Jasmine Xuereb

Extending the Limits of Monitorability

Supervised by Adrian Francalanza and Antonis Achilleos

Submitted in partial fulfilment of the requirements for the degree of PhD in Computer Science

University of Malta and Reykjavik University · *June 19, 2026*

ISBN 978-9935-577-08-5 · Electronic version

0000-0002-1676-5476 · Jasmine Xuereb

Declaration of Authenticity

I, the undersigned, declare that the dissertation titled

Extending the Limits of Monitorability

is my work, except where acknowledged and referenced.

Jasmine Xuereb · *June 19, 2026*

Acknowledgements

First and foremost, I would like to express my deepest gratitude to my supervisors, Adrian Francalanza and Antonis Achilleos. Their unrelenting guidance, feedback and moral support whenever my spirits wavered have been invaluable, and I truly couldn't have done it without them.

This journey would not have been the same without the friendship formed during my studies. Duncan, Gerard, Caroline and Marietta (among others), thank you for making this time more enjoyable, for always being ready to discuss and share ideas, and for all the laughter and encouragement along the way. To my other friends, Emma, Noella, Manuela and Anna-Marie, thank you for understanding and putting up with my long absences, missed celebrations, and last-minute cancellations whenever deadlines were looming—I promise to make it up to you in the times ahead.

A heartfelt thanks goes to my (back then) fiancé, Josmar, for helping me keep things in perspective and reminding me that there is more to life than just work and deadlines. You encouraged me to pursue a PhD, and then graciously endured being left behind for months at a time while I disappeared to Iceland in the name of research. You've been my steady anchor (and occasional reality check) throughout it all.

Last but certainly not least, a special thanks goes to my family, especially my parents, Jacqueline and Joseph, and my brother, Charlon, who have been a never-ending source of motivation (and remarkable patience). You've endured more than your fair share of self-doubt, complaints, and never-ending rants, always responding with nothing than calm reassurance and a steady supply of of encouragement. Your unwavering belief in me, even when I wasn't so sure myself, meant everything. This PhD is as much yours as it is mine.

"Begin at the beginning," the King said, gravely,
"and go on till you come to the end; then stop."

— Lewis Carroll, *Alice in Wonderland*

Publications

Part of this work is taken from the paper entitled *"If At First You Don't Succeed: Extended Monitorability through Multiple Executions"*, to appear in LICS 2025.

Contribution I hereby acknowledge that I was actively involved in the writing of this paper and that I am solely responsible for the development of all definitions, theoretical results, and corresponding proofs. Section 1 of the paper contributes to Chapter 1, whereas most of the material in Section 2 is split between Chapters 2 and 6. Section 3 forms the basis of Sections 3.1, 7.1 and 7.2. Section 4 contributes to Chapters 4 and 8. Most of the material in Section 5 is integrated in Chapter 9, but certain parts of it are also similar to Chapter 5. Appendices D and E of the paper forms the basis of the proofs in Sections 5.2, 5.4 and 9.3. Section 5 (and Appendix G) contribute to Chapter 11, whereas the material in Sections 6 and 7 of the paper are integrated in Section 10.2 and Chapter 14 respectively. The remaining appendices of the paper are integrated in Appendix I to Appendix III.

Abstract

One of the most fundamental and long-standing issues in computer science is making sure that a system behaves as intended. To address this challenge, systems may be verified dynamically using a lightweight monitoring technique called runtime verification. Even though runtime verification circumvents issues associated with traditional techniques, it is limited by the fact that observations are typically constrained to the current computation path of the system. This limitation is even more acute for correctness properties describing the system's computational graph, known as branching-time properties. This thesis investigates whether runtime verification can be extended to meaningfully verify a wider range of branching-time properties. We depart from the classical setup where monitoring is limited to a single execution and investigate the enhanced observational capabilities when monitoring a system over multiple runs. To ensure generality in our results, we focus on branching-time properties expressed in the modal μ-calculus, a well-studied foundational logic used by state-of-the-art model checkers.

As part of a comprehensive theory, the first part of this thesis focuses on deterministic systems and demonstrates that the proposed multi-run monitoring setup can systematically extend previously established monitorability limits for branching-time properties. The second part extends the multi-run framework and corresponding results to systems that may exhibit non-deterministic behaviour. To show that the proposed setup is implementable, the third part outlines the steps for a full automation, as well as proves bounds that capture the correspondence between the syntactic structure of a property and the number of system runs required to adequately verify it. We also validate our results by instantiating the proposed multi-run monitoring setup to verify actor-based concurrent systems, a widely used concurrency paradigm. Given that these systems are inherently non-deterministic, the fourth part presents a general study of a highly non-deterministic calculus, where we systematically explore how the defining characteristics of the actor model enable the recovery of a certain degree of determinism.

Contents

Figures

1 Introduction

Modern systems are becoming increasingly complex and are often integrated in safety-critical domains such as medical devices, autonomous vehicles, and air traffic control systems. Ensuring that they behave as intended under all conditions is imperative as a failure in these systems can have severe consequences. The challenge of making sure that a system meets its specification is known as the *correctness problem* [59, 58, 118, 98, 36, 38] and is one of the most fundamental and long-standing issues in computer science. To address this challenge, systems are often verified statically using well-established techniques such as model checking [51, 34]. Unfortunately, these verification techniques cannot be used whenever the system model is either too expensive to build and analyse (*e.g.* due to the state-explosion problems), poorly understood (*e.g.* when the system logic is partly governed by machine-learning procedures) or downright unavailable (*e.g.* restrictions due to intellectual property rights). As a result, many verification tasks will most likely employ some complementary dynamic analysis technique.

Runtime verification (RV) [36, 100] is a post-deployment technique that *dynamically* checks whether a system-under-scrutiny (SUS) satisfies a correctness property of interest. Instead of analysing the state space, as is done in static verification techniques, RV analyses the *current execution* of the SUS. Correctness properties are typically formalised using a high-level formalism, *e.g.* logic, regular expressions or automata, to unambiguously describe the expected behaviour. From these properties, RV generates computational entities called *monitors*, which are occasionally also termed sequence recognisers [122, 101, 41, 73]. These monitors are *instrumented* to run alongside the SUS and *passively* observe its execution, expressed as a *trace* of events, to determine whether the SUS satisfies or violates the correctness property of interest by issuing corresponding acceptance or rejection *verdicts*. Accordingly, a property is said to be *monitorable for satisfactions* (resp. *violations*) if the monitor can reach an acceptance (resp. rejection) verdict after observing some trace. RV is particularly applicable in settings where traditional static verification techniques cannot be effectively employed, and has therefore emerged as a promising technique [38, 62, 100]. Nevertheless, it is inherently weaker than static verification methods. This is due to the fact that monitor observations are typically constrained to the *current* computation path of the SUS, which limits both the *range* of properties that can be verified, as well as *how* verification itself is conducted. The aim of this thesis is to investigate whether these RV limitations can be mitigated, even if only partially, to enable the verification of a broader range of correctness properties.

Branching-time logics, such as LTL, CTL and CTL* and the μ-calculus [34, 51, 12, 94] are widely used formalisms for specifying correctness properties as they allow reasoning about the *computation graph* of the SUS, where each state may have many possible futures. Recent work has shown that RV can be used effectively (either in isolation or in conjunction with other verification techniques as part of multing-pronged approach) to verify certain branching-time properties [93, 115, 55, 88, 24, 76, 58, 91, 10,

32, 125, 33, 65, 16]. However, prior work [76, 10] has shown that the RV limitations mentioned earlier are even more acute for these kinds of properties.

Example 1.1. Consider the system p that acts as a server that might exhibit four events: receive queries (r), service queries (s), allocate more memory (a), and close connection (c). One correctness property that might be expected of system p might be that

"all interactions can only start with a receive query (r)" (Prop 0)

The specification logic that we use to formalise properties employs the formula $[\alpha]\mathsf{ff}$ to describe system states that *cannot* perform any α transitions, meaning that if the SUS p can exhibit an α action, it violates $[\alpha]\mathsf{ff}$. Using this formalism, property Prop 0 can be expressed as the formula

$$\varphi_0 \stackrel{\text{def}}{=} [s]\mathsf{ff} \wedge [a]\mathsf{ff} \wedge [c]\mathsf{ff}$$

where interactions cannot be service, $[s]\mathsf{ff}$, allocate, $[a]\mathsf{ff}$, or close, $[c]\mathsf{ff}$, actions. It is not hard to see that formula φ_0 can be runtime verified for violations using a single trace. Concretely, any SUS execution observed that starts with event s, a or c confirms that the running SUS violates the property (irrespective of any events that may follow in that execution). This means φ_0 is monitorable in the classical RV setup.

The same cannot be said for the branching-time property stating that

"systems that can perform a receive action (r) cannot *also perform a close action (c)"* (Prop 1)

To formalise Prop 1, our specification logic employs formulae of the form $\langle\alpha\rangle\mathsf{tt}$ to describe states that *can* emit at least one α action. We also say that two formulae φ and ψ are equivalent, denoted $\varphi \equiv \psi$, if they are satisfied by exactly the same set of states. This allows us to express Prop 1 as formula φ_1 below.

$$\varphi_1 \stackrel{\text{def}}{=} \langle r\rangle\mathsf{tt} \Longrightarrow [c]\mathsf{ff} \;\equiv\; [r]\mathsf{ff} \vee [c]\mathsf{ff}$$

To simplify reasoning, we use the equivalent formula $[r]\mathsf{ff} \vee [c]\mathsf{ff}$ [1], which is satisfied if the SUS p cannot produce both r and c events, and violated otherwise. In contrast with φ_0 from Example 1.1, formula φ_1 is monitorable with respect to neither satisfactions nor violations. Specifically, no (single) trace prefix can provide enough evidence for a verifying monitor to conclude that the system p satisfies φ_1. Conversely, an observed trace starting with event r (dually c) is not enough to conclude that system p violates the property: the monitor would also need additional evidence that p can emit event c (dually r). ■

There are various approaches that one could adopt to extend the set of monitorable properties. One method is to weaken the detection requirements expected of the monitors effecting the verification [12, 13] (*e.g.* by flagging all possible violations that can be determined with a finite trace and allowing those that cannot go undetected). This, in turn, impinges on what it means for a property to be monitorable. Another approach is to increase the monitors' observational capabilities. Aceto *et al.* [7] investigate the increased observational power in monitors as a result of augmenting the information recorded in the trace: apart from reporting computational steps *that happened*, they consider trace events that can also record branching information such as the computation steps that *could have* happened at a

[1] This follows from the propositional logic identity $p \Longrightarrow q \equiv \neg p \vee q$ and the duality of $\langle r\rangle\mathsf{tt}$ and $[r]\mathsf{ff}$, *i.e.*, $\neg\langle r\rangle\mathsf{tt} \equiv [r]\mathsf{ff}$.

particular state, or the computation steps that could *not* have happened. This approach treats the SUS as a *grey-box* [125, 132] since the augmented traces reveal *specific* system information about the SUS states reached. This thesis builds on Aceto *et al.*'s work [7] while sticking, as much as possible, to a *black-box* treatment of the SUS. More precisely, we *systematically* study the increase in the observational power of monitors obtained from considering *multiple* execution traces for the same SUS *without* relying directly on information about the specific intermediary states reached during monitoring.

Example 1.2. Property φ_1 from Example 1.1 can be monitored for violations over *two* executions of the same system: a first trace starting with event r and a second trace starting with event c is sufficient evidence for the verifying monitor to conclude that the SUS violates formula φ_1. ■

Observing and analysing multiple traces is *not* always sufficient for the verifying monitor to conclude that a system violates a property with disjunctions since the same trace prefix could, in principle, be emitted by different SUS states. This is better illustrated in Example 1.3.

