' \overset{t}{\Longrightarrow} s' \tag{B.11}$$

$$m' = \mathsf{id}. \tag{B.12}$$

Since by the definition of $\overset{\varepsilon}{\Longrightarrow}$ we know that $s \overset{\varepsilon}{\Longrightarrow} s$, we can combine it with (B.8) and (B.11) to conclude that

$$s \overset{\varepsilon;\alpha t}{\Longrightarrow} s'. \tag{B.13}$$

and similarly, by the definition of $\overset{\varepsilon}{\Longrightarrow}$ from (B.9) we deduce that

$$m \overset{\varepsilon}{\Longrightarrow} m \xrightarrow{\alpha \blacktriangleright \gamma} \not\rightarrow \text{ and } m \xrightarrow{\bullet \blacktriangleright \alpha} \not\rightarrow. \tag{B.14}$$

Since we know (B.10), we can deduce that the identity monitor m'' in (B.4) can only follow the system actions via rule EID and so we have that

$$u = \alpha t = \varepsilon; \alpha t \tag{B.15}$$

and hence, this case holds since $\mathit{zip}(\varepsilon, \varepsilon) = \varepsilon$, which means that by (B.12), (B.13), (B.14) and (B.15) we can deduce that

$$\begin{pmatrix} u = \mathit{zip}(\varepsilon, \varepsilon); \alpha t \text{ and } m \overset{\varepsilon}{\Longrightarrow} m \xrightarrow{\alpha \blacktriangleright \gamma} \not\rightarrow \text{ and } m \xrightarrow{\bullet} \not\rightarrow \\ \text{and } s \overset{\varepsilon;\alpha t}{\Longrightarrow} s' \text{ and } m' = \mathsf{id} \end{pmatrix}$$

as required.

- IINS: From (B.3) we can now deduce that $\mu = \alpha$ and that

$$m \xrightarrow{\bullet \blacktriangleright \alpha} m'' \tag{B.16}$$

$$s'' = s' \tag{B.17}$$

and so by the definition of *zip* we deduce that $\mathit{zip}(t, (\bullet \blacktriangleright \alpha)\kappa) = \alpha u$ where $\mathit{zip}(t, \kappa)$,

so that by (B.16), (B.17) and (B.5) we conclude that

$$\alpha u{=}\mathit{zip}(t,(\bullet\triangleright\alpha)\kappa)\ \mathit{and}\ m \xRightarrow{(\bullet\triangleright\alpha)\kappa} m'\ \mathit{and}\ s \xRightarrow{t} s';\ \text{or that}$$
$$\left(\begin{array}{c}\alpha u = \mathit{zip}(t,(\bullet\triangleright\alpha)\kappa);\alpha t'\ \mathit{and}\ m \xRightarrow{(\bullet\triangleright\alpha)\kappa} m'''\not\xrightarrow{\alpha\triangleright\gamma}\ \mathit{and}\\ m'''\not\xrightarrow{\bullet}\ \mathit{and}\ s \xRightarrow{t;\alpha t'} s'\ \mathit{and}\ m' = \mathsf{id}\end{array}\right)$$

as required.

- rTrn: When considering rule rTrn, from (B.3) we infer that $\mu = \beta$ and that

$$m \xrightarrow{\alpha\triangleright\beta} m'' \tag{B.18}$$
$$s \xRightarrow{\alpha} s'' \tag{B.19}$$

and hence since $\mathit{zip}(\alpha t,(\alpha\triangleright\beta)\kappa) = \beta u$ where $u = \mathit{zip}(t,\kappa)$, by combining (B.18) and (B.19) to the respective reductions in (B.5) we can conclude that

$$\beta u{=}\mathit{zip}(\alpha t,(\alpha\triangleright\beta)\kappa)\ \mathit{and}\ m \xRightarrow{(\alpha\triangleright\beta)\kappa} m'\ \mathit{and}\ s \xRightarrow{\alpha t} s';\ \text{or that}$$
$$\left(\begin{array}{c}\beta u = \mathit{zip}(\alpha t,(\alpha\triangleright\beta)\kappa);\alpha t'\ \mathit{and}\ m \xRightarrow{(\alpha\triangleright\beta)\kappa} m'''\not\xrightarrow{\alpha\triangleright\gamma}\ \mathit{and}\\ m'''\not\xrightarrow{\bullet}\ \mathit{and}\ s \xRightarrow{\alpha t;\alpha t'} s'\ \mathit{and}\ m' = \mathsf{id}\end{array}\right)$$

as required.

- iAsy: We now assume that (B.3) results from rule iAsy and so we have that $\mu = \tau$, $s \xrightarrow{\tau} s''$ and $m'' = m$, and so since $\tau u = u$ and $\tau t = t$ from (B.5) we can conclude that

$$u{=}\mathit{zip}(t,\kappa)\ \mathit{and}\ m \xRightarrow{\kappa} m'\ \mathit{and}\ s \xRightarrow{t} s';\ \text{or that}$$
$$\left(\begin{array}{c}u = \mathit{zip}(t,\kappa);\alpha t'\ \mathit{and}\ m \xRightarrow{\kappa} m'''\not\xrightarrow{\alpha\triangleright\gamma}\ \mathit{and}\\ m'''\not\xrightarrow{\bullet}\ \mathit{and}\ s \xRightarrow{t;\alpha t'} s'\ \mathit{and}\ m' = \mathsf{id}\end{array}\right)$$

as required, and so we are done. □

B.1.2 Proving Proposition 3.2 (Zipping)

We must prove Proposition 3.2 by ensuring that

(a) $m \xRightarrow{\kappa} m'$ *and* $s \xRightarrow{t} s'$ *and* $\mathit{zip}(t,\kappa) = u$ *imply* $m[s] \xRightarrow{u} m'[s']$, and that

(b) $m \stackrel{\kappa}{\Longrightarrow} m'' \stackrel{\alpha \blacktriangleright \gamma}{\not\longrightarrow}$ *and* $m'' \stackrel{\bullet}{\not\longrightarrow}$ *and* $s \stackrel{t;\alpha t'}{\Longrightarrow} s'$ *and* $\mathit{zip}(t, \kappa) = u$ *imply* $m[s] \stackrel{u;\alpha t'}{\Longrightarrow} \mathsf{id}[s']$.

Proof for (a). We proceed by rule induction on $\mathit{zip}(t, \kappa)$.

Case $\mathit{zip}(\varepsilon, \varepsilon)$. We initially assume that $\mathit{zip}(\varepsilon, \varepsilon) = \varepsilon$ and that

$$s \stackrel{\varepsilon}{\Longrightarrow} s' \tag{B.20}$$

$$m \stackrel{\varepsilon}{\Longrightarrow} m'. \tag{B.21}$$

Since from (B.21) and by the definition of $\stackrel{\varepsilon}{\Longrightarrow}$ we know that $m' = m$, and since $\stackrel{\varepsilon}{\Longrightarrow} \stackrel{\text{def}}{=} \stackrel{\tau}{\longrightarrow}\!\!^*$ we can apply rule IASY for zero or more times on (B.20) to deduce that $m[s] \stackrel{\varepsilon}{\Longrightarrow} m'[s']$ as required.

Case $\mathit{zip}(\alpha t, (\alpha_{\blacktriangleright} \bullet)\kappa)$. We start by assuming that $\mathit{zip}(\alpha t, (\alpha_{\blacktriangleright} \bullet)\kappa) = u$ because

$$\mathit{zip}(t, \kappa) = u \tag{B.22}$$

and that $s \stackrel{\alpha t}{\Longrightarrow} s'$ and $m \stackrel{(\alpha \blacktriangleright \bullet)\kappa}{\Longrightarrow} m'$ from which by the definitions of $\stackrel{t}{\Longrightarrow}$ and $\stackrel{\kappa}{\Longrightarrow}$ respectively we can infer that

$$s \stackrel{\alpha}{\Longrightarrow} s'' \tag{B.23}$$

$$s'' \stackrel{t}{\Longrightarrow} s' \tag{B.24}$$

$$m \stackrel{\alpha \blacktriangleright \bullet}{\longrightarrow} m'' \tag{B.25}$$

$$m'' \stackrel{\kappa}{\Longrightarrow} m'. \tag{B.26}$$

Since $\stackrel{\alpha}{\Longrightarrow} \stackrel{\text{def}}{=} \stackrel{\tau}{\longrightarrow}\!\!^* \stackrel{\alpha}{\longrightarrow}$ from (B.23) we have that $s \stackrel{\tau}{\longrightarrow}\!\!^* s'''$ and that $s''' \stackrel{\alpha}{\longrightarrow} s''$, and so knowing the former we can apply rule IASY for zero or more times, and subsequently rule ISUP, since we know (B.25) and the latter, to deduce that

$$m[s] \stackrel{\varepsilon}{\Longrightarrow} m''[s'']. \tag{B.27}$$

Finally, since we know (B.22), (B.24) and (B.26) we can invoke the inductive hypothesis and deduce that $m''[s''] \stackrel{u}{\Longrightarrow} m'[s']$, which when combined with (B.27) we are able to conclude that $m[s] \stackrel{u}{\Longrightarrow} m'[s']$ as required.

Case $\mathit{zip}(t, (\bullet \triangleright \alpha)\kappa)$. We start by assuming that $\mathit{zip}(t, (\bullet \triangleright \alpha)\kappa) = \alpha u$ because

$$u = \mathit{zip}(t, \kappa) \tag{B.28}$$

and that

$$s \xRightarrow{t} s' \tag{B.29}$$

$$m \xRightarrow{(\bullet \triangleright \alpha)\kappa} m' \tag{B.30}$$

and so by applying the definition $\xRightarrow{\kappa}$ to (B.30) we have that

$$m'' \xRightarrow{\kappa} m'. \tag{B.31}$$

and that $m \xrightarrow{\bullet \triangleright \alpha} m''$ from which by rule iIns we can infer that

$$m[s] \xRightarrow{\alpha} m''[s]. \tag{B.32}$$

Finally, since we know (B.28), (B.29) and (B.31) we can apply the inductive hypothesis and deduce that $m''[s] \xRightarrow{u} m'[s']$, which we can combine with (B.32) to conclude that $m[s] \xRightarrow{\alpha u} m'[s']$ as required.

Case $\mathit{zip}(\alpha t, (\alpha \triangleright \beta)\kappa)$. We first assume that $\mathit{zip}(\alpha t, (\alpha \triangleright \beta)\kappa) = \beta u$ since

$$\mathit{zip}(t, \kappa) = u \tag{B.33}$$

and that $s \xRightarrow{\alpha t} s'$ and $m \xRightarrow{(\alpha \triangleright \beta)\kappa} m'$ on which we can apply the definitions of $\xRightarrow{t}$ and $\xRightarrow{\kappa}$ respectively to infer that

$$s \xRightarrow{\alpha} s'' \tag{B.34}$$

$$s'' \xRightarrow{t} s' \tag{B.35}$$

$$m \xrightarrow{\alpha \triangleright \beta} m'' \tag{B.36}$$

$$m'' \xRightarrow{\kappa} m'. \tag{B.37}$$

Since $\xRightarrow{\alpha} \overset{\text{def}}{=} \xrightarrow{\tau}{}^{*} \xrightarrow{\alpha}$, from (B.34) we have that $s \xrightarrow{\tau}{}^{*} s'''$ and that $s''' \xrightarrow{\beta} s''$, and so knowing the former we can apply rule iAsy for zero or more times, and subsequently

rule ıTrn since we know (B.36) and the latter, thus deducing that

$$m[s] \stackrel{\beta}{\Longrightarrow} m''[s'']. \tag{B.38}$$

Finally, since we know (B.33), (B.35) and (B.37) we apply the inductive hypothesis and deduce that $m''[s''] \stackrel{u}{\Longrightarrow} m'[s']$, which when combined with (B.38) we can conclude that $m[s] \stackrel{\beta u}{\Longrightarrow} m'[s']$ as required, and so we are done. □

Proof for (b). Initially we assume that

$$\mathit{zip}(t, \kappa) = u \tag{B.39}$$

$$m \stackrel{\kappa}{\Longrightarrow} m'' \tag{B.40}$$

$$m'' \stackrel{\alpha \blacktriangleright \gamma}{\not\longrightarrow} \textit{ and } m'' \stackrel{\bullet}{\not\longrightarrow} \tag{B.41}$$

and that $s \stackrel{t;\alpha t'}{\Longrightarrow} s'$ which can be further subdivided as

$$s \stackrel{t}{\Longrightarrow} s'' \tag{B.42}$$

$$s'' \stackrel{\tau}{\longrightarrow}\!\!^{*}\ s''' \tag{B.43}$$

$$s''' \stackrel{\alpha}{\longrightarrow} s'''' \tag{B.44}$$

$$s'''' \stackrel{t''}{\Longrightarrow} s' \tag{B.45}$$

and so by the *Proposition 3.2 (a)*, from (B.39), (B.40) and (B.42) we deduce that

$$m[s] \stackrel{u}{\Longrightarrow} m''[s'']. \tag{B.46}$$

We also apply rule ıAsy to (B.43) from which we infer that $m''[s''] \stackrel{\tau}{\longrightarrow}\!\!^{*}\ m''[s''']$, and by applying rule ıDef to (B.41) and (B.44) we deduce that $m''[s'''] \stackrel{\alpha}{\longrightarrow} \mathsf{id}[s'''']$, and so by combining these two results we thus conclude that

$$m''[s''] \stackrel{\alpha}{\Longrightarrow} \mathsf{id}[s'''']. \tag{B.47}$$

Finally, since id only defines identity transformations and always recurses back to the same state, from (B.45) we can infer that

$$\mathsf{id}[s''''] \stackrel{t'}{\Longrightarrow} \mathsf{id}[s']. \tag{B.48}$$

Finally, by combining (B.46), (B.47) and (B.48) we conclude that $\mathsf{id}[s] \stackrel{u;\alpha t'}{\Longrightarrow} \mathsf{id}[s']$ as required, and so we are done. □

B.2 Missing proofs from Chapter 4

We now provide the proofs for the supporting lemmas used in Theorem 4.2 *i.e.*, Lemmas 4.1 and 4.2.

B.2.1 Proving Lemma 4.1

We need to prove that for every system s, sHML formula φ and trace $t \in \mathit{traces}(s)$ when $s \in [\![\varphi]\!]$ then $\mathsf{sys}(t) \in [\![\varphi]\!]$. Recall that when restricted to sHML $s \in [\![\varphi]\!]$ can be defined in terms of the coinductive satisfaction rules of Figure 5.1 given in Chapter 5. We therefore prove that $\mathcal{R} \stackrel{\text{def}}{=} \{ (\mathsf{sys}(t), \varphi) \mid s \vDash \varphi \text{ and } t \in \mathit{traces}(s) \}$ is a satisfation relation that follows the rules of Figure 5.1.

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

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

$$t \in \mathit{traces}(s) \tag{B.49}$$

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

$$s \stackrel{\alpha}{\Longrightarrow} s' \tag{B.50}$$

$$\mathsf{mtch}(p, \alpha) = \sigma \text{ and } c\sigma \Downarrow \mathsf{true} \tag{B.51}$$

$$s' \vDash \varphi\sigma. \tag{B.52}$$

Since from (B.50) we know that s transitions to s' over α, from (B.49) we can infer that $\alpha t' \in \mathit{traces}(s)$ where $t' \in \mathit{traces}(s')$ which means that by (B.52) and the definition of $\mathcal{R}$ we have that

$$(\mathsf{sys}(t'), \varphi\sigma) \in \mathcal{R}. \tag{B.53}$$

Therefore, this case holds by (B.51), (B.53) and since $\mathsf{sys}(\alpha t') \stackrel{\alpha}{\Longrightarrow} \mathsf{sys}(t')$ and so we are done. □

B.2.2 Proving Lemma 4.2

We need to prove that for every action α, sHML formula φ and trace t, if $\mathsf{sys}(t) \in [\![\mathit{after}(\varphi, \alpha)]\!]$ then $\mathsf{sys}(\alpha t) \in [\![\varphi]\!]$.

Proof. We proceed by rule induction on *after*.

Case $\mathit{after}(\mathsf{ff}, \alpha)$. This case does not apply since $\mathit{after}(\mathsf{ff}, \alpha) = \mathsf{ff}$ and so the assumption that $\mathsf{sys}(t) \in [\![\mathit{after}(\mathsf{ff}, \alpha)]\!]$ is invalid.

Case $\mathit{after}(\mathsf{tt}, \alpha)$. This case holds trivially since $\mathsf{sys}(\alpha t) \in [\![\mathsf{tt}]\!]$.

Case $\mathit{after}(\bigwedge_{i\in I} \varphi_i, \alpha)$. Assume that $\mathsf{sys}(t) \in [\![\mathit{after}(\bigwedge_{i\in I} \varphi_i, \alpha)]\!]$ from which by the definition of *after* we have that

$$\mathsf{sys}(t) \in [\![\textstyle\bigwedge_{i\in I} \mathit{after}(\varphi_i, \alpha)]\!] \;=\; \forall i \in I \cdot \mathsf{sys}(t) \in [\![\mathit{after}(\varphi_i, \alpha)]\!]. \tag{B.54}$$

Hence, knowing (B.54) 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 $\mathit{after}(\max X.\varphi, \alpha)$ *where* X *is guarded in* φ. Now let's assume that $\mathsf{sys}(t) \in [\![\mathit{after}(\max X.\varphi, \alpha)]\!]$ so that by the definition of *after* we infer that

$$\mathsf{sys}(t) \in [\![\mathit{after}(\varphi\{\max X.\varphi/X\}, \alpha)]\!] \tag{B.55}$$

and since by (B.55) 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 after$([\{p, c\}]\varphi, \alpha)$. Assume that

$$\mathsf{sys}(t) \in [\![\mathit{after}([\{p, c\}]\varphi, \alpha)]\!] \tag{B.56}$$

and consider the following two cases:

- $\mathsf{mtch}(p, \alpha) = \sigma$ and $c\sigma \Downarrow \mathsf{true}$: By (B.56) and the definition of *after* we have that

  $$\mathsf{sys}(t) \in [\![\varphi\sigma]\!]. \tag{B.57}$$

  Since $\mathsf{sys}(\alpha t)$ is a trace process that can only perform α and transition to $\mathsf{sys}(t)$, *i.e.,* $\mathsf{sys}(\alpha t) \stackrel{\alpha}{\Longrightarrow} \mathsf{sys}(t)$, and since from (B.57) 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 $\mathsf{sys}(\alpha t) \stackrel{\alpha}{\Longrightarrow} \mathsf{sys}(t)$ and that $\mathsf{mtch}(p, \alpha) = \mathsf{undef}$ or that $\exists\sigma \cdot c\sigma \Downarrow \mathsf{ff}$, by the definition of $[\![-]\!]$ we can immediately conclude that $\mathsf{sys}(\alpha t) \in [\![[\{p, c\}]\varphi]\!]$ as required. □

B.3 Missing proofs from Chapter 5

In this section we provide the proofs for the supporting lemmas that were omitted from the main text of Chapter 5. Specifically, we prove Proposition 5.3 and lemmas 5.1 to 5.4, 5.8, 5.10, 5.11 and 5.13 along with their auxiliary lemmas. At certain points we also refer to the τ-closure property of sHML, Proposition B.1, that was proven in [10].

Proposition B.1. if $s \xrightarrow{\tau} s'$ and $s \vDash \varphi$ then $s' \vDash \varphi$.

B.3.1 Proving Proposition 5.3

We must prove that for every state s and $\mathrm{sHML}_{\mathbf{nf}}$ formula φ if $s \in [\![\varphi]\!]$ then $(\!|\varphi|\!)[s] \sim s$.

Proof. To prove this proposition we show that relation $\mathcal{R} \stackrel{\text{def}}{=} \{(s, (\!|\varphi|\!)[s]) \mid s \vDash \varphi\}$ 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 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

$$s \xrightarrow{\mu} s' \tag{B.58}$$

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

- $\mu = \tau$*:* Since $\mu = \tau$, we can apply rule ıAsʏ on (B.58) and get that

  $$(\!|\mathsf{tt}|\!)[s] \xrightarrow{\tau} (\!|\mathsf{tt}|\!)[s'] \tag{B.59}$$

  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

  $$(s', (\!|\mathsf{tt}|\!)[s']) \in \mathcal{R} \tag{B.60}$$

  as required, which means that this case is done by (B.59) and (B.60).

- $\mu = \alpha$*:* Since id encodes the 'catch-all' monitor $\mathsf{rec}\, Y.\{(x)!(y)\}.Y + \{(x)?(y)\}.Y$, by rules ᴇRec and ᴇTrn we deduce that $\mathsf{id} \xrightarrow{\alpha \blacktriangleright \alpha} \mathsf{id}$. Hence, since $(\!|\mathsf{tt}|\!) = \mathsf{id}$, from (B.58) and by rule ıTrn we conclude that

  $$(\!|\mathsf{tt}|\!)[s] \xrightarrow{\alpha} (\!|\mathsf{tt}|\!)[s'] \tag{B.61}$$

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

  $$(s', (\!|\mathsf{tt}|\!)[s']) \in \mathcal{R} \tag{B.62}$$

  as required, and so this case is done by (B.61) and (B.62).

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

$$s \vDash \bigwedge_{i\in I} [\{p_i, c_i\}]\varphi_i \tag{B.63}$$

$$s \xrightarrow{\mu} s' \tag{B.64}$$

and so by the definition of $\vDash$ and (B.63) we have that for every index $i \in I$ and action $\beta \in \text{Act}$,

$$s \xRightarrow{\beta} s', \mathsf{mtch}(p_i, \beta) = \sigma \textit{ and } c_i\sigma \Downarrow \mathsf{true} \textit{ implies } s \vDash \bigwedge_{i\in I} [\{p_i, c_i\}]\varphi_i. \tag{B.65}$$

Since $\mu \in \{\tau, \alpha\}$, we must consider both possibilities for (B.64).

- $\mu = \tau$*:* Since $\mu = \tau$, we can apply rule ıAsy on (B.64) and obtain

  $$(\!| \bigwedge_{i\in I} [\{p_i, c_i\}]\varphi_i |\!)[s] \xrightarrow{\tau} (\!| \bigwedge_{i\in I} [\{p_i, c_i\}]\varphi_i |\!)[s'] \tag{B.66}$$

  as required. Since $\mu = \tau$, and since we know that sHML is τ-closed, from (B.63), (B.64) and Proposition B.1, 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

  $$(s', (\!| \bigwedge_{i\in I} [\{p_i, c_i\}]\varphi_i |\!)[s']) \in \mathcal{R} \tag{B.67}$$

  as required. This subcase is therefore done by (B.66) and (B.67).

- $\mu = \alpha$*:* Since $\mu = \alpha$, from (B.64) we know that

  $$s \xrightarrow{\alpha} s' \tag{B.68}$$

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

  $$(\!| \bigwedge_{i\in I} [\{p_i, c_i\}]\varphi_i |\!) = \mathsf{rec}\, Y.\Big(\sum_{i\in I} \begin{cases} \{p_i, c_i, \bullet\}.Y & \text{if } \varphi_i = \mathsf{ff} \\ \{p_i, c_i\}.(\!| \varphi_i |\!) & \text{otherwise} \end{cases}\Big). \tag{B.69}$$

  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.,* $\nexists j \in I \cdot \mathit{mtch}(p_j, \alpha) = \sigma$ and $c_j\sigma \Downarrow \mathit{true}$*):* Since none of the symbolic transformations in (B.69) 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 (B.68) we have that

$$(\!|\textstyle\bigwedge_{i\in I}[\{p_i,c_i\}]\varphi_i|\!)[s] \xrightarrow{\alpha} (\!|\,\mathsf{tt}\,|\!)[s'] \quad \text{(since id} = (\!|\,\mathsf{tt}\,|\!)\text{)} \tag{B.70}$$

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

$$(s', (\!|\,\mathsf{tt}\,|\!)[s']) \in \mathcal{R} \tag{B.71}$$

as required. This case is therefore done by (B.70) and (B.71).

– *One matching branch (i.e., $\exists j \in I \cdot$ mtch$(p_j,\alpha) = \sigma$ and $c_j\sigma \Downarrow$ true):* From (B.69) we infer that the synthesised monitor can only suppress actions that are defined by violating necessities. However, from (B.65) we also deduce that s is *incapable* of executing such an action as otherwise would contradict assumption (B.63). Hence, since we now assume that there exists some index $j \in I$ so that $\mathsf{mtch}(p_j,\alpha) = \sigma$ and $c_j\sigma \Downarrow \mathsf{true}$, from (B.69) we deduce that this action can only be transformed by an identity transformation and so by rule eTrn we have that

$$\{p_j,c_j\}.(\!|\,\varphi_j\,|\!) \xrightarrow{\alpha\blacktriangleright\alpha} (\!|\,\varphi_j\sigma\,|\!). \tag{B.72}$$

By applying rules eSel, eRec on (B.72) and by (B.68), (B.69) and iTrn we get that

$$(\!|\textstyle\bigwedge_{i\in I}[\{p_i,c_i\}]\varphi_i|\!)[s] \xrightarrow{\alpha} (\!|\,\varphi_j\sigma\,|\!)[s'] \tag{B.73}$$

as required. By (B.65), (B.68) and since we assume that there exists some index $j \in I$ so that $\mathsf{mtch}(p_j,\alpha) = \sigma$ and $c_j\sigma \Downarrow \mathsf{true}$, we have that $s' \vDash \varphi_j\sigma$, and so by the definition of $\mathcal{R}$ we conclude that

$$(s', (\!|\,\varphi_j\sigma\,|\!)[s']) \in \mathcal{R} \tag{B.74}$$

as required. Hence, this subcase is done by (B.73) and (B.74).

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

$$s \xrightarrow{\mu} s' \tag{B.75}$$

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

$$s \vDash \varphi\{\max X.\varphi/X\}. \tag{B.76}$$

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

$$\varphi = \max Y_0. \ldots \max Y_n. \bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i \tag{B.77}$$

and so by (B.76), (B.77) and the definition of $\vDash$ we have that

$$s \vDash (\textstyle\bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i)\{\cdot\cdot\} \quad \textit{where} \tag{B.78}$$

$$\{\cdot\cdot\} = \{\max X.\varphi/X, (\max Y_0. \ldots \max Y_n. \bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i)/Y_0, \ldots, (\max Y_n. \bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i)/Y_n\}.$$

Since we know (B.75) and (B.78), from this point onwards the proof proceeds as per the previous case. We thus omit this part of the proof and immediately deduce that

$$\exists m' \cdot (\!| (\textstyle\bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i)\{\cdot\cdot\} |\!)[s] \xrightarrow{\mu} (\!| m' |\!)[s'] \tag{B.79}$$

$$(s', (\!| m' |\!)[s']) \in \mathcal{R} \tag{B.80}$$

and so since $(\!| (\bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i)\{\cdot\cdot\} |\!)$ produces the *unfolded equivalent* of $(\!| \varphi\{\max X.\varphi/X\} |\!)$, from (B.79) we can conclude that

$$\exists m' \cdot (\!| \varphi\{\max X.\varphi/X\} |\!)[s] \xrightarrow{\mu} (\!| m' |\!)[s'] \tag{B.81}$$

as required, and so this case holds by (B.80) and (B.81).

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

$$(\!|\,\mathsf{tt}\,|\!)[s] \xrightarrow{\mu} r'. \tag{B.82}$$

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

- $\mu = \tau$*:* Since $\mu = \tau$, the transition in (B.82) can be performed either via iSup, or iAsy. We must therefore consider these cases.
  - *iAsy:* From rule iAsy and (B.82) 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 (B.82) 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 (B.82) 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 $n \xrightarrow{\alpha}$ and $n \xrightarrow{\bullet}$.
  - *iIns:* This case does not apply since from (B.82) and by the definition of $(\!|-|\!)$ we know that the synthesised monitor does not include action insertions.
  - *iTrn:* By applying rule iTrn on (B.82) we know that $r' = m'[s']$ such that

$$s \xrightarrow{\beta} s' \tag{B.83}$$

$$(\!|\,\mathsf{tt}\,|\!) \xrightarrow{\alpha \blacktriangleright \beta} m'. \tag{B.84}$$

    Since $(\!|\,\mathsf{tt}\,|\!) = \mathsf{id} = \mathsf{rec}\, Y.\{(x)!(y), \mathsf{true}, x!y\}.Y + \{(x)?(y), \mathsf{true}, x?y\}.Y$, by applying rules eRec, eSel and eTrn to (B.84) 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

$$(s', (\!|\, \mathsf{tt} \,|\!)[s']) \in \mathcal{R}. \tag{B.85}$$

Hence, we are done by (B.83) and (B.85) since we know that $\alpha = \beta$.

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

$$s \vDash \textstyle\bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i \tag{B.86}$$

$$(\!|\, \textstyle\bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i \,|\!)[s] \xrightarrow{\mu} r'. \tag{B.87}$$

From (B.86) and by the definition of $\vDash$ we can deduce that

$$\forall i \in I, \alpha \in \textsc{Act} \cdot s \stackrel{\alpha}{\Longrightarrow} s', \mathsf{mtch}(p_i, \alpha) = \sigma \textit{ and } c_i\sigma \Downarrow \mathsf{true} \textit{ implies } s' \vDash \varphi_i\sigma \tag{B.88}$$

and from (B.87) and by the definition of $(\!|-|\!)$ we have that

$$\Big(\mathsf{rec}\, Y.\sum_{i \in I} \begin{cases} \{p_i, c_i, \bullet\}.Y & \text{if } \varphi_i = \mathsf{ff} \\ \{p_i, c_i\}.(\!|\, \varphi_i \,|\!) & \text{otherwise} \end{cases}\Big)[s'] \xrightarrow{\mu} r'. \tag{B.89}$$

From (B.89) 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 (B.88) since $s' \vDash \mathsf{ff}$ does not hold. Hence, we can safely conclude that the synthesised monitor in (B.89) does *not* suppress any actions of s, and so we conclude that

$$\forall \alpha \in \textsc{Act}, s' \in \textsc{Sys} \cdot s \xrightarrow{\alpha} s' \textit{ implies } (\!|\, \textstyle\bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i \,|\!) \xrightarrow{\alpha \blacktriangleright \bullet} . \tag{B.90}$$

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

- $\mu = \tau$*:* Since $\mu = \tau$, from (B.87) we know that

$$(\!|\, \textstyle\bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i \,|\!)[s] \xrightarrow{\tau} r' \tag{B.91}$$

  The τ-transition in (B.91) can be the result of rules ıAsy or ıSup; we thus consider each eventuality.

  - *ıAsy:* As we assume that the reduction in (B.91) is the result of rule ıAsy,

we know that $r' = (\!| \bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i |\!)[s']$ and that

$$s \xrightarrow{\tau} s' \tag{B.92}$$

as required. Also, since sHML is τ-closed, by (B.86), (B.92) and Proposition B.1 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

$$(s', (\!| \bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i |\!)[s']) \in \mathcal{R} \tag{B.93}$$

and so we are done by (B.92) and (B.93).

– *iSup:* As we now assume that the reduction in (B.91) results from iSup, we have that $r' = m'[s']$ and that

$$s \xrightarrow{\alpha} s' \tag{B.94}$$

$$(\!| \bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i |\!) \xrightarrow{\alpha \blacktriangleright \bullet} m'. \tag{B.95}$$

This case does not apply since by (B.90) and (B.94) we can deduce that $(\!| \bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i |\!) \not\xrightarrow{\alpha \blacktriangleright \bullet}$ which contradicts with (B.95).

• $\mu = \alpha$*:* When $\mu = \alpha$, the transition in (B.89) can be performed via rules iDef, iIns or iTrn, we consider both possibilities.

– *iDef:* If (B.89) results from iDef, we have that

$$r' = (\!| \mathsf{tt} |\!)[s'] \quad (\text{since } (\!| \mathsf{tt} |\!) = \mathsf{id}) \tag{B.96}$$

$$s \xrightarrow{\alpha} s'. \tag{B.97}$$

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 (B.96) we can conclude that

$$(s', r') \in \mathcal{R} \tag{B.98}$$

as required. Hence this case is done by (B.97) and (B.98).

– *iIns:* This case does not apply since from (B.89) and by the definition of $(\!| - |\!)$

we know that the synthesised monitor does not include action insertions.

– *ITRN:* By assuming that (B.89) is obtained from rule ITRN we know that

$$(\mathsf{rec}\, Y.\sum_{i\in I} \begin{cases} \{p_i, c_i, \bullet\}.Y & \text{if } \varphi_i = \mathsf{ff} \\ \{p_i, c_i\}.(\!|\, \varphi_i \,|\!) & \text{otherwise} \end{cases}) \xrightarrow{\beta \blacktriangleright \alpha} m' \tag{B.99}$$

$$s \xrightarrow{\beta} s' \tag{B.100}$$

$$r' = m'[s']. \tag{B.101}$$

Since from (B.90) we know that the synthesised monitor in (B.99) 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

$$\alpha = \beta. \tag{B.102}$$

With the knowledge of (B.102), from (B.100) we can thus deduce that

$$s \xrightarrow{\alpha} s' \tag{B.103}$$

as required. Knowing (B.102) we can also deduce that in (B.99) 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 (B.99) we deduce that

$$\exists j \in I \cdot \mathsf{mtch}(p_j, \alpha) = \sigma \textit{ and } c_j\sigma \Downarrow \mathsf{true} \tag{B.104}$$

$$m' = (\!|\, \varphi_j\sigma \,|\!). \tag{B.105}$$

and so from (B.103), (B.104) and (B.88) 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 (B.101) and (B.105) we can conclude that

$$(s', r') \in \mathcal{R} \tag{B.106}$$

as required, and so this case is done by (B.103) and (B.106).

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

$$(\!|\max X.\varphi|\!)[s] \xrightarrow{\mu} r' \tag{B.107}$$

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

$$s \vDash \varphi\{\max X.\varphi/X\}. \tag{B.108}$$

Since $\varphi\{\max X.\varphi/X\} \in \text{sHML}_{\mathbf{nf}}$, by the restrictions imposed by $\text{sHML}_{\mathbf{nf}}$ we know that: φ cannot be X because (bound) logical variables are required to be *guarded*, and it also cannot be $\mathtt{tt}$ or $\mathtt{ff}$ 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

$$\varphi = \max Y_0 \ldots \max Y_n.\bigwedge_{i\in I} [\{p_i, c_i\}]\varphi_i \tag{B.109}$$

and so by (B.108), (B.109) and the definition of $\vDash$ we have that

$$s \vDash (\textstyle\bigwedge_{i\in I} [\{p_i, c_i\}]\varphi_i)\{\cdot\cdot\} \quad \textit{where} \tag{B.110}$$

$$\{\cdot\cdot\} = \{\max X.\varphi/X, (\max Y_0 \ldots \max Y_n.\bigwedge_{i\in I} [\{p_i, c_i\}]\varphi_i)/Y_0, \ldots, (\max Y_n.\bigwedge_{i\in I} [\{p_i, c_i\}]\varphi_i)/Y_n\}.$$

Since $(\!|(\bigwedge_{i\in I} [\{p_i, c_i\}]\varphi_i)\{\cdot\cdot\}|\!)$ synthesises the *unfolded equivalent* of $(\!|\max X.\varphi|\!)$, from (B.107) we know that

$$(\!|(\textstyle\bigwedge_{i\in I} [\{p_i, c_i\}]\varphi_i)\{\cdot\cdot\}|\!)[s] \xrightarrow{\mu} r'. \tag{B.111}$$

Hence, since we know (B.110) and (B.111), from this point onwards the proof proceeds as per the previous case. We thus omit showing the remainder of this proof.

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

B.3.2 Proving Lemma 5.1

We now prove that for every formula $\varphi \in \text{sHML}_{\mathbf{nf}}$, if we assume that $(\!|\varphi|\!)[s] \stackrel{t}{\Longrightarrow} m'[s']$ then there must exist $\text{sHML}_{\mathbf{nf}}$ some formula ψ, such that $\psi = \textit{after}(\varphi, t)$ and $(\!|\psi|\!) = m'$. This proof relies on the following lemma whose proof is given in **??** B.3.2.1.

Lemma B.1. For every formula of the form $\bigwedge_{i\in I}[\{p_i, c_i\}]\varphi_i$ and system states s and r, if $(\!|\bigwedge_{i\in I}[\{p_i, c_i\}]\varphi_i|\!)[s] \xrightarrow{\tau}{}^{*} r$ then $\exists s', u \cdot s \overset{u}{\Longrightarrow} s'$ and $r = (\!|\bigwedge_{i\in I}[\{p_i, c_i\}]\varphi_i|\!)[s']$.

Proof. We proceed by induction on the length of t.

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

Case $t = \alpha u$. Assume that $(\!|\varphi|\!)[s] \overset{\alpha u}{\Longrightarrow} m'[s']$ from which by the definition $\overset{t}{\Longrightarrow}$ we can infer that

$$(\!|\varphi|\!)[s] \xrightarrow{\tau}{}^{*} r \tag{B.112}$$

$$r \xrightarrow{\alpha} r' \tag{B.113}$$

$$r' \overset{u}{\Longrightarrow} m'[s']. \tag{B.114}$$

We now proceed by case analysis on φ.

- $\varphi \in \{\mathsf{ff}, X\}$: These cases do not apply since $(\!|\mathsf{ff}|\!)$ and $(\!|X|\!)$ do not yield a valid monitor.
- $\varphi = \mathsf{tt}$: Since $(\!|\mathsf{tt}|\!) = \mathsf{id}$ we know that the τ-reductions in (B.112) are only possible via rule ıAsy which means that $s \xrightarrow{\tau}{}^{*} s''$ and $r = (\!|\mathsf{tt}|\!)[s'']$. The latter allows us to deduce that the reduction in (B.113) is only possible via rule ıTrn and so we also know that $s'' \xrightarrow{\alpha}{}^{*} s'''$ and $r' = (\!|\mathsf{tt}|\!)[s''']$. Hence, by (B.114) and the *inductive hypothesis* we conclude that

  $$\exists \psi \in \mathrm{sHML}_{\mathbf{nf}} \cdot \psi = \mathit{after}(\mathsf{tt}, u) \tag{B.115}$$

  $$(\!|\psi|\!) = m'. \tag{B.116}$$

  As by the definition of *after* we know that $\mathit{after}(\mathsf{tt}, \alpha u) = \mathit{after}(\mathit{after}(\mathsf{tt}, \alpha), u)$ and $\mathit{after}(\mathsf{tt}, \alpha) = \mathsf{tt}$, from (B.115) we can conclude that

  $$\exists \psi \in \mathrm{sHML}_{\mathbf{nf}} \cdot \psi = \mathit{after}(\mathsf{tt}, \alpha u) \tag{B.117}$$

  and so this case holds by (B.116) and (B.117).
- $\varphi = \bigwedge_{i\in I}[\{p_i, c_i\}]\varphi_i$ and $\#_{i\in I}\{p_i, c_i\}$: Since $\varphi = \bigwedge_{i\in I}[\{p_i, c_i\}]\varphi_i$, by the definition of

$(\!|-|\!)$ we know that

$$\begin{array}{rcl}(\!|\bigwedge_{i\in I}[\{p_i,c_i\}]\varphi_i|\!) & = & \mathsf{rec}\,Y.\sum_{i\in I}\begin{cases}\{p_i,c_i,\bullet\}.Y & \textit{if } \varphi_i = \mathsf{ff}\\ \{p_i,c_i\}.(\!|\varphi_i|\!) & \textit{otherwise}\end{cases}\\ & = & \sum_{i\in I}\begin{cases}\{p_i,c_i,\bullet\}.(\!|\bigwedge_{i\in I}[\{p_i,c_i\}]\varphi_i|\!) & \textit{if } \varphi_i = \mathsf{ff}\\ \{p_i,c_i\}.(\!|\varphi_i|\!) & \textit{otherwise}\end{cases}\end{array} \tag{B.118}$$

and so by (B.112), (B.118) and Lemma B.1 we conclude that $\exists s'' \cdot s \xRightarrow{u} s''$ and

$$r = (\!|\bigwedge_{i\in I}[\{p_i,c_i\}]\varphi_i|\!)[s'']. \tag{B.119}$$

Hence, by (B.118) and (B.119) we know that the reduction in (B.113) can only happen if $\exists s''' \cdot s'' \xrightarrow{\alpha} s'''$ and α matches an identity transformation $\{p_j,c_j\}.(\!|\varphi_j|\!)$ (for some $j \in I$) which was derived from $[\{p_j,c_j\}]\varphi_j$ ($\neq \mathsf{ff}$). Hence we can deduce that

$$r' = (\!|\varphi_j\sigma|\!)[s'''] \tag{B.120}$$

$$\mathsf{mtch}(p_j,\alpha) = \sigma \textit{ and } c_j\sigma \Downarrow \mathsf{true} \tag{B.121}$$

and so by (B.114), (B.120) and the *inductive hypothesis* we deduce that

$$\exists \psi \in \text{sHML}_{\mathbf{nf}} \cdot \psi = \textit{after}(\varphi_j\sigma, u) \tag{B.122}$$

$$(\!|\psi|\!) = m'. \tag{B.123}$$

Now since we know (B.121), by the definition of *after* we infer that

$$\textit{after}(\textstyle\bigwedge_{i\in I}[\{p_i,c_i\}]\varphi_i, \alpha u) = \textit{after}(\textit{after}(\textstyle\bigwedge_{i\in I}[\{p_i,c_i\}]\varphi_i, \alpha), u) = \textit{after}(\varphi_j\sigma, u) \tag{B.124}$$

and so from (B.122) and (B.124) we conclude that

$$\exists \psi \in \text{sHML}_{\mathbf{nf}} \cdot \psi = \textit{after}(\textstyle\bigwedge_{i\in I}[\{p_i,c_i\}]\varphi_i, \alpha u). \tag{B.125}$$

Hence, this case is done by (B.123) and (B.125).

- $\varphi = \mathsf{max}\,X.\psi$ and $X \in \mathbf{fv}(\psi)$: Since $\varphi = \mathsf{max}\,X.\psi$, by the syntactic rules of $\text{sHML}_{\mathbf{nf}}$ we know that $\psi \notin \{\mathsf{ff},\mathsf{tt}\}$ since $X \notin \mathbf{fv}(\psi)$, and that $\psi \neq X$ since logical variables

must be guarded, hence we know that ψ can only be of the form

$$\psi = \max Y_1 \ldots . \max Y_n . \bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i. \tag{B.126}$$

where $\max Y_1 \ldots . \max Y_n .$ denotes an arbitrary number of fixpoint declarations, possibly none. Hence, knowing (B.126), by unfolding every fixpoint in $\max X.\psi$ we reduce the formula to $\bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i\{^{\max X.\max Y_1 \ldots . \max Y_n .} \bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i /_X, \ldots\}$. This implies that from this point onwards, the proof proceeds as per that of case $\varphi = \bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i$ which allows us to deduce that

$$\exists \psi' \in \text{sHML}_{\textbf{nf}} \cdot \psi' = \mathit{after}(\textstyle\bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i\{\ldots\}, \alpha u) \tag{B.127}$$

$$(\!| \psi' |\!) = m'. \tag{B.128}$$

From (B.126), (B.127) and the definition of *after* we can thus conclude that

$$\exists \psi' \in \text{sHML}_{\textbf{nf}} \cdot \psi' = \mathit{after}(\max X.\psi, \alpha u) \tag{B.129}$$

and so this case holds by (B.128) and (B.129).

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

B.3.2.1 Proving Lemma B.1

To prove this lemma we must show that for every formula of the form $\bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i$ and system states s and r, if $(\!| \bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i |\!)[s] \xrightarrow{\tau}{}^* r$ then there must exist some state s' and trace u such that $s \stackrel{u}{\Longrightarrow} s'$ and $r = (\!| \bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i |\!)[s']$.

Proof. The proof proceeds by mathematical induction on the number of τ transitions.

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

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

$$(\!| \textstyle\bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i |\!)[s] \xrightarrow{\tau} r' \quad \text{(for some } r\text{')} \tag{B.130}$$

$$r' \xrightarrow{\tau}{}^{k} r. \tag{B.131}$$

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

$$(\!|\bigwedge_{i\in I}[\{p_i,c_i\}]\varphi_i|\!) = \sum_{i\in I}\begin{cases}\{p_i,c_i,\bullet\}.(\!|\bigwedge_{i\in I}[\{p_i,c_i\}]\varphi_i|\!) & \textit{if } \varphi{=}\mathsf{ff}\\ \{p_i,c_i\}.(\!|\varphi_i|\!) & \textit{otherwise}\end{cases} \tag{B.132}$$

and so from (B.132) we know that the τ-reduction in (B.130) can be the result of rule iAsy or iSup. We therefore inspect both cases.

- iAsy: By rule iAsy, from (B.130) we can deduce that

$$\exists s''\cdot s \xrightarrow{\tau} s'' \tag{B.133}$$

$$r' = (\!|\bigwedge_{i\in I}[\{p_i,c_i\}]\varphi_i|\!)[s''] \tag{B.134}$$

  and so by (B.131), (B.134) and the *inductive hypothesis* we know that

$$\exists s', u\cdot s'' \xRightarrow{u} s' \textit{ and } r = (\!|\bigwedge_{i\in I}[\{p_i,c_i\}]\varphi_i|\!)[s']. \tag{B.135}$$

  Hence, by (B.133) and (B.135) we conclude that there exists a system state s' and trace u so that $s \xRightarrow{u} s'$ and $r = (\!|\bigwedge_{i\in I}[\{p_i,c_i\}]\varphi_i|\!)[s']$.

- iSup: By rule iSup and from (B.130) we infer that

$$\exists s''\cdot s \xrightarrow{\alpha} s'' \tag{B.136}$$

$$(\!|\bigwedge_{i\in I}[\{p_i,c_i\}]\varphi_i|\!) \xrightarrow{\alpha\blacktriangleright\bullet} m' \tag{B.137}$$

$$r' = m'[s''] \tag{B.138}$$

  and from (B.132) we know that the reduction in (B.137) occurs when α matches a suppression transformation which then reduces back to $(\!|\bigwedge_{i\in I}[\{p_i,c_i\}]\varphi_i|\!)$ allowing us to infer that

$$m' = (\!|\bigwedge_{i\in I}[\{p_i,c_i\}]\varphi_i|\!). \tag{B.139}$$

  Hence, by (B.131), (B.138) and (B.139) we can apply the *inductive hypothesis* and deduce that

$$\exists s', u\cdot s'' \xRightarrow{u} s' \textit{ and } r = (\!|\bigwedge_{i\in I}[\{p_i,c_i\}]\varphi_i|\!)[s'] \tag{B.140}$$

so that by (B.136) and (B.140) we finally conclude that $\exists s', u \cdot s \xRightarrow{\alpha u} s'$ and that $r = (\!| \bigwedge_{i \in I} [\{p_i, c_i\}] \varphi_i |\!)[s']$ as required, and so we are done. □

B.3.3 Proving Lemma 5.2

This lemma proves that for every suppression monitor $m \in \textsc{SupTrn}$ and explicit trace t_τ, $\mathit{mc}(m, t_\tau) = N$.

Proof. The proof proceeds by induction on the length of t_τ.

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

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

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

- $\forall \mu \cdot m[\mathsf{sys}(\mu t'_\tau)] \xrightarrow{\mu}$: This case does not apply since rule iDef prevents the monitor from blocking the composite system.
- $\exists \mu', m', s \cdot m[\mathsf{sys}(\mu t'_\tau)] \xrightarrow{\mu'} m'[s]$: When considering only suppression monitors, by the rules in our model we can infer that this instrumented reduction over action μ' can be attained via rules iDef, iAsy, iSup and iTrn. We thus consider each case.
  - iSup: Since by rule iSup we know that $\mu = \alpha$, $\mu' = \tau$ and $s = \mathsf{sys}(t'_\tau)$, by the definition of *mc* we deduce that $\mathit{mc}(m, \alpha t'_\tau) = \mathit{mc}(m', t'_\tau) + 1$ and since by the *inductive hypothesis* we know that $\mathit{mc}(m', t'_\tau) = N$, then we conclude that $\mathit{mc}(m, \alpha t'_\tau) = N + 1$ as required.
  - iDef: Since by rule iDef we know that $\mu = \mu' = \alpha$, $m' = \mathsf{id}$ and $s = \mathsf{sys}(t'_\tau)$, by the definition of *mc* we deduce that $\mathit{mc}(m, \alpha t'_\tau) = \mathit{mc}(\mathsf{id}, t'_\tau)$ and since by the *inductive hypothesis* we know that $\mathit{mc}(\mathsf{id}, t'_\tau) = N$, then we can conclude that $\mathit{mc}(m, \alpha t'_\tau) = N$.

– iAsy and iTrn: We omit the proofs for these cases as they are very similar to that of case iDef, and so we are done. □

B.3.4 Proving Lemma 5.3

The aim of this proof is to show that for every action α and suppression monitors $m, m' \in \text{SupTrn}$, if $\mathsf{enf}(m, \bigwedge_{i\in I}[\{p_i, c_i\}]\varphi_i)$, $m \xrightarrow{\alpha\blacktriangleright\alpha} m'$, $\mathsf{mtch}(p_i, \alpha) = \sigma$ and $c_i\sigma \Downarrow \mathsf{true}$ (for some $j \in I$) then $\mathsf{enf}(m', \varphi_j\sigma)$.

Proof. We start this proof by assuming that

$$m \xrightarrow{\alpha\blacktriangleright\alpha} m' \tag{B.141}$$

$$\exists j \in I \cdot \mathsf{mtch}(p_i, \alpha) = \sigma \text{ and } c_i\sigma \Downarrow \mathsf{true}. \tag{B.142}$$

and also that $\mathsf{enf}(m, \bigwedge_{i\in I}[\{p_i, c_i\}]\varphi_i)$ from which we can infer that

$$\mathsf{senf}(m, \textstyle\bigwedge_{i\in I}[\{p_i, c_i\}]\varphi_i) \stackrel{\text{def}}{=} \forall s \cdot m[s] \vDash \bigwedge_{i\in I}[\{p_i, c_i\}]\varphi_i \tag{B.143}$$

$$\mathsf{evtenf}(m, \textstyle\bigwedge_{i\in I}[\{p_i, c_i\}]\varphi_i) \stackrel{\text{def}}{=} \forall s, s'', t \cdot \textit{if } m[s] \stackrel{t}{\Longrightarrow} m''[s''] \textit{ and } s'' \vDash \textit{after}(\textstyle\bigwedge_{i\in I}[\{p_i, c_i\}]\varphi_i, t)$$
$$\textit{then } m''[s''] \sim s''. \tag{B.144}$$

Since both (B.143) and (B.144) quantify on every s, we must consider the following two cases, namely, when $m[s]$ transitions over α and reach m', *i.e.*, $m[s] \xrightarrow{\alpha} m'[s']$ (for some system state s'), and when $m[s]$ does not reach m' over α, *i.e.*, $m[s] \stackrel{\alpha}{\not\longrightarrow} m'[s']$.

- $m[s] \stackrel{\alpha}{\not\longrightarrow} m'[s']$: This case does not apply since, as stated by assumption (B.141), we only consider the cases where the instrumented system causes the monitor to perform the identity transformation of (B.141) via rule iTrn.
- $m[s] \xrightarrow{\alpha} m'[s']$: Since $m[s] \xrightarrow{\alpha} m'[s']$, from (B.143), (B.142) and by the definition of $\vDash$ we get that

  $$\mathsf{senf}(m', \varphi_j\sigma) \stackrel{\text{def}}{=} \forall s' \cdot m'[s'] \vDash \varphi_j\sigma \tag{B.145}$$

  as required. Now, lets assume that

  $$\forall s''', u \cdot m'[s'] \stackrel{u}{\Longrightarrow} m'''[s'''] \tag{B.146}$$

  $$s''' \vDash \textit{after}(\varphi_j\sigma, u) \tag{B.147}$$

  and since $m[s] \xrightarrow{\alpha} m'[s']$ when combined with (B.146) we know that $m[s] \stackrel{\alpha u}{\Longrightarrow} m'''[s''']$ and so from (B.144) and (B.147) we can deduce that

  $$m'''[s'''] \sim s'''. \tag{B.148}$$

Hence, from assumptions (B.146), (B.147) and conclusion (B.148) we can introduce the implication and conclude that

$$\mathsf{evtenf}(m', \varphi_j\sigma) \stackrel{\text{def}}{=} \forall s', s'', u \cdot \textit{ if } m'[s'] \stackrel{u}{\Longrightarrow} m'''[s'''] \textit{ and } s''' \vDash \textit{after}(\varphi_j\sigma, u) \textit{ then } m'''[s'''] \sim s'''. \tag{B.149}$$

Therefore, by (B.145) and (B.149) we can finally conclude that $\mathsf{enf}(m', \varphi_j\sigma)$ holds as required, and so we are done.

□

B.3.5 Proving Lemma 5.4

We now show that for every action α and suppression monitors $m, m' \in \textsc{SupTrn}$, if $\mathsf{enf}(m, \bigwedge_{i\in I} [\{p_i, c_i\}]\varphi_i)$ and $m \xrightarrow{\alpha\blacktriangleright\bullet} m'$ then $\mathsf{enf}(m', \bigwedge_{i\in I} [\{p_i, c_i\}]\varphi_i)$.

Proof. We start this proof by assuming that

$$m \xrightarrow{\alpha\blacktriangleright\bullet} m' \tag{B.150}$$

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

$$\mathsf{senf}(m, \bigwedge_{i\in I} [\{p_i, c_i\}]\varphi_i) \stackrel{\text{def}}{=} \forall s \cdot m[s] \vDash \bigwedge_{i\in I} [\{p_i, c_i\}]\varphi_i \tag{B.151}$$

$$\mathsf{evtenf}(m, \bigwedge_{i\in I} [\{p_i, c_i\}]\varphi_i) \stackrel{\text{def}}{=} \forall s, s'', t \cdot \textit{ if } m[s] \stackrel{t}{\Longrightarrow} m''[s''] \textit{ and } s'' \vDash \textit{after}(\bigwedge_{i\in I} [\{p_i, c_i\}]\varphi_i, t) \textit{ then } m''[s''] \sim s''. \tag{B.152}$$

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

- $m[s] \stackrel{\tau}{\not\longrightarrow} m'[s']$: This case does not apply since, as stated by assumption (B.150), we only consider the cases where the instrumented system causes the monitor to perform the suppression transformation of (B.150) via rule ıSup.
- $m[s] \xrightarrow{\tau} m'[s']$: Since $m[s] \xrightarrow{\tau} m'[s']$, from (B.151) and by Proposition B.1 we deduce that

$$\mathsf{senf}(m', \bigwedge_{i\in I} [\{p_i, c_i\}]\varphi_i) \stackrel{\text{def}}{=} \forall s' \cdot m'[s'] \vDash \bigwedge_{i\in I} [\{p_i, c_i\}]\varphi_i \tag{B.153}$$

as required. We now assume that

$$\forall s''', u \cdot m'[s'] \stackrel{u}{\Longrightarrow} m'''[s'''] \tag{B.154}$$

$$s''' \vDash \mathit{after}(\textstyle\bigwedge_{i\in I} [\{p_i, c_i\}]\varphi_i, u) \tag{B.155}$$

and since $m[s] \xrightarrow{\tau} m'[s']$, by (B.154) and the definition of $\stackrel{u}{\Longrightarrow}$ we have that $m[s] \xrightarrow{u} m'''[s''']$ and so from (B.152) and (B.155) we can deduce that

$$m'''[s'''] \sim s'''. \tag{B.156}$$

Hence, from assumptions (B.154), (B.155) and conclusion (B.156) we can introduce the implication and conclude that

$$\begin{aligned}\mathsf{evtenf}(m', \textstyle\bigwedge_{i\in I} [\{p_i, c_i\}]\varphi_i) \stackrel{\text{def}}{=} \forall s', s'', u \cdot \mathit{if}\ m'[s'] \stackrel{u}{\Longrightarrow} m'''[s'''] \ \mathit{and}\\ s''' \vDash \mathit{after}(\textstyle\bigwedge_{i\in I} [\{p_i, c_i\}]\varphi_i, u)\ \mathit{then}\ m'''[s'''] \sim s'''.\end{aligned} \tag{B.157}$$

Hence, by (B.153) and (B.157) we can finally conclude that $\mathsf{enf}(m', \bigwedge_{i\in I} [\{p_i, c_i\}]\varphi_i)$ as required, and so we are done. □

B.3.6 Proving Lemma 5.8

To prove this lemma we must show that for every $\varphi \in \text{sHML}^{\#}$, $[\![\langle\!\langle \varphi \rangle\!\rangle_4]\!] = [\![\varphi]\!]$. To simplify the proof we instead prove the following two results:

(a) $\forall s \in \text{Sys},\ \mathit{if}\ s \vDash \langle\!\langle \varphi \rangle\!\rangle_4\ \mathit{then}\ s \vDash \varphi$; and that

(b) $\forall s \in \text{Sys},\ \mathit{if}\ s \vDash \varphi\ \mathit{then}\ s \vDash \langle\!\langle \varphi \rangle\!\rangle_4$.