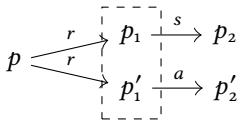
Example 1.3. Consider the property stating that

"after any receive query (r), if a system can service it (s), then (Prop 2)
(this should take precedence and) it should not be able to allocate more memory (a)"

In our specification logic, Prop 2 can be expressed as φ_2 below, where formulae of the form $[\alpha]\varphi$ describe system states where *all* of their α-labelled transitions (possibly none) lead to states that satisfy φ.

$$\varphi_2 \stackrel{\text{def}}{=} [r](\langle s\rangle \mathsf{tt} \Rightarrow [a]\mathsf{ff}) \;\equiv\; [r]([s]\mathsf{ff} \vee [a]\mathsf{ff})$$

Intuitively, φ_2 is violated whenever any state reached after performing event r can perform *both* events s and a. However, observing the trace prefixes rs and ra along two distinct SUS executions is not enough for a monitor to conclude that the SUS violates φ_2. Concretely, although both executions start from the (same) SUS state, say p, distinct intermediary states could have been reached after exhibiting event r:



When $p_1 = p'_1$, the trace prefixes rs and ra share the same r-derivative state, meaning a monitor can conclude that φ_2 is violated since the (SUS) state p_1 can exhibit both events s and a. In contrast, if $p_1 \neq p'_1$, the trace prefixes rs and ra only share the initial state, p, and formula φ_2 is *not* violated. ■

1.1 Contributions Summary and Motivation

We present an augmented monitoring setup that repeatedly analyses the SUS, across multiple executions, and investigate how the notions of determinism can be leveraged to increase the analytical capabilities of the verifying monitor. Based on this, we study how the monitorability limits established in [10, 76] for branching-time properties are affected. The contributions of this thesis are six-fold:

(a) A formalisation of an augmented monitoring setup that gathers information over multiple monitored executions of the same SUS.

(b) An analysis, formalised as a proof system, that uses sets of partial trace prefixes aggregated from the monitoring setup in (a) to runtime verify the SUS against a branching-time property.

(c) Definitions formalising what it means for a monitor to *correctly* analyse a property over multiple runs and, dually, what it means for a property to be *monitorable* over multiple runs.

(d) The identification of an *extended* logical fragment that is monitorable over the augmented monitoring setup handling multiple runs, and a result establishing that it is maximally expressive, *i.e.,* every property that can be monitored for correctly using our multi-run monitors can always be expressed as a formula in this fragment.

(e) An algorithm and a complexity analysis of the proposed multi-run RV framework, showing its implementability, as well as a method for systematically determining lower bounds for the number of SUS executions required to conduct RV from the syntactic structure of the formula being verified.

(f) An instantiation of the proposed multi-run RV framework to actor-based concurrent systems, a widely used concurrency paradigm. Since these systems are inherently non-deterministic, we also present a general study of a highly non-deterministic calculus, where we systematically investigate how the defining characteristics of the actor model enable us to recover a certain degree of determinism.

Sections 1.1.1 to 1.1.4 detail the systematic approach adopted in this thesis to realise each of these contributions, while also highlighting their importance.

1.1.1 Repeated Monitoring for Deterministic Systems

Although non-deterministic SUS behaviour cannot be ruled out in general, there is still a considerable number of systems that behave deterministically, such as single-threaded ones and those type-checked to rule out concurrency races. As a first step towards a comprehensive theory, we limit our study to deterministic systems to curb complications such as those described in Example 1.3.

There are various candidate definitions for determinism, *e.g.* [34, 113]. For instance, one could require that for each possible action η, there is at most one reachable (system) state, *i.e.,* if $p \xrightarrow{\eta} q$ and $p \xrightarrow{\eta} r$ then $q = r$. In certain cases, this definition is too stringent and is relaxed by allowing the system to perform any number of η-labelled transitions, provided that all resulting states, say q and r, are, in some sense, behaviourally equivalent, denoted as $q \equiv r$. Again, what it means for two states to be behaviourally equivalent can be formalised in many different ways. For instance, one might consider two states behaviourally equivalent if they are confluent w.r.t. internal actions, *i.e.,* they can perform different internal transitions, as long as they exhibit the same observable behaviour. The notion of determinism we adopt for this first part of the thesis is of the first kind, *i.e.,* if $p \xrightarrow{\eta} q$ and $p \xrightarrow{\eta} r$ then $q = r$. Under that assumption, observing multiple traces suffices for the monitor conducting RV to infer that the SUS violates a correctness property with disjunctions (see Example 1.3 where $p_1 = p_1'$ for details).

We follow the route outlined in [76]. Concretely, we first present a multi-run monitoring framework for deterministic systems where monitors repeatedly aggregate and analyse traces generated by the same deterministic SUS, corresponding with contributions (a) and (b). In order to pin down a correspondence between the behaviour of the verifying monitor and the semantic meaning of the property it is monitoring for, we formally define what it means for a monitor to *correctly monitor* for a property; prior work [76]

formalised this as single-run monitoring whereas we redefine it as multi-run monitoring. Monitor correctness then serves as the basis for a concrete definition of *monitorability*, which is a function of the underlying monitoring setup, resulting in contribution (c). Finally, we identify which properties are monitorable at runtime, resulting in contribution (d). This delineation directly guides the decision on whether to employ RV or some alternative verification technique. It is also useful for the construction of RV tools, as automated monitor synthesis can exclusively target the identified monitorable logic, with the assurance that all monitorable properties are still covered.

1.1.2 Repeated Monitoring for Non-Deterministic Systems

Since our ultimate goal is to runtime verify *any* system, even if it might exhibit certain non-deterministic behaviour, the second step is to extend our study to potentially non-deterministic systems. We start by observing that certain property violations can be detected by a verifying monitor even in the presence of non-deterministic actions. This is illustrated in Example 1.4 below.

Example 1.4. Consider the property that, in addition to the behaviour described by φ_2, requires that

> *"…the SUS does* not *exhibit any action after a close event"* (Prop 3)

Using the logical constructs described in Examples 1.1 to 1.3, Prop 3 can be expressed as φ_3 below.

$$\varphi_3 \stackrel{\text{def}}{=} \big([r]([s]\mathrm{ff} \vee [a]\mathrm{ff})\big) \wedge \big([c]([r]\mathrm{ff} \wedge [s]\mathrm{ff} \wedge [a]\mathrm{ff} \wedge [c]\mathrm{ff})\big)$$

There are cases where it might be reasonable to assume that a SUS behaves deterministically for receive actions (*e.g.* when a single thread is in charge of receiving queries). As already shown in Example 1.3, whenever this assumption cannot be made, a verifying monitor cannot detect violations of the subformula $\varphi_2 \stackrel{\text{def}}{=} [r]([s]\mathrm{ff} \vee [a]\mathrm{ff})$. However, *no* determinism assumption is required for close actions (c) to runtime verify the subformula $[c]([r]\mathrm{ff} \wedge [s]\mathrm{ff} \wedge [a]\mathrm{ff} \wedge [c]\mathrm{ff})$. Thus, any trace prefix from either cr, cs, ca or cc suffices to infer that the SUS violates the property φ_3. ■

Since correctness properties are only concerned with the observable behaviour of the system, we relax the determinism definition adopted in Section 1.1.1 by allowing non-determinism as long as it internal. In particular, we require that all states reached after performing a η-labelled transition to be behaviourally equivalent, *i.e.,* if $p \xrightarrow{\eta} q$ and $p \xrightarrow{\eta} r$ then $q \equiv r$, where $q \equiv r$ whenever q and r are confluent w.r.t. internal actions. Although non-deterministic SUS behaviour cannot be ruled out in general, many classes of LTSs are deterministic w.r.t. a *subset* of actions, such as asynchronous LTSs and output actions [123, 89] (*e.g.* if r was an asynchronous output in Example 1.3 then $p_1 \equiv p_1'$.)

Based on these observations, we extend the multi-run monitoring setup of Section 1.1.1 to potentially non-deterministic systems by augmenting the instrumentation mechanism coupling the behaviour of an executing SUS with that of the verifying monitor in two ways. First, our augmented instrumentation equips the monitor with additional information about the determinism of each SUS action; this information can be obtained a priori by analysing all the possible actions that can be emitted by a certain class of SUSs. This would allow a verifying monitor to determine whether the SUS performed a non-deterministic branching, *e.g.* the monitor in Example 1.3 would be able to tell whether the (system) states p_1 and p_1' can exhibit the same (observable) behaviour. Second, since a system could also internally

branch, our augmented instrumentation makes certain internal computations of the SUS behaviour visible to the monitor. This permits the monitor to differentiate between confluent internal moves and internal moves that potentially lead to non-deterministic branching.

We observe that although the augmented instrumentation equips the monitor with additional information about the SUS, our treatment of the SUS is still black-box for two reasons. First, the information on internal actions, together with the determinism properties of all actions, characterises an *infinite* class of systems, not one specific SUS. Second, the determinism guarantees permit a verifying monitor to reason about states within the same equivalence class (*i.e.,* $q \equiv r$) but not directly on specific SUS states.

We adopt the same approach as in Section 1.1.1. Concretely, we start by extending the multi-run monitoring framework of Section 1.1.1 to potentially *non-deterministic* systems, using the described augmented instrumentation mechanism to increase the analytical capabilities of the verifying monitors. Based on this, we study how the monitorability limits identified in [10, 76] are affected. Again, these result in contributions (a) to (d), described in detail in Section 1.1.1, but in the context of potentially non-deterministic systems.

1.1.3 Practical Aspects

The theoretical results in Section 1.1.2 raise several practical questions. The first one is whether the proposed verification technique lends itself well to the implementation of a tool that runtime verifies systems over multiple executions. Although we do not provide a RV tool that shows that the proposed verified technique is indeed implementable, we outline the steps towards a full automation and give a corresponding complexity analysis. In order to attain an efficient implementation, we observe that the traces recorded by the verifying monitor should not be analysed unnecessarily. For this reason, we would also like to understand better the correlation between the structure of the specification formula expressing the property (*e.g.* the number of disjunctions used) and the number of SUS executions required to monitor adequately for that property. This results in contribution (e) outlined above.

Another crucial practical concern is whether the proposed verification technique can be used to runtime verify real-world systems. We address this by outlining a possible instantiation of our multi-run monitoring framework to actor-based systems, a widely used concurrency paradigm that has been adopted by numerous programming languages, including Erlang [48], Akka [79], Elixir [90] and Swift [29]. This results in the completion of the first part of contribution (f).

1.1.4 Confluence and Determinism in Concurrent Systems

The multi-run RV setup proposed in Section 1.1.2 is underpinned by an instrumentation that largely relies on the notions of *determinism* and *confluence*. In particular, we recall that our instrumentation (i) makes certain internal computation of system behaviour visible to the monitor to allow it to differentiate between *confluent* internal moves and moves that potentially lead to non-deterministic branching, and (ii) provides the monitor with information about whether the observed actions are *deterministic*.

For this reason, we would also like to examine these two notions in a wider context. In particular, confluence and determinism have been studied extensively in the context of the π-calculus [78, 105, 106, 80, 113, 126, 116, 111, 81, 103, 130], a touchstone model for expressing concurrent systems. Although the π-calculus is highly general and capable of describing a wide range of computational patterns,

concurrency in modern systems is often more structured. For instance, another prominent concurrent paradigm is Hewitt's actor model [87], later formalised by Agha in [22]; we have already shown that our multi-run RV framework can be instantiated to actor-based systems in Section 1.1.3. Despite its conceptual similarities with the π-calculus, the actor model has several key characteristics that make it easier to distribute and implement in real-world systems [69].

We systematically investigate how each of these defining characteristics of the actor model affects confluence and determinism of concurrent systems, resulting in the completion of contribution (f). Concretely, we depart from the π-calculus and incrementally introduce *semantic* constraints that capture these defining actor properties, assessing whether they provide any confluence or determinism guarantees at each step. Since semantic conditions are, in general, undecidable, we also identify *syntactic* conditions that guarantee their semantic counterparts.

1.2 Structure of the thesis

As part of a comprehensive theory, we present this thesis in four parts, which respectively correspond to Sections 1.1.1 to 1.1.4. We briefly describe the contents of each chapter.

Chapter 2 introduces the classical RV setup and the system language that we use throughout the first part of this thesis. It also formalises the branching-time specification logic called Hennessy-Milner Logic with Recursion [97] (recHML), used to describe correctness properties about the behaviour of the SUS. We conclude by giving a syntactic characterisation for the subset of recHML formulae that are monitorable for *violations* with one system execution.

Part I: Repeated Monitoring for Deterministic Systems

Chapter 3 presents an extended monitoring setup designed to operate over multiple runs of the same deterministic SUS. It formalises the operational model of the monitors, detailing how they record SUS traces, and an instrumentation relation connecting their operational behaviour with that of the running SUS. This chapter concludes by defining how the aggregated traces are rejected by a monitor through a proof system.