The proofs for (a) and (b) also rely on the following lemmas which we prove in **??** B.3.6.1**??** B.3.6.2 respectively.

Lemma B.2. For all $\varphi \in \text{sHML}^{\#}$, if $X \in \mathsf{fvars}(\varphi)$ then $X \in \mathsf{fvars}(\langle\!\langle \varphi \rangle\!\rangle_4)$.

Lemma B.3. For all $\varphi \in \text{sHML}^{\#}$, if $X \in \mathsf{fvars}(\varphi)$ *and* $X \in \mathsf{fvars}(\langle\!\langle \psi \rangle\!\rangle_4)$ then $\langle\!\langle \varphi\{^{\max X.\psi}/_X\} \rangle\!\rangle_4$ $= \langle\!\langle \varphi \rangle\!\rangle_4\{^{\max X.\langle\!\langle \psi \rangle\!\rangle_4}/_X\}$.

Proof for (a). Let $\mathcal{R} \stackrel{\text{def}}{=} \{\ (s, \varphi) \mid s \vDash \langle\!\langle \varphi \rangle\!\rangle_4\ \}$, we must prove that $\mathcal{R}$ is a satisfaction relation by showing that it obeys the rules of Figure 5.1. We conduct this proof by case analysis on φ.

Cases $\varphi \in \{\mathsf{ff}, X\}$. These cases do not apply since $\langle\!\langle \varphi \rangle\!\rangle_4 = \varphi$ and so the assumption that $s \vDash \langle\!\langle \varphi \rangle\!\rangle_4$ does not hold when $\varphi \in \{\mathsf{ff}, X\}$.

Case $\varphi = \mathtt{tt}$. This case is satisfied trivially since any process satisfies $\mathtt{tt}$ which confirms that $(s, \mathtt{tt}) \in \mathcal{R}$.

Case $\varphi = \bigwedge_{i\in I} [\{p_i, c_i\}]\varphi_i$. In this case we must prove that $(s, \bigwedge_{i\in I} [\{p_i, c_i\}]\varphi_i) \in \mathcal{R}$ by showing that for every α and $i \in I$, if $s \stackrel{\alpha}{\Longrightarrow} s'$ so that $\mathsf{mtch}(p_i, \alpha) = \sigma$ and $c_i\sigma \Downarrow \mathsf{true}$ then $(s', \langle\!\langle \varphi_i\sigma \rangle\!\rangle_4) \in \mathcal{R}$. Hence we assume that $s \vDash \langle\!\langle \bigwedge_{i\in I} [\{p_i, c_i\}]\varphi_i \rangle\!\rangle_4$ and since by the definition of $\langle\!\langle - \rangle\!\rangle_4$ we know that $s \vDash \bigwedge_{i\,\in I} [\{p_i, c_i\}]\langle\!\langle \varphi_i \rangle\!\rangle_4$ then by the definition of $\vDash$ we have that

$$\forall i \in I, \alpha \in \textsc{Act} \cdot \ \textit{if}\ s \stackrel{\alpha}{\Longrightarrow} s'\ \textit{s.t.}\ \mathsf{mtch}(p_i, \alpha) = \sigma \text{ and } c_i\sigma \Downarrow \mathsf{true} \text{ then } s' \vDash \langle\!\langle \varphi_i\sigma \rangle\!\rangle_4. \quad \text{(B.158)}$$

Hence by (B.158) and the definition of $\mathcal{R}$ we can finally conclude that

$$\forall i \in I, \alpha \in \textsc{Act} \cdot \ \textit{if}\ s \stackrel{\alpha}{\Longrightarrow} s'\ \textit{s.t.}\ \mathsf{mtch}(p_i, \alpha) = \sigma \text{ and } c_i\sigma \Downarrow \mathsf{true} \text{ then } (s', \varphi_i\sigma) \in \mathcal{R}$$

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

$$s \vDash \langle\!\langle \mathsf{max}\, X.\varphi \rangle\!\rangle_4 \quad \text{(B.159)}$$

and consider the following two subcases for $\langle\!\langle \mathsf{max}\, X.\varphi \rangle\!\rangle_4$.

- when $X \in \mathsf{fvars}(\varphi)$: Since $X \in \mathsf{fvars}(\varphi)$, from (B.159) and the definition of $\langle\!\langle - \rangle\!\rangle_4$ we have that $s \vDash \mathsf{max}\, X.\langle\!\langle \varphi \rangle\!\rangle_4$ and so by the definition of $\vDash$ we can deduce that

$$s \vDash \langle\!\langle \varphi \rangle\!\rangle_4 \{^{\mathsf{max}\, X.\langle\!\langle \varphi \rangle\!\rangle_4}/_X\}. \quad \text{(B.160)}$$

  Since $X \in \mathsf{fvars}(\varphi)$ and by Lemma B.2 we have that $X \in \mathsf{fvars}(\langle\!\langle \varphi \rangle\!\rangle_4)$ and therefore by Lemma B.3, from (B.160) we deduce that

$$s \vDash \langle\!\langle \varphi\{\mathsf{max}\, X.\varphi/X\} \rangle\!\rangle_4. \quad \text{(B.161)}$$

  Hence, by (B.161) and the definition of $\mathcal{R}$ we deduce that

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

as required.

- $X \notin \mathsf{fvars}(\varphi)$: Since $X \notin \mathsf{fvars}(\varphi)$, we know that the maximal fixpoint in (B.159) is redundant, and so from (B.159) and the definition of $\langle\!\langle - \rangle\!\rangle_4$ we have that

$$s \vDash \langle\!\langle \varphi \rangle\!\rangle_4. \tag{B.162}$$

Since $X \notin \mathsf{fvars}(\varphi)$, from (B.162) we infer that $\langle\!\langle \varphi \rangle\!\rangle_4$ is logically equivalent to $\langle\!\langle \varphi\{\mathsf{max}\, X.\varphi/X\} \rangle\!\rangle_4$ since X is unused in φ which means that from (B.162) we can deduce that

$$s \vDash \langle\!\langle \varphi\{\mathsf{max}\, X.\varphi/X\} \rangle\!\rangle_4. \tag{B.163}$$

Hence from (B.163) 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 $\mathcal{R} \stackrel{\text{def}}{=} \{\, (s, \langle\!\langle \varphi \rangle\!\rangle_4) \mid s \vDash \varphi \,\}$, 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 case holds trivially since $\langle\!\langle \mathsf{tt} \rangle\!\rangle_4 = \mathsf{tt}$ and since all systems satisfy tt. Hence, we affirm that $(s, \langle\!\langle \mathsf{tt} \rangle\!\rangle_4) \in \mathcal{R}$.

Case $\varphi = \bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i$. To prove this case we confirm that $(s, \langle\!\langle \bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i \rangle\!\rangle_4) \in \mathcal{R}$. Since $\langle\!\langle \bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i \rangle\!\rangle_4 = \bigwedge_{i \in I} [\{p_i, c_i\}]\langle\!\langle \varphi_i \rangle\!\rangle_4$, we instead prove that $(s, \bigwedge_{i \in I} [\{p_i, c_i\}]\langle\!\langle \varphi_i \rangle\!\rangle_4) \in \mathcal{R}$ by showing that for every α and $i \in I$, if $s \stackrel{\alpha}{\Longrightarrow} s'$ *s.t.* $\mathsf{mtch}(p_i, \alpha) = \sigma$ and $c_i\sigma \Downarrow \mathsf{true}$ then $(s', \langle\!\langle \varphi_i\sigma \rangle\!\rangle_4) \in \mathcal{R}$. Hence we start by assuming that $s \vDash \bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i$ and so by the definition of $\vDash$ we have that

$$\forall i \in I, \alpha \in \text{Act} \cdot \ \textit{if}\ s \stackrel{\alpha}{\Longrightarrow} s' \ \textit{s.t.}\ \mathsf{mtch}(p_i, \alpha) = \sigma \text{ and } c_i\sigma \Downarrow \mathsf{true} \text{ then } s' \vDash \varphi_i\sigma \tag{B.164}$$

and so by (B.164) and the definition of $\mathcal{R}$ we conclude that

$$\forall i \in I, \alpha \in \text{Act} \cdot \ \textit{if}\ s \xRightarrow{\alpha} s'\ \textit{s.t.}\ \mathsf{mtch}(p_i, \alpha) = \sigma \text{ and } c_i\sigma \Downarrow \mathsf{true} \text{ then } (s', \langle\!\langle \varphi_i\sigma \rangle\!\rangle_4) \in \mathcal{R}$$

as required.

Case $\varphi = \max X.\varphi$. To prove this case we must confirm that $(s, \langle\!\langle \max X.\varphi \rangle\!\rangle_4) \in \mathcal{R}$ and so we start by assuming that $s \vDash \max X.\varphi$ from which by the definitions of $\vDash$ and $\mathcal{R}$ we deduce that

$$(s, \langle\!\langle \varphi\{\max X.\varphi/X\} \rangle\!\rangle_4) \in \mathcal{R}. \tag{B.165}$$

We now consider two subcases for $\langle\!\langle \max X.\varphi \rangle\!\rangle_4$.

- $\langle\!\langle \max X.\varphi \rangle\!\rangle_4 = \max X.\langle\!\langle \varphi \rangle\!\rangle_4$ when $X \in \mathsf{fvars}(\varphi)$: To confirm that $(s, \langle\!\langle \max X.\varphi \rangle\!\rangle_4) \in \mathcal{R}$, in this case we must affirm that $(s, \max X.\langle\!\langle \varphi \rangle\!\rangle_4) \in \mathcal{R}$. This can be achieved by showing that $(s, \langle\!\langle \varphi \rangle\!\rangle_4\{^{\max X.\langle\!\langle \varphi \rangle\!\rangle_4}/_X\}) \in \mathcal{R}$ as well. Hence, since we assume that $X \in \mathsf{fvars}(\varphi)$, by Lemma B.2 we deduce that $X \in \mathsf{fvars}(\langle\!\langle \varphi \rangle\!\rangle_4)$ and so by Lemma B.3 and from (B.165) we can conclude that $(s, \langle\!\langle \varphi \rangle\!\rangle_4\{^{\max X.\langle\!\langle \varphi \rangle\!\rangle_4}/_X\}) \in \mathcal{R}$ as required.
- $\langle\!\langle \max X.\varphi \rangle\!\rangle_4 = \langle\!\langle \varphi \rangle\!\rangle_4$ when $X \notin \mathsf{fvars}(\varphi)$: To confirm that $(s, \langle\!\langle \max X.\varphi \rangle\!\rangle_4) \in \mathcal{R}$, we must now affirm that $(s, \langle\!\langle \varphi \rangle\!\rangle_4) \in \mathcal{R}$. Since we now assume that $X \notin \mathsf{fvars}(\varphi)$, we know that $\varphi\{\max X.\varphi/X\} \equiv \varphi$ and so from (B.165) we confirm that $(s, \langle\!\langle \varphi \rangle\!\rangle_4) \in \mathcal{R}$ as required. □

B.3.6.1 Proving Lemma B.2

We now prove that for every $\varphi \in \text{sHML}^{\#}$, if $X \in \mathsf{fvars}(\varphi)$ then $X \in \mathsf{fvars}(\langle\!\langle \varphi \rangle\!\rangle_4)$.

Proof. We carry out this proof by structural induction on φ.

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

Case $\varphi = \bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i$. We first assume that $X \in \mathsf{fvars}(\bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i)$ and so by the definition of fvars we know that for every $i \in I$ we have that $X \in \mathsf{fvars}(\varphi_i)$. Hence, by applying the inductive hypothesis for every $i \in I$ we infer that $X \in \mathsf{fvars}(\langle\!\langle \varphi_i \rangle\!\rangle_4)$. With this result and by the definitions of fvars and $\langle\!\langle - \rangle\!\rangle_4$, we thus conclude that $X \in \mathsf{fvars}(\langle\!\langle \bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i \rangle\!\rangle_4)$ as required, and so we are done.

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

- when $Y = X$: This case holds trivially since $\langle\!\langle Y \rangle\!\rangle_4 = Y = X$ and so since $X \in \mathsf{fvars}(X)$ we can infer that $X \in \mathsf{fvars}(\langle\!\langle Y \rangle\!\rangle_4)$ as required.
- when $Y \neq X$: This case does not apply since $X \notin \mathsf{fvars}(Y)$ when $Y \neq X$.

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

$$X \in \mathsf{fvars}(\mathsf{max}\, Y.\varphi) \tag{B.166}$$

and consider the following cases:

- when $Y = X$: This case does not apply since $X \notin \mathsf{fvars}(\mathsf{max}\, Y.\varphi)$ when $Y = X$.
- when $Y \neq X$: From (B.166) and by the definition of fvars we can deduce that

  $$X \in \mathsf{fvars}(\varphi) \tag{B.167}$$

  and so by the inductive hypothesis we have that $X \in \mathsf{fvars}(\langle\!\langle \varphi \rangle\!\rangle_4)$ from which we can deduce that

  $$X \in \mathsf{fvars}(\mathsf{max}\, Y.\langle\!\langle \varphi \rangle\!\rangle_4). \tag{B.168}$$

  Finally, since $Y \in \mathsf{fvars}(\langle\!\langle \varphi \rangle\!\rangle_4)$ from (B.168) and the definition of $\langle\!\langle - \rangle\!\rangle_4$ we can conclude that $X \in \mathsf{fvars}(\langle\!\langle \mathsf{max}\, Y.\varphi \rangle\!\rangle_4)$ as required, and so we are done. □

B.3.6.2 Proving Lemma B.3

This proof determines that for every $\varphi \in \mathrm{sHML}^{\#}$, if $X \in \mathsf{fvars}(\varphi)$ *and* $X \in \mathsf{fvars}(\langle\!\langle \psi \rangle\!\rangle_4)$ then $\langle\!\langle \varphi\{^{\mathsf{max}\, X.\psi}/_X\} \rangle\!\rangle_4 = \langle\!\langle \varphi \rangle\!\rangle_4\{^{\mathsf{max}\, X.\langle\!\langle \psi \rangle\!\rangle_4}/_X\}$.

Proof. We conduct this proof by structural induction on φ.

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

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

$$X \in \mathsf{fvars}(\textstyle\bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i) \tag{B.169}$$

$$X \in \mathsf{fvars}(\langle\!\langle \psi \rangle\!\rangle_4). \tag{B.170}$$

By (B.169) and the definition of fvars we know that $\forall i \in I \cdot X \in \mathsf{fvars}(\varphi_i)$, and so by (B.170) we can apply the inductive hypothesis for every $i \in I$ and infer that

$$\forall i \in I \cdot \langle\!\langle \varphi_i \{^{\mathsf{max}\, X.\psi}/_X\} \rangle\!\rangle_4 = \langle\!\langle \varphi_i \rangle\!\rangle_4 \{^{\mathsf{max}\, X.\langle\!\langle \psi \rangle\!\rangle_4}/_X\}. \tag{B.171}$$

Finally, by (B.171) and the definition of $\langle\!\langle - \rangle\!\rangle_4$ we thus conclude that

$$\langle\!\langle \textstyle\bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i \{^{\mathsf{max}\, X.\psi}/_X\} \rangle\!\rangle_4 = \langle\!\langle \textstyle\bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i \rangle\!\rangle_4 \{^{\mathsf{max}\, X.\langle\!\langle \psi \rangle\!\rangle_4}/_X\}$$

as required.

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

$$X \in \mathsf{fvars}(Y) \tag{B.172}$$

$$X \in \mathsf{fvars}(\langle\!\langle \psi \rangle\!\rangle_4) \tag{B.173}$$

and consider the following cases:

- when $Y \neq X$: This case does not apply since (B.172) 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

  $$\langle\!\langle Y \{^{\mathsf{max}\, X.\psi}/_X\} \rangle\!\rangle_4 = \langle\!\langle X \{^{\mathsf{max}\, X.\psi}/_X\} \rangle\!\rangle_4 = \langle\!\langle \mathsf{max}\, X.\psi \rangle\!\rangle_4. \tag{B.174}$$

  Since $\langle\!\langle Y \rangle\!\rangle_4 = Y$ and $Y = X$ we can deduce that

  $$\langle\!\langle Y \rangle\!\rangle_4 \{^{\mathsf{max}\, X.\langle\!\langle \psi \rangle\!\rangle_4}/_X\} = X \{^{\mathsf{max}\, X.\langle\!\langle \psi \rangle\!\rangle_4}/_X\} = \mathsf{max}\, X.\langle\!\langle \psi \rangle\!\rangle_4. \tag{B.175}$$

  Since by (B.173) and the definition of $\langle\!\langle - \rangle\!\rangle_4$ we know that $\langle\!\langle \mathsf{max}\, X.\psi \rangle\!\rangle_4 = \mathsf{max}\, X.\langle\!\langle \psi \rangle\!\rangle_4$ and so from (B.174) and (B.175) we can conclude that

  $$\langle\!\langle Y \{^{\mathsf{max}\, X.\psi}/_X\} \rangle\!\rangle_4 = \langle\!\langle Y \rangle\!\rangle_4 \{^{\mathsf{max}\, X.\langle\!\langle \psi \rangle\!\rangle_4}/_X\}.$$

as required.

Case $\varphi = \max Y.\varphi$. We assume that

$$X \in \mathsf{fvars}(\max Y.\varphi) \tag{B.176}$$

$$X \in \mathsf{fvars}(\langle\!\langle \psi \rangle\!\rangle_4) \tag{B.177}$$

and consider the following cases:

- when $Y = X$: This case does not apply since $X \notin \mathsf{fvars}(\max Y.\varphi)$ when $Y = X$.
- when $Y \neq X$: From (B.176) and by the definition of fvars we can deduce that $X \in \mathsf{fvars}(\varphi)$ and so by (B.177) and the inductive hypothesis we have that

$$\langle\!\langle \varphi \rangle\!\rangle_4 \{^{\max X.\langle\!\langle \psi \rangle\!\rangle_4}/_X\} = \langle\!\langle \varphi \{^{\max X.\psi}/_X\} \rangle\!\rangle_4. \tag{B.178}$$

  Hence, by applying the definition of $\langle\!\langle - \rangle\!\rangle_4$ on both sides of equation (B.178) we get that $\langle\!\langle \max Y.\varphi \{^{\max X.\psi}/_X\} \rangle\!\rangle_4 = \langle\!\langle \max Y.\varphi \rangle\!\rangle_4 \{^{\max X.\langle\!\langle \psi \rangle\!\rangle_4}/_X\}$ as required, and so we are done. □

B.3.7 Proving Lemma 5.10.

We now need to prove that for every equation set *Eq*, if $\mathsf{traverse}(Eq, \{0\}, \mathsf{partition}, \varnothing)$=$\zeta$ then ζ is a *well-formed* map for *Eq*. In our proof, we rely on Lemma B.4 which we subsequently prove in **??** B.3.7.1.

Lemma B.4. 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*.

Proof. Assume that

$$\mathsf{traverse}(Eq, \{0\}, \mathsf{partition}, \varnothing)=\zeta \tag{B.179}$$

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

$$\varnothing \text{ is a } \textit{Well-formed} \text{ map for } Eq_{//\mathbf{dom}(\varnothing)} \tag{B.180}$$

and hence by (B.179), (B.180) and Lemma B.4 we can conclude that

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

as required. □

B.3.7.1 Proving Lemma B.4

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

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

$$\mathsf{traverse}(\varnothing, I, \mathsf{partition}, \zeta){=}\zeta' \tag{B.181}$$

$$\zeta \text{ is a } \textit{well-formed} \text{ map for } Eq_{/\!/\mathbf{dom}(\zeta)}. \tag{B.182}$$

Since $Eq'{=}\varnothing$, by (B.181) and the definition of traverse we have that $\zeta = \zeta'$ and so from (B.182) we can deduce that

$$\zeta' \text{ is a } \textit{well-formed} \text{ map for } Eq_{/\!/\mathbf{dom}(\zeta')}. \tag{B.183}$$

From (B.181) and the definition of 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 traverse, the equation set Eq' becomes smaller since when traverse recurses it does so with respect to 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 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 traverse function is recursively applied with respect to some $Eq'''{=}\varnothing$, 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, since the map ζ' was created as a result of traversing *every* equation in Eq, than we can deduce that $\mathbf{dom}(\zeta')$ contains the index of every equation in Eq and so we can deduce that

$$Eq_{/\!/\mathbf{dom}(\zeta')} = Eq \tag{B.184}$$

Therefore, from (B.183) and (B.184) we can conclude that

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

as required.

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

$$\mathsf{traverse}(Eq', I, \mathsf{partition}, \zeta) = \zeta' \tag{B.185}$$

$$\zeta \text{ is a } \textit{well-formed} \text{ map for } Eq_{//\mathbf{dom}(\zeta)} \tag{B.186}$$

$$Eq' \subseteq Eq \tag{B.187}$$

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

– $\boldsymbol{I = \varnothing}$: Since $I = \varnothing$, by (B.185) and the definition of traverse we know that $\zeta = \zeta'$ and so from (B.186) we can deduce that

$$\zeta' \text{ is a } \textit{well-formed} \text{ map for } Eq_{//\mathbf{dom}(\zeta')}. \tag{B.188}$$

Since $I = \varnothing$, 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 (B.188) 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.

– $\boldsymbol{I \neq \varnothing}$: By the definition of traverse and from (B.185) we can infer that

$$\zeta'' = \mathsf{partition}(Eq', I, \zeta) \tag{B.189}$$

$$Eq'' = Eq' \setminus Eq'_{//I} \tag{B.190}$$

$$I' = \bigcup_{j \in I} \mathsf{child}(Eq', j) \tag{B.191}$$

$$\mathsf{traverse}(Eq'', I', \mathsf{partition}, \zeta'') = \zeta' \tag{B.192}$$

By (B.186) 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. (B.193)
- renaming any *reference* to a data variable that is bound by a *renamed parent necessity* defined in $Eq_{/\!/\mathbf{dom}(\zeta)}$ (B.194)

and by the definition of partition from (B.189) we have that

$$\zeta'' = \zeta \,\dot\cup \left\{ \begin{array}{l} j \mapsto \zeta(i) \,\dot\cup\, \{{}^{z_0}/_{x_0} \ldots {}^{z_n}/_{x_n}\} \\ k \mapsto \zeta(l) \,\dot\cup\, \{{}^{z_0}/_{y_0} \ldots {}^{z_n}/_{y_n}\} \end{array} \middle| \begin{array}{l} \forall i, l \in I \cdot \textit{if } \mathit{Eq}(i) = \bigwedge_{j \in I'} [\{p_j[x_0 \ldots x_n], c_j\}] X_j \wedge \varphi', \\ \mathit{Eq}(l) = \bigwedge_{k \in I''} [\{p_k[y_0 \ldots y_n], c_k\}] X_k \wedge \varphi'' \textit{ and} \\ p_j \textit{ and } p_k \textit{are pattern equiv., (when } j \neq k\textit{) then} \\ \textit{we assign the } \text{same} \textit{ fresh variables } z_0 \ldots z_n. \end{array} \right\} \tag{B.195}$$

From (B.195) 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 (B.191) 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 (B.193) we can deduce that ζ'' provides a set of mappings which allow for

- renaming the *data variables* of each *pattern equivalent sibling necessity*, defined in $Eq_{/\!/\mathbf{dom}(\zeta) \cup I'}$, to the *same* set of fresh variables. (B.196)

Similarly, from (B.195) 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 (B.194) we can deduce

that the mappings provided by ζ'' also allow for

- renaming any *reference* to a data variable that is bound by a *renamed parent necessity* defined in $Eq_{//\mathbf{dom}(\zeta)\cup I'}$. (B.197)

By (B.196), (B.197) and the definition of a *well-formed* map we thus infer that

$$\zeta'' \text{ is a } \textit{well-formed} \text{ map for } Eq_{//\mathbf{dom}(\zeta)\cup I'}. \tag{B.198}$$

From (B.195) 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 (B.198) we can infer that

$$\zeta'' \text{ is a } \textit{well-formed} \text{ map for } Eq_{//\mathbf{dom}(\zeta'')}. \tag{B.199}$$

Finally, since from (B.190) and (B.187) we have that $Eq'' \subseteq Eq$, by (B.192), (B.199) 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. □

B.3.8 Proving Lemma 5.11.

In this lemma we must show that 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. We conduct this proof by induction on the structure of Eq.

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

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

$$\zeta \text{ is a } \textit{well-formed} \text{ map for } Eq \tag{B.200}$$

and so by (B.200) 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. (B.201)
- renaming any *reference* to a data variable that is bound by a *renamed parent necessity* defined in *Eq*. (B.202)

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

$$\begin{aligned} & \mathsf{uni}(\{X_i = \textstyle\bigwedge_{j\in I} [\eta_j]\varphi_j \wedge \varphi\} \dot\cup Eq', \zeta) \\ = \;& \{X_i = \textstyle\bigwedge_{j\in I} [\eta_j \underline{\zeta(j)}]\varphi_j \wedge \varphi\} \dot\cup \mathsf{uni}(Eq', \zeta) \end{aligned} . \tag{B.203}$$

Now if we assume that η_j defines an arbitrary pattern $p_j[x_0 \dots x_n]$ (where $x_0 \dots x_n$ are newly bound variables), along with some condition $c_j[x_0 \dots x_n, y^m_{<i}]$ whose evaluation depends on $x_0 \dots x_n$ and the values of m variables $y^m_{<i}$ that are bound by parent modal necessities. Hence, from (B.201) we can deduce that mapping $\zeta(j)$ in (B.203) produces a substitution environment which renames the data bindings $x_0 \dots x_n$ to some fresh variables $z_0 \dots z_n$, which are the *same* for all the other conjunct sibling necessities that are pattern equivalent to η_j. From (B.202) we can also deduce that any reference being made to some variable $y^m_{<i}$ will also be renamed accordingly by $\zeta(j)$. Hence, by the definition of a *uniform equation*, we can deduce that

$$\text{equation } X_i = \textstyle\bigwedge_{j\in I} [\eta_j]\varphi_j \wedge \varphi \text{ is } \textit{uniform}. \tag{B.204}$$

Moreover, from (B.201) and (B.202) 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 (B.203), *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 $\{p_j[x_0 \dots x_n], c_j[x_0 \dots x_n, y^m_{<i}]\}$ this will be-

come $\{p_j[z_0 \dots z_n], c_j[z_0 \dots z_n, z^m_{<i}]\}$. Notice that the new pattern $p_j[z_0 \dots z_n]$ is *equivalent* to the original one $p_j[x_0 \dots x_n]$ since it only varies by the name of the data variables it binds. The new condition $c_j[z_0 \dots z_n, z^m_{<i}]$ is also equivalent to $c_j[x_0 \dots x_n, y^m_{<i}]$ since by (B.202) we know that $\zeta(j)$ (where $\zeta(j)$ also contains $\zeta(i)$ where i is the parent of j) renames $x_0 \dots x_n$ to $z_0 \dots z_n$ and $y^m_{<i}$ to the variable names, $z^m_{<i}$, that are bound by the renamed parent necessities. This preserves the semantics of the equation by keeping it closed with respect to data variables. Hence, we can deduce

$$\begin{aligned}
& X_i{=}\textstyle\bigwedge_{j\in I}[\eta_j]\varphi_j \wedge \varphi \\
\equiv\ & X_i{=}\textstyle\bigwedge_{j\in I}[\{p_j[x_0 \dots x_n], c_j[x_0 \dots x_n, y^m_{<i}]\}]\varphi_j \wedge \varphi \\
\equiv\ & X_i{=}\textstyle\bigwedge_{j\in I}[\{p_j[z_0 \dots z_n], c_j[z_0 \dots z_n, z^m_{<i}]\}]\varphi_j \wedge \varphi \\
\equiv\ & X_i{=}\textstyle\bigwedge_{j\in I}[\{p_j[x_0 \dots x_n], c_j[x_0 \dots x_n, y^m_{<i}]\}\zeta(j)]\varphi_j \wedge \varphi \\
\equiv\ & X_i{=}\textstyle\bigwedge_{j\in I}[\eta_j\zeta(j)]\varphi_j \wedge \varphi.
\end{aligned} \tag{B.205}$$

Now since $Eq' \subset Eq$ from (B.200) we can infer that ζ is also a *well-formed* map for Eq' which allows us to apply the inductive hypothesis and deduce that

$$\text{every equation } (X_k{=}\psi_k) \in \mathsf{uni}(Eq', \zeta) \text{ is } \textit{uniform}, \text{ and that} \tag{B.206}$$

$$\mathsf{uni}(Eq', \zeta) \equiv Eq'. \tag{B.207}$$

Hence, by (B.203), (B.206) and (B.204) we can conclude that

$$\text{every equation } (X_k{=}\psi_k) \in \mathsf{uni}(Eq, \zeta) \text{ is } \textit{uniform} \tag{B.208}$$

and by (B.203), (B.207) and (B.205) we can conclude

$$\{X_i{=}\textstyle\bigwedge_{j\in I}[\eta_j]\varphi_j \wedge \varphi\} \dot\cup Eq' \equiv \{X_i{=}\textstyle\bigwedge_{j\in I}[\eta_j\underline{\zeta(j)}]\varphi_j \wedge \varphi\} \dot\cup \mathsf{uni}(Eq', \zeta) \tag{B.209}$$

as required, and so this case is done by (B.208) and (B.209). □

B.3.9 Proving Lemma 5.13.

As a result of this lemma we prove that for every equation $(X_j{=}\varphi_j) \in Eq$, if $X_j{=}\varphi_j$ is *uniform* then $Eq \equiv \mathsf{traverse}(Eq, \{0\}, \mathsf{cond\_comb}, \varnothing)$ and that every equation $(X_k{=}\psi_k) \in \mathsf{traverse}(Eq, \{0\}, \mathsf{cond\_comb}, \varnothing)$ is *equi-disjoint*.

The proof for Lemma 5.13 depends on the following which is proven in **??** B.3.9.1.

Lemma B.5. 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 $Eq_{/\!/_{\mathsf{dom}_{\mathsf{ind}}(\omega)}} {\equiv} \omega$ 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 $Eq {\equiv} \omega'$.

Proof. Assume that

$$\forall (X_j{=}\varphi_j) \in Eq \cdot \text{ equation } X_j{=}\varphi_j \text{ is } \textit{uniform}. \tag{B.210}$$

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

$$\mathsf{traverse}(Eq, \{0\}, \mathsf{cond\_comb}, \omega) = \omega' \tag{B.211}$$

and so since $\omega{=}\varnothing$, by the definition of $Eq_{/\!/_I}$ we have that $Eq_{/\!/_{\mathbf{dom}(\varnothing)}} = \varnothing = \omega$ 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 (B.210) and (B.211) we can use Lemma B.5 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. □

B.3.9.1 Proving Lemma B.5.

This 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 with respect to 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 $\mathsf{dom}_{\mathsf{ind}}(\omega) \stackrel{\text{def}}{=} \{\, i \mid X_i \in \mathbf{dom}(\omega) \,\}$.

Proof. We proceed by induction on the structure of I.

Case $I = \varnothing$. Lets start by assuming that

$$Eq' \subseteq Eq, \tag{B.212}$$

$$\mathsf{traverse}(Eq', \varnothing, \mathsf{cond\_comb}, \omega){=}\omega', \tag{B.213}$$

$$Eq_{/\!/_{\mathsf{dom}_{\mathsf{ind}}(\omega)}} {\equiv} \omega, \tag{B.214}$$

$$\text{every equation } (X_j{=}\varphi_j) \in Eq' \text{ is } \textit{uniform}, \textit{andthat} \tag{B.215}$$

$$\text{every equation } (X_k{=}\psi_k) \in \omega \text{ is } \textit{equi-disjoint}. \tag{B.216}$$

By (B.213) and the definition of traverse we know that $\omega = \omega'$ and so from (B.214) and (B.216) we can deduce that

$$\text{every equation } (X_k{=}\psi_k) \in \omega' \text{ is } \textit{equi-disjoint} \tag{B.217}$$

$$Eq_{/\!/\mathsf{dom}_{\mathsf{ind}}(\omega')} \equiv \omega'. \tag{B.218}$$

Since $I{=}\varnothing$, by the definition of traverse and (B.213) 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 (B.218) 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 so we can conclude that

$$\omega' \equiv Eq \tag{B.219}$$

as required, and so this case is done by (B.217) and (B.219).

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

$$Eq' \subseteq Eq \tag{B.220}$$

$$\mathsf{traverse}(Eq', I, \mathsf{cond\_comb}, \omega){=}\omega' \tag{B.221}$$

$$Eq_{/\!/\mathsf{dom}_{\mathsf{ind}}(\omega)} \equiv \omega \tag{B.222}$$

$$\text{every equation } (X_j{=}\varphi_j) \in Eq' \text{ is } \textit{uniform} \tag{B.223}$$

$$\text{every equation } (X_k{=}\psi_k) \in \omega \text{ is } \textit{equi-disjoint} \tag{B.224}$$

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

- $Eq' = \varnothing$: Since $Eq' = \varnothing$, by (B.221) and the definition of traverse we know that $\omega = \omega'$ and so from (B.222) and (B.224) we can deduce that

$$Eq_{/\!/\mathsf{dom}_{\mathsf{ind}}(\omega')} \equiv \omega', \text{ and that} \tag{B.225}$$

$$\text{every equation } (X_k{=}\psi_k) \in \omega' \text{ is } \textit{equi-disjoint}. \tag{B.226}$$

  By (B.221) 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.,* $\omega{=}\varnothing$,

that would eventually contain the resultant equi-disjoint equation set. Every recursive application of the traverse function is then performed with respect to: a *smaller* version Eq, *i.e.,* $Eq'{=}Eq{\setminus}Eq_{/\!/I}$, 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 $\varnothing$ 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 (B.225) we can conclude that

$$Eq \equiv \omega' \tag{B.227}$$

as required, and so this case holds by (B.226) and (B.227).

– $Eq' \neq \varnothing$: By (B.221) and the definition of traverse we have that

$$\mathsf{cond\_comb}(Eq', I, \omega){=}\omega'' \tag{B.228}$$

$$Eq'' = Eq' \setminus Eq'_{/\!/I} \tag{B.229}$$

$$I' = \bigcup_{l \in I} \mathsf{child}(Eq, l) \tag{B.230}$$

$$\mathsf{traverse}(Eq'', I', \mathsf{cond\_comb}, \omega'') = \omega', \tag{B.231}$$

By applying definition of cond_comb to (B.228) we deduce that

$$\omega'' = \omega \,\dot\cup \left\{ X_i{=}\bigwedge_{c_k \in \mathbb{C}(j,I')} [\{p, c_k\}] X_j \wedge \varphi (= \psi_i) \;\middle|\; \begin{array}{l} (X_i{=}\bigwedge_{j \in I''} [\{p, c_j\}] X_j \wedge \varphi) {\in} Eq_{/\!/I} \\ \text{and } I'{=}\bigcup_{l \in I} \mathsf{child}(Eq, l) \\ \textit{such that } I'' \subseteq I' \end{array} \right\}. \tag{B.232}$$

Now from (B.232) 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 (B.223) 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 (B.232) and the definition of *equi-disjoint*, we can deduce that

$$\text{every equation } (X_k{=}\psi_k) \in \left\{ X_i{=}\bigwedge_{c_k \in \mathbb{C}(j,I')} [\{p, c_k\}] X_j \wedge \varphi (= \psi_i) \;\middle|\; \begin{array}{l} (X_i{=}\bigwedge_{j \in I''} [\{p, c_j\}] X_j \wedge \varphi) \in \mathit{Eq}_{//I} \\ \text{and } I'{=}\bigcup_{l \in I} \mathsf{child}(\mathit{Eq}, l) \\ \textit{such that } I'' \subseteq I' \end{array} \right\} \text{ is } \textit{equi-disjoint}. \tag{B.233}$$

This means that from (B.224), (B.232) and (B.233) we can conclude that

$$\text{every equation } (X_k{=}\psi_k) \in \omega'' \text{ is } \textit{equi-disjoint} \tag{B.234}$$

as required. We also argue that the reconstructed equations in (B.232) (*i.e.*, $X_i{=}\psi_i$) are in fact *semantically equivalent* to the original ones (*i.e.*, $(X_i{=}\varphi_i){\in}\mathit{Eq}_{//I}$), 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

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

which means that from (B.222) and (B.232) we can infer that

$$\mathit{Eq}_{//\mathsf{dom}_{\mathsf{ind}}(\omega'')} \equiv \omega''. \tag{B.235}$$

Finally, since from (B.220) and (B.229) we know that $\mathit{Eq}'' \subseteq \mathit{Eq}$, from (B.223) we can infer that every equation $(X_j{=}\varphi_j) \in \mathit{Eq}''$ is *uniform*. Hence, with this result along with (B.231), (B.234) and (B.235) we can apply the inductive hypothesis

and conclude that $\mathit{Eq} \equiv \omega'$ and that every equation $(X_k{=}\psi_k) \in \omega'$ is *equi-disjoint* as required, and so we are done. □

B.4 Missing proofs from Chapter 6

In this section we provide the necessary proofs for Lemmas 6.1, 6.4 and 6.5 as these were omitted from the main text of Chapter 6.

B.4.1 Proving Lemma 6.1

We must prove that for every μHML formula φ, suppression monitors m, m', identity transformation trace κ_{ID}, and trace u if $m \xRightarrow{\kappa_{\text{ID}}} m'$ and $\mathsf{sys}(u) \notin [\![\varphi]\!]$ and $\mathit{zip}(u, \kappa_{\text{ID}}) = u$ then $\neg\mathsf{enf}(m, \varphi)$.

Proof. We proceed by induction on the length of u.

Case $u = \varepsilon$. This case holds trivially since $\mathsf{sys}(\varepsilon) = \mathsf{nil} \notin [\![\varphi]\!]$ and so since we assume that our monitors cannot perform insertions we can infer that $m[\mathsf{nil}] \sim \mathsf{nil}$ and so by the Hennessy-Milner Theorem we have that $m[\mathsf{nil}] \notin [\![\varphi]\!]$ which means that $\neg\mathsf{enf}(m, \varphi)$ as required.

Case $u = t\alpha$. Let's start by assuming that

$$m \xRightarrow{\kappa_{\text{ID}}} m' \tag{B.236}$$

$$\mathsf{sys}(t\alpha) \notin [\![\varphi]\!] \tag{B.237}$$

$$\mathit{zip}(t\alpha, \kappa_{\text{ID}}) = t\alpha \tag{B.238}$$

and consider the following two cases for the trace system of t.

$\mathsf{sys}(t) \notin [\![\varphi]\!]$: Since κ_{ID} is a trace of identity transformations, from (B.238) we can deduce that $\kappa_{\text{ID}} = \kappa'_{\text{ID}}(\alpha_{\blacktriangleright}\alpha)$ and that $\mathit{zip}(t, \kappa'_{\text{ID}}) = t$. Hence, since from (B.236) we know that $m \xRightarrow{\kappa'_{\text{ID}}} m''$ (for some m'') and since $\mathsf{sys}(t) \notin [\![\varphi]\!]$, by the *inductive hypothesis* we can deduce that $\neg\mathsf{enf}(m, \varphi)$ as required.

$\mathsf{sys}(t) \in [\![\varphi]\!]$: Let us now assume that

$$\mathsf{enf}(m, \varphi) \tag{B.239}$$

so that we can immediately infer that for every system s,

$$m[s] \in [\![\varphi]\!] \tag{B.240}$$

$$\textit{if } s \in [\![\varphi]\!] \textit{ then } m[s] \sim s. \tag{B.241}$$

Since we assume that $\mathsf{sys}(t) \in [\![\varphi]\!]$, from (B.241) we can therefore infer that $m[\mathsf{sys}(t)] \sim \mathsf{sys}(t)$ and so, as stated by (B.236), monitor m does not modify the executions of $\mathsf{sys}(t)$, and so we are guaranteed that trace t remains intact. Hence, since we know (B.237), we can also infer that α is the action that causes φ to be violated, and so in order to ensure soundness, *i.e.,* (B.240), m is required to somehow modify trace $t\alpha$. Since m is a suppression monitor, it must therefore suppress the trailing α and transform trace $t\alpha$ into t. However, from (B.236) and (B.238) we know that $\kappa_{\text{ID}} = \kappa'_{\text{ID}}(\alpha \triangleright \alpha)$ and that $\textit{zip}(t, \kappa'_{\text{ID}}) = t$ which means that m might not always suppress α, and so this means that $m[\mathsf{sys}(t\alpha)] \notin [\![\varphi]\!]$ which contradicts with (B.240). This contradiction allows us to conclude that our initial assumption (B.239) was false and thus that $\neg\mathsf{enf}(m, \varphi)$ as required, and so we are done. □

B.4.2 Proving Lemma 6.4

To prove this lemma we must show that for every $m \in$ PSupTrn, if $s \in [\![\langle\!\langle\, m \,\rangle\!\rangle]\!]$ then $m[s] \sim s$. We adopt a coinductive approach in which we show that relation $\mathcal{R} \stackrel{\text{def}}{=} \{\, (m[s], s) \mid s \in [\![\langle\!\langle\, m \,\rangle\!\rangle]\!] \,\}$ is a bisimulation relation, and so we prove the following results:

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

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

Proof for (a). We proceed by case analysis on m.

Case $m = X$. This case does not apply since $s \notin [\![\langle\!\langle\, X \,\rangle\!\rangle]\!] = [\![X]\!]$.

Case $m = \sum_{i\in I} m_i$. Assume that

$$\sum_{i\in I} m_i[s] \xrightarrow{\mu} q \tag{B.242}$$

$$s \in [\![\langle\!\langle \sum_{i\in I} m_i \rangle\!\rangle]\!] = [\![\bigwedge_{i\in I} \langle\!\langle m_i \rangle\!\rangle]\!]. \tag{B.243}$$

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

- $\mu = \tau$: In this case we can infer that the reduction in (B.242) can be caused by either rules iSup or iAsy, we thus consider both cases.

  - iAsy: By rule iAsy from (B.242) we have that

    $$s \xrightarrow{\tau} s' \tag{B.244}$$

    and that $q = \sum_{i\in I} m_i[s']$ and since the μHML semantics are agnostic of τ-actions, by (B.243) and (B.244) we have that $s' \in [\![\langle\!\langle \sum_{i\in I} m_i \rangle\!\rangle]\!]$ so that by the definition of $\mathcal{R}$ we conclude that

    $$(\sum_{i\in I} m_i[s'], s') \in \mathcal{R} \tag{B.245}$$

    as required, and so this case holds by (B.244) and (B.245).

  - iSup: In this case from (B.242) we know that

    $$s \xrightarrow{\alpha} s' \tag{B.246}$$

    and that $q = m'[s']$ and by rule eSel we have that

    $$\exists j \in I \cdot m_j \xrightarrow{\alpha \blacktriangleright \bullet} m' \tag{B.247}$$

    and so we can deduce that m_j must have the following structure, *i.e.*, $m_j = \mathsf{rec}\, X_0, \ldots X_n.\{p_j, c_j, \bullet\}.m'_j$ (where $\mathsf{rec}\, X_0, \ldots X_n.$ represents an arbitrary number of $\mathsf{rec}\,.$ statements, possibly none) so that by rules eRec and eTrn

from (B.247) we have that

$$\mathsf{mtch}(p_j, \alpha) = \sigma \textit{ and } c_j\sigma \Downarrow \mathsf{true} \tag{B.248}$$

$$m' = m'_j\sigma\{^{\mathsf{rec}\, X_0,\ldots X_n.\{p_j,c_j,\bullet\}.m'_j}/_{X_0}, \ldots\} \tag{B.249}$$

Since from (B.243) we infer that $s \in [\![\langle\!\langle\, m_j \,\rangle\!\rangle]\!] = [\![\langle\!\langle\, \{p_j, c_j, \bullet\}.m'_j\{\ldots\} \,\rangle\!\rangle]\!]$ and subsequently by the definition of $\langle\!\langle - \rangle\!\rangle$ that $s \in [\![[\{p_j, c_j\}]\mathsf{ff}]\!]$, we can thus deduce that for every action β such that $\mathsf{mtch}(p_j, \beta) = \sigma$ and $c_j\sigma \Downarrow \mathsf{true}$ then $s \xrightarrow{\beta}\!\!\!\!\!\!/\;\;$ as otherwise (B.243) would not hold, which means that this case does not apply since (B.246) and (B.248) contradict with (B.243).

- $\mu = \alpha$: In this case we can deduce that (B.242) can be the result of rules iDef or iTrn and so we inspect both eventualities.

  - iDef: By rule iDef from (B.242) we have that

$$s \xrightarrow{\alpha} s' \tag{B.250}$$

$$q = \mathsf{id}[s'] \tag{B.251}$$

$$\sum_{i \in I} m_i \xrightarrow{\alpha}\!\!\!\!\!\!/\;\;\; \tag{B.252}$$

    and since $\mathsf{id} \stackrel{\text{def}}{=} \mathsf{rec}\, Y.\{(x)!(y), \mathsf{true}, x!y\}.Y + \{(x)?(y), \mathsf{true}, x?y\}.Y$, knowing that $\langle\!\langle\, \mathsf{id} \,\rangle\!\rangle$ produces $\mathsf{max}\, X.[(x)?(y)]X \wedge [(x)!(y)]X$ which is logically equivalent to tt, we can deduce that $s' \in [\![\mathsf{tt}]\!] = [\![\langle\!\langle\, \mathsf{id} \,\rangle\!\rangle]\!]$ and so by the definition of $\mathcal{R}$ we have that

$$(\mathsf{id}[s'], s') \in \mathcal{R} \tag{B.253}$$

    as required. Therefore, this case is done by (B.250) and (B.253).

  - iTrn: By rule iTrn from (B.242) we have that

$$s \xrightarrow{\alpha} s' \tag{B.254}$$

    and that $q = m'[s']$ and (by rule eSel) that

$$\exists j \in I \cdot m_j \xrightarrow{\alpha \blacktriangleright \alpha} m' \tag{B.255}$$

and so we can deduce that m_j must have the following structure, *i.e.,* $m_j = \mathsf{rec}\, X_0, \ldots X_n.\{p_j, c_j, \underline{p_j}\}.m'_j$ (where $\mathsf{rec}\, X_0, \ldots X_n.$ represents an arbitrary number of $\mathsf{rec}.$ statements, possibly none) so that by rules eRec and eTrn from (B.255) we know that

$$\mathsf{mtch}(p_j, \alpha) = \sigma \text{ and } c_j\sigma \Downarrow \mathsf{true} \tag{B.256}$$

$$m' = m'_j\sigma\{^{\mathsf{rec}\, X_0,\ldots X_n.\{p_j,c_j,\bullet\}.m'_j}/_{X_0}, \ldots\}. \tag{B.257}$$

Since from (B.243) we infer that $s \in [\![\langle\!\langle\, m_j \,\rangle\!\rangle]\!] = [\![\langle\!\langle\, \{p_j, c_j, \underline{p_j}\}.m'_j\{\ldots\} \,\rangle\!\rangle]\!]$ which is equal to $[\![[\{p_j, c_j\}] \langle\!\langle\, m'_j\{^{\mathsf{rec}\, X_0,\ldots X_n.\{p_j,c_j,\bullet\}.m'_j}/_{X_0}, \ldots\} \,\rangle\!\rangle]\!]$, knowing (B.254) and (B.256) we can thus conclude that $s' \in [\![\langle\!\langle\, m'_j\sigma\{\ldots\} \,\rangle\!\rangle]\!]$. This means that by the definition of $\mathcal{R}$ we can conclude that

$$(\langle\!\langle\, m'_j\sigma\{^{\mathsf{rec}\, X_0,\ldots X_n.\{p_j,c_j,\bullet\}.m'_j}/_{X_0}, \ldots\} \,\rangle\!\rangle [s'], s') \in \mathcal{R} \tag{B.258}$$

as required. Hence, this case holds by (B.254) and (B.258).

Case $m = \mathsf{rec}\, X.m'$. Assume that

$$\mathsf{rec}\, X.m'[s] \xrightarrow{\mu} q \tag{B.259}$$

$$s \in [\![\langle\!\langle\, \mathsf{rec}\, X.m' \,\rangle\!\rangle]\!] = [\![\mathsf{max}\, X.\langle\!\langle\, m' \,\rangle\!\rangle]\!] \tag{B.260}$$

and so from the monitor syntax we can deduce that m' must have the following structure, *i.e.,* $m' = \mathsf{rec}\, Y_0 \ldots Y_n.\sum_{i\in I} m_i$ where $\mathsf{rec}\, Y_0 \ldots Y_n.$ represents an arbitrary number of $\mathsf{rec}.$ statements, while $|\, I \,|$ can range from 1 to n. Hence, as by rule eRec we know that the monitor in (B.259) is unfolded prior to reducing, we infer that $(\sum_{i\in I} m_i\{^{\mathsf{rec}\, X.m'}/_X, \ldots, ^{\mathsf{rec}\, Y_n.\sum_{i\in I} m_i}/_{Y_n}\}) \xrightarrow{\mu} q$ and so from (B.260) we deduce that $s \in [\![\langle\!\langle\, \mathsf{rec}\, X.m' \,\rangle\!\rangle]\!] = [\![\sum_{i\in I} m_i\{^{\mathsf{rec}\, X.m'}/_X, \ldots\}]\!]$. From this point onwards the proof proceeds as per case $m = \sum_{i\in I} m_i$ and so we omit the remainder of this proof.

Cases $m \in \{\{p, c, \underline{p}\}.m', \{p, c, \bullet\}.m'\}$. We omit the proof of these two cases as they are special cases of case $m = \sum_{i\in I} m_i$.

The above cases thus suffice to prove that (a) holds. □

Proof for (b). Once again, we carry out this proof by case analysis on m.

Case $m = X$. This case does not apply since $s \notin [\![\langle\!\langle\, X \,\rangle\!\rangle]\!] = [\![X]\!]$.

Case $m = \sum_{i \in I} m_i$. Assume that

$$s \xrightarrow{\mu} s' \tag{B.261}$$

$$s \in [\![\langle\!\langle\, \textstyle\sum_{i \in I} m_i \,\rangle\!\rangle]\!] = [\![\textstyle\bigwedge_{i \in I} \langle\!\langle\, m_i \,\rangle\!\rangle]\!] \tag{B.262}$$

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

- $\mu = \tau$: By (B.261) and rule iAsy we have that

  $$\textstyle\sum_{i \in I} m_i[s] \xrightarrow{\tau} \sum_{i \in I} m_i[s'] \tag{B.263}$$

  and since the μHML semantics abstract over τ-actions, by (B.261) and (B.262) we can infer that $s' \in [\![\langle\!\langle\, \sum_{i \in I} m_i \,\rangle\!\rangle]\!]$ and so by the definition of $\mathcal{R}$ we conclude that

  $$(\textstyle\sum_{i \in I} m_i[s'], s') \in \mathcal{R} \tag{B.264}$$

  as required, and so this case holds by (B.263) and (B.264).

- $\mu = \alpha$: When $\mu = \alpha$, the monitor can react to (B.261) via different instrumentation rules, namely rules iDef, iSup or iTrn. We inspect each eventuality.

  - iDef: By (B.261) and rule iDef we have that

    $$\textstyle\sum_{i \in I} m_i[s] \xrightarrow{\alpha} \mathsf{id}[s'] \tag{B.265}$$

    and since $\langle\!\langle\, \mathsf{id} \,\rangle\!\rangle = \mathsf{tt}$ and so $s' \in [\![\langle\!\langle\, \mathsf{id} \,\rangle\!\rangle]\!]$, by the definition of $\mathcal{R}$ we have that

    $$(\mathsf{id}[s'], s') \in \mathcal{R} \tag{B.266}$$

    as required. Therefore, this case is done by (B.265) and (B.266).

  - iTrn: By rule iTrn, from (B.261) we have that

    $$\textstyle\sum_{i \in I} m_i[s] \xrightarrow{\alpha} m'[s'] \tag{B.267}$$

    $$\textstyle\sum_{i \in I} m_i \xrightarrow{\beta \blacktriangleright \alpha} m' \tag{B.268}$$

    and since we assume that our monitors can only suppress actions from

(B.268) we can deduce that $\beta = \alpha$ and so by rule eSel we can also deduce that

$$\exists j \in I \cdot m_j \xrightarrow{\alpha \blacktriangleright \alpha} m'. \tag{B.269}$$

By the reduction rules in our model we can infer that in order for m_j to perform the reduction in (B.269) it must have a specific form, that is

$$m_j = \mathsf{rec}\, X_0, \ldots, X_n.\{p_j, c_j, \underline{p_j}\}.m'_j \tag{B.270}$$

where $\mathsf{rec}\, X_0, \ldots X_n.$ represents an arbitrary number of $\mathsf{rec}\,.$ statements, possibly none. Knowing (B.270) we can apply rules eRec and eTrn to (B.269) and deduce that

$$\exists j \in I \cdot m'_j\{^{\mathsf{rec}\, X_0, \ldots X_n.\{p_j, c_j, \underline{p_j}\}.m'_j}/_{X_0}, \ldots\} \xrightarrow{\alpha \blacktriangleright \alpha} m' \tag{B.271}$$

$$m' = m'_j\sigma\{^{\mathsf{rec}\, X_0, \ldots X_n.\{p_j, c_j, \underline{p_j}\}.m'_j}/_{X_0}, \ldots\} \tag{B.272}$$

$$\mathsf{mtch}(p_j, \alpha) = \sigma \textit{ and } c_j\sigma \Downarrow \mathsf{true}. \tag{B.273}$$

Now, since from (B.262) we know that $s \in [\![\langle\!\langle\, m_j \,\rangle\!\rangle]\!]$, from (B.270) we can deduce that $s \in [\![\langle\!\langle\, \{p_j, c_j, \underline{p_j}\}.m'_j\{\ldots\} \,\rangle\!\rangle]\!] = [\![[\{p_j, c_j\}] \langle\!\langle\, m'_j\{\ldots\} \,\rangle\!\rangle]\!]$, and so knowing (B.261), (B.273) and that $\mu = \alpha$ we can thus deduce that $s' \in [\![\langle\!\langle\, m'_j\sigma\{\ldots\} \,\rangle\!\rangle]\!]$. Hence, by the definition of $\mathcal{R}$ we can conclude that

$$(m'[s'], s') \in \mathcal{R} \tag{B.274}$$

as required. Therefore, this case holds by (B.267) and (B.274).