Chapter 4 provides several definitions of what it means for a monitor to correctly verify a property. Among the most fundamental criteria are *soundness* and *completeness*, respectively stating that the monitor never report false violations and that it flags all system violations.

Chapter 5 gives a concrete definition for *monitorability*, which is parametric w.r.t. the notion of correct monitoring from Chapter 4. Based on this, we examine how the monitorability limits identified in [10, 76] are affected by considering multiple executions of a deterministic SUS.

Part II: Repeated Monitoring for Non-deterministic Systems

Chapter 6 revisits the system language presented in Chapter 2, lifting the determinism assumption and enriching the system model with additional properties.

The rest of this part of the thesis is organised as in Part I.

Chapter 7 extends Chapter 3 by augmenting the instrumentation relation to make certain internal SUS actions visible to the monitor, while also forwarding information about the determinism of all the

observed actions. This chapter concludes by redefining how a monitor, privy to this additional SUS information, rejects the aggregated traces through a proof system.

Chapter 8 revisits the notion of monitor correctness in the context of the extended multi-run monitoring setup of Chapter 7.

Chapter 9 recasts the notion of monitorability in terms of the redefined definition for monitor correctness. Based on this, it studies how the identified monitorability limits [10, 76] are affected when analysing multiple executions of a potentially non-deterministic SUS.

Part III: Practical Aspects

Chapter 10 shows that the verification technique proposed in Part II lends itself well to the implementation of an RV tool that analyses a system over multiple runs. This is done by first outlining the steps towards a full automation and then giving a corresponding complexity analysis. This chapter concludes by presenting a method for systematically determining lower bounds for the number of SUS executions required to conduct RV from the syntactic structure of the formula being verified.

Chapter 11 formalises the language for actor-based concurrent systems and shows how they can be runtime verified using the proposed multi-run monitoring framework.

Part IV: Confluence and Determinism in Concurrent Systems

Chapter 12 introduces three defining characteristics of the actor model. It also presents the system language that we will use throughout this part of the thesis, namely a standard version of the synchronous π-calculus with name matching. This chapter concludes by formally defining the notions of confluence and determinism, adapted from Philippou *et al.*'s work in [113].

Chapter 13 incrementally introduces *semantic* constraints capturing the characteristic in Chapter 12 to the π-calculus and assesses whether they provide any confluence or determinism guarantees at each step. To validate the realisability of these constraints, this chapter also identifies a number of *syntactic* conditions that capture their semantic counterparts.

1.3 Scope of the Study

The properties discussed thus far, as well as those considered for the rest of our study, are formalised in terms of a variant of the modal μ-calculus [94] called Hennessy-Milner Logic with Recursion [97], RECHML. This logic is a natural choice for describing branching-time properties and is employed by state-of-the-art model checkers, including MCRL2 [54] and UPPAAL [39], as well as a stable RV tool for specifying and verifying asynchronous component systems[2]. It is a highly-expressive formalism that has been shown to embed other standard logics such as LTL, CTL and CTL* [34, 51]. For instance, prior work [10, 12] provided translations from LTL to a linear-time interpretation of RECHML, which indicates that our results can be carried over to these logics. In addition, the chosen logic allows for continuity by building upon existing maximality results for branching-time logics [76, 7, 13], which have only been established for the modal μ-calculus. Our exposition focusses on "safety" properties that can be monitored for violations; monitoring for satisfactions of branching-time properties is symmetric [76].

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

2 Preliminaries

In this chapter, we present the main technical machinery that we use throughout this thesis. In particular, we formalise the system model and the specification logic we will use to specify correctness properties of interest. We also show that constraining monitor observations to the current (single) computation path of the SUS—as is standard in the classical RV setting—severely limits the range of properties that can be runtime verified.

Structure of the chapter. In Section 2.1, we introduce the system language and endow it with an operational semantics that is defined in terms of an LTS. In Section 2.2, we formalise the specification logic, a variant of the modal μ-calculus [94] called Hennessy-Milner Logic with Recursion [97], recHML, and give it a branching-time interpretation. In Section 2.3, we revisit Aceto *et al.*'s [76] notion of monitorability, and demonstrate, through a series of pathological examples, that a property may either be monitored for satisfactions or violations, but not both; in the rest of this thesis, we focus on the latter as monitoring for satisfactions is symmetric. We also present an important result from [76] stating that the subset of recHML properties that are monitorable for violations corresponds to a syntactic fragment called sHML. Section 2.4 concludes.

2.1 The Model

We presuppose a finite set of actions $\eta,\xi,\ldots \in \textsc{Act} = \textsc{eAct} \cup \{\tau\}$, with a set of *external actions* $\alpha,\beta,\ldots \in \textsc{eAct}$ and a distinguished silent action $\tau \notin \textsc{eAct}$.

A SUS is modelled as a Labelled Transition System (LTS) [92], a triple of the form

$$\langle\, \textsc{Prc}, \textsc{Act}, \longrightarrow \,\rangle$$

where SUS states are denoted by a set of processes $p,q,\ldots \in \textsc{Prc}$ and $\longrightarrow \subseteq (\textsc{Prc} \times \textsc{Act} \times \textsc{Prc})$ is a ternary transition relation between SUS states labelled by actions from Act. We also write $p \xrightarrow{\eta} q$ instead of $(p,\eta,q) \in \longrightarrow$, and $p \overset{\eta}{\not\longrightarrow}$ whenever $\nexists q$ such that $p \xrightarrow{\eta} q$. Weak transitions, $p \overset{\alpha}{\Longrightarrow} q$, denote that p can transition to q with a single α action and a number (possibly zero) of τ actions, *i.e.,* $p(\xrightarrow{\tau})^* \xrightarrow{\alpha} (\xrightarrow{\tau})^* q$, referring to q as the α-derivative of p. Systems in this LTS are deterministic, that is:

$$\text{for any } \eta \in \textsc{Act}, \text{ if } p \xrightarrow{\eta} q \text{ and } p \xrightarrow{\eta} r \text{ then } q = r$$

External actions can be sequenced to form *traces*, $t,u \in \textsc{Trc} = \textsc{eAct}^*$, representing finite prefixes of system runs. As usual, ϵ denotes the empty trace. A trace with action α at its head and continuation t is denoted as αt, whereas a trace with prefix t and action α at its tail is denoted as $t\alpha$. For trace $t = \alpha_1 \cdots \alpha_n$, we write $p \overset{t}{\Longrightarrow} q$ instead of the sequence of transitions $p \overset{\alpha_1}{\Longrightarrow} \cdots \overset{\alpha_n}{\Longrightarrow} q$. A system p *produces* trace t if $\exists q$

such that $p \overset{t}{\Longrightarrow} q$, where the set of all the traces produced by p is denoted by $T_p \subseteq \textsc{Trc}$. *Histories*, denoted $H \in \textsc{Hst}$ where $\textsc{Hst} \subseteq \textsc{Trc}$, are finite sets of traces. We also write H,t as a shorthand for the disjoint union $H \uplus \{t\}$ *i.e.,* $H \cap \{t\} = \emptyset$.

Throughout this thesis, we often denote systems $p,q \in \textsc{Prc}$ using standard CCS notation (Definition 2.1).

Definition 2.1 (CCS notation [106]). The regular fragment of CCS is inductively defined as follows:

$$p,q \in \textsc{Prc} \ ::= \ \mathbf{0} \quad \mid \eta.p \quad \mid p+q \quad \mid \mathrm{rec}X.p \quad \mid X$$

where $X, Y, \ldots$ are from some countably infinite set of recursion variables $\textsc{TVars}$ and $\eta \in \textsc{Act}$. The transition relation $\longrightarrow$ is defined as the least relation satisfying the following rules:

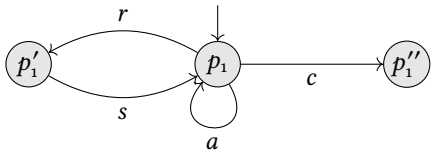
$$\textsc{act}\ \frac{}{\eta.p \xrightarrow{\eta} p} \qquad \textsc{rec}\ \frac{}{\mathrm{rec}X.p \xrightarrow{\tau} p[\mathrm{rec}X.p/X]} \qquad \textsc{selL}\ \frac{p \xrightarrow{\eta} p'}{p+q \xrightarrow{\eta} p'} \qquad \textsc{selR}\ \frac{q \xrightarrow{\eta} q'}{p+q \xrightarrow{\eta} q'}$$ ■

Remark 2.2. Rule $\textsc{rec}$ in Definition 2.1 differs from the standard CCS recursion rule, which derives transitions directly from the unfolded process. In contrast, we represent recursion explicitly through an internal τ-transition corresponding to one unfolding step [76, 109]. ■

Example 2.3. Consider the system p_1 below (formalised as a CCS process) that describes a possible implementation of the server in Example 1.1.

$$p_1 \overset{\text{def}}{=} \mathrm{rec}X.(r.s.X + a.X + c.\mathbf{0})$$

System p_1 repeatedly receives queries (r) and services (s) them. After any number of serviced queries (*i.e.,* alternating r and s actions), it may also allocate more memory (a) before servicing more received queries or close the connection (c). The directed graph below depicts the LTS for p_1 where nodes represent system states and arcs represent weak system transitions between states. For instance, the arc from p_1 to p_1' represents the transitions $p_1 \xrightarrow{\tau} r.s.p_1 + (a.p_1 + c.\mathbf{0}) \xrightarrow{r} s.p_1$, *i.e.,* $p_1 \overset{r}{\Longrightarrow} s.p_1$ where $p_1' \overset{\text{def}}{=} s.p_1$.



We note that these graphs do *not* describe the full branching structure of the depicted systems as they abstract away from τ-derivatives [1], such as $r.s.p_1 + (a.p_1 + c.\mathbf{0})$ for system p_1. This omission is intentional: such graphs only serve as an abstract depiction of LTSs to better explain the intuitions behind the logical correctness properties that follow. ■

[1] Internal (silent) transitions are treated as unobservable and omitted, so only visible actions are represented.

2.2 The Specification Logic

Correctness properties are formulated for the *external* SUS view in a variant of the modal μ-calculus [94] called Hennessy-Milner Logic with Recursion [97], RECHML. This logic is defined by the negation free grammar in Figure 2.1, which assumes a countably infinite set of formula variables $X, Y, \ldots \in \text{TVARS}$. Apart from the standard constructs for truth, falsity, conjunction and disjunction, this logic includes existential modalities, $\langle\alpha\rangle\varphi$, and universal modalities, $[\alpha]\varphi$, that operate over the set of *external* actions EACT, *i.e.,* $\alpha \in \text{EACT}$, and respectively express the dual notions of *possibility* and *necessity*. Least and greatest fixed points, $\min X.\varphi$ and $\max X.\varphi$ respectively, bind free instances of variable X in φ.

In the sequel, we assume standard definitions for open and closed formulae and work up to α-conversion, assuming formulae to be closed and guarded, unless otherwise stated. A formula is said to be *guarded* if every variable X appears within the scope of a modality that is itself in the scope of X. *E.g.* the formula $\max X.[\alpha]\mathsf{ff} \wedge [\beta]X$ is guarded but $\max X.[\alpha]\mathsf{ff} \wedge X$ is not. For formulae φ and ψ, and variable X, we use $\varphi[\psi/X]$ to denote the capture-avoiding substitution of all free occurrences of X in φ with ψ.