– iSup: By rule iSup, from (B.261) we have that $\sum_{i \in I} \xrightarrow{\alpha \blacktriangleright \bullet} m'$ and so by rule eSel we know that $\exists j \in I \cdot m_j \xrightarrow{\alpha \blacktriangleright \bullet} m'$ where $m_j = \mathsf{rec}\, X_0, \ldots, X_n.\{p_j, c_j, \bullet\}.m'_j$. Hence, by applying rules eRec and eTrn we can deduce that

$$m_j = \{p_j, c_j, \bullet\}.m'_j\{^{\mathsf{rec}\, X_0, \ldots X_n.\{p_j, c_j, \bullet\}.m'_j}/_{X_0}, \ldots\} \tag{B.275}$$

$$\mathsf{mtch}(p_j, \alpha) = \sigma \textit{ and } c_j\sigma \Downarrow \mathsf{true}. \tag{B.276}$$

Now, since from (B.262) we have that $s \in [\![\langle\!\langle\, m_j \,\rangle\!\rangle]\!]$, from (B.275) we know

that $s \in [\![[\{p_j, c_j\}]\mathsf{ff}]\!]$ which means that for every action β where $\mathsf{mtch}(p_j, \beta) = \sigma$ and $c_j\sigma \Downarrow \mathsf{true}$, then $s \stackrel{\beta}{\not\longrightarrow}$. Hence, this case does not apply since this result contradicts with (B.261) and (B.276).

Cases $m \in \{\{p, c, \underline{p}\}.m', \{p, c, \bullet\}.m'\}$. These cases are a special case of case $m = \sum_{i \in I} m_i$, *i.e.,* when $|\, I \,| = 1$ and $m_i \in \{\{p, c, \underline{p}\}.m', \{p, c, \bullet\}.m'\}$. We thus omit the proof for these two cases.

Case $m = \mathit{rec}\, X.m'$. Assume that

$$s \xrightarrow{\mu} s' \tag{B.277}$$

$$s \in [\![\langle\!\langle\, \mathsf{rec}\, X.m' \,\rangle\!\rangle]\!] = [\![\mathsf{max}\, X.\langle\!\langle\, m' \,\rangle\!\rangle]\!] \tag{B.278}$$

and since from the monitor's syntax we can infer that m' must adhere to the following structure, *i.e.,* $m' = \mathsf{rec}\, Y_0 \ldots Y_n. \sum_{i \in I} m_i$ where $\mathsf{rec}\, Y_0 \ldots Y_n.$ represents an arbitrary number of $\mathsf{rec}\,.$ statements, and $|\, I \,|$ can range from 1 to n. Hence, since from (B.278) we know that $s \in [\![\langle\!\langle\, \mathsf{rec}\, X, Y_0, \ldots, Y_n. \sum_{i \in I} m_i \,\rangle\!\rangle]\!] = [\![\mathsf{max}\, X, Y_0, \ldots, Y_n. \langle\!\langle \sum_{i \in I} m_i \,\rangle\!\rangle]\!]$ which is equal to $[\![\langle\!\langle \sum_{i \in I} m_i \,\rangle\!\rangle \{^{\mathsf{max}\, X, Y_0, \ldots, Y_n. \langle\!\langle \sum_{i \in I} m_i \rangle\!\rangle}/_X, \ldots\}]\!]$. From this point onwards the proof proceeds as per case $m = \sum_{i \in I} m_i$ and so we elide the remainder of this proof. □

B.4.3 To prove Lemma 6.5.

We prove that for every $m, m' \in$ PSupTrn, system s and trace t, if $m \xrightarrow{\alpha \blacktriangleright \bullet} m'$ and $s \in [\![\mathit{after}(\langle\!\langle\, m \,\rangle\!\rangle, t)]\!]$ then $s \in [\![\mathit{after}(\langle\!\langle\, m' \,\rangle\!\rangle, t)]\!]$. To simplify this proof, we refer to Lemma B.6 whose proof is given in **??** B.4.3.1.

Lemma B.6. For every suppression monitor m, $m' \in$ PSupTrn, system action α, and system state r, if there exists an action β, a monitor m'' and system state r' so that $m \xrightarrow{\alpha \blacktriangleright \bullet} m'$, $m' \xrightarrow{\beta \blacktriangleright \bullet} m''$ and $r \xrightarrow{\beta} r'$ then there cannot exist a μHML formula φ so that $r' \in [\![\varphi]\!]$ and $\mathsf{enf}(m, \varphi)$. □

Proof. Assume that

$$m \xrightarrow{\alpha \blacktriangleright \bullet} m' \tag{B.279}$$

$$s \in [\![\mathit{after}(\langle\!\langle\, m \,\rangle\!\rangle, t)]\!] \tag{B.280}$$

and consider the following cases, namely, the case when there exists an action β so that $s \xrightarrow{\beta} s'$ and $m' \xrightarrow{\beta\blacktriangleright\bullet} m''$, and the case when such an action does not exist. The latter can be further subdivided into two cases, namely, the case when for every action β, $s \not\xrightarrow{\beta}$, and when $s \xrightarrow{\beta} s'$ and $m' \not\xrightarrow{\beta\blacktriangleright\bullet}$.

- $\exists\beta \cdot s \xrightarrow{\beta} s'$ and $m' \xrightarrow{\beta\blacktriangleright\bullet} m''$: Since $s \xrightarrow{\beta} s'$ and $m' \xrightarrow{\beta\blacktriangleright\bullet} m''$, by (B.279) and Lemma B.6 we know that, in this case, there does not exist a μHML formula φ that is both satisfied by s (such as $\langle\!\langle\, m \,\rangle\!\rangle$ in (B.280)) and that is also enforceable by m, and hence this case does not apply.
- $\forall\beta \cdot s \not\xrightarrow{\beta}$: This case holds trivially since if s cannot perform any action, then it cannot violate the safety property $\mathit{after}(\langle\!\langle\, m' \,\rangle\!\rangle, t) \in$ sHML, and so we can simply conclude that $s \in [\![\mathit{after}(\langle\!\langle\, m' \,\rangle\!\rangle, t)]\!]$ as required.
- $\forall\beta \cdot s \xrightarrow{\beta} s'$ and $m' \not\xrightarrow{\beta\blacktriangleright\bullet}$: In this case we assume that

$$\forall\beta \cdot s \xrightarrow{\beta} s' \text{ and } m' \not\xrightarrow{\beta\blacktriangleright\bullet} . \tag{B.281}$$

  Since we only consider suppression monitors that are *persistent*, *i.e.*, $m, m' \in$ PSupTrn, from (B.279) we know that α is a violating action and that m' is also able to suppress it. From (B.281) we thus infer that any action β that can be performed by s is not a violating action since m' is unable to suppress it. Hence, since m' does not attempt to suppress more actions than m, and by the definition of $\langle\!\langle -\rangle\!\rangle$ (since $\langle\!\langle\, \{p, c, \bullet\}.m \,\rangle\!\rangle = [\{p, c\}]\mathsf{ff}$), we can infer that $\langle\!\langle\, m \,\rangle\!\rangle$ and $\langle\!\langle\, m' \,\rangle\!\rangle$ define the same violating modal necessities, $[\{p, c\}]\mathsf{ff}$, and so since we know (B.280) we can infer that $s \in [\![\mathit{after}(\langle\!\langle\, m' \,\rangle\!\rangle, \beta t')]\!]$ (where $t = \beta t'$, for all β) as required, and we are done. □

B.4.3.1 Proving Lemma B.6

We prove that for every suppression monitor m, $m' \in$ PSupTrn, system action α, and system state r, if there exists an action β, a monitor m'' and system state r' so that $m \xrightarrow{\alpha\blacktriangleright\bullet} m'$, $m' \xrightarrow{\beta\blacktriangleright\bullet} m''$ and $r \xrightarrow{\beta} r'$ then there cannot exist a μHML formula φ so that $r' \in [\![\varphi]\!]$ and $\mathsf{enf}(m, \varphi)$.

Proof. Assume that

$$m \xrightarrow{\alpha \blacktriangleright \bullet} m' \tag{B.282}$$

$$\exists \beta \cdot m' \xrightarrow{\beta \blacktriangleright \bullet} m'' \tag{B.283}$$

$$r \xrightarrow{\beta} r' \tag{B.284}$$

and additionally assume that

$$\exists \varphi \in \mu\text{HML} \cdot \mathsf{enf}(m, \varphi) \textit{ and } r \in [\![\varphi]\!] \tag{B.285}$$

from which we can deduce that

$$\mathsf{evtenf}(m, \varphi) \stackrel{\text{def}}{=} \forall s, t \cdot \textit{ if } m[s] \stackrel{t}{\Longrightarrow} n[s'] \textit{ and } s' \in [\![\textit{after}(\varphi, t)]\!] \textit{ then } n[s'] \sim s'. \tag{B.286}$$

Consider a system state r'' that performs α and reduces to s, *i.e.*, $r'' \xrightarrow{\alpha} r$ and so from (B.282) and by rule iSup we know that

$$m[r''] \xrightarrow{\tau} m'[r.] \tag{B.287}$$

Since $\textit{after}(\varphi, \tau) = \varphi$, and knowing (B.287) and (from (B.285)) that $r \in [\![\varphi]\!]$, from (B.286) we infer that

$$m'[r] \sim r. \tag{B.288}$$

However, by (B.283), (B.284) and iSup we also know that $m'[r] \xrightarrow{\tau} m''[X']$, and so from (B.284) we can infer that $m'[r] \not\sim r$ which contradicts with (B.288). This means that assumption (B.285) is *false*, and so we conclude that *there does not exist* a μHML formula φ such that $\mathsf{enf}(m, \varphi)$ and $r \in [\![\varphi]\!]$, as required. □

B.5 Missing proofs from Chapter 7

In this section we present the proofs for Theorem 7.1 and Proposition 7.1 which were omitted from Chapter 7.

B.5.1 Proving Theorem 7.1

To prove this theorem we must show that for all system states s and r, $\textit{traces}(s) = \textit{traces}(r)$ iff s and r satisfy the same set of safety properties.

The proof for this theorem, however, relies on the work on detection (runtime verification) monitors by Francalanza *et al.* in [56]. Detection monitors m_{rv} in [56] can reject a trace t at runtime by issuing the verdict no, whenever they detect that an

sHML formula has been violated by t, *i.e.*, $m_{\mathsf{rv}} \stackrel{t}{\Longrightarrow} \mathsf{no}$. In respect to these detection monitors, they prove the following results.

- **Detection Soundness:** For every formula φ, system s, trace t and detection monitor m_{rv}, if $s \stackrel{t}{\Longrightarrow}$ and $m_{\mathsf{rv}} \stackrel{t}{\Longrightarrow} \mathsf{no}$ then $s \notin [\![\varphi]\!]$.
- **Detection Completeness:** For every formula φ, system s, if $s \notin [\![\varphi]\!]$ then there exists a trace t and a detection monitor m_{rv} such that $s \stackrel{t}{\Longrightarrow}$ and $m_{\mathsf{rv}} \stackrel{t}{\Longrightarrow} \mathsf{no}$.

Detection soundness states that if a system state s executes a trace t that gets rejected by a detection monitor m_{rv}, then s violates φ. Completeness states the opposite. Using this framework we can now easily prove Theorem 7.1 as follows.

Proof. Assume that

$$\mathit{traces}(s) \subseteq \mathit{traces}(r) \quad \text{and that} \tag{B.289}$$

$$s \notin [\![\varphi]\!]. \tag{B.290}$$

Knowing (B.290) and *Detection Completeness* from [56], we can infer that there exists a trace t, and runtime verification monitor m_{rv}, such that when s executes t, m_{rv} *rejects* it for being invalid, *i.e.*, $m_{\mathsf{rv}} \stackrel{t}{\Longrightarrow} \mathsf{no}$. Hence, since from (B.289) we know that the invalid trace t can also be executed by r, in which case by *Detection Soundness* from [56] we can also conclude that $r \notin [\![\varphi]\!]$ as required, and we are done. □

B.5.2 Proving Proposition 7.1

We must prove that for every $\text{sHML}_{\mathbf{nf}}$ formula φ, system s and trace t, when $(\!|\varphi|\!)=m$ then $t \in \mathit{traces}(m[s])$ iff $t \in \mathit{traces}((s, \varphi))$. We thus prove the if case and only-if case separately. As an aide for our proofs we rely on Lemma B.7 which is proven in **??** B.5.2.1.

Lemma B.7. For every system s and r, $\text{sHML}_{\mathbf{nf}}$ formula φ and action α, if $(\!|\varphi|\!)[s] \stackrel{\alpha}{\Longrightarrow} r$ then $(\!|\varphi|\!)[s] \xrightarrow{\alpha} r$. □

Proof for the only-if case. We proceed by induction on the length of t.

Case $t = \varepsilon$. This case holds trivially since the empty trace can be executed by every system, that is, $\varepsilon \in \mathit{traces}((s, \varphi))$ as required.

Case $t = \alpha t'$. Assume that $\alpha t' \in \mathit{traces}(\llparenthesis \varphi \rrparenthesis[s])$ and so by the definition of *traces* we know that there exists a system r such that

$$t' \in \mathit{traces}(r) \tag{B.291}$$

and that $\llparenthesis \varphi \rrparenthesis[s] \stackrel{\alpha}{\Longrightarrow} r$. Hence, by Lemma B.7 we get that

$$\llparenthesis \varphi \rrparenthesis[s] \stackrel{\alpha}{\longrightarrow} r. \tag{B.292}$$

We now proceed by case analysis on φ.

- $\varphi \in \{X, \mathsf{ff}\}$: These cases do not apply since they contradict assumption (B.292), namely since $\nexists m \cdot \llparenthesis X \rrparenthesis = m$, and since $\forall t \in \textsc{Act} \cdot \llparenthesis \mathsf{ff} \rrparenthesis[s] \stackrel{t}{\not\Longrightarrow}$ where $\llparenthesis \mathsf{ff} \rrparenthesis = \mathsf{sup}$.

- $\varphi = \mathsf{tt}$: Since $\llparenthesis \mathsf{tt} \rrparenthesis = \mathsf{id}$, by rule rTrn, from (B.292) we get that

  $$s \stackrel{\alpha}{\longrightarrow} s' \tag{B.293}$$

  and that $r = \mathsf{id}[s'] = \llparenthesis \mathsf{tt} \rrparenthesis[s']$ which in conjunction with (B.291) and the *inductive hypothesis* we can deduce that

  $$t' \in \mathit{traces}((s', \mathsf{tt})). \tag{B.294}$$

  Since $\varphi = \mathsf{tt}$ and knowing (B.293) we can synthesise the controlled transition $(s, \mathsf{tt}) \stackrel{\alpha}{\longrightarrow} (s', \mathsf{tt})$ so that from (B.294) we conclude that $\alpha t' \in \mathit{traces}((s, \mathsf{tt}))$ as required.

- $\varphi = \bigwedge_{i \in I} [\alpha_i]\varphi_i$: Since $\llparenthesis \bigwedge_{i \in I} [\alpha_i]\varphi_i \rrparenthesis = \left(\sum_{i \in I} \begin{cases} \{\alpha_i, \alpha_i\}.\llparenthesis \varphi_i \rrparenthesis & \mathit{if}\ \varphi_i \neq \mathsf{ff} \\ \{\alpha_i, \bullet\}.\llparenthesis \mathsf{ff} \rrparenthesis & \mathit{otherwise} \end{cases} \right)$ we must explore the instrumentation rules that permit for reduction in (B.292).

  - rTrn: By applying rule rTrn to (B.292) we have that

    $$s \stackrel{\alpha}{\longrightarrow} s' \tag{B.295}$$

    $$r = m[s'] \tag{B.296}$$

    and that $\left(\sum_{i \in I} \begin{cases} \{\alpha_i, \alpha_i\}.\llparenthesis \varphi_i \rrparenthesis & \mathit{if}\ \varphi_i \neq \mathsf{ff} \\ \{\alpha_i, \bullet\}.\llparenthesis \mathsf{ff} \rrparenthesis & \mathit{otherwise} \end{cases} \right) \stackrel{\alpha \blacktriangleright \alpha}{\longrightarrow} m$ so that by rules eSel and

eTrn we know that

$$\exists j \in I \cdot \alpha_j = \alpha \tag{B.297}$$

$$m = (\!|\varphi_j|\!) \text{ (where } \varphi_j \neq \mathsf{ff}). \tag{B.298}$$

Knowing (B.295), (B.297) and that $\varphi_j \neq \mathsf{ff}$ we can synthesise the controlled transition

$$(s, [\alpha_j]\varphi_j) \overset{\alpha}{\longmapsto} (s', \varphi_j). \tag{B.299}$$

Moreover, since the conjunct modal necessities are *pairwise disjoint*, from (B.297) we infer that for every $i \in I \setminus \{j\}$, $\alpha_i \neq \alpha$ and so we can synthesise the controlled transition $(s, [\alpha_i]\varphi_i) \overset{\alpha}{\longmapsto} (s', \mathsf{tt})$ which in conjunction with (B.299) can be synthesised as the transition

$$(s, \textstyle\bigwedge_{i\in I} [\alpha_i]\varphi_i) \xrightarrow{\alpha} (s', \varphi_j) \tag{B.300}$$

since $\mathit{mini}(\varphi_j \wedge \bigwedge_{i\in I\setminus\{j\}} \mathsf{tt}) = \varphi_j$. Finally, since by (B.291), (B.296), (B.298) and the *inductive hypothesis* we have that $t' \in \mathit{traces}((s', \varphi_j))$, from (B.300) we conclude that $\alpha t' \in \mathit{traces}((s, \bigwedge_{i\in I} [\alpha_i]\varphi_i))$ as required.

– iDef: By rule iDef we get that

$$s \xrightarrow{\alpha} s' \tag{B.301}$$

$$r = \mathsf{id}[s'] \tag{B.302}$$

and that $\left(\sum_{i\in I} \begin{cases} \{\alpha_i, \alpha_i\}.(\!|\varphi_i|\!) & \text{if } \varphi_i \neq \mathsf{ff} \\ \{\alpha_i, \bullet\}.(\!|\mathsf{ff}|\!) & \text{otherwise} \end{cases}\right) \overset{\alpha}{\not\longrightarrow}$ from which we can infer that for every $i \in I$, $\alpha_i \neq \alpha$. With this result, from (B.301) we can thus synthesise the controlled transition $(s, \bigwedge_{i\in I} [\alpha_i]\varphi_i) \xrightarrow{\alpha} (s', \mathsf{tt})$ and so since $(\!|\mathsf{tt}|\!) = \mathsf{id}$ and by (B.291), (B.302) and the *inductive hypothesis* we have that $t' \in \mathit{traces}((s', \mathsf{tt}))$. Hence, we can conclude that $\alpha t' \in \mathit{traces}((s, \bigwedge_{i\in I} [\alpha_i]\varphi_i))$ as required.

- $\varphi = \max X.\varphi'$: Since $X \in \mathbf{fv}(\varphi')$ we can deduce that $\varphi' \notin \{\mathsf{tt}, \mathsf{ff}\}$, and also that $\varphi' \neq X$ since logical variables are required to be guarded in sHML$_{\mathbf{nf}}$. We can

thus infer that φ' adheres to a specific structure, that is, $\mathsf{max}\, Y_0 ... Y_n.\, \bigwedge_{i \in I} [\alpha_i]\varphi_i$ (where $\mathsf{max}\, Y_0 ... Y_n.$ is an arbitrary number of fixpoint declarations, possibly none). Hence, since from (B.292) we infer $\langle\!|\, \mathsf{max}\, X.\mathsf{max}\, Y_0 ... Y_n.\, \bigwedge_{i \in I} [\alpha_i]\varphi_i \,|\!\rangle[s] \xrightarrow{\alpha} r$, and since fixpoint unfolding preserves semantics, we get that

$$\langle\!|\, \textstyle\bigwedge_{i \in I} [\alpha_i]\varphi_i\{\mathsf{max}\, X Y_0 ... Y_n.\, \bigwedge_{i \in I} [\alpha_i]\varphi_i / X, \ldots\} \,|\!\rangle[s] \xrightarrow{\alpha} r. \tag{B.303}$$

After reaching the point where we know (B.303), the proof proceeds as per the previous case (*i.e.,* when $\varphi = \bigwedge_{i \in I} [\alpha_i]\varphi_i$). We thus skip this part of the proof and simply deduce that $\alpha t' \in \mathit{traces}((s, \bigwedge_{i \in I} [\alpha_i]\varphi_i\{\mathsf{max}\, X Y_0 ... Y_n.\, \bigwedge_{i \in I} [\alpha_i]\varphi_i / X, \ldots\}))$. Since unfolded recursive formulas are equivalent to their folded versions, and since $\varphi' = \mathsf{max}\, Y_0 ... Y_n.\, \bigwedge_{i \in I} [\alpha_i]\varphi_i$, we deduce that $\alpha t' \in \mathit{traces}((s, \mathsf{max}\, X.\varphi'))$ as required, and so we are done. □

Proof for the if case. We again proceed by induction on the structure of t.

Case $t = \varepsilon$. This case holds trivially since the empty trace can be executed by every system, *i.e.,* $\varepsilon \in \mathit{traces}(\langle\!|\, \varphi \,|\!\rangle[s])$ as required.

Case $t = \alpha t'$. Assume that $\alpha t' \in \mathit{traces}((s, \varphi))$ and so by the definition of *traces* we know that there exists a system r such that

$$(s, \varphi) \xrightarrow{\alpha} r \tag{B.304}$$

$$t' \in \mathit{traces}(r). \tag{B.305}$$

We proceed by case analysis on φ.

- $\varphi \in \{\mathsf{ff}, X\}$: These cases do not apply because state (s,φ) is invalid.
- $\varphi = \mathsf{tt}$: Since $(s, \mathsf{tt}) \xrightarrow{\alpha} (s', \mathsf{tt})$ this case holds trivially since $r = (s', \mathsf{tt})$ and so by (B.305) and the *inductive hypothesis* we get that $t' \in \mathit{traces}(\langle\!|\, \mathsf{tt} \,|\!\rangle[s'])$ and since $\langle\!|\, \mathsf{tt} \,|\!\rangle = \mathsf{id}$, by rules ɪTRN and ETRN we have that $\langle\!|\, \mathsf{tt} \,|\!\rangle[s] \xrightarrow{\alpha} \langle\!|\, \mathsf{tt} \,|\!\rangle[s']$ which allows us to conclude that $\alpha t' \in \mathit{traces}(\langle\!|\, \mathsf{tt} \,|\!\rangle[s])$.
- $\varphi = \bigwedge_{i \in I} [\alpha_i]\varphi_i$ and $\#_{i \in I}\, \alpha_i$: In this case we have that

$$(s, \textstyle\bigwedge_{i \in I} [\alpha_i]\varphi_i) \xmapsto{\alpha} r \tag{B.306}$$

$$\exists \psi \cdot r = (s', \mathit{mini}(\psi)) \tag{B.307}$$

and so since the branches of the conjunction are disjoint, we only need to further investigate the following cases:

– $\forall i \in I \cdot \alpha_i \neq \alpha$: Hence, from (B.306) we can infer that for every $i \in I$ we have that $(s, [\alpha_i]\varphi_i) \overset{\alpha}{\longmapsto} (s', \mathsf{tt})$ and that

$$s \xrightarrow{\alpha} s' \tag{B.308}$$

$$\mathit{mini}(\psi) = \mathsf{tt} \text{ (since } \psi = \textstyle\bigwedge_{i \in I} \mathsf{tt}). \tag{B.309}$$

Therefore, as we know (B.308) and that for every $i \in I$ then $\alpha_i \neq \alpha$, by rules eTrn and eSel we can infer that $\left(\sum_{i \in I} \begin{cases} \{\alpha_i, \alpha_i\}.(\!|\varphi_i|\!) & \textit{if } \varphi_i \neq \mathsf{ff} \\ \{\alpha_i, \bullet\}.(\!|\mathsf{ff}|\!) & \textit{otherwise} \end{cases}\right) \overset{\alpha}{\not\longmapsto}$ and so by the definition of $(\!|-|\!)$ and rule iDef we conclude that

$$(\!|\textstyle\bigwedge_{i \in I} [\alpha_i]\varphi_i|\!)[s] \xrightarrow{\alpha} (\!|\mathsf{tt}|\!)[s']. \tag{B.310}$$

Finally, from (B.305), (B.307), (B.309) and by the *inductive hypothesis* we have that $t' \in \mathit{traces}((\!|\mathsf{tt}|\!)[s'])$ and so knowing (B.310), we can infer that $\alpha t' \in \mathit{traces}((\!|\bigwedge_{i \in I} [\alpha_i]\varphi_i|\!)[s])$.

– $\exists j \in I \cdot \eta_j = \alpha$ but $\forall i \in I \setminus \{j\} \cdot \alpha_i \neq \alpha$: In this case, from the controlled synthesis rules and from (B.306) and (B.307) we can infer that $\exists j \in I \cdot (s, [\alpha_j]\varphi_j) \overset{\alpha}{\longmapsto} (s', \varphi_j)$ and that $\forall i \in I \setminus \{j\} \cdot (s, [\alpha_i]\varphi_i) \overset{\alpha}{\longmapsto} (s', \mathsf{tt})$ and finally that

$$s \xrightarrow{\alpha} s' \tag{B.311}$$

$$\mathit{mini}(\psi) = \mathit{mini}(\varphi_j \wedge \textstyle\bigwedge_{i \in I} \mathsf{tt}) = \varphi_j \tag{B.312}$$

where $\varphi_j \neq \mathsf{ff}$ as otherwise the resulting state $(s', \mathit{mini}(\mathsf{ff}))$ would be invalid and thus removed by the synthesis along with any transitions leading to it (including (B.306)). Knowing that there exists $j \in I$ so that $\alpha_j \neq \alpha$ and by rule eTrn we can also deduce that $\{\alpha_j, \alpha_j\}.(\!|\varphi_j|\!) \xrightarrow{\alpha \blacktriangleright \alpha} (\!|\varphi_j|\!)$ and so by rule eSel we have that $\left(\sum_{i \in I} \begin{cases} \{\alpha_i, \alpha_i\}.(\!|\varphi_i|\!) & \textit{if } \varphi_i \neq \mathsf{ff} \\ \{\alpha_i, \bullet\}.(\!|\mathsf{ff}|\!) & \textit{otherwise} \end{cases}\right) \xrightarrow{\alpha \blacktriangleright \alpha} (\!|\varphi_j|\!)$. This means that by (B.311), rule iTrn and the definition of $(\!|-|\!)$ we conclude that

$$(\!|\textstyle\bigwedge_{i \in I} [\alpha_i]\varphi_i|\!)[s] \xrightarrow{\alpha} (\!|\varphi_j|\!)[s']. \tag{B.313}$$

Finally, since from (B.305), (B.307), (B.312) and the *inductive hypothesis* we know that $t' \in \mathit{traces}(\llparenthesis \varphi_j \rrparenthesis[s'])$, from (B.313) we can subsequently infer that $\alpha t' \in \mathit{traces}(\llparenthesis \bigwedge_{i\in I} [\alpha_i]\varphi_i \rrparenthesis[s])$ as required.