The denotational semantics function $[\![-]\!]$ in Figure 2.1 gives a *branching-time* interpretation to RECHML by mapping formulae to sets of system states, $[\![-]\!] : \text{RECHML} \to \mathcal{P}(\text{PRC})$. This function is defined with respect to an *environment* ρ, which maps formula variables to sets of states, $\rho : X \to \mathcal{P}(\text{PRC})$. Given an environment ρ, variable X and a set of states P, we write $\rho[X \mapsto P]$ for the environment mapping X to P, acting like ρ and agreeing with ρ on all other variables. The cases for truth, falsity, conjunction and disjunction are standard; all systems satisfy tt but no system satisfies ff, conjunctions $\varphi \wedge \psi$ are satisfied by systems that satisfy both sub-formulae φ and ψ, whereas disjunctions $\varphi \vee \psi$ are satisfied by systems that satisfy either φ or ψ (or both). Existential modalities $\langle\alpha\rangle\varphi$ denote the set of system states that can perform *at least one* α-labelled transition and reach a state that satisfies the continuation φ. Conversely, universal modalities $[\alpha]\varphi$ denote the set of systems that reach states satisfying φ for *all* (possibly none) their α-transitions. The set of systems that satisfy least fixed point formulae (resp. greatest fixed point) is given by the intersection (resp. union) of all pre-fixed points (resp. post-fixed points) of the function induced by the corresponding binding formula. Since the interpretation of closed formulae is independent of ρ, we write $[\![\varphi]\!]$ in lieu of $[\![\varphi, \rho]\!]$ for closed φ. A state p *satisfies* φ if $p \in [\![\varphi]\!]$ and *violates* it if $p \notin [\![\varphi]\!]$. Two formulae φ and ψ are said to be equivalent, denoted as $\varphi \equiv \psi$, whenever $[\![\varphi]\!] = [\![\psi]\!]$. The negation of a formula can be obtained by duality in the usual way, *e.g.* $\neg(\min X.\langle\alpha\rangle\mathsf{tt} \vee [\beta]X) = \max X.[\alpha]\mathsf{ff} \wedge \langle\beta\rangle X$.

We highlight two basic RECHML formulae that are used throughout the rest of the thesis: the formula $\langle\alpha\rangle\mathsf{tt}$ describes systems that can produce action α, whereas $[\alpha]\mathsf{ff}$ describes systems that *cannot* produce action α, as no system can perform an α-labelled transition and reach a state that satisfies ff. We also note that apart from the usual identities, *e.g.* $\mathsf{tt} \wedge \varphi \equiv \varphi$ and $\mathsf{tt} \vee \varphi \equiv \mathsf{tt}$, we also have $\langle\alpha\rangle\mathsf{ff} \equiv \mathsf{ff}$ and $[\alpha]\mathsf{tt} \equiv \mathsf{tt}$.

Example 2.4. Consider an additional property for the system of Examples 1.1 to 1.3, stating that

> *"after any number of serviced queries,* $[r][s]\ldots$*, a state that can close a connection,* $\langle c\rangle tt$*,* cannot *also allocate memory,* $[a]ff$*"*

Using the specification logic presented in Figure 2.1, this property can be formalised by the RECHML formula φ_4 below:

$$\varphi_4 \stackrel{\text{def}}{=} \max X.\big([r][s]X \wedge (\langle c\rangle\mathsf{tt} \Rightarrow [a]\mathsf{ff})\big) \;\equiv\; \max X.\big([r][s]X \wedge ([c]\mathsf{ff} \vee [a]\mathsf{ff})\big)$$

recHML Syntax

$$
\begin{array}{lllll}
\varphi,\psi \in \textsc{recHML} ::= & \mathsf{tt} & \text{(truth)} & \mid \mathsf{ff} & \text{(falsehood)} \\
\mid & \varphi \vee \psi & \text{(disjunction)} & \mid \varphi \wedge \psi & \text{(conjunction)} \\
\mid & \langle\alpha\rangle\varphi & \text{(existential modality)} & \mid [\alpha]\varphi & \text{(universal modality)} \\
\mid & \mathsf{min}X.\varphi & \text{(least fixed point)} & \mid \mathsf{max}X.\varphi & \text{(greatest fixed point)} \\
\mid & X & \text{(recursion variable)} & &
\end{array}
$$

Branching-Time Semantics

$$
\begin{array}{ll}
[\![\mathsf{tt},\rho]\!] \stackrel{\text{def}}{=} \textsc{Prc} & [\![\mathsf{ff},\rho]\!] \stackrel{\text{def}}{=} \emptyset \\
[\![[\alpha]\varphi,\rho]\!] \stackrel{\text{def}}{=} \{p \mid \forall q \cdot p \stackrel{\alpha}{\Longrightarrow} q \text{ implies } q \in [\![\varphi,\rho]\!]\} & [\![\varphi \vee \psi,\rho]\!] \stackrel{\text{def}}{=} [\![\varphi,\rho]\!] \cup [\![\psi,\rho]\!] \\
[\![\langle\alpha\rangle\varphi,\rho]\!] \stackrel{\text{def}}{=} \{p \mid \exists q \cdot p \stackrel{\alpha}{\Longrightarrow} q \text{ and } q \in [\![\varphi,\rho]\!]\} & [\![\varphi \wedge \psi,\rho]\!] \stackrel{\text{def}}{=} [\![\varphi,\rho]\!] \cap [\![\psi,\rho]\!] \\
[\![\mathsf{min}X.\varphi,\rho]\!] \stackrel{\text{def}}{=} \bigcap \{P \mid [\![\varphi,\rho[X \mapsto P]]\!] \subseteq P\} & [\![\mathsf{max}X.\varphi,\rho]\!] \stackrel{\text{def}}{=} \bigcup \{P \mid P \subseteq [\![\varphi,\rho[X \mapsto P]]\!]\} \\
[\![X,\rho]\!] \stackrel{\text{def}}{=} \rho(X) &
\end{array}
$$

Figure 2.1. *Syntax and Semantics of recHML in the branching-time setting.*

The greatest fixed point in φ_4 captures the intuition that a system invariantly satisfies the sub-formula $[c]\mathsf{ff} \vee [a]\mathsf{ff}$ in all of the states that can be reached after any sequence of alternating r and s actions (*i.e.,* serviced queries). This means that a system violates φ_4 if it is capable of producing *both* a and c action after an unbounded, but *finite*, sequence of r and s actions. For instance, system p_1 from Example 2.3 clearly violates this property, *i.e.,* $p_1 \notin [\![\varphi_4]\!]$, since $p_1 \stackrel{a}{\Longrightarrow} p_1$ and $p_1 \stackrel{c}{\Longrightarrow} \mathbf{0}$. ■

2.3 Monitorability in the Classical RV Setup

The classical RV setup can be decomposed into three components: the SUS, the monitor conducting the verification, and the instrumentation coupling the monitor with the SUS. At runtime, instrumentation permits the monitor m to observe the *current* execution of the SUS p until it detects certain behaviour and either *accepts* or *rejects* the SUS, as defined below [2].

Definition 2.5 (Monitor Acceptance and Rejection). Given system $p \in \textsc{Prc}$ and monitor $m \in \textsc{Mon}$,

- m *accepts* p, denoted $\mathbf{sacc}(p,m)$, if p can produce some trace t that leads m to an acceptance verdict.
- m *rejects* p, denoted $\mathbf{srej}(p,m)$, if p can produce some trace t that leads m to a rejection verdict. ■

Importantly, once the monitor reaches a verdict, this verdict is *irrevocable* [9]. This means that the monitor *cannot* alter its conclusion, regardless of system behaviour it might observe in the future.

[2] In [76, 10], monitor acceptances and rejections are denoted by the predicates **acc** and **rej**. Here, we use **sacc** and **srej** to make it explicit that the verdict is reached after observing a *single* execution. This allows us to clearly distinguish these predicates from those used in Part I, which describe acceptances/rejections reached after observing *multiple* system executions.

To understand which properties are monitorable w.r.t. the classical RV setup, where monitoring is limited to the current system execution, we first need to define what we understand by the statement

"Monitor m *correctly monitors* for property φ"

In our setting, this translates to a correspondence between monitor acceptances ($\mathbf{sacc}(p,m)$) and rejections ($\mathbf{srej}(p,m)$) to property satisfactions ($p \in [\![\varphi]\!]$) and violations ($p \notin [\![\varphi]\!]$). This allows us to determine if a monitor, in some sense, adequately represents a property.

The least correctness requirement expected of monitors is *soundness* of Definition 2.6, stating that monitor acceptances (resp. rejections) imply property satisfactions (resp. violations). Concretely, a monitor m monitors *soundly* for a property φ whenever for *any* monitored system p, if a monitor accepts (resp. rejects) this system, *i.e.,* $\mathbf{sacc}(p,m)$, then it must be the case that p satisfies (resp. violates) the property φ, *i.e.,* $p \in [\![\varphi]\!]$. This ensures that the verdicts reached by the monitors do not contradict the semantic meaning of the property they are monitoring for.

Definition 2.6 (Soundness [76]). Monitor m monitors *soundly* for the formula φ if for any $p \in \textsc{Prc}$,

- $\mathbf{sacc}(p,m)$ implies $p \in [\![\varphi]\!]$
- $\mathbf{srej}(p,m)$ implies $p \notin [\![\varphi]\!]$ ■

A few comments about Definition 2.6 are in order. First, the universal quantification over all systems (*i.e.,* for any system p) manifests our black-box treatment of the SUS. Second, it also rules out contradicting monitor verdicts. Concretely, suppose monitor m monitors soundly for φ. If it accepts a system p, *i.e.,* $\mathbf{sacc}(p,m)$ then by Definition 2.6, we have that $p \in [\![\varphi]\!]$. By the contrapositive of Definition 2.6, we also have that $\neg\mathbf{srej}(p,m)$. A symmetric argument applies for monitor rejections.

Soundness is arguably the least requirement for relating a monitor with a property. Further to this, one might also require the dual of Definition 2.6, *i.e., completeness* for m and φ, where all property satisfactions (resp. violations) imply monitor acceptances (resp. rejections). Such a requirement turns out to be too restrictive for the logic in Figure 2.1: as shown in [76, Theorem 2], monitors that can perform *both* acceptances and rejections are necessarily *unsound.* This is better illustrated in Example 2.7.

Example 2.7. Consider the basic formula $[\alpha]\mathsf{ff}$. A sound monitor m for it would reject any violating system $p \notin [\![[\alpha]\mathsf{ff}]\!]$ but it *cannot* accept any satisfying systems. Assume, by way of contradiction, that $q \in [\![[\alpha]\mathsf{ff}]\!]$ implies $\mathbf{sacc}(q,m)$. Pick any satisfying system $q \in [\![[\alpha]\mathsf{ff}]\!]$, *i.e.,* $q \overset{\alpha}{\not\longrightarrow}$. By our assumption, we have $\mathbf{sacc}(q,m)$. By definition, we know that m reaches an acceptance verdict after observing some trace t produced by p, *i.e.,* $q \overset{t}{\Longrightarrow} q'$. From q, we can build a violating system $p = q + \alpha.\mathbf{0} \notin [\![[\alpha]\mathsf{ff}]\!]$. But since p can also produce trace t, *i.e.,* $p \overset{t}{\Longrightarrow} q'$, then m must also accept it, *i.e.,* $\mathbf{sacc}(p,m)$. This makes m unsound since it can *both* (incorrectly) accept and reject the violating system $p = q + \alpha.\mathbf{0}$, contradicting the initial assumption that m is sound (apart from being complete). ■

We therefore require a weaker form of completeness, where a monitor is either complete w.r.t. property satisfactions or property violations but *not* both. For the rest this thesis, we focus on the latter; monitoring for satisfactions is symmetric [76]. In what follows, we omit the term (rejection-) in Definition 2.8.

Definition 2.8 (Completeness [76]). Monitor m monitors *(rejection-)completely* for the (closed) formula φ if for any system $p \in \textsc{Prc}$, $p \notin [\![\varphi]\!]$ implies $\mathbf{srej}(p,m)$. ■

Soundness (Definition 2.6) and completeness (Definition 2.8) characterise the relation between monitors and property in the classical RV setup. Together, they constitute the basis for the definition of monitor correctness in this setting, formalised in Definition 2.9 below.

Definition 2.9 (Monitor Correctness [76]). Monitor m monitors *correctly* for the (closed) formula φ if it can do so soundly and completely. ■

Monitorability [62, 76, 36, 12] describes the ability of properties to be adequately runtime verified by a corresponding monitor. This means that the definition of monitorability is *dependent* on the underlying monitoring setup, which is especially relevant to our investigation on the increase in expressive power when moving from single-run monitoring to multi-runs; the interested reader should consult [12] for an extensive comparison between various notions of monitorability found in the literature.

Definition 2.10 (Monitorability [76]). A formula $\varphi \in \text{recHML}$ is *monitorable* iff there exists a monitor that monitors correctly for it. ■

Several logical formulae from Figure 2.1 are not monitorable (for violations) w.r.t. a classical RV setup that is limited to one execution of the system. This claim is substantiated by the following example.