- $\varphi = \max X.\varphi'$ and $X \in \mathbf{fv}(\varphi')$: We now have that $(s, \max X.\varphi') \xrightarrow{\alpha} (s', \psi)$ because

$$(s, \varphi'\{^{\max X.\varphi'}/_X\}) \overset{\alpha}{\longmapsto} (s', \psi) \tag{B.314}$$

and so since φ' can neither be X (since sHML$_{\mathbf{nf}}$ requires fixpoint variables to be guarded) nor ff or tt (since $X \in \mathbf{fv}(\varphi')$) we can deduce that φ' must have the form $\max Y_0 ... Y_n.\, \bigwedge_{i\in I} [\alpha_i]\varphi_i$. Since fixpoint unfolding preserves formula semantics, from (B.314) we can then deduce that

$$(s, \bigwedge_{i\in I} [\alpha_i]\varphi_i\{\max X Y_0 ... Y_n.\, \bigwedge_{i\in I} [\alpha_i]\varphi_i / X, \ldots\}) \overset{\alpha}{\longmapsto} (s', \psi). \tag{B.315}$$

From this point onwards the proof proceeds as per the previous case ($\varphi = \bigwedge_{i\in I} [\alpha_i]\varphi_i$), we thus skip this part of the proof and safely conclude that

$$\alpha t' \in \mathit{traces}(\llparenthesis \bigwedge_{i\in I} [\alpha_i]\varphi_i\{\max X Y_0 ... Y_n.\, \bigwedge_{i\in I} [\alpha_i]\varphi_i / X, \ldots\} \rrparenthesis[s]). \tag{B.316}$$

Since fixpoint folding preserves semantics and $\varphi' = \max Y_0 ... Y_n.\, \bigwedge_{i\in I} [\alpha_i]\varphi_i$, from (B.316) we thus conclude that $\alpha t' \in \mathit{traces}(\llparenthesis \max X.\varphi' \rrparenthesis[s])$ as required, and so we are done. □

B.5.2.1 Proving Lemma B.7

We must prove that for every system state s and r, sHML$_{\mathbf{nf}}$ formula φ and action α, if $\llparenthesis \varphi \rrparenthesis[s] \overset{\alpha}{\Longrightarrow} r$ then $\llparenthesis \varphi \rrparenthesis[s] \xrightarrow{\alpha} r$.

Since we assume that the SuS s does not perform τ actions, by the rules in our enforcement model we know that the only case when a τ reduction is part of a monitored execution occurs when the monitor suppresses a (visible) action of s.

Proof. We proceed by case analysis on φ.

Case $\varphi \in \{X, \mathsf{ff}\}$. These cases do not apply since $\nexists m \cdot \llparenthesis X \rrparenthesis = m$ and since $\llparenthesis \mathsf{ff} \rrparenthesis = \mathsf{sup}$ and so $\nexists \beta \in \text{Act} \cdot \mathsf{sup}[s] \overset{\beta}{\Longrightarrow}$.

Case $\varphi = \mathsf{tt}$. Since $(\!|\mathsf{tt}|\!) = \mathsf{id}$ cannot suppress any action, we can deduce that the delayed transition in $(\!|\mathsf{tt}|\!), s) \stackrel{\alpha}{\Longrightarrow} r$ is in fact a strong one and so that $(\!|\mathsf{tt}|\!), s) \stackrel{\alpha}{\longrightarrow} r$ as required.

Case $\varphi = \bigwedge_{i\in I} [\alpha_i]\varphi_i$. Assume that

$$(\sum_{i\in I} \begin{cases} \{\alpha_i\}.(\!|\varphi_i|\!) & \textit{if } \varphi_i \neq \mathsf{ff} \\ \{\alpha_i, \tau\}.(\!|\mathsf{ff}|\!) & \textit{otherwise} \end{cases})[s] \stackrel{\alpha}{\Longrightarrow} r. \tag{B.317}$$

From the delayed transition in (B.317) we infer that the system must perform some action β which is then suppressed by one of the monitor's branches, and so there must exist an index $j \in I$ so that $\alpha_j = \beta$ and $\{\alpha_j, \tau\}.(\!|\mathsf{ff}|\!) \stackrel{\beta\blacktriangleright\tau}{\Longrightarrow} (\!|\mathsf{ff}|\!)$. However, since $(\!|\mathsf{ff}|\!) = \mathsf{sup}$, we know that if any invalid action β were to be executed by s and, as a consequence, suppressed by the monitor, any subsequent action (including α) would also be suppressed by sup, in which case the instrumented system in (B.317) would be unable to eventually execute α and thus yield a contradiction. Therefore, the only way that transition (B.317) can happen is when the monitor *does not* suppress any action prior to executing α, which thus means that the delayed transition in (B.317) is in fact a strong one, *i.e.*, $(\sum_{i\in I} \begin{cases} \{\alpha_i\}.(\!|\varphi_i|\!) & \textit{if } \varphi_i \neq \mathsf{ff} \\ \{\alpha_i, \tau\}.(\!|\mathsf{ff}|\!) & \textit{otherwise} \end{cases})[s] \stackrel{\alpha}{\longrightarrow} r$ as required.

Case $\varphi = \mathsf{max}\, X.\varphi'$ *where* $X \in \boldsymbol{fv}(\varphi')$. Assume that $(\!|\mathsf{max}\, X.\varphi'|\!)[s] \stackrel{\alpha}{\Longrightarrow} r$ and so since $[\![\mathsf{max}\, X.\varphi']\!] = [\![\varphi'\{^{\mathsf{max}\, X.\varphi'}/_X\}]\!]$ we can deduce that

$$(\!|\varphi'\{^{\mathsf{max}\, X.\varphi'}/_X\}|\!)[s] \stackrel{\alpha}{\Longrightarrow} r. \tag{B.318}$$

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

$$\varphi' = \mathsf{max}\, Y_0 ... Y_n.\, \bigwedge_{i\in I} [\alpha_i]\varphi_i \tag{B.319}$$

where $\mathsf{max}\, Y_0 \ldots Y_n \ldots$ represents an arbitrary number of fixpoint declarations, possibly none. Hence, since $[\![\varphi']\!] = [\![\bigwedge_{i\in I} [\alpha_i]\varphi_i\{\mathsf{max}\, X Y_0 ... Y_n.\, \bigwedge_{i\in I} [\alpha_i]\varphi_i / X, \ldots\}]\!]$, from

(B.318) and (B.319) we have that

$$\langle\!| \bigwedge_{i\in I} [\alpha_i]\varphi_i\{\mathsf{max}\, XY_0...Y_n.\, \bigwedge_{i\in I} [\alpha_i]\varphi_i/X,\ldots\} |\!\rangle[s] \stackrel{\alpha}{\Longrightarrow} r. \tag{B.320}$$

Having reached the point where we know (B.320), the proof becomes identical as per the previous case ($\varphi = \bigwedge_{i\in I} [\alpha_i]\varphi_i$), we thus skip this part of the proof and safely conclude that $\langle\!| \bigwedge_{i\in I} [\alpha_i]\varphi_i\{\mathsf{max}\, XY_0...Y_n.\, \bigwedge_{i\in I} [\alpha_i]\varphi_i/X,\ldots\} |\!\rangle[s] \xrightarrow{\alpha} r$. Hence, knowing (B.319) and that $[\![\varphi']\!] = [\![\bigwedge_{i\in I} [\alpha_i]\varphi_i\{\mathsf{max}\, XY_0...Y_n.\, \bigwedge_{i\in I} [\alpha_i]\varphi_i/X,\ldots\}]\!]$, from (B.318) and (B.319) we conclude that $\langle\!|\mathsf{max}\, X.\varphi'|\!\rangle[s] \xrightarrow{\alpha} r$ as required, and so we are done. □

C. Missing Proofs from Part II

In this chapter we present the proofs for any result that was omitted from the main text of Part II.

C.1 Missing proofs from Chapter 9

In this section we prove Propositions 9.1 and 9.2 as these proofs were omitted from the main text of Chapter 9.

C.1.1 Proving Proposition 9.1

Since these proofs are very similar to the ones proven for Propositions 3.1 and 3.2 in Appendix B.1, we omit the similar cases and focus on showing the differing ones. We thus prove that for every instrumented transition $m[s] \stackrel{u}{\Longrightarrow} m'[s']$ we can deduce:

(a) $u = \mathit{zip}_{bi}(t, \kappa)$ and $m \stackrel{\kappa}{\Longrightarrow} m'$ and $s \stackrel{t}{\Longrightarrow} s'$; or

(b) $u = \mathit{zip}_{bi}(t, \kappa); (\mathsf{a}!v)t'$ and $m \stackrel{\kappa}{\Longrightarrow} m''\mathsf{a}!v$ and $m'' \stackrel{\bullet}{\not\rightarrow}$ and $s \stackrel{t;(\mathsf{a}!v)t'}{\Longrightarrow} s'$ and $m' = \mathsf{id}$.

Proof. We proceed by induction on the number of μ reductions in $m[s] \stackrel{u}{\Longrightarrow} m'[s']$. We omit showing the base case (for 0 reductions) as it is *identical* to that of Proposition 3.1.

Case $k+1$ reductions. As we now assume $k+1$ reductions we have that

$$m[s] \xrightarrow{\mu} m''[s''] \tag{C.1}$$

$$m''[s''] \stackrel{\mu}{\rightarrow}{}^{k} m'[s'] \quad (\equiv m''[s''] \stackrel{u}{\Longrightarrow} m'[s']) \tag{C.2}$$

and so by (C.2) and the *inductive hypothesis* we can immediately deduce either that

$$\begin{gathered}\mathit{zip}_{bi}(t,\kappa) = u \text{ and } m'' \stackrel{\kappa}{\Longrightarrow} m' \text{ and } s'' \stackrel{t}{\Longrightarrow} s'; \qquad \text{or that} \\ \left(\begin{array}{c} u = \mathit{zip}_{bi}(t,\kappa);(\mathsf{a}!v)t' \text{ and } m'' \stackrel{\kappa}{\Longrightarrow} m''' \stackrel{\mathsf{a}!v}{\not\longrightarrow} \text{ and } m''' \stackrel{\bullet}{\not\longrightarrow} \\ \text{and } s'' \stackrel{t;(\mathsf{a}!v)t'}{\Longrightarrow} s' \text{ and } m' = \mathsf{id} \end{array}\right).\end{gathered} \tag{C.3}$$

Since the reduction in (C.1) can be the result of any instrumentation rule, we must consider each eventuality. We omit showing the cases for rules iAsy and iDef as these are identical to those proven for Proposition 3.1, and the cases for iDisO, iEnO and iTrnO/I as these are very similar to those of rules iSup, iIns and iTrn respectively in Proposition 3.1.

- iDisI: In this case from (C.1) we can deduce that $\mu = \tau$ and that

$$m \xrightarrow{\bullet \triangleright \mathsf{b}?w} m'' \tag{C.4}$$

$$s \xrightarrow{\mathsf{b}?w} s'' \tag{C.5}$$

  and so since $\mathit{zip}_{bi}((\mathsf{b}?w)t, (\bullet \triangleright \mathsf{b}?w)\kappa) = u = \mathit{zip}_{bi}(t,\kappa)$, if we combine (C.4) and (C.5) to the respective reductions in (C.3) we can conclude that

$$\begin{gathered}\mathit{zip}_{bi}((\mathsf{b}?w)t, (\bullet \triangleright \mathsf{b}?w)\kappa) = u \text{ and } m \stackrel{(\bullet \triangleright \mathsf{b}?w)\kappa}{\Longrightarrow} m' \text{ and } s \stackrel{(\mathsf{b}?w)t}{\Longrightarrow} s'; \text{ or that} \\ \left(\begin{array}{c} u = \mathit{zip}_{bi}((\mathsf{b}?w)t, (\bullet \triangleright \mathsf{b}?w)\kappa);(\mathsf{a}!v)t' \text{ and } m \stackrel{(\bullet \triangleright \mathsf{b}?w)\kappa}{\Longrightarrow} m''' \stackrel{\mathsf{a}!v}{\not\longrightarrow} \text{ and} \\ m''' \stackrel{\bullet}{\not\longrightarrow} \text{ and } s \stackrel{(\mathsf{b}?w)t;(\mathsf{a}!v)t'}{\Longrightarrow} s' \text{ and } m' = \mathsf{id} \end{array}\right)\end{gathered}$$

  as required.

- iEnI: From (C.1) we can now deduce that $\mu = \mathsf{b}?w$ and that

$$m \xrightarrow{(\mathsf{b}?w)\triangleright \bullet} m'' \tag{C.6}$$

$$s'' = s' \tag{C.7}$$

  and so by the definition of zip_{bi} we deduce that $\mathit{zip}_{bi}(t, (\mathsf{b}?w_{\triangleright\bullet})\kappa) = (\mathsf{b}?w)u$ where

$\mathit{zip}_{bi}(t,\kappa)=u$, so that by (C.6), (C.7) and (C.3) we conclude that

$$\mathit{zip}_{bi}(t,(\mathsf{b}?w_{\blacktriangleright}\bullet)\kappa)=(\mathsf{b}?w)u \text{ and } m \xRightarrow{(\mathsf{b}?w_{\blacktriangleright}\bullet)\kappa} m' \text{ and } s \xRightarrow{t} s';\quad \text{or that}$$

$$\left(\begin{array}{c}(\mathsf{b}?w)u=\mathit{zip}_{bi}(t,(\mathsf{b}?w_{\blacktriangleright}\bullet)\kappa);(\mathsf{a}!v)t' \text{ and } m \xRightarrow{(\mathsf{b}?w_{\blacktriangleright}\bullet)\kappa} m''' \xrightarrow{\mathsf{a}!v} \text{ and}\\ m''' \xrightarrow{\bullet} \text{ and } s \xRightarrow{t;(\mathsf{a}!v)t'} s' \text{ and } m'=\mathsf{id}\end{array}\right)$$

as required, and so we are done. □

C.1.2 Proving Proposition 9.2

To prove Proposition 9.2 we must show that the following implications hold.

(a) if $m \xRightarrow{\kappa} m'$, $s \xRightarrow{t} s'$ and $\mathit{zip}_{bi}(t,\kappa)=u$ then $m[s] \xRightarrow{u} m'[s']$, and that

(b) if $m \xRightarrow{\kappa} m'' \xrightarrow{\mathsf{a}!v}$, $m'' \xrightarrow{\bullet}$, $s \xRightarrow{t;(\mathsf{a}!v)t'} s'$ and $\mathit{zip}_{bi}(t,\kappa)=u$ then $m[s] \xRightarrow{u;(\mathsf{a}!v)t'} \mathsf{id}[s']$.

We omit showing the proof for (b) as it is almost identical to Proposition 3.2 (b).

Proof for (a). We proceed by rule induction on $\mathit{zip}_{bi}(t,\kappa)$. We omit case $\mathit{zip}_{bi}(\varepsilon,\varepsilon)$ as it is identical to the one proven for Proposition 3.2 (a). We also elide the following cases: (i) $\mathit{zip}_{bi}((\mathsf{a}!v)t',((\mathsf{a}!v)_{\blacktriangleright}\bullet)\kappa')$, (ii) $\mathit{zip}_{bi}(t,(\bullet_{\blacktriangleright}(\mathsf{a}!v))\kappa')$, (iii) $\mathit{zip}_{bi}((\mathsf{b}!w)t',((\mathsf{b}!w)_{\blacktriangleright}(\mathsf{a}!v))\kappa')$ and $\mathit{zip}_{bi}((\mathsf{b}?w)t',((\mathsf{a}?v)_{\blacktriangleright}(\mathsf{b}?w))\kappa')$. Cases (i) and (ii) are respectively similar to the cases $\mathit{zip}_{bi}(\alpha t',(\alpha_{\blacktriangleright}\bullet)\kappa')$ and $\mathit{zip}_{bi}(t,(\bullet_{\blacktriangleright}\alpha)\kappa')$ from Proposition 3.2 (a), and the cases in (iii) are both similar to case $\mathit{zip}_{bi}(\beta t',(\beta_{\blacktriangleright}\alpha)\kappa')$ of the same proof.

Case $\mathit{zip}_{bi}((\mathsf{b}?w)t,(\mathsf{b}?w_{\blacktriangleright}\bullet)\kappa)$. Assume that $\mathit{zip}_{bi}((\mathsf{b}?w)t,(\bullet_{\blacktriangleright}\mathsf{b}?w)\kappa)=u$ where

$$\mathit{zip}_{bi}(t,\kappa)=u \tag{C.8}$$

and that $s \xRightarrow{(\mathsf{b}?w)t} s'$ and $m \xRightarrow{(\bullet_{\blacktriangleright}\mathsf{b}?w)\kappa} m'$ from which by the definitions of $\xRightarrow{t}$ and $\xRightarrow{\kappa}$ respectively, we can infer that

$$s \xRightarrow{\mathsf{b}?w} s'' \tag{C.9}$$

$$s'' \xRightarrow{t} s' \tag{C.10}$$

$$m \xrightarrow{\bullet_{\blacktriangleright}\mathsf{b}?w} m'' \tag{C.11}$$

$$m'' \xRightarrow{\kappa} m'. \tag{C.12}$$

Since $\stackrel{\mathsf{b}?w}{\Longrightarrow} \stackrel{\text{def}}{=} \stackrel{\tau}{\longrightarrow}\!\!^* \stackrel{\mathsf{b}?w}{\longrightarrow}$ from (C.9) we have that $s \stackrel{\tau}{\longrightarrow}\!\!^* s'''$ and that $s''' \stackrel{\mathsf{b}?w}{\longrightarrow} s''$, and so knowing the former we can apply rule ıAsy for zero or more times, and subsequently rule ıDisI, since we know (C.11) and the latter, to deduce that

$$m[s] \stackrel{\varepsilon}{\Longrightarrow} m''[s''']. \tag{C.13}$$

Finally, since we know (C.8), (C.10) and (C.12) we can invoke the inductive hypothesis and deduce that $m''[s''] \stackrel{u}{\Longrightarrow} m'[s']$, which when combined with (C.13) we are able to conclude that $m[s] \stackrel{u}{\Longrightarrow} m'[s']$ as required.

Case $\mathit{zip}_{bi}(t, (\mathsf{b}?w_{\blacktriangleright}\bullet)\kappa)$. Now assume that $\mathit{zip}_{bi}(t, (\mathsf{b}?w_{\blacktriangleright}\bullet)\kappa) = (\mathsf{b}?w)u$ where

$$u = \mathit{zip}_{bi}(t, \kappa) \tag{C.14}$$

and that

$$s \stackrel{t}{\Longrightarrow} s' \tag{C.15}$$

$$m \stackrel{(\mathsf{b}?w_{\blacktriangleright}\bullet)\kappa}{\Longrightarrow} m'. \tag{C.16}$$

Therefore, by applying the definition $\stackrel{\kappa}{\Longrightarrow}$ to (C.16) we have that

$$m'' \stackrel{\kappa}{\Longrightarrow} m'. \tag{C.17}$$

and that $m \stackrel{(\mathsf{b}?w)\blacktriangleright\bullet}{\longrightarrow} m''$ from which by rule ıEnI we can infer that

$$m[s] \stackrel{\mathsf{b}?w}{\Longrightarrow} m''[s]. \tag{C.18}$$

Finally, since we know (C.14), (C.15) and (C.17) we can apply the inductive hypothesis and deduce that $m''[s] \stackrel{u}{\Longrightarrow} m'[s']$, which we can combine with (C.18) to conclude that $m[s] \stackrel{(\mathsf{b}?w)u}{\Longrightarrow} m'[s']$ as required, and so we are done. □

C.2 Missing proofs from Chapter 11

This appendix chapter presents the proofs for the main results omitted from the main text of Chapter 11. Specifically, we prove Propositions 11.1 and 11.2 (sound-

ness and eventual transparency) and Theorem 11.2 (optimality) along with their supporting lemmas. Once again, to facilitate these proofs, we occasionally use the satisfaction semantics for sHML [10, 61] presented in Figure 5.1 of Chapter 5 which allows us to use $s \vDash \varphi$ in lieu of $s \in [\![\varphi]\!]$. We also refer to the τ-closure property of sHML, Proposition B.1 (restated below as Proposition C.1), that was proven in [10].

Proposition C.1. if $s \xrightarrow{\tau} s'$ and $s \vDash \varphi$ then $s' \vDash \varphi$.

C.2.1 Proving Proposition 11.1 (soundness)

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

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

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

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

We prove this by showing that relation $\mathcal{R}$ (below) is a *satisfaction relation* ($\vDash$) and so that it abides by the rules in Figure 5.1 of Chapter 5.

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

Proof. We thus proceed by case analysis on the structure of φ.

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

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

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

$$[\![\max X.\varphi]\!] \neq \varnothing \tag{C.1}$$

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

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

(C.2) we have that

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

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

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

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

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

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

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

which can be further unfolded as

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

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

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

To prove that $(r, [\{(x)!(y), c_j\}]\mathsf{ff}) \in \mathcal{R}$ we must show that (a) $[\![[\{(x)!(y), c_j\}]\mathsf{ff}]\!] \neq \varnothing$, (b) $r \sqsubseteq (\!| [\{(x)!(y), c_j\}]\mathsf{ff}, \Pi |\!)[s]$, and finally that (c) for every port a and payload value v, if $\mathsf{mtch}((x)!(y), \mathsf{a}!v) = \sigma$ and $c_j\sigma \Downarrow \mathsf{true}$ then there does *not* exist a system r' such that $r \xRightarrow{\mathsf{a}!v} r'$. From (C.4) and the definition of $[\![-]\!]$ we can immediately infer that (a) holds, and so we have that

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

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

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

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

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

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

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

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

(ii) *A violating input branch,* $[\{(x)?(y), c_j\}]\mathsf{ff}$ where $Y \notin \mathbf{fv}(c_j)$:

To prove that $(r, [\{(x)?(y), c_j\}]\mathsf{ff}) \in \mathcal{R}$ we show that: (a) $[\![[\{(x)?(y), c_j\}]\mathsf{ff}]\!] \neq \varnothing$, (b) $r \sqsubseteq_{\approx} (\!|[\{(x)?(y), c_j\}]\mathsf{ff}, \Pi|\!)[s]$ and finally that (c) for every port a and payload value v,

if $\mathsf{mtch}((x)?(y), \mathsf{a}?v) = \sigma$ and $c_j\sigma \Downarrow \mathsf{true}$, then there does *not* exist a system r' such that $r \xrightarrow{\mathsf{a}?v} r'$.

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

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

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

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

Therefore from (C.12) we can deduce that monitor

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

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

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

as required. Finally, to show that (c) holds, recall that every system s *is unable perform an input unless the environment provides it*, and that the monitor *cannot allow an input to go through* unless it has an *identity* (or replacement) branch that forwards the environment's input to the system. From (C.12) and (C.13) we thus know that the synthesised monitor m does not include such an identity (or replacement) branch for $[\{(x)?(y), c_j\}]\mathsf{ff}$, and unless $\Pi = \varnothing$, it instead provides a summation of insertion transformations that allow the monitor to *insert a default domain value* w on every port b in Π whenever $c_j\{^{\mathsf{b}}/_x\}$ evaluates to true at runtime. In this way, whenever the system is expecting to erroneously procure an input from the environment, the monitor *blocks and disables* the

input and unless $\Pi{=}\varnothing$ it also (non-deterministically) selects one of the synthesised branches in (C.12) via rule eSel and performs the insertion via rule eTrn which subsequently allows the instrumentation to proceed via rule iDisI that forwards the generated input to the system. This causes the composite system $m[s]$ to transition over a silent τ action to the recursive derivative m. Since the erroneous input is *blocked* regardless of whether the monitor inserts a value or not, we can infer that $m[s] \stackrel{\mathsf{a}?v}{\Longrightarrow\!\!\!\!\!/}$ and so from (C.5) we can also deduce that

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

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

(iii) *A non-violating branch,* $[\{p_j, c_j\}]\varphi_j$ *(where* $\varphi_j \neq \mathsf{ff}$*)*:
To prove that this branch is in $\mathcal{R}$, $(r, [\{p_j, c_j\}]\varphi_j){\in}\mathcal{R}$, we show that (a) $[\![[\{p_j, c_j\}]\varphi_j]\!] \neq \varnothing$, (b) $r \sqsubseteq\!\!\!\!\sim (\!| [\{p_j, c_j\}]\varphi_j, \Pi |\!)[s]$ and then that (c) for every action α and derivative r', when $\mathsf{mtch}(p_j, \alpha) = \sigma$, $c_j\sigma \Downarrow \mathsf{true}$ and $r \stackrel{\alpha}{\Longrightarrow} r'$ then $(r', \varphi_j\sigma){\in}\mathcal{R}$.
From (C.4) and by the definition of $[\![-]\!]$ we can immediately determine that (a) holds, and so that

$$[\![[\{p_j, c_j\}]\varphi_j]\!] \neq \varnothing \tag{C.16}$$

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

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

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

$$r \sqsubseteq\!\!\!\!\sim m[s] \sqsubseteq\!\!\!\!\sim (\!| [\{p_j, c_j\}]\varphi_j, \Pi |\!)[s] \tag{C.17}$$

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

$$\mathsf{mtch}(p_j, \alpha) = \sigma \text{ and } c_j\sigma \Downarrow \mathsf{true} \tag{C.18}$$

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

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

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

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

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

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

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

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

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

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

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

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

Since we know that $[\{p_j, c_j\}]\varphi_j$ and $\varphi_j \neq \mathsf{ff}$, from (C.6) we know that m de-

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

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

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

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

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

$$\forall \alpha, r' \cdot \ \textit{if}\ \mathsf{mtch}(p_j, \alpha) = \sigma, c_j\sigma \Downarrow \mathsf{true}\ \textit{and}\ r \xRightarrow{\alpha} r'\ \text{then}\ (r', \varphi_j\sigma) \in \mathcal{R}. \tag{C.27}$$

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

C.2.2 Proving Proposition 11.2 (eventual transparency)

We must prove that for every formula $\varphi \in \text{sHML}_{\mathbf{nf}}$ if $(\!|\, \varphi, \Pi \,|\!) = m$ then $\mathsf{evtenf}(m, \varphi)$. Since $\text{sHML}_{\mathbf{nf}}$ is equally expressive as sHML we prove that for every $\varphi \in \text{sHML}_{\mathbf{nf}}$, if $(\!|\, \varphi, \Pi \,|\!)[s] \xRightarrow{t} m'[s']$ and $s' \vDash \textit{after}(\varphi, t)$ then $m'[s'] \sim s'$. We also refer to Proposition C.2 (Transparency) and Lemma C.1, defined below, and whose proofs are provided in **??** C.2.2.1**??** C.2.2.2 respectively.