Example 2.11. According to Definition 2.10, the basic formula $\langle\alpha\rangle\mathsf{tt}$ is *not* monitorable. Assume, by way of contradiction, that there exists a monitor m that monitors soundly and completely for it. Consider the violating system $\mathbf{0} \notin [\![\langle\alpha\rangle\mathsf{tt}]\!]$. Since m is complete for $\langle\alpha\rangle\mathsf{ff}$, then $\mathbf{srej}(\mathbf{0},m)$. From the structure of the system, m could have only rejected $\mathbf{0}$ after observing the empty trace ϵ. Since monitor verdicts are irrevocable, then m must also reject the *satisfying* system $\alpha.\mathbf{0}$, *i.e.,* $\mathbf{srej}(\alpha.\mathbf{0},m)$, as it can also produce the trace ϵ by not transitioning. This makes m unsound, contradicting our initial assumption.

Following a similar argument, we can also show that the disjunction formula $[\alpha]\mathsf{ff} \vee [\beta]\mathsf{ff}$ is *not* monitorable according to Definition 2.10. Assume, towards a contradiction, that there exists a monitor m that monitors soundly and completely for it. Since $\alpha.\mathbf{0}+\beta.\mathbf{0} \notin [\![[\alpha]\mathsf{ff} \vee [\beta]\mathsf{ff}]\!]$ then $\mathbf{srej}(\alpha.\mathbf{0}+\beta.\mathbf{0},m)$. From the structure of the system, m could have only rejected $\alpha.\mathbf{0}+\beta.\mathbf{0}$ after observing either of the traces ϵ, α or β. For the first case, we would also have $\mathbf{srej}(\mathbf{0},m)$ since $\mathbf{0}$ can also produce the empty trace ϵ via the transition $\mathbf{0} \stackrel{\epsilon}{\Rightarrow} \mathbf{0}$. For the second case, we would also have $\mathbf{srej}(\alpha.\mathbf{0},m)$ since $\alpha.\mathbf{0}$ can also produce the trace α with $\alpha.\mathbf{0} \stackrel{\alpha}{\longrightarrow} \mathbf{0}$. Similarly, for the third case, we would also have $\mathbf{srej}(\beta.\mathbf{0},m)$ since $\beta.\mathbf{0} \stackrel{\beta}{\longrightarrow} \mathbf{0}$. These make m unsound since the systems $\mathbf{0}$, $\alpha.\mathbf{0}$ and $\beta.\mathbf{0}$ all satisfy $[\alpha]\mathsf{ff} \vee [\beta]\mathsf{ff}$.

Similarly, the least fixed point formula $\min X.([\alpha]X \wedge [\beta]\mathsf{ff})$, describing systems that eventually stop producing action α and never produce action β, is *not* monitorable for violations. Assume, towards a contradiction, that there exists a monitor m that monitors soundly and completely for it. The single state system p with the sole transition $p \stackrel{\alpha}{\longrightarrow} p$ violates the formula since it continues performing action α indefinitely. By Definition 2.8, we must have $\mathbf{srej}(p,m)$. Since rejections are based on *finite* observations, from the structure of p, we also know that m must have rejected p after observing a *finite* trace $t = \alpha\cdots\alpha$ where $|t| = k$. Consider the system q consisting of $k+1$ states that exclusively has the transitions $q = q_0 \stackrel{\alpha}{\longrightarrow} \cdots \stackrel{\alpha}{\longrightarrow} q_{k+1}$ (and nothing else). Clearly, this system satisfies $\min X.([\alpha]X \wedge [\beta]\mathsf{ff})$, *i.e.,* $q \in [\![\min X.([\alpha]X \wedge [\beta]\mathsf{ff})]\!]$. However, q can also produce trace t, which means that m must also reject it, *i.e.,* $\mathbf{srej}(q,m)$, contradicting the initial assumption that m is sound. ■

Prior work has shown that the subset of (violation) monitorable recHML formulae is characterised by the syntactic fragment sHML , describing the set of safety properties [25] *i.e.,* "something bad does not happen". This means that every formula that can be monitored for correctly in the classical RV setup (with a single execution), then it can always be expressed as a formula in this fragment.

Theorem 2.12 (Safety Fragment [76]). Any formula $\varphi \in$ recHML is *monitorable* (for violations) iff $\exists \psi \in$ sHML and $\varphi \equiv \psi$ where the syntactic fragment sHML defined as follows:

$$\varphi, \psi \in \text{sHML} \ ::= \quad \mathsf{tt} \quad | \quad \mathsf{ff} \quad | \quad [\alpha]\varphi \quad | \quad \varphi \wedge \psi \quad | \quad \max X.\varphi \quad | \quad X$$ ■

At an intuitive level, the fragment sHML captures all recHML properties such that any violating system can produce at least one witness trace that enables a verifying monitor to detect the violation.

Example 2.13. Recall $\varphi_4 \stackrel{\text{def}}{=} \max X.\big([r][s]X \wedge ([c]\mathsf{ff} \vee [a]\mathsf{ff})\big)$ from Example 2.4. According to Theorem 2.12, this formula is *not* monitorable for violations in the classical monitoring setup since $\varphi_4 \notin$ sHML. For instance, even though Example 2.4 showed that $p_1 \stackrel{\text{def}}{=} \mathrm{rec}X.\big(r.s.X + (a.X + c.\mathbf{0})\big)$ from Example 2.3 violates this property, *i.e.,* $p_1 \notin [\![\varphi_4]\!]$, no single trace prefix provides enough evidence for a monitor to detect this. We could, however, detect this over two executions, *e.g.* $p_1 \stackrel{rsc}{\Longrightarrow}$ and $p_1 \stackrel{rsa}{\Longrightarrow}$. ■

2.4 Summary

This chapter introduced the system language and the branching-time logic recHML. It also showed that the subset of monitorable recHML formulae is characterised by the syntactic fragment sHML. As indicated by the limited syntax of sHML, restricting RV to monitorable properties severely limits its applicability. Indeed, many relevant correctness properties still fall outside of this scope as some violations cannot be determined from single (finite) system executions. This sets the foundation for our study, which investigates how these monitorability limits are affected when enhancing the monitors' observational capabilities by allowing them to observe the system across multiple runs.

Part I

Repeated Monitoring for Deterministic Systems

3 The Framework

The classical RV setup can be decomposed into three components: the SUS, the monitor conducting the verification, and the instrumentation coupling the monitor with the SUS. At runtime, the instrumentation permits the monitor to observe the current execution of the SUS until it detects certain behaviour. In this chapter, we formalise an *extended* setup, where monitoring is performed in two steps, namely *history aggregation* and *history analysis*. During *aggregation*, monitors gather SUS information over *multiple* executions. Then, each time a new trace is added to the history, the *analysis* step uses a proof system to determine whether the SUS generating such a history is rejected. If it fails to reject that history, these two steps are repeated again until a permanent verdict is reached.

Structure of the chapter. Section 3.1 first presents our monitor language. We adopt the modular approach advocated by Aceto *et al.* [9, 10] by keeping the specification formalism (in this case RECHML) and the operational description of monitors separate from one another. We then formally define the instrumentation transition relation connecting the operational behaviour of monitors with the running SUS. Subsequently, Section 3.2 formalises how a history is rejected by the monitor through a proof system. Section 3.3 concludes.

3.1 History Aggregation

Monitors are often classified based on *when* and *how* they retrieve information from the SUS [63]. The timing of this retrieval, *i.e., when*, results in two categories: *online* monitors and *offline* monitors. *Online* monitors run alongside the SUS while it executes, observing events generated by the SUS up to the current execution point (*i.e.,* partial execution traces). In contrast, *offline* monitors run as soon as the SUS terminates, giving them access to the complete SUS execution (*e.g.* a log file). Monitors in our framework are of the former kind; this approach is more *lightweight* as monitors can stop running and analyse the trace as soon as they potentially have enough SUS information.

Remark 3.1. Although the monitors in our framework are of the online kind, a log file containing observations from multiple system executions can be viewed as a history of the SUS. In such cases, a monitor can perform repeated monitoring by focusing exclusively on history analysis (Section 3.2) and skipping the history aggregation step (Section 3.1). Nevertheless, the inherent benefits of online monitoring still carry over to this setting. ■

The question of *how* monitors retrieve information is determined by the *instrumentation*, which refers to the mechanism employed to extract events of interest from the SUS during its execution and

Monitor Syntax

$$
\begin{array}{llllll}
m,n \in \textsc{Mon} ::= & \mathsf{no} & \text{(rejection verdict)} & \mid \mathsf{end} & \text{(inconclusive verdict)} \\
\mid & X & \text{(recursion variable)} & \mid m \otimes n & \text{(parallel conjunction)} \\
\mid & \alpha.m & \text{(sequencing)} & \mid m \oplus n & \text{(parallel disjunction)} \\
\mid & \mathrm{rec}X.m & \text{(recursion)} & &
\end{array}
$$

$(\odot \in \{\oplus, \otimes\})$

Monitor Semantics

$$\textsc{mEnd}\ \frac{}{(t,\mathsf{end}) \xrightarrow{\alpha}_H (t\alpha,\mathsf{end})} \qquad \textsc{mVrP1L}\ \frac{t \in H}{(t,\mathsf{no} \odot n) \xrightarrow{\tau}_H (t,n)} \qquad \textsc{mVrP2L}\ \frac{t \notin H}{(t,\mathsf{no} \odot n) \xrightarrow{\tau}_H (t,\mathsf{no})} \qquad \textsc{mAct}\ \frac{}{(t,\alpha.m) \xrightarrow{\alpha}_H (t\alpha,m)}$$

$$\textsc{mRec}\ \frac{}{(t,\mathrm{rec}X.m) \xrightarrow{\tau}_H (t,m[\mathrm{rec}X.m/X])} \qquad \textsc{mTauL}\ \frac{(t,m) \xrightarrow{\tau}_H (t,m')}{(t,m \odot n) \xrightarrow{\tau}_H (t,m' \odot n)}$$

$$\textsc{mPar1}\ \frac{(t,m) \xrightarrow{\alpha}_H (t',m') \quad (t,n) \xrightarrow{\alpha}_H (t',n')}{(t,m \odot n) \xrightarrow{\alpha}_H (t',m' \odot n')} \qquad \textsc{mPar2L}\ \frac{n \neq \mathsf{no} \quad (t,m) \xrightarrow{\alpha}_H (t',m') \quad (t,n) \not\xrightarrow{\alpha}_H \quad (t,n) \not\xrightarrow{\tau}_H}{(t,m \odot n) \xrightarrow{\alpha}_H (t',m')}$$

Instrumentation

$$\textsc{iNo}\ \frac{}{H \triangleright (t,\mathsf{no}) \triangleleft p \xrightarrow{\tau} H, t \triangleright (t,\mathsf{end}) \triangleleft p} \qquad \textsc{iTer}\ \frac{m \neq \mathsf{no} \quad p \xrightarrow{\alpha} p' \quad (t,m) \not\xrightarrow{\alpha}_H \quad (t,m) \not\xrightarrow{\tau}_H}{H \triangleright (t,m) \triangleleft p \xrightarrow{\alpha} H \triangleright (t\alpha,\mathsf{end}) \triangleleft p'}$$

$$\textsc{iAsP}\ \frac{p \xrightarrow{\tau} p'}{H \triangleright (t,m) \triangleleft p \xrightarrow{\tau} H \triangleright (t,m) \triangleleft p'} \qquad \textsc{iAsM}\ \frac{(t,m) \xrightarrow{\tau}_H (t',m')}{H \triangleright (t,m) \triangleleft p \xrightarrow{\tau} H \triangleright (t',m') \triangleleft p} \qquad \textsc{iMon}\ \frac{p \xrightarrow{\alpha} p' \quad (t,m) \xrightarrow{\alpha}_H (t',m')}{H \triangleright (t,m) \triangleleft p \xrightarrow{\alpha} H \triangleright (t',m') \triangleleft p'}$$

Figure 3.1. *Monitors and Instrumentation for deterministic SUSs*

report them to the monitor [36, 18, 46]. Instrumentation can follow either of two approaches: *inline* or *outline*. In the *inline* approach, monitor code is directly injected into the system's code, *e.g.* by adding additional checks or logging statements in specific points. This approach typically yields lower overheads and is more expressive since the monitor has full access to the SUS implementation. However, inline instrumentation *cannot* be used when the system model is unavailable (*e.g.* in case of restrictions due to intellectual property rights, third parties services, *etc.*). In the *outline* approach, the SUS and the monitor are kept as separate entities, and an external tracing infrastructure gathers SUS events and forwards them to the monitor. Since this thesis aims to stick, as much as possible, to a black-box treatment of the SUS, our monitoring framework adopts the second approach, *i.e.*, outline instrumentation. This permits the instrumented monitors to analyse the events emitted by the SUS without relying on any information about the specific intermediate states reached during execution.

3.1.1 Monitors

The runtime analysis conducted by our monitors, defined in Figure 3.1, *records* the SUS traces that lead to rejection states. An *executing-monitor* state is described as a tuple $(t,m) \in \textsc{Trc} \times \textsc{Mon}$, where t is the trace that some initial monitor has analysed from the beginning of the run, up to the current execution point, and m is the current state of the monitor after this analysis.

Executing-monitor transitions are defined w.r.t. a history H that stores the trace prefixes accumulated in prior monitored executions of the same SUS. They take the form

$$(t,m) \xrightarrow{\eta}_H (t',m')$$

where $\eta \in \textsc{Act}$, to denote that (t,m) transitions to (t',m') either by observing an action α produced by the SUS, *i.e.,* $\eta = \alpha \in \textsc{eAct}$, or by evolving autonomously using the silent action τ, *i.e.,* $\eta = \tau$.

A monitor execution can reach one of two final states: either a *rejection* verdict, no, or an *inconclusive* state, end. The inconclusive state end behaves like an identity, transitioning to itself when analysing any SUS action (without aggregating the trace further); this is described by rule mEnd. Differently, a rejection state no indicates to the instrumentation that the (partial) trace analysed thus far should be aggregated to the history. After aggregating the trace, it then behaves as end; this is described by instrumentation rule iNo to be discussed later. Note that rule iNo is the only rule that extends the history H by the aggregated trace t as (H,t).

The current recorded trace is accrued via monitor sequencing, $\alpha.m$, described in rule mAct. Besides sequencing, (sub-)monitors can be composed together as a parallel *conjunction*, $m \otimes n$, or a parallel *disjunction*, $m \oplus n$. Parallel monitors, $m \odot n$ where $\odot \in \{\oplus, \otimes\}$, either move autonomously with silent τ actions, following rule mTauL (symmetric mTauR elided), or in unison when analysing system actions α, following rule mPar1. When the sub-monitor m can analyse the action proffered by the system *but* the other sub-monitor n cannot, the transition is still permitted but the blocked sub-monitor n is discarded, following rule mPar2L (the symmetric rule, mPar2R, is elided). This reason for this is that although the sub-monitor n cannot analyse the system action α produced, *i.e.,* $(t,n) \xrightarrow{\alpha}\!\!\!\!\!\!/\;\;_H$, it does not prohibit the other sub-monitor m from potentially recording a new trace. An analogous mechanism is also implemented by the instrumentation rule iTer to be discussed later.

Four rules determine how a rejection verdict *sub-monitor* is handled (symmetric rules elided). Rule mVrP2L asserts that the verdict no supersedes its parallel counterpart whenever the accumulated (violating) trace is new, *i.e.,* $t \notin H$; when $\mathsf{no} \odot n$ transitions to no, it allows the instrumentation rule iNo to add t to the history. Dually, if the trace aggregated is already part of the history, *i.e.,* $t \in H$, the rejection verdict is discarded, *i.e.,* $\mathsf{no} \odot n$ transitions to n, to allow the other sub-monitor, n, to potentially collect other violating traces with common prefixes over multiple runs, following rule mVrP1L.

The remaining monitor rules are standard, where symmetric rules are elided. We note that, although trace collection does *not* distinguish between parallel conjunction and disjunctions, history analysis does. This can be seen in Figure 3.3, which will be thoroughly discussed in Section 3.2.

3.1.2 Instrumentation

The operational behaviour of an executing-monitor is connected to that of a running SUS via the instrumentation transition relation in Figure 3.1. It is defined over *monitored systems*, which are triples

that consist of a SUS state p, an executing-monitor (t,m), and a history H, denoted as

$$H \triangleright (t,m) \triangleleft p$$

For action $\eta \in \text{Act}$, monitored system transitions take the form

$$H \triangleright (t,m) \triangleleft p \xrightarrow{\eta} H' \triangleright (t',m') \triangleleft p'$$

and denote that the executing-monitor (t,m) transitions to (t',m') when analysing the SUS transitioning from p to p' via action η, updating the history from H to H' in the process.

Rule iMon formalises the analysis of an action: whenever a system p transitions to p' via action α and the executing-monitor (m,t) can analyse α and transition to (m',t'), the two transition in lockstep to $H \triangleright (m',t') \triangleleft p'$. Rule iNo, previewed earlier, handles the storing of new traces t that lead to a rejection verdict by extending the history H to (H,t) whenever the executing-monitor is in a rejection state (m,no), forcing the executing-monitor to terminate in the process. Instrumentation also allows the SUS and executing-monitor to (internally) transition independent of one another via silent actions, as described in rules iAsP and iAsM. In case (t,m) can neither analyse a SUS action, nor perform an internal transition, instrumentation forces it to terminate prematurely by transitioning to the inconclusive verdict (rule iTer); this ensures instrumentation transparency, *i.e.,* the instrumented monitors interfere minimally with the SUS execution [73, 72]. Consequently, the monitoring infrastructure does not block the behaviour of the SUS whenever the executing monitor cannot analyse a system event.

We adopt a similar convention to that of Chapter 2. For instance, we define weak transitions in a similar manner and write $H \triangleright (t,m) \triangleleft p \overset{u}{\Longrightarrow} H' \triangleright (t',m') \triangleleft p'$ instead of $H \triangleright (t,m) \triangleleft p \overset{\alpha_1}{\Longrightarrow} \cdots \overset{\alpha_n}{\Longrightarrow} H' \triangleright (t',m') \triangleleft p'$ when $u = \alpha_1 \cdots \alpha_n$.

Remark 3.2. A major departure from existing work on formalising monitor instrumentation [76, 10, 73] is that ours does *not* flag property violations. Instead, it limits itself to aggregating traces. Indeed, each monitored SUS execution starts with the empty trace $t = \epsilon$ and the history size can increase by *at most* one trace along each execution, namely the current trace accrued. ■

Our monitors are designed to work over multiple runs of the *same* SUS. Starting from an empty history $H_0 = \emptyset$, a set of trace prefixes, each leading to rejection monitor states no, can be accumulated in the history over a sequence of monitored system executions by passing the resulting history H_i obtained from the i^{th} monitored execution on to the subsequent execution $i+1$. As depicted in Figure 3.2, this induces a (finite) totally-ordered sequence of histories, $H_0 \subseteq H_1 \subseteq H_2 \subseteq \cdots$ where $H_0 = \emptyset$.

Example 3.3. Consider monitor m_1 below, which reaches a rejection verdict after observing either action a or action c, following a sequence of serviced queries (*i.e.,* alternating r and s actions).

$$m_1 \overset{\text{def}}{=} \text{rec}X.\big(r.s.X \otimes (a.\text{no} \oplus c.\text{no})\big)$$

When instrumented with system $p_1 \overset{\text{def}}{=} \text{rec}X.\big(r.s.X + (a.X + c.\mathbf{0})\big)$ from Example 2.3, the executing-monitor

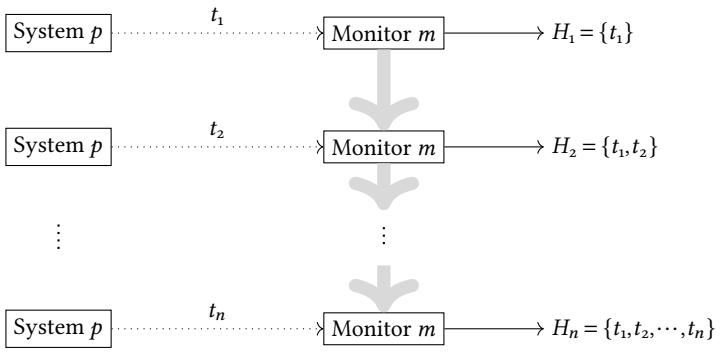


Figure 3.2. *History aggregation over n monitored SUS executions*

(ϵ, m_1) with history $H_0 = \emptyset$ can aggregate the trace prefix $t_1 = rsa$ as follows:

$$
\begin{aligned}
H_0 \triangleright (\epsilon, m_1) \triangleleft p_1 &\xrightarrow{\tau} H_0 \triangleright (\epsilon, m_1) \triangleleft r.s.p_1 + (a.p_1 + c.\mathbf{0}) && \text{(IAsP)} \\
&\xrightarrow{\tau} H_0 \triangleright (\epsilon, r.s.m_1 \otimes (a.\mathsf{no} \oplus c.\mathsf{no})) \triangleleft r.s.p_1 + (a.p_1 + c.\mathbf{0}) && \text{(IAsM)} \\
&\xrightarrow{r} H_0 \triangleright (r, s.m_1) \triangleleft s.p_1 && \text{(IMon)} \\
&\xrightarrow{s} H_0 \triangleright (rs, m_1) \triangleleft p_1 && \text{(IMon)} \\
&\xrightarrow{\tau} H_0 \triangleright (rs, m_1) \triangleleft r.s.p_1 + (a.p_1 + c.\mathbf{0}) && \text{(IAsP)} \\
&\xrightarrow{\tau} H_0 \triangleright (rs, r.s.m_1 \otimes (a.\mathsf{no} \oplus c.\mathsf{no})) \triangleleft r.s.p_1 + (a.p_1 + c.\mathbf{0}) && \text{(IAsM)} \\
&\xrightarrow{a} H_0 \triangleright (rsa, \mathsf{no}) \triangleleft p_1 && \text{(IMon)} \\
&\xrightarrow{\tau} H_0 \cup \{rsa\} \triangleright (rsa, \mathsf{end}) \triangleleft p_1 && \text{(INo)}
\end{aligned}
$$

In a subsequent run with the augmented history $H_1 = \{rsa\}$, the executing-monitor (ϵ, m_1) can then aggregate the trace prefix $t_2 = rsc$ via the execution sequence

$$
H_1 \triangleright (\epsilon, m_1) \triangleleft p_1 \xRightarrow{t_2} H_1 \triangleright (t_2, \mathsf{no}) \triangleleft p_1 \xrightarrow{\tau} H_1 \cup \{t_2\} \triangleright (t_2, \mathsf{end}) \triangleleft p_1
$$

Note that, since monitors assume a *passive* role [73], they cannot steer the behaviour of the SUS. This means there is *no* guarantee the SUS will exhibit *different* behaviour each time it is executed. ■

Since the behaviour of executing-monitors (t,m) in monitored systems $H \triangleright (t, m) \triangleleft p$ is determined by the accumulated history H, which governs whether m should stop monitoring and add t to the history, or continue observing p, the instrumentation mechanism must be able to handle overlapping trace prefixes that may lead to rejection states.