Proposition C.2 (Transparency)**.** For every state $s \in \text{Sys}$ and $\varphi \in \text{sHML}_{\mathbf{nf}}$, if $s \in [\![\varphi]\!]$ then $(\!|\, \varphi, \Pi \,|\!)[s] \sim s$. □

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

Proof. Assume that

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

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

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

$$\exists \psi \in \mathbf{sHML_{nf}} \cdot \psi = \mathit{after}(\varphi, t) \tag{C.30}$$

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

Hence, knowing (C.29) and (C.30), by Proposition C.2 (Transparency) we conclude that $(\!|\, \mathit{after}(\varphi, t), \Pi \,|\!)[s'] \sim s'$ as required, and so we are done. □

C.2.2.1 Proving Proposition C.2 (transparency)

We need to prove that for every system s, if $s \in [\![\varphi]\!]$ then $m[s] \sim s$. Since $s \in [\![\varphi]\!]$ is analogous to $s \vDash \varphi$ we prove that relation $\mathcal{R} \stackrel{\text{def}}{=} \{\, (s, (\!|\, \varphi, \Pi \,|\!)[s]) \mid s \vDash \varphi \,\}$ is a *strong bisimulation relation* that satisfies the following transfer properties:

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

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

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

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

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

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

$$s \vDash \mathsf{tt} \tag{C.32}$$

$$s \xrightarrow{\mu} s' \tag{C.33}$$

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

- $\mu = \tau$*:* Since $\mu = \tau$, we can apply rule ıAsy on (C.33) and get that

$$(\!|\, \mathsf{tt}, \Pi \,|\!)[s] \xrightarrow{\tau} (\!|\, \mathsf{tt}, \Pi \,|\!)[s'] \tag{C.34}$$

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

$$(s', (\!|\, \mathsf{tt}, \Pi \,|\!)[s']) \in \mathcal{R} \tag{C.35}$$

as required. This means that this case is done by (C.34) and (C.35).

- $\mu = \alpha$*:* Since $(\!|\, \mathsf{tt}, \Pi \,|\!) = \mathsf{id}$ encodes the 'catch-all' monitor, $\mathsf{rec}\, Y.\{(x)!(y), \mathsf{true}, x!y\}.Y + \{(x)?(y), \mathsf{true}, x?y\}.Y$, by rules ᴇRᴇc and ᴇTRN we can apply rule ɪTRNI/O and deduce that $\mathsf{id} \xrightarrow{\alpha \blacktriangleright \alpha} \mathsf{id}$, which we can further refine as

$$(\!|\, \mathsf{tt}, \Pi \,|\!)[s] \xrightarrow{\alpha} (\!|\, \mathsf{tt}, \Pi \,|\!)[s'] \tag{C.36}$$

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

$$(s', (\!|\, \mathsf{tt}, \Pi \,|\!)[s']) \in \mathcal{R} \tag{C.37}$$

as required, and so this case is done by (C.36) and (C.37).

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

$$s \vDash \textstyle\bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i \tag{C.38}$$

$$s \xrightarrow{\mu} s' \tag{C.39}$$

and by the definition of $\vDash$ and (C.38) we have that for every index $i \in I$ and action $\beta \in$ Aᴄᴛ,

$$\textit{if } s \xRightarrow{\beta} s', \mathsf{mtch}(p_i, \beta) = \sigma \textit{ and } c_i\sigma \Downarrow \mathsf{true} \textit{ then } s \vDash \textstyle\bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i. \tag{C.40}$$

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

- $\mu = \tau$*:* Since $\mu = \tau$, we can apply rule ɪAsʏ on (C.39) and obtain

$$(\!|\, \textstyle\bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i, \Pi \,|\!)[s] \xrightarrow{\tau} (\!|\, \textstyle\bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i, \Pi \,|\!)[s'] \tag{C.41}$$

as required. Since $\mu = \tau$, and since we know that sHML is τ-closed, from (C.38), (C.39) and Proposition C.1, we can deduce that $s' \vDash \bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i$, so that by

the definition of $\mathcal{R}$ we conclude that

$$(s', (\!| \bigwedge_{i\in I} [\{p_i, c_i\}]\varphi_i, \Pi |\!)[s']) \in \mathcal{R} \tag{C.42}$$

as required. This subcase is therefore done by (C.41) and (C.42).

- $\mu = \alpha$*:* Since $\mu = \alpha$, from (C.39) we know that

$$s \xrightarrow{\alpha} s' \tag{C.43}$$

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

$$(\!| \varphi_\wedge, \Pi |\!) = \mathsf{rec}\, Y.\Big(\sum_{i\in I} \begin{cases} \mathsf{dis}(p_i, c_i, Y, \Pi) & \text{if } \varphi_i = \mathsf{ff} \\ \{p_i, c_i\}.(\!| \varphi_i, \Pi |\!) & \text{otherwise} \end{cases} \Big) + \mathsf{def}(\varphi_\wedge) \tag{C.44}$$

where $\varphi_\wedge \stackrel{\text{def}}{=} \bigwedge_{i\in I} [\{p_i, c_i\}]\varphi_i$. Since the branches in the conjunction are all disjoint, $\#_{i\in I}\{p_i, c_i\}$, we know that *at most one* of the branches can match the same (input or output) action α. Hence, we consider two cases, namely:

  - *No matching branches (i.e.,* $\nexists j \in I \cdot$ *mtch*$(p_j, \alpha) = \sigma$ and $c_j\sigma \Downarrow$ *true):* Since none of the symbolic actions in (C.44) can match action α, we can infer that if α is an *input*, *i.e.,* $\alpha = \mathsf{a}?v$, then it will match the default monitor $\mathsf{def}(\varphi_\wedge)$ and transition via rule iTrnI, while if it is an *output*, *i.e.,* $\alpha = \mathsf{a}!v$, rule iDef handles the underspecification. In both cases, the monitor reduces to id. Also, notice that rules iDisO and iDisI cannot be applied since if they do, it would mean that s can also perform an erroneous action, which is not the case since we assume (C.38). Hence, we infer that

$$(\!| \bigwedge_{i\in I} [\{p_i, c_i\}]\varphi_i, \Pi |\!)[s] \xrightarrow{\alpha} (\!| \mathsf{tt}, \Pi |\!)[s'] \quad (\text{since } \mathsf{id} = (\!| \mathsf{tt}, \Pi |\!)) \tag{C.45}$$

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

$$(s', (\!| \mathsf{tt}, \Pi |\!)[s']) \in \mathcal{R} \tag{C.46}$$

    as required. This case is therefore done by (C.45) and (C.46).

  - *One matching branch (i.e.,* $\exists j \in I \cdot$ *mtch*$(p_j, \alpha) = \sigma$ and $c_j\sigma \Downarrow$ *true):* From (C.44)

we can infer that the synthesised monitor can only disable the (input or output) actions that are defined by violating modal necessities. However, from (C.40) we also deduce that s is *incapable* of executing such an action as otherwise would contradict assumption (C.38). Hence, since we now assume that branch $\{p_j, c_j\}$ matches α, from (C.44) we deduce that this action can only be transformed by an identity transformation and so by rule eTrn we have that

$$\{p_j, c_j\}.(\!|\, \varphi_j, \Pi \,|\!) \xrightarrow{\alpha \blacktriangleright \alpha} (\!|\, \varphi_j\sigma, \Pi \,|\!). \tag{C.47}$$

By applying rules eSel, eRec on (C.47) and by (C.43), (C.44) and iTrnI/O (depending on whether α is an input or output action) we get that

$$(\!|\, \textstyle\bigwedge_{i\in I} [\{p_i, c_i\}]\varphi_i, \Pi \,|\!)[s] \xrightarrow{\alpha} (\!|\, \varphi_j\sigma, \Pi \,|\!)[s'] \tag{C.48}$$

as required. By (C.40), (C.43) and since we assume that $\{p_j, c_j\}$ matches α we have that $s' \vDash \varphi_j\sigma$, and so by the definition of $\mathcal{R}$ we conclude that

$$(s', (\!|\, \varphi_j\sigma, \Pi \,|\!)[s']) \in \mathcal{R} \tag{C.49}$$

as required. Hence, this subcase holds by (C.48) and (C.49).

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

$$s \xrightarrow{\mu} s' \tag{C.50}$$

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

$$s \vDash \varphi\{\max X.\varphi/X\}. \tag{C.51}$$

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

$$\varphi = \max Y_0. \ldots \max Y_n. \bigwedge_{i\in I} [\{p_i, c_i\}]\varphi_i \tag{C.52}$$

and so by (C.51), (C.52) and the definition of $\vDash$ we have that

$$s \vDash (\bigwedge_{i\in I} [\{p_i, c_i\}]\varphi_i)\{\cdot\cdot\} \quad \textit{where} \tag{C.53}$$
$$\{\cdot\cdot\} = \{\max X.\varphi/X, (\max Y_0. \ldots \max Y_n. \bigwedge_{i\in I} [\{p_i, c_i\}]\varphi_i)/Y_0, \ldots\}.$$

Since we know (C.50) and (C.53), from this point onwards the proof proceeds as per the previous case. We thus omit this part of the proof and immediately deduce that

$$\exists m' \cdot (\!| (\bigwedge_{i\in I} [\{p_i, c_i\}]\varphi_i)\{\cdot\cdot\}, \Pi |\!)[s] \xrightarrow{\mu} (\!| m', \Pi |\!)[s'] \tag{C.54}$$
$$(s', (\!| m', \Pi |\!)[s']) \in \mathcal{R} \tag{C.55}$$

and so since $(\!| (\bigwedge_{i\in I} [\{p_i, c_i\}]\varphi_i)\{\cdot\cdot\}, \Pi |\!)$ synthesises a monitor that is the *unfolded equivalent* of that synthesised by $(\!| \varphi\{\max X.\varphi/X\}, \Pi |\!)$, from (C.54) we can conclude that

$$\exists m' \cdot (\!| \varphi\{\max X.\varphi/X\}, \Pi |\!)[s] \xrightarrow{\mu} (\!| m', \Pi |\!)[s'] \tag{C.56}$$

as required, and so this case holds by (C.55) and (C.56). □

Proof for (b). To prove (b) we proceed by using the same case analysis approach as the one we used for (a).

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

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

$$s \vDash \mathsf{tt} \tag{C.57}$$
$$(\!| \mathsf{tt}, \Pi |\!)[s] \xrightarrow{\mu} r' \tag{C.58}$$

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

- $\mu = \tau$*:* Since $\mu = \tau$, the transition in (C.58) can be performed via ıDısI, ıDısO or ıAsy. We must therefore consider these cases.
  - *ıAsy:* From rule ıAsy and (C.58) we thus know that $r' = (\!| \mathsf{tt}, \Pi |\!)[s']$ and that $s \xrightarrow{\tau} s'$ as required. Also, since every process satisfies tt, we know that

$s' \vDash \mathsf{tt}$ as well, and so we are done since by the definition of $\mathcal{R}$ we know that $(s', (\!|\, \mathsf{tt}, \Pi \,|\!)[s']) \in \mathcal{R}$.

- *ıDisI:* From rule ıDisI and (C.58) we know that: $r' = m'[s']$, $s \xrightarrow{\mathsf{a}?v} s'$ and that

$$(\!|\, \mathsf{tt}, \Pi \,|\!) \xrightarrow{\bullet \blacktriangleright (\mathsf{a}?v)} m'. \tag{C.59}$$

  Since $(\!|\, \mathsf{tt}, \Pi \,|\!) = \mathsf{id}$ we can deduce that (C.59) is *false* and hence this case does not apply.

- *ıDisO:* The proof for this case is analogous as to that of case ıDisI.

- $\mu = a?v$*:* Since $\mu = \mathsf{a}?v$, the transition in (C.58) can be performed either via ıTrnI or ıEnI. We consider both cases.

  - *ıEnI:* This case also does not apply since if the transition in (C.58) is caused by rule ıEnI we would have that $(\!|\, \mathsf{tt}, \Pi \,|\!) \xrightarrow{\mathsf{a}?v \blacktriangleright \bullet} m$ which is *false* since $(\!|\, \mathsf{tt}, \Pi \,|\!) = \mathsf{id} = \mathsf{rec}\, Y.\{(x)!(y), \mathsf{true}, x!y\}.Y + \{(x)?(y), \mathsf{true}, x?y\}.Y$ and rules eRec, eSel and eTrn state that for every $\mathsf{a}?v$, $\mathsf{id} \xrightarrow{\mathsf{a}?v \blacktriangleright \mathsf{a}?v} \mathsf{id}$, thus leading to a contradiction.

  - *ıTrnI:* By applying rule ıTrnI on (C.58) we know that $r' = m'[s']$ such that

$$(\!|\, \mathsf{tt}, \Pi \,|\!) \xrightarrow{\mathsf{a}?v \blacktriangleright \mathsf{b}?w} m'. \tag{C.60}$$

$$s \xrightarrow{\mathsf{b}?w} s' \tag{C.61}$$

    Since $(\!|\, \mathsf{tt}, \Pi \,|\!) = \mathsf{id} = \mathsf{rec}\, Y.\{(x)!(y), \mathsf{true}, x!y\}.Y + \{(x)?(y), \mathsf{true}, x?y\}.Y$, by applying rules eRec, eSel and eTrn to (C.60) we know that $\mathsf{a}?v = \mathsf{b}?w$, $m' = \mathsf{id} = (\!|\, \mathsf{tt}, \Pi \,|\!)$, meaning that $r' = (\!|\, \mathsf{tt}, \Pi \,|\!)[s']$. Hence, since every process satisfies tt we know that $s' \vDash \mathsf{tt}$, so that by the definition of $\mathcal{R}$ we conclude

$$(s', (\!|\, \mathsf{tt}, \Pi \,|\!)[s']) \in \mathcal{R}. \tag{C.62}$$

    Hence, we are done by (C.61) and (C.62) since we know that $\mathsf{a}?v = \mathsf{b}?w$.

- $\mu = a!v$*:* When $\mu = \mathsf{a}!v$, the transition in (C.58) can be performed via ıDef,

ıTrnO or ıEnO. We omit this proof as it is very similar to that of case $\mu = \mathsf{a}?v$.

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

$$s \vDash \bigwedge_{i\in I} [\{p_i, c_i\}]\varphi_i \tag{C.63}$$

$$(\!| \bigwedge_{i\in I} [\{p_i, c_i\}]\varphi_i, \Pi |\!)[s] \xrightarrow{\mu} r'. \tag{C.64}$$

From (C.63) and by the definition of $\vDash$ we can deduce that

$$\forall i \in I, \beta \in \textsc{Act} \cdot \ \textit{if}\ s \xRightarrow{\beta} s', \mathsf{mtch}(p_i, \beta) = \sigma\ \textit{and}\ c_i\sigma \Downarrow \mathsf{true}\ \textit{then}\ s' \vDash \varphi_i\sigma \tag{C.65}$$

and from (C.64) and the definition of $(\!| - |\!)$ we have that

$$\Big(\mathsf{rec}\, Y.\Big(\sum_{i\in I} \begin{cases} \mathsf{dis}(p_i, c_i, Y, \Pi) & \text{if } \varphi_i = \mathsf{ff} \\ \{p_i, c_i\}.(\!| \varphi_i, \Pi |\!) & \text{otherwise} \end{cases}\Big) + \mathsf{def}(\bigwedge_{i\in I} [\{p_i, c_i\}]\varphi_i)\Big)[s'] \xrightarrow{\mu} r'. \tag{C.66}$$

From (C.66) we can deduce that the synthesised monitor can only disable an (input or output) action β when this satisfies a violating modal necessity. However, we also know that s is *unable* to perform such an action as otherwise it would contradict assumption (C.65). Hence, we can safely conclude that the synthesised monitor in (C.66) does *not* disable any (input or output) actions of s, and so by the definition of dis we conclude that

$$\begin{array}{l} \forall \mathsf{a}?v, \mathsf{a}!v \in \textsc{Act}, s' \in \textsc{Sys} \cdot \\ \quad \left(\begin{array}{l} s \xrightarrow{\mathsf{a}?v} s'\ \textit{implies}\ (\!| \bigwedge_{i\in I} [\{p_i, c_i\}]\varphi_i, \Pi |\!) \xrightarrow{\bullet \blacktriangleright \mathsf{a}?w} \text{(for all } w) \ \textit{and} \\ s \xrightarrow{\mathsf{a}!v} s'\ \textit{implies}\ (\!| \bigwedge_{i\in I} [\{p_i, c_i\}]\varphi_i, \Pi |\!) \xrightarrow{\mathsf{a}?v \blacktriangleright \bullet} \end{array} \right). \end{array} \tag{C.67}$$

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

- $\mu = \tau$*:* Since $\mu = \tau$, from (C.64) we know that

$$(\!| \bigwedge_{i\in I} [\{p_i, c_i\}]\varphi_i, \Pi |\!)[s] \xrightarrow{\tau} r' \tag{C.68}$$

  The τ-transition in (C.68) can be the result of rules ıAsy, ıDisI or ıDisO; we thus consider each eventuality.

  - *ıAsy:* As we assume that the reduction in (C.68) is the result of rule ıAsy,

we know that $r' = (\!| \bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i, \Pi |\!)[s']$ and that

$$s \xrightarrow{\tau} s' \tag{C.69}$$

as required. Also, since sHML is τ-closed, by (C.63), (C.69) and Proposition C.1 we deduce that $s' \vDash \bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i$ as well, so that by the definition of $\mathcal{R}$ we conclude that

$$(s', (\!| \bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i, \Pi |\!)[s']) \in \mathcal{R} \tag{C.70}$$

and so we are done by (C.69) and (C.70).

– *ıDisI:* By assuming that reduction (C.68) results from ıDisI, we have that $r' = m'[s']$ and that

$$(\!| \bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i, \Pi |\!) \xrightarrow{\bullet \blacktriangleright \mathsf{a}?v} m' \tag{C.71}$$

$$s \xrightarrow{\mathsf{a}?v} s' \tag{C.72}$$

By (C.67) and (C.72) we can, however, deduce that for every value w, $(\!| \bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i, \Pi |\!) \xrightarrow{\bullet \blacktriangleright \mathsf{a}?w} \not\rightarrow$. This contradicts with (C.71) and so this case does not apply.

– *ıDisO:* As we now assume that the reduction in (C.68) results from ıDisO, we have that $r' = m'[s']$ and that

$$s \xrightarrow{\mathsf{a}!v} s' \tag{C.73}$$

$$(\!| \bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i, \Pi |\!) \xrightarrow{\mathsf{a}!v \blacktriangleright \bullet} m'. \tag{C.74}$$

Again, this case does not apply since from (C.67) and (C.73) we can deduce that $(\!| \bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i, \Pi |\!) \xrightarrow{\mathsf{a}!v \blacktriangleright \bullet} \not\rightarrow$ which contradicts with (C.74).

- $\mu = a?v$*:* When $\mu = \mathsf{a}?v$, the transition in (C.66) can be performed via rules ıEnI or ıTrnI, we consider both possibilities.

  – *ıEnI:* This case does not apply since from (C.66) and by the definition of $(\!| - |\!)$ we know that the synthesised monitor does not include action enabling transformations.

– *ıTrnI:* By assuming that (C.66) is obtained from rule ıTrnI we know that

$$\mathsf{rec}\, Y.\Big(\sum_{i\in I}\begin{cases}\mathsf{dis}(p_i,c_i,Y,\Pi) & \text{if } \varphi_i=\mathsf{ff}\\ \{p_i,c_i\}.(\!|\,\varphi_i,\Pi\,|\!) & \text{otherwise}\end{cases}\Big)+\mathsf{def}(\bigwedge_{i\in I}[\{p_i,c_i\}]\varphi_i)\xrightarrow{\mathsf{a}?v\blacktriangleright\mathsf{b}?w} m' \tag{C.75}$$

$$s\xrightarrow{\mathsf{b}?w}s' \tag{C.76}$$

$$r'=m'[s']. \tag{C.77}$$

Since from (C.67) we know that the synthesised monitor in (C.75) does not disable any action performable by s, and since from the definition of $(\!|-|\!)$ we know that the synthesis is incapable of producing action replacing monitors, we can deduce that

$$\mathsf{a}?v=\mathsf{b}?w. \tag{C.78}$$

With the knowledge of (C.78), from (C.76) we can thus deduce that

$$s\xrightarrow{\mathsf{a}?v}s' \tag{C.79}$$

as required. Knowing (C.78) we can also deduce that in (C.75) the monitor transforms an action $\mathsf{a}?v$ either (i) via an identity transformation that was synthesised from one of the *disjoint* conjunction branches, *i.e.,* from a branch $\{p_j,c_j\}.(\!|\,\varphi_j,\Pi\,|\!)$ for some $j\in I$, or else (ii) via the default monitor synthesised by $\mathsf{def}(\bigwedge_{i\in I}[\{p_i,c_i\}]\varphi_i)$. We consider both eventualities.

(i) In this case we apply rules eRec, eSel and eTrn on (C.75) and deduce that

$$\exists j\in I\cdot \mathsf{mtch}(p_j,\mathsf{a}?v)=\sigma \textit{ and } c_j\sigma\Downarrow\mathsf{true} \tag{C.80}$$

$$m'=(\!|\,\varphi_j\sigma,\Pi\,|\!). \tag{C.81}$$

and so from (C.79), (C.80) and (C.65) we infer that $s'\vDash\varphi_j\sigma$ from which by the definition of $\mathcal{R}$ we have that $(s',(\!|\,\varphi_j\sigma,\Pi\,|\!)[s'])\in\mathcal{R}$, and so from

(C.77) and (C.81) we can conclude that

$$(s', r') \in \mathcal{R} \tag{C.82}$$

as required, and so this case is done by (C.79) and (C.82).

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

$$m' = (\!|\, \mathsf{tt}, \Pi \,|\!). \tag{C.83}$$

Consequently, as every process satisfies tt, we know that $s' \vDash \mathsf{tt}$ and so by the definition of $\mathcal{R}$ we have that $(s', (\!|\, \mathsf{tt}, \Pi \,|\!)[s']) \in \mathcal{R}$, so that from (C.77) and (C.83) we can conclude that

$$(s', r') \in \mathcal{R} \tag{C.84}$$

as required. Hence this case is done by (C.79) and (C.84).

- $\mu = a!v$: When $\mu = \mathsf{a}!v$, the transition in (C.66) can be performed via iDef, iTrnO or iEnO. We omit the proof for this case due to its strong resemblance to that of case $\mu = \mathsf{a}?v$.

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

$$(\!|\, \max X.\varphi, \Pi \,|\!)[s] \xrightarrow{\mu} r' \tag{C.85}$$

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

$$s \vDash \varphi\{\max X.\varphi / X\}. \tag{C.86}$$

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

$$\varphi = \max Y_0. \ldots \max Y_n. \bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i \tag{C.87}$$

and so by (C.86), (C.87) and the definition of $\vDash$ we have that

$$s \vDash \bigwedge_{i\in I} [\{p_i, c_i\}]\varphi_i\{\cdot\cdot\} \quad \textit{where} \tag{C.88}$$
$$\{\cdot\cdot\} = \{\max X.\varphi/X, (\max Y_0. \ldots \max Y_n. \bigwedge_{i\in I} [\{p_i, c_i\}]\varphi_i)/Y_0, \ldots\}.$$

Since $(\!| \bigwedge_{i\in I} [\{p_i, c_i\}]\varphi_i\{\cdot\cdot\}, \Pi |\!)$ synthesises the *unfolded equivalent* of $(\!| \max X.\varphi, \Pi |\!)$, from (C.85) we know that

$$(\!| \bigwedge_{i\in I} [\{p_i, c_i\}]\varphi_i\{\cdot\cdot\}, \Pi |\!)[s] \xrightarrow{\mu} r'. \tag{C.89}$$

Hence, since we know (C.88) and (C.89), from this point onwards the proof proceeds as per the previous case. We thus omit showing the remainder of this proof. □

C.2.2.2 Proving Lemma C.1

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

Lemma C.2. For every formula of the form $\bigwedge_{i\in I} [\{p_i, c_i\}]\varphi_i$ and system states s and r, if $(\!| \bigwedge_{i\in I} [\{p_i, c_i\}]\varphi_i, \Pi |\!)[s] \xrightarrow{\tau}{}^* r$ then there exists some state s' and trace u such that $s \stackrel{u}{\Longrightarrow} s'$ and $r = (\!| \bigwedge_{i\in I} [\{p_i, c_i\}]\varphi_i, \Pi |\!)[s']$.

Proof. We now proceed by induction on the length of t.

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

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

$$(\!| \varphi, \Pi |\!)[s] \xrightarrow{\tau}{}^* r \tag{C.90}$$
$$r \xrightarrow{\alpha} r' \tag{C.91}$$
$$r' \stackrel{u}{\Longrightarrow} m'[s']. \tag{C.92}$$

We now proceed by case analysis on φ.

- $\varphi \in \{\mathsf{ff}, X\}$: These cases do not apply since $(\!|\, \mathsf{ff}, \Pi \,|\!)$ and $(\!|\, X, \Pi \,|\!)$ do not yield a valid monitor.
- $\varphi = \mathsf{tt}$: Since $(\!|\, \mathsf{tt}, \Pi \,|\!) = \mathsf{id}$ we know that the τ-reductions in (C.90) are only possible via rule ıAsy which means that $s \xrightarrow{\tau} {}^* s''$ and $r = (\!|\, \mathsf{tt}, \Pi \,|\!)[s'']$. The latter allows us to deduce that the reduction in (C.91) is only possible via rule ıTrn and so we also know that $s'' \xrightarrow{\alpha} {}^* s'''$ and $r' = (\!|\, \mathsf{tt}, \Pi \,|\!)[s''']$. Hence, by (C.92) and the *inductive hypothesis* we conclude that

$$\exists \psi \in \mathsf{sHML}_{\mathbf{nf}} \cdot \psi = \mathit{after}(\mathsf{tt}, u) \tag{C.93}$$

$$(\!|\, \psi, \Pi \,|\!) = m'. \tag{C.94}$$

  Since from the definition of *after* we know that $\mathit{after}(\mathsf{tt}, \alpha u)$ equates to $\mathit{after}(\mathit{after}(\mathsf{tt}, \alpha), u)$ and $\mathit{after}(\mathsf{tt}, \alpha) = \mathsf{tt}$, from (C.93) we can conclude that $\psi = \mathit{after}(\mathsf{tt}, \alpha u)$ and so this case holdssince we also know (C.94).

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

$$(\!|\, \bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i, \Pi \,|\!) = \sum_{i \in I} \begin{cases} \mathsf{dis}(p_i, c_i, m, \Pi) & \textit{if } \varphi = \mathsf{ff} \\ \{p_i, c_i\}.(\!|\, \varphi_i, \Pi \,|\!) & \textit{otherwise} \end{cases} \tag{C.95}$$

  and so by (C.90), (C.95) and Lemma C.2 we conclude that $\exists s'' \cdot s \stackrel{u}{\Longrightarrow} s''$ and that

$$r = (\!|\, \textstyle\bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i, \Pi \,|\!)[s''']. \tag{C.96}$$

  Hence, by (C.95) and (C.96) we know that the reduction in (C.91) can only happen if $\exists s''' \cdot s'' \xrightarrow{\alpha} s'''$ and α matches an identity transformation $\{p_j, c_j\}.(\!|\, \varphi_j, \Pi \,|\!)$ (for some $j \in I$) which was derived from $[\{p_j, c_j\}]\varphi_j$ (where $\varphi_j \neq \mathsf{ff}$). We can thus deduce that

$$r' = (\!|\, \varphi_j \sigma, \Pi \,|\!)[s'''] \tag{C.97}$$

$$\mathsf{mtch}(p_j, \alpha) = \sigma \textit{ and } c_j \sigma \Downarrow \mathsf{true} \tag{C.98}$$

and so by (C.92), (C.97) and the *inductive hypothesis* we deduce that

$$\exists \psi \in \text{sHML}_{\mathbf{nf}} \cdot \psi = \mathit{after}(\varphi_j \sigma, u) \tag{C.99}$$

$$(\!| \psi, \Pi |\!) = m'. \tag{C.100}$$

Now since we know (C.98), by the definition of *after* we infer that

$$\begin{array}{rcl} \mathit{after}(\bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i, \alpha u) & = & \mathit{after}(\mathit{after}(\bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i, \alpha), u) \\ & = & \mathit{after}(\varphi_j \sigma, u) \end{array} \tag{C.101}$$

and so from (C.99) and (C.101) we conclude that

$$\psi = \mathit{after}(\bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i, \alpha u). \tag{C.102}$$

Hence, this case is done by (C.100) and (C.102).