Example 3.4. Monitor m_2 below is a revised version of monitor m_1 from Example 3.3, where sequences of rs actions can be interleaved with memory allocations, *i.e.,* finite sequences of a actions described by the sub-monitor $a.X$ in m_2.

$$
m_2 \stackrel{\text{def}}{=} \mathsf{rec}X.\big(r.s.X \otimes a.X \otimes (a.\mathsf{no} \oplus c.\mathsf{no})\big)
$$

Suppose the initial executing-monitor (ϵ, m_2) is instrumented with system p_1 from Example 2.3, which can generate the trace $rsarsa\cdots$. For history $H_0 = \emptyset$, monitor m_2 stores the trace prefix rsa via a transition

sequence similar to that in Example 3.3. However, in a subsequent run with the augmented history $H_1 = \{rsa\}$, we observe the following behaviour:

$$\begin{aligned} H_1 \triangleright (\epsilon, m_2) \triangleleft p_1 &\xRightarrow{rs} H_1 \triangleright (rs, r.s.m_2 \otimes a.m_2 \otimes (a.\mathsf{no} \oplus c.\mathsf{no})) \triangleleft a.p_1 \\ &\xrightarrow{a} H_1 \triangleright (rsa, m_2 \otimes \mathsf{no}) \triangleleft p_1 \\ &\xrightarrow{\tau} H_1 \triangleright (rsa, m_2) \triangleleft p_1 \qquad (*) \\ &\xRightarrow{rsa} H_1 \triangleright (rsarsa, m_2 \otimes \mathsf{no}) \triangleleft p_1 \\ &\xrightarrow{\tau} H_1 \triangleright (rsarsa, \mathsf{no}) \triangleleft p_1 \\ &\xrightarrow{\tau} H_1 \cup \{rsarsa\} \triangleright (rsarsa, \mathsf{end}) \triangleleft p_1 \qquad (\dagger) \end{aligned}$$

Transition $(*)$ follows rule mVrP1R with $(rsa, m_2 \otimes \mathsf{no}) \xrightarrow{\tau}_{H_1} (rsa, m_2)$ since $rsa \in H_1$. The executing-monitor does *not* stop accruing at the trace prefix rsa but continues monitoring the SUS until it encounters a *new* rejecting trace, $rsarsa$. This is then aggregated to history H_1 in transition $(\dagger)$ using rule iNo. ■

Rule iNo encodes the design decision to stop monitoring (by transitioning to the inconclusive monitor end) as soon as a new trace is aggregated to the history. This provides the monitors conducting the verification with a clear cut-off point at which to stop monitoring and pass the aggregated history to the subsequent execution. Consequently, these successive monitored executions form the chain of histories $H_0 \subseteq H_1 \subseteq H_2 \subseteq \cdots$ (*i.e.,* totally-ordered sequence of histories) outlined earlier.

3.2 History Analysis

We formalise how a history is rejected by a monitor through a proof system. This is used in conjunction with the history aggregating mechanism of Section 3.1 to determine whether the SUS generating such a history is rejected. Concretely, every time a new trace is added to the history, the proof system is invoked. If it fails to reject that history, the SUS is executed again until a permanent verdict is reached.

The main judgement of this proof system is $\mathbf{rej}(H, m)$, *i.e.,* monitor m *rejects* history H. It can be seen as an attempt to reconstruct a partial model of the SUS from the traces aggregated by the monitors in Section 3.1, large enough to infer the property violation. This analysis is the least relation defined by the rules in Figure 3.3, which rely on a helper function

$$\mathit{suffix}(H, \eta) = \{\, t \mid \eta t \in H \,\}$$

Intuitively, this function returns the continuation of all the traces in H that are prefixed by an η action, and corresponds to the classical notion of the derivative of a language, also known as the Brzozowski derivative [45, 110]. For instance, when $H = \{rsa, rsc, ars\}$, the function $\mathit{suffix}(H, r)$ returns the set of traces $\{sa, sc\}$ whereas $\mathit{suffix}(H, s)$ returns $\emptyset$.

The axiom NO states that a no monitor rejects all *non-empty* histories. This implicitly means that a monitor cannot reject a SUS outright, without any observation. In rule ACT, a sequenced monitor $\alpha.m$ rejects H if the sub-monitor m rejects the history returned by $\mathit{suffix}(H, \alpha)$. Parallel disjunction monitors $m \oplus n$ reject history H if *both* constituent monitors m and n reject H (rule PARO). Conversely, parallel conjunction monitors $m \otimes n$ reject history H if either *one* of them rejects it (rules PARAL and the elided symmetric rule PARAR). Rule REC states that a recursive monitor rejects a history if its unfolding does.

$$\textsc{no}\ \frac{H \neq \emptyset}{\mathbf{rej}(H,\mathsf{no})} \qquad \textsc{act}\ \frac{\mathbf{rej}(\mathit{suffix}(H,\alpha),m)}{\mathbf{rej}(H,\alpha.m)} \qquad \textsc{parO}\ \frac{\mathbf{rej}(H,m)\ \ \mathbf{rej}(H,n)}{\mathbf{rej}(H,m \oplus n)} \qquad \textsc{parAL}\ \frac{\mathbf{rej}(H,m)}{\mathbf{rej}(H,m \otimes n)} \qquad \textsc{rec}\ \frac{\mathbf{rej}(H,m[^{\mathrm{rec}X.m}/_X])}{\mathbf{rej}(H,\mathrm{rec}X.m)}$$

Figure 3.3. *Proof System for deterministic SUSs*

We say that *monitor m rejects system p* if and only if system p is capable of producing a history H that m rejects (using the proof system in Figure 3.3). This is formalised in Definition 3.5.

Definition 3.5 (System rejections). A monitor m rejects system p, denoted as $\mathbf{rej}(p,m)$, whenever $\exists H \subseteq T_p$ such that $\mathbf{rej}(H,m)$. ■

Example 3.6. Recall monitor $m_1 \stackrel{\text{def}}{=} \mathrm{rec}X.\big(r.s.X \otimes (a.\mathsf{no} \oplus c.\mathsf{no})\big)$ from Example 3.3. One can show that m_1 rejects the history $\{rsa,rsc\}$, denoted $\mathbf{rej}(\{rsa,rsc\},m_1)$, through the left-hand proof derivation below.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\mathbf{rej}(\{\epsilon\},\mathsf{no})}\textsc{no}}{\mathbf{rej}(\{a,c\},a.\mathsf{no})}\textsc{act} \quad \dfrac{\dfrac{}{\mathbf{rej}(\{\epsilon\},\mathsf{no})}\textsc{no}}{\mathbf{rej}(\{a,c\},c.\mathsf{no})}\textsc{act}}{\mathbf{rej}(\{a,c\},a.\mathsf{no} \oplus c.\mathsf{no})}\textsc{parO}}{\mathbf{rej}(\{a,c\},r.s.m_1 \otimes (a.\mathsf{no} \oplus c.\mathsf{no}))}\textsc{parAR}}{\mathbf{rej}(\{a,c\},m_1)}\textsc{rec}}{\mathbf{rej}(\{sa,sc\},s.m_1)}\textsc{act}}{\mathbf{rej}(\{rsa,rsc\},r.s.m_1)}\textsc{act}}{\mathbf{rej}(\{rsa,rsc\},r.s.m_1 \otimes (a.\mathsf{no} \oplus c.\mathsf{no}))}\textsc{parAL}}{\mathbf{rej}(\{rsa,rsc\},m_1)}\textsc{rec}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\mathbf{rej}(\{\epsilon\},\mathsf{no})}\textsc{no}}{\mathbf{rej}(\{a\},a.\mathsf{no})}\textsc{act} \quad \dfrac{\mathbf{rej}(\emptyset,\mathsf{no}) \quad (**)}{\mathbf{rej}(\{a\},c.\mathsf{no})}\textsc{act}}{\mathbf{rej}(\{a\},a.\mathsf{no} \oplus c.\mathsf{no})}\textsc{parO}}{\mathbf{rej}(\{a\},r.s.m_1 \otimes (a.\mathsf{no} \oplus c.\mathsf{no}))}\textsc{parAR}}{\mathbf{rej}(\{a\},m_1)}\textsc{rec}}{\mathbf{rej}(\{sa\},s.m_1)}\textsc{act}}{\mathbf{rej}(\{rsa\},r.s.m_1)}\textsc{act}}{\mathbf{rej}(\{rsa\},r.s.m_1 \otimes (a.\mathsf{no} \oplus c.\mathsf{no}))}\textsc{parAL}}{\mathbf{rej}(\{rsa\},m_1)}\textsc{rec}$$

In contrast, the right-hand derivation above shows why $\neg\mathbf{rej}(\{rsa\},m_1)$ since no rule from Figure 3.3 can justify the judgement $\mathbf{rej}(\emptyset,\mathsf{no})$ at $(**)$. ■

Remark 3.7. Derivations for $\mathbf{rej}(H,m)$ are not necessarily unique since Figure 3.3 allows a level of non-determinism. For instance, the judgement $\mathbf{rej}(\{rsa,rsc\},r.s.a.\mathsf{no} \otimes r.s.c.\mathsf{no})$ admits two derivations:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\mathbf{rej}(\{\epsilon\},\mathsf{no})}\textsc{no}}{\mathbf{rej}(\{a,c\},a.\mathsf{no})}\textsc{act}}{\mathbf{rej}(\{sa,sc\},s.a.\mathsf{no})}\textsc{act}}{\mathbf{rej}(\{rsa,rsc\},r.s.a.\mathsf{no})}\textsc{act}}{\mathbf{rej}(\{rsa,rsc\},r.s.a.\mathsf{no} \otimes r.s.c.\mathsf{no})}\textsc{parAL}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\mathbf{rej}(\{\epsilon\},\mathsf{no})}\textsc{no}}{\mathbf{rej}(\{a,c\},c.\mathsf{no})}\textsc{act}}{\mathbf{rej}(\{sa,sc\},s.c.\mathsf{no})}\textsc{act}}{\mathbf{rej}(\{rsa,rsc\},r.s.c.\mathsf{no})}\textsc{act}}{\mathbf{rej}(\{rsa,rsc\},r.s.a.\mathsf{no} \otimes r.s.c.\mathsf{no})}\textsc{parAR}$$

This, however, does not affect our theory. ■

Example 3.8 below illustrates how a monitor rejects a SUS by repeatedly aggregating and analysing trace prefixes using the history aggreation and history analysis mechanisms.

Example 3.8. Recall the following execution sequences from Example 3.3, where $H_0 = \emptyset$ and $H_1 = \{t_1\}$.

$$H_0 \triangleright (\epsilon,m_1) \triangleleft p_1 \stackrel{t_1}{\Longrightarrow} H_0 \cup \{t_1\} \triangleright (t_1,\mathsf{end}) \triangleleft p_1 \qquad \text{and} \qquad H_1 \triangleright (\epsilon,m_1) \triangleleft p_1 \stackrel{t_2}{\Longrightarrow} H_1 \cup \{t_2\} \triangleright (t_2,\mathsf{end}) \triangleleft p_1$$

During the first monitored execution, instrumentation records the trace prefix $t_1 = rsa$ but monitor m_1 fails to reject the accrued history, *i.e.,* $\neg\mathbf{rej}(\{t_1\},m_1)$, as shown in Example 3.6. The monitored system $H_1 \triangleright (\epsilon,m_1) \triangleleft p_1$ is thus executed again, this time with the augmented history H_1, resulting in the addition of the trace prefix $t_2 = rsc$ to the history. Monitor m_1 can now reject the extended history, $\mathbf{rej}(\{t_1,t_2\},m_1)$; see Example 3.6. This means that m_1 rejects p_1 via the witness history $\{t_1,t_2\}$, denoted as $\mathbf{rej}(p_1,m_1)$. ■

We also note that, the history analysis in Figure 3.3 models the fact that monitor rejections are always evidence-based. This is better illustrated in the example below.

Example 3.9. Although the (verdict) monitor, no, trivially rejects *any* system p, it can only do so after it observes at least one execution. Once it is instrumented with p and initial history $H_o = \emptyset$, the setup in Figure 3.1 immediately triggers rule INo, *i.e.,*

$$\emptyset \triangleright (\epsilon,\mathsf{no}) \triangleleft p \xrightarrow{\tau} \{\epsilon\} \triangleright (\epsilon,\mathsf{end}) \triangleleft p$$

Since a new trace, ϵ, has been added to the history, *i.e.,* $H_1 = \{\epsilon\}$, the monitoring framework invokes the proof system of Figure 3.3 to immediately conclude $\mathbf{rej}(\{\epsilon\},\mathsf{no})$ via rule NO, which implies $\mathbf{rej}(p,m)$. ■

3.3 Summary

In this chapter, we presented an extended online monitoring setup designed to operate over multiple runs of the same (deterministic) SUS. Monitoring is performed in two stages: history aggregation and history analysis. History aggregation is conducted by executing-monitors, which collect information about the SUS over multiple executions. We equipped these executing-monitors with an operational semantics and formalised how their behaviour is connected to that of a running SUS via an instrumentation transition relation. For the analysis phase, we introduced a proof system that determines how a monitor rejects a history. This, in turn, establishes whether the SUS generating such a history is rejected, which is defined in terms of the predicate **rej**. The resulting multi-run setup serves as the basis for defining rigorous notions of monitor correctness in the following chapter and, later, for studying how multiple runs affect the monitorability limits identified in prior works.