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

$$\psi = \max Y_1 \ldots . \max Y_n . \bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i. \tag{C.103}$$

where $\max Y_1 \ldots . \max Y_n .$ denotes an arbitrary number of fixpoint declarations, possibly none. Hence, knowing (C.103), by unfolding every fixpoint in $\max X.\psi$ we reduce the formula to

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

and so from this point onwards the proof proceeds as per when $\varphi = \bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i$. This therefore allows us to deduce that

$$\exists \psi' \in \text{sHML}_{\mathbf{nf}} \cdot \psi' = \mathit{after}(\bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i \{\ldots\}, \alpha u) \tag{C.104}$$

$$(\!| \psi', \Pi |\!) = m'. \tag{C.105}$$

From (C.103), (C.104) and the definition of *after* we can therefore conclude

that

$$\exists \psi' \in \mathbf{sHML_{nf}} \cdot \psi' = \mathit{after}(\max X.\psi, \alpha u) \tag{C.106}$$

and so this case holds by (C.105) and (C.106).

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

C.2.2.3 Proving Lemma C.2

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

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

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

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

$$(\!|\bigwedge_{i\in I} [\{p_i, c_i\}]\varphi_i, \Pi|\!)[s] \xrightarrow{\tau} r' \quad \text{(for some } r') \tag{C.107}$$

$$r' \xrightarrow{\tau}{}^{k} r. \tag{C.108}$$

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

$$(\!|\bigwedge_{i\in I} [\{p_i, c_i\}]\varphi_i, \Pi|\!) = \sum_{i\in I} \begin{cases} \mathsf{dis}(p_i, c_i, m, \Pi) & \mathit{if}\ \varphi{=}\mathsf{ff} \\ \{p_i, c_i\}.(\!|\varphi_i, \Pi|\!) & \mathit{otherwise} \end{cases} \tag{C.109}$$

and so from (C.109) we know that the τ-reduction in (C.107) can be the result of rules iAsy, iDisO or iDisI. We therefore inspect each case.

- iAsy: By rule iAsy, from (C.107) we can deduce that

$$\exists s'' \cdot s \xrightarrow{\tau} s'' \tag{C.110}$$

$$r' = (\!|\bigwedge_{i\in I} [\{p_i, c_i\}]\varphi_i|\!)[s''] \tag{C.111}$$

and so by (C.108), (C.111) and the *inductive hypothesis* we know that

$$\exists s', u \cdot s'' \xRightarrow{u} s' \textit{ and } r = (\!| \textstyle\bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i |\!)[s']. \tag{C.112}$$

Finally, by (C.110) and (C.112) we can thus conclude that $\exists s', u \cdot s \xRightarrow{u} s'$ and $r = (\!| \bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i |\!)[s']$.

- ıDısI: By rule ıDısI and from (C.107) we infer that

$$\exists s'' \cdot s \xrightarrow{(\mathsf{a}?v)} s'' \tag{C.113}$$

$$(\!| \textstyle\bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i, \Pi |\!) \xrightarrow{\bullet\blacktriangleright(\mathsf{a}?v)} m' \tag{C.114}$$

$$r' = m'[s''] \tag{C.115}$$

and from (C.109) and by the definition of dis we can infer that the reduction in (C.114) occurs when the synthesised monitor inserts action $\mathsf{a}?v$ and then reduces back to $(\!| \bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i, \Pi |\!)$ allowing us to infer that

$$m' = (\!| \textstyle\bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i, \Pi |\!). \tag{C.116}$$

Hence, by (C.108), (C.115) and (C.116) we can apply the *inductive hypothesis* and deduce that

$$\exists s', u \cdot s'' \xRightarrow{u} s' \textit{ and } r = (\!| \textstyle\bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i, \Pi |\!)[s'] \tag{C.117}$$

so that by (C.113) and (C.117) we finally conclude that $\exists s', u \cdot s \xRightarrow{(\mathsf{a}?v)u} s'$ and that $r = (\!| \bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i, \Pi |\!)[s']$ as required, and so we are done.

- ıDısO: We omit showing the proof for this case as it is very similar to that of case ıDısI. □

C.2.3 Proving Theorem 11.2 (weak DIS-optimal enforcement)

To simplify our proof we represent the set of action disabling monitors as DısTrn which is defined as $\text{DısTrn} \stackrel{\text{def}}{=} \{n \mid \text{if } ec_{bi}(n) \subseteq \{\mathsf{DIS}\}\}$. We also refer to the following lemmas that are proven in **??** C.2.3.1–C.2.3.4.

Lemma C.3. For every $m \in \text{DısTrn}$ and explicit trace t_τ, $mc(m, t_\tau) = N$. □

Lemma C.4. For every (input/output) action α and monitor $m \in \text{DisTrn}$, if $m \xrightarrow{\alpha \blacktriangleright \alpha} m'$, $\mathsf{enf}(m, \bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i)$, $\mathsf{mtch}(p_j, \alpha) = \sigma$ and $c_j\sigma \Downarrow \mathsf{true}$ (for some $j \in I$) then $\mathsf{enf}(m', \varphi_j\sigma)$. □

Lemma C.5. For every port a, value v and monitor $m \in \text{DisTrn}$, if $\mathsf{enf}(m, \bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i)$ and $m \xrightarrow{(\mathsf{a}!v)\blacktriangleright\bullet} m'$ then $\mathsf{enf}(m', \bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i)$. □

Lemma C.6. For every port a, value v and monitor $m \in \text{DisTrn}$, if $\mathsf{enf}(m, \bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i)$ and $m \xrightarrow{\bullet\blacktriangleright(\mathsf{a}?v)} m'$ then $\mathsf{enf}(m', \bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i)$. □

Since from Lemma C.3 we know that for every $m \in \text{DisTrn}$, $\mathit{mc}(m, t_\tau) = N$, we can prove that for every monitor $m \in \text{DisTrn}$, if $\mathsf{enf}(m, \varphi)$, $s \xRightarrow{t_\tau}$ and $\mathit{mc}(\langle\!| \varphi, \Pi |\!\rangle, t_\tau) = N$ then $N \leq \mathit{mc}(m, t_\tau)$.

Proof. We thus proceed by rule induction on $\mathit{mc}(\langle\!| \varphi, \Pi |\!\rangle, t_\tau)$.

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

$$\mathit{mc}(\langle\!| \varphi, \Pi |\!\rangle, \mu t'_\tau) = \mathit{mc}(m'_\varphi, t'_\tau) = N \tag{C.118}$$

which implies that

$$\langle\!| \varphi, \Pi |\!\rangle[\mathsf{sys}(\mu t'_\tau)] \xrightarrow{\mu} m'_\varphi[\mathsf{sys}(t'_\tau)] \tag{C.119}$$

and also assume that

$$\mathsf{enf}(m, \varphi) \tag{C.120}$$

and that $s \xRightarrow{\mu t'_\tau}$. By the rules in our model we can infer that the reduction in (C.119) can result from rule iAsy when $\mu = \tau$, iDef and rTrnO when $\mu = \mathsf{a}!v$, or rTrnI when $\mu = \mathsf{a}?v$. We consider each case individually.

- iAsy: By rule iAsy from (C.119) we know that $\mu = \tau$ and that

$$m'_\varphi = \langle\!| \varphi, \Pi |\!\rangle. \tag{C.121}$$

  Since from (C.120) we know that m is sound and eventual transparent, we can thus deduce that m does not hinder internal τ-actions from occurring and so

the composite system $(\!|\, \varphi, \Pi \,|\!)[\mathsf{sys}(\tau t'_\tau)]$ can always transition over τ via rule ıAsy, that is,

$$m[\mathsf{sys}(\tau t'_\tau)] \xrightarrow{\tau} m[\mathsf{sys}(t'_\tau)]. \tag{C.122}$$

Hence, by (C.118), (C.120) and since $s \xrightarrow{\tau t'_\tau} \Longrightarrow$ entails $s \xrightarrow{\tau} \Longrightarrow s'$ and $s' \xrightarrow{t'_\tau} \Longrightarrow$ we can apply the *inductive hypothesis* and deduce that $N \leq \mathit{mc}(m, t'_\tau)$ so that by (C.122) and the definition of *mc*, we conclude that $N \leq \mathit{mc}(m, \tau t'_\tau)$ as required.

- ıDef: From (C.119) and rule ıDef we know that $\mu = \mathsf{a}!v$, $(\!|\, \varphi, \Pi \,|\!) \xrightarrow{\mathsf{a}!v} \not\Longrightarrow$ and that $m'_\varphi = \mathsf{id}$. Since id does not modify actions, we can deduce that $\mathit{mc}(m'_\varphi, t'_\tau) = 0$ and so by the definition of *mc* we know that $\mathit{mc}((\!|\, \varphi, \Pi \,|\!), (\mathsf{a}!v)t'_\tau) = 0$ as well. This means that we cannot find a monitor that performs fewer transformations, and so we conclude that $0 \leq \mathit{mc}(m, (\mathsf{a}!v)t'_\tau)$ as required.
- ıTrnI: From (C.119) and rule ıTrnI we know that $\mu = \mathsf{a}?v$ and that

$$(\!|\, \varphi, \Pi \,|\!) \xrightarrow{(\mathsf{a}?v) \blacktriangleright (\mathsf{a}?v)} m'_\varphi. \tag{C.123}$$

We now inspect the cases for φ.

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

$$\begin{aligned} &(\!|\, \varphi_\wedge = \textstyle\bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i, \Pi \,|\!) \\ &= \mathsf{rec}\, Y. \left(\sum_{i \in I} \begin{cases} \mathsf{dis}(p_i, c_i, Y, \Pi) & \textit{if } \varphi_i = \mathsf{ff} \\ \{p_i, c_i\}.(\!|\, \varphi_i, \Pi \,|\!) & \textit{otherwise} \end{cases} \right) + \mathsf{def}(\textstyle\bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i) \\ &= \left(\sum_{i \in I} \begin{cases} \mathsf{dis}(p_i, c_i, (\!|\, \varphi_\wedge, \Pi \,|\!), \Pi) & \textit{if } \varphi_i = \mathsf{ff} \\ \{p_i, c_i\}.(\!|\, \varphi_i, \Pi \,|\!) & \textit{otherwise} \end{cases} \right) + \mathsf{def}(\textstyle\bigwedge_{i \in I} [\{p_i, c_i\}]\varphi_i) \end{aligned} \tag{C.124}$$

    Since normalized conjunctions are disjoint, *i.e.,* $\#_{i \in I} \{p_i, c_i\}$, from (C.124) we can infer that the identity reduction in (C.123) can only happen when $\mathsf{a}?v$ matches an identity branch, $\{p_j, c_j\}.(\!|\, \varphi_j, \Pi \,|\!)$ (for some $j \in I$), and so we

have that

$$\mathsf{mtch}(p_j, \mathsf{a}?v) = \sigma \text{ and } c_j\sigma \Downarrow \mathsf{true}. \tag{C.125}$$

Therefore, from (C.123), (C.125) and by rule eTrn we infer that $m'_\varphi = (\!|\, \varphi_j\sigma, \Pi \,|\!)$ and so by (C.118) we can infer that

$$\mathit{mc}(m'_\varphi, t'_\tau) = N \;\; \textit{where } m'_\varphi = (\!|\, \varphi_j\sigma, \Pi \,|\!). \tag{C.126}$$

Since from (C.124) we also know that the monitor branch $\{p_j, c_j\}.(\!|\, \varphi_j, \Pi \,|\!)$ is derived from a non-violating modal necessity, *i.e.*, $[\{p_j, c_j\}]\varphi_j$ where $\varphi_j \neq \mathsf{ff}$, we can infer that $\mathsf{a}?v$ is not a violating action and so it should not be modified by any other monitor m, as otherwise it would infringe the eventual transparency constraint of assumption (C.120) and so we know that

$$m \xrightarrow{(\mathsf{a}?v)\blacktriangleright(\mathsf{a}?v)} m' \quad (\text{for some } m'). \tag{C.127}$$

Hence, knowing that $t_\tau = (\mathsf{a}?v)t'_\tau$ and also that $\mathsf{sys}((\mathsf{a}?v)t'_\tau) \xrightarrow{\mathsf{a}?v} \mathsf{sys}(t'_\tau)$, from (C.127) and by rule iTrnI and the definition of *mc* we infer that

$$\mathit{mc}(m, (\mathsf{a}?v)t'_\tau) = \mathit{mc}(m', t'_\tau). \tag{C.128}$$

As by (C.120), (C.123), (C.125) and Lemma C.4 we know that $\mathsf{enf}(m', \varphi_j\sigma)$, by (C.126) and since $s \xRightarrow{(\mathsf{a}?v)t'_\tau}$ entails that $s \xrightarrow{\mathsf{a}?v} s'$ and $s' \xRightarrow{t'_\tau}$, we can apply the *inductive hypothesis* and deduce that $N \leq \mathit{mc}(m', t'_\tau)$ and so from (C.128) we conclude that $N \leq \mathit{mc}(m, (\mathsf{a}?v)t'_\tau)$ as required.

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

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

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

$$\mathit{mc}(\langle\!|\, \varphi, \Pi \,|\!\rangle, \mu t'_\tau) = 1 + M \tag{C.129}$$

$$\textit{where } M = \mathit{mc}(m'_\varphi, t'_\tau) \tag{C.130}$$

which implies that

$$\langle\!|\, \varphi, \Pi \,|\!\rangle[\mathsf{sys}(\mu t'_\tau)] \xrightarrow{\mu'} m'_\varphi[\mathsf{sys}(t'_\tau)] \;\; \textit{where } \mu' \neq \mu \tag{C.131}$$

and also assume that

$$\mathsf{enf}(m, \varphi) \tag{C.132}$$

and that $s \xrightarrow{\mu t'_\tau}$. Since we only consider action disabling monitors, the μ' reduction of (C.131) can only be achieved via rules ıDısO or ıDısI. We thus explore both cases.

- ıDısI: From (C.131) and by rule ıDısI we have that $\mu = \mathsf{a}?v$ and $\mu' = \tau$ and that

$$\langle\!|\, \varphi, \Pi \,|\!\rangle \xrightarrow{\bullet\blacktriangleright \mathsf{a}?v} m'_\varphi. \tag{C.133}$$

  We now inspect the cases for φ.

  - $\varphi \in \{\mathsf{ff}, \mathsf{tt}, X\}$: These cases do not apply since $\langle\!|\, \mathsf{ff}, \Pi \,|\!\rangle$ and $\langle\!|\, X, \Pi \,|\!\rangle$ do not yield a valid monitor, while $\langle\!|\, \mathsf{tt}, \Pi \,|\!\rangle = \mathsf{id}$ does not perform the reduction in (C.133).
  - $\varphi = \bigwedge_{i\in I} [\{p_i, c_i\}]\varphi_i$ where $\#_{i\in I}\{p_i, c_i\}$: Since $\varphi = \bigwedge_{i\in I} [\{p_i, c_i\}]\varphi_i$, by the definition of $\langle\!|\, - \,|\!\rangle$ we have that

$$\begin{array}{rcl} \langle\!|\, \varphi_\wedge = \bigwedge_{i\in I} [\{p_i, c_i\}]\varphi_i, \Pi \,|\!\rangle & & \\ = & \mathsf{rec}\, Y. \left(\sum_{i\in I} \begin{cases} \mathsf{dis}(p_i, c_i, Y, \Pi) & \textit{if } \varphi_i = \mathsf{ff} \\ \{p_i, c_i\}.\langle\!|\, \varphi_i, \Pi \,|\!\rangle & \textit{otherwise} \end{cases} \right) + \mathsf{def}(\bigwedge_{i\in I} [\{p_i, c_i\}]\varphi_i) & \\ = & \left(\sum_{i\in I} \begin{cases} \mathsf{dis}(p_i, c_i, \langle\!|\, \varphi_\wedge, \Pi \,|\!\rangle, \Pi) & \textit{if } \varphi_i = \mathsf{ff} \\ \{p_i, c_i\}.\langle\!|\, \varphi_i, \Pi \,|\!\rangle & \textit{otherwise} \end{cases} \right) + \mathsf{def}(\bigwedge_{i\in I} [\{p_i, c_i\}]\varphi_i) & \end{array} \tag{C.134}$$

    Since normalized conjunctions are disjoint *i.e.,* $\#_{i\in I}\{p_i, c_i\}$, and since $s \xrightarrow{\mu t'_\tau}$ where $\mu = (\mathsf{a}?v)$, by the definition of dis, from (C.134) we can deduce that

the reduction in (C.133) can only be performed by an insertion branch of the form, $\{\bullet, c_j\{\mathsf{a}/x\}, \mathsf{a}?v\}.(\!|\bigwedge_{i\in I}[\{p_i, c_i\}]\varphi_i, \Pi|\!)$ that can only be derived from a violating modal necessity $[\{p_j, c_j\}]\mathsf{ff}$ (for some $j \in I$). Hence, we can infer that

$$m'_\varphi = (\!|\bigwedge_{i\in I}[\{p_i, c_i\}]\varphi_i, \Pi|\!) \tag{C.135}$$

$$p_j = (x)?(y) \text{ and } c_j\{\mathsf{a}/x\} \Downarrow \mathsf{true}. \tag{C.136}$$

Knowing (C.136) and that $[\{p_j, c_j\}]\mathsf{ff}$ we can deduce that any input on port a is erroneous and so for the soundness constraint of assumption (C.132) to hold, any other monitor m is obliged to somehow *block* this input port. As we consider action disabling monitors, *i.e.,* $m \in$ DisTrn, we can infer that monitor m may block this input in two ways, namely, either by not reacting to the input action, *i.e.,* $m \xrightarrow{\mathsf{a}?v}\!\!\!\!\!/\;\;$, or by additionally inserting a default value v, *i.e.,* $m \xrightarrow{\bullet\blacktriangleright(\mathsf{a}?v)} m'$. We explore both cases.

* $m \xrightarrow{\mathsf{a}?v}\!\!\!\!\!/\;\;$: Since $\mathsf{sys}((\mathsf{a}?v)t'_\tau) \xrightarrow{\mathsf{a}?v} \mathsf{sys}(t'_\tau)$ and since $m \xrightarrow{\mathsf{a}?v}\!\!\!\!\!/\;\;$, by the rules in our model we know that for every action μ', $m[\mathsf{sys}((\mathsf{a}?v)t'_\tau)] \xrightarrow{\mu'}\!\!\!\!\!/\;\;$ and so by the definition of *mc* we have that $mc(m, (\mathsf{a}?v)t'_\tau) = |\,(\mathsf{a}?v)t'_\tau\,|$ meaning that by blocking inputs on a, m also blocks (and thus modifies) every subsequent action of trace t'_τ. Hence, this suffices to deduce that *at worst* $1 + M$ is equal to $|\,(\mathsf{a}?v)t'_\tau\,|$, that is $1 + M \leq |\,(\mathsf{a}?v)t'_\tau\,|$, and so from (C.129) we can deduce that $1 + M \leq mc((\!|\varphi, \Pi|\!), \mu t'_\tau)$ as required.
* $m \xrightarrow{\bullet\blacktriangleright(\mathsf{a}?v)} m'$: Since $\mathsf{sys}((\mathsf{a}?v)t'_\tau) \xrightarrow{\mathsf{a}?v} \mathsf{sys}(t'_\tau)$ and since $m \xrightarrow{\bullet\blacktriangleright(\mathsf{a}?v)} m'$, by rule iDisI we know that $m[\mathsf{sys}((\mathsf{a}?v)t'_\tau)] \xrightarrow{\tau} m[\mathsf{sys}(t'_\tau)]$ and so by the definition of *mc* we have that

  $$mc(m, (\mathsf{a}?v)t'_\tau) = 1 + mc(m', t'_\tau). \tag{C.137}$$

  Since we know (C.132), (C.133) from Lemma C.6 we can infer that $\mathsf{enf}(m', \bigwedge_{i\in I}[\{p_i, c_i\}]\varphi_i)$. Hence, by (C.130), (C.137) and since $s \xRightarrow{(\mathsf{a}?v)t'_\tau}$ entails that $s \xrightarrow{(\mathsf{a}?v)} s'$ and $s' \xRightarrow{t'_\tau}$, we can apply the *inductive hypothesis* and deduce that $M \leq mc(m', t'_\tau)$ and so from (C.129), (C.130) and

(C.137) we conclude that $1 + M \leq \mathit{mc}(m, (\mathsf{a}?v)t'_\tau)$ as required.

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

- iDisO: We omit showing the proof for this subcase as it is very similar to that of case iDisI. Apart from the obvious differences (*e.g.,* $\mathsf{a}!v$ instead of $\mathsf{a}?v$), Lemma C.5 is used instead of Lemma C.6.

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

$$\mathit{mc}(\!|\, \varphi, \Pi \,|\!), t_\tau) = |\, t_\tau \,| \quad (\text{where } t_\tau \in \{\mu t'_\tau, \varepsilon\}) \tag{C.138}$$

$$(\!|\, \varphi, \Pi \,|\!)[\mathsf{sys}(\mu t'_\tau)] \overset{\mu'}{\not\longrightarrow} \tag{C.139}$$

$$\mathsf{enf}(m, \varphi) \tag{C.140}$$

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

- $t_\tau = \varepsilon$: This case holds trivially since by (C.138), (C.139) and the definition of *mc*, $\mathit{mc}(\!|\, \varphi, \Pi \,|\!), \varepsilon) = |\, \varepsilon \,| = 0$.
- $t_\tau = \mu t'_\tau$: Since $t_\tau = \mu t'_\tau$ we can immediately exclude the cases when $\mu \in \{\tau, \mathsf{a}!v\}$ since rules iAsy and iDef make it impossible for (C.139) to be attained in such cases. Particularly, rule iAsy always permits the SuS to independently perform an internal τ-move, while rule iDef allows the monitor to default to id whenever the system performs an unspecified output $\mathsf{a}!v$. However, in the case of inputs, $\mathsf{a}?v$, the monitor may completely block inputs on a port a and as a consequence cause the entire composite system $(\!|\, \varphi, \Pi \,|\!)[\mathsf{sys}(\mu t'_\tau)]$ to block, thereby making (C.139) a possible scenario. We thus inspect the cases for φ vis-a-vis $\mu = \mathsf{a}?v$.

  – $\varphi \in \{\mathsf{ff}, \mathsf{tt}, X\}$: These cases do not apply since $(\!|\, \mathsf{ff}, \Pi \,|\!)$ and $(\!|\, X, \Pi \,|\!)$ do not yield a valid monitor and since $(\!|\, \mathsf{tt}, \Pi \,|\!) = \mathsf{id}$ is incapable of attaining (C.139).

  – $\varphi = \bigwedge_{i\in I} [\{p_i, c_i\}]\varphi_i$ where $\#_{i\in I} \{p_i, c_i\}$: Since $\varphi = \bigwedge_{i\in I} [\{p_i, c_i\}]\varphi_i$, by the definition of

$(\!|-|\!)$ we have that

$$\begin{aligned}
&(\!|\varphi_\wedge = \textstyle\bigwedge_{i\in I}[\{p_i,c_i\}]\varphi_i, \Pi|\!) \\
&\quad= \quad \mathsf{rec}\,Y.\left(\sum_{i\in I}\begin{cases}\mathsf{dis}(p_i,c_i,Y,\Pi) & \textit{if } \varphi_i = \mathsf{ff}\\ \{p_i,c_i\}.(\!|\varphi_i,\Pi|\!) & \textit{otherwise}\end{cases}\right) + \mathsf{def}(\textstyle\bigwedge_{i\in I}[\{p_i,c_i\}]\varphi_i) \qquad \text{(C.141)}\\
&\quad= \quad \left(\sum_{i\in I}\begin{cases}\mathsf{dis}(p_i,c_i,(\!|\varphi_\wedge,\Pi|\!),\Pi) & \textit{if } \varphi_i = \mathsf{ff}\\ \{p_i,c_i\}.(\!|\varphi_i,\Pi|\!) & \textit{otherwise}\end{cases}\right) + \mathsf{def}(\textstyle\bigwedge_{i\in I}[\{p_i,c_i\}]\varphi_i)
\end{aligned}$$

Since $\mu = \mathsf{a}?v$, from (C.141) and by the definitions of dis and def we can infer that the only case when (C.139) is possible is when the inputs on port a satisfy a violating modal necessity, that is, there exists some $j \in I$ such that $[\{p_j, c_j\}]\mathsf{ff}$ and for every $v \in \textsc{Val}$, $\mathsf{mtch}(p_j, \mathsf{a}?v) = \sigma$ and $c_j\sigma \Downarrow \mathsf{true}$. At the same time, the monitor is also *unaware* of the port on which the erroneous input can be made, *i.e.,* $\mathsf{a} \notin \Pi$. Hence, this case does not apply since we limit ourselves to $\textsc{Sys}_\Pi$, *i.e.,* states of system that can only input values via the ports specified in Π.

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

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

C.2.3.1 Proving Lemma C.3

We must prove that for every monitor $m \in \textsc{DisTrn}$ and explicit trace t_τ, $mc(m, t_\tau) = N$.

Proof. We proceed by induction on the length of t_τ.

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

- $\forall \mu \cdot m[\mathsf{sys}(\varepsilon)] \not\xrightarrow{\mu}$: This case holds trivially since by the definition of *mc* we have that $mc(m, \varepsilon) = |\,\varepsilon\,| = 0$.
- $\exists \mu, m', s \cdot m[\mathsf{sys}(\varepsilon)] \xrightarrow{\mu} m'[s]$: Since $\mathsf{sys}(\varepsilon) = \mathsf{nil} \not\xrightarrow{\mu}$, by the rules in our model we can infer that such a transition is only possible when the monitor *enables*

an action β via rules iEnO and iEnI, and so this case does not apply since $m \notin \text{DisTrn}$.

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

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

C.2.3.2 Proving Lemma C.4

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

Proof. We therefore start this proof by assuming that

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

$$\exists j \in I \cdot \mathsf{mtch}(p_i, \alpha) = \sigma \textit{ and } c_i\sigma \Downarrow \mathsf{true} \tag{C.143}$$

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

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

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

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

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

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

  as required. Now, lets assume that

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

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

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

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

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

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

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

C.2.3.3 Proving Lemma C.5

In this proof we show that for every port a, value v and monitors $m, m' \in \text{DisTrn}$, if $\mathsf{enf}(m, \bigwedge_{i\in I} [\{p_i, c_i\}]\varphi_i)$ and $m \xrightarrow{(\mathsf{a}!v)\blacktriangleright\bullet} m'$ then $\mathsf{enf}(m', \bigwedge_{i\in I} [\{p_i, c_i\}]\varphi_i)$.

Proof. Therefore, let's assume that

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

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

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

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

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

- $m[s] \stackrel{\tau}{\not\longrightarrow} m'[s']$: This case does not apply since, as stated by assumption (C.151), we only consider the cases where the instrumented system causes the monitor to perform the suppression transformation of (C.151) via rule iDisO.
- $m[s] \xrightarrow{\tau} m'[s']$: Since $m[s] \xrightarrow{\tau} m'[s']$, from (C.152) and by Proposition C.1 we deduce that

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

  as required. We now assume that

  $$\forall s''', u \cdot m'[s'] \stackrel{u}{\Longrightarrow} m'''[s'''] \tag{C.155}$$

  $$s''' \vDash \textit{after}(\textstyle\bigwedge_{i\in I} [\{p_i, c_i\}]\varphi_i, u) \tag{C.156}$$

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

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

  Hence, from assumptions (C.155), (C.156) and conclusion (C.157) we can in-

troduce the implication and conclude that

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

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

C.2.3.4 Proving Lemma C.6

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