4 Monitor Correctness

RV establishes a correspondence between the operational behaviour of a monitor and the semantic meaning of the property it is monitoring for [75, 12]. This is underpinned by what we understand by the following statement:

"Monitor *m correctly monitors* for property φ."

What is actually expected of monitors is seldom defined in precise terms, making it hard to ascertain monitor correctness. For instance, in the classical RV setup (see Section 2.3 for details), the least correctness requirement one often expects is *soundness* [75], which ensures that the verdicts flagged by the monitor conducting RV do not contradict the semantic meaning of the property being monitored for. Another common requirement is *completeness* [13, 12], though its definition varies depending on the monitoring setup. In certain cases, a monitor might be considered complete if it can flag a verdict at least once, even if it misses other possible detections. However, missed detections could be harmful in certain contexts, such as safety-critical or autonomous systems. In such scenarios, stricter completeness constraints might be necessary, such as the ability of flagging *all* possible satisfactions or *all* possible violations for a property. One might also require attributes that are independent of the property being monitored for and instead focus solely on the verifying monitor. Common requirement include *verdict irrevocability*, stating that once a monitor reaches a conclusion, that conclusion cannot be changed [73, 72, 12, 10], and *passivity*, where the monitor cannot interfere with the SUS execution [36]. The monitor might also need to exhibit a certain degree of determinism, such as *strong eventual consistency* [60] or *observational verdict determinism* [71, 72, 11], and conceal any non-determinism, as long as it is internal.

This chapter pins down what it means for a monitor from the multi-run monitoring setup of Chapter 3 to correctly monitor for a property φ, building on the theoretical foundations of Aceto *et al.* in [7].

Structure of the chapter. The monitors of Chapter 3 conduct two interdependent tasks, namely history aggregation and history analysis. In Section 4.1, we prove correctness properties of the history aggregation mechanism of Section 3.1. Similarly, in Section 4.2, we shift our focus to the history analysis mechanism of Section 3.2 and prove correctness properties about it. We conclude Section 4.2 by formally defining the notions of soundness, completeness and correct monitoring for our multi-run monitors.

4.1 Correctness for History Aggregation

The first correctness result we show concerns the *history aggregation* mechanism of Chapter 3. Proposition 4.1 (below) states that the traces collected by the monitor are indeed traces of the system with which it is instrumented. It can be used to show that whenever a history H is accumulated over a sequence of

monitored executions of any system p, *i.e.,* $H_0 \subseteq H_1 \subseteq \cdots \subseteq H_n$ where $H_0 = \emptyset$ and $H_n = H$, we have that $H \subseteq T_p$. Following Proposition 4.1, we assume that all monitors considered are *veracious*.

Proposition 4.1 (Veracity). For any history H, monitor m, system p, and sequence of actions $\eta_1,\ldots,\eta_n$,

$$\text{if } \; H \triangleright (\epsilon,m) \triangleleft p \xrightarrow{\eta_1} \cdots \xrightarrow{\eta_n} H' \triangleright (t,m') \triangleleft p' \; \text{ then } \; p \overset{t}{\Longrightarrow} p'$$

Proof Outline. The proof is by induction on n.

For the base case, the result is immediate since $p = p'$, $t = \epsilon$ and $p \Longrightarrow p$.

For the inductive case, suppose $H \triangleright (\epsilon,m) \triangleleft p \xrightarrow{\eta_1} \cdots \xrightarrow{\eta_k} H' \triangleright (t,m') \triangleleft p' \xrightarrow{\eta_{k+1}} H'' \triangleright (t',m'') \triangleleft p''$. We show $p \overset{t'}{\Longrightarrow} p''$. By the IH and $H \triangleright (\epsilon,m) \triangleleft p \xrightarrow{\eta_1} \cdots \xrightarrow{\eta_k} H' \triangleright (t,m') \triangleleft p'$, we deduce $p \overset{t}{\Longrightarrow} p'$. By case analysis on the rules that could have been used to derive $H' \triangleright (t,m') \triangleleft p' \xrightarrow{\eta_{k+1}} H'' \triangleright (t',m'') \triangleleft p''$, we know

- if $\eta = \tau$ then $t' = t$ and $p' \Longrightarrow p''$
- if $\eta \neq \tau$ then $t' = t\eta$ and $p' \overset{\eta}{\Longrightarrow} p''$

Together with $p \overset{t}{\Longrightarrow} p'$ and the definition for weak transitions, these imply that $p \overset{t'}{\Longrightarrow} p''$.

For the complete proof, see Appendix B.1. □

Another important correctness criteria for our multi-run monitoring setup is that executing-monitors behave *deterministically*, *i.e.,* they produce consistent outcomes for the same SUS observations [72, 11]. Establishing that executing-monitors indeed exhibit such behaviour relies on two additional results, namely Propositions 4.2 and 4.3. More specifically, Proposition 4.2 shows that τ-moves do not affect the analytical capabilities of a monitor, *i.e.,* the executing-monitor can internally transition only if it cannot analyse any SUS actions, and vice versa. Conversely, Proposition 4.3 assures us that monitor behaviour is *confluent* w.r.t. τ-moves, *i.e.,* the order of τ-moves does not affect the final monitor state.

Proposition 4.2 (τ-Race Absence). For all $\alpha \in \textsc{eAct}$, if $(t,m) \xrightarrow{\tau}_H (t,n)$ then $(t,m) \xrightarrow{\alpha}\!\!\!\!\!\!\not\;\;\;_H$.

Proof Outline. Straightforward by case analysis. □

Proposition 4.3 (τ-confluence). For monitors $m,m',m'' \in \textsc{Mon}$, trace $t \in \textsc{Trc}$ and history $H \in \textsc{Hst}$, if $(t,m) \xrightarrow{\tau}_H (t,m')$ and $(t,m) \xrightarrow{\tau}_H (t,m'')$, then there exist weak τ-moves $(t,m')\,(\xrightarrow{\tau}_H)^*\,(t,n)$ and $(t,m'')\,(\xrightarrow{\tau}_H)^*\,(t,n)$ for some monitor $n \in \textsc{Mon}$.

Proof Outline. By induction on the inference of the first move $(t,m) \xrightarrow{\tau}_H (t,m')$, with sub-induction on the derivation of the second move $(t,m) \xrightarrow{\tau}_H (t,m'')$. For the full proof, refer to Appendix B.2. □

Together, Propositions 4.2 and 4.3 allow us to equate monitor states up to τ-moves, a concept formalised as *monitor τ-equivalence* in Definition 4.4 below.

Definition 4.4 (Monitor τ-Equivalence). The executing-monitors (t,m_1) and (t,m_2) are τ-equivalent w.r.t. history H, denoted as $(t,m_1) \approx_H (t,m_2)$, if there exists a common monitor $n \in \textsc{Mon}$ such that $(t,m_1)\,(\xrightarrow{\tau}_H)^*\,(t,n)$ and $(t,m_2)\,(\xrightarrow{\tau}_H)^*\,(t,n)$. ■

Importantly, for a given history, the executing-monitors of Chapter 3 deterministically reach equivalent states when analysing a (partial) trace exhibited by the SUS, formalised in Proposition 4.5. This is also illustrated in Example 4.6 below.

Proposition 4.5 (Monitor Determinism). For all traces u,t,t_1,t_2, history H, and monitors m,n_1,n_2,

$$\text{if } (t,m) \overset{u}{\Longrightarrow}_H (t_1,n_1) \text{ and } (t,m) \overset{u}{\Longrightarrow}_H (t_2,n_2) \text{ then } t_1 = t_2 \text{ and } (t_1,n_1) \approx_H (t_2,n_2)$$

Proof Outline. We can show $\approx_H$ is an equivalence relation, *i.e.,* it is symmetric, reflexive and transitive. Our result then follows from the stronger statement below:

$$\text{if } (t,m_1) \overset{u}{\Longrightarrow}_H (t_1,n_1) \text{ and } (t,m_2) \overset{u}{\Longrightarrow}_H (t_2,n_2) \text{ where } (t,m_1) \approx_H (t,m_2), \text{ then } t_1 = t_2 \text{ and } (t_1,n_1) \approx_H (t_2,n_2)$$

We prove this by induction on the number of transitions in the first execution sequence $(t,m_1) \overset{u}{\Longrightarrow}_H (t_1,n_1)$.

For the base case, we have $u = \epsilon$ and $m_1 = n_1$. Since executing-monitors cannot record silent actions, for the second move, *i.e.,* $(t,m_2)(\xrightarrow{\tau}_H)^*(t_2,n_2)$, we can show $t = t_2$. By Definition 4.4, we thus get $(t,m_1) \approx_H (t,m_2) \approx_H (t,n_2)$. Our result follows by transitivity.

For the inductive case, we can expand the first transition sequence to $(t,m_1) \xrightarrow{\eta}_H (v_1,n_1') \overset{u'}{\Longrightarrow}_H (t_1,n_1)$ for some traces u',v_1, monitor n_1' and action η. There are two cases to consider:

- If $\eta = \tau$, we can show $u = u'$, $t = v_1$ and $(t,m_1) \approx_H (v_1,n_1')$. From the assumption $(t,m_1) \approx_H (t,m_2)$ and the fact that $\approx_H$ is an equivalence relation, we obtain $(v_1,n_1') \approx_H (t,m_2)$. By $(t,n_1') \overset{u}{\Longrightarrow}_H (t_1,n_1)$, the second transition sequence $(t,m_2) \overset{u}{\Longrightarrow}_H (t_2,n_2)$ and the IH, we conclude $t_1 = t_2$ and $(t_1,n_1) \approx_H (t_2,n_2)$.
- If $\eta = \alpha \in \text{EAct}$, we know $u = \alpha u'$ for some trace u'. The second transition sequence $(t,m_2) \overset{u}{\Longrightarrow}_H (t_2,n_2)$ can be expanded as

  $$(t,m_2)\ (\xrightarrow{\tau}_H)^*\ (t,n_2') \xrightarrow{\alpha}_H (v_2,n_2'') \overset{u'}{\Longrightarrow}_H (t_2,n_2)$$

  for some trace v_2. By Definition 4.4, we also know $(t,m_2) \approx_H (t,n_2')$. By this, the assumption $(t,m_1) \approx_H (t,m_2)$ and the fact that $\approx_H$ is an equivalence relation, we deduce $(t,m_1) \approx_H (t,n_2')$. Using Proposition 4.2, we can also show that since $(t,m_1) \xrightarrow{\alpha}_H (v_1,n_1')$ and $(t,n_2') \xrightarrow{\alpha}_H (v_2,n_2'')$ where $(t,m_1) \approx_H (t,n_2')$, we have $m_1 = n_2'$ and $n_1' = n_2''$ and $v_1 = v_2$. Our result then follows by the IH.

For the full proof, refer to Appendix B.2. □

Example 4.6. Recall $m_2 \overset{\text{def}}{=} \text{rec}X.\big(r.s.X \otimes a.X \otimes (a.\text{no} \oplus c.\text{no})\big)$ from Example 3.4. Given history $H = \emptyset$ and trace $u = rsa$, the executing-monitor (ϵ,m_2) can reach *both* monitor states (u,no) and $(u,m_2 \otimes \text{no})$, that is

$$(\epsilon,m_2) \overset{u}{\Longrightarrow}_H (u,\text{no}) \qquad \text{and} \qquad (\epsilon,m_2) \overset{u}{\Longrightarrow}_H (u,m_2 \otimes \text{no})$$

However, the two executing-monitors (u,no) and $(u,m_2 \otimes \text{no})$ can converge at (u,no) by allowing the latter to τ-transition with rule MVRP2R (since $u \notin H$). Following Definition 4.4, this means that (u,no) and $(u,m_2 \otimes \text{no})$ are τ-equivalent w.r.t. history H, *i.e.,* $(u,\text{no}) \approx_H (u,m_2 \otimes \text{no})$. ■

4.2 Correctness for History Analysis

Another sanity check characteristic of RV concerns the *history analysis* of Section 3.2 and is termed *verdict irrevocability* [73, 72, 12, 10]. In our case, this translates to Propositions 4.7 and 4.8 stating that, once the SUS is rejected by a monitor after exhibiting history H (using the history analysis of Section 3.2), further observations (in terms of additional traces, *width*, or longer traces, *length*) do *not* alter this conclusion. This is crucial whenever the remedial action taken is based on such verdicts. For instance, when the monitor rejects a history, the monitor might step in and terminate the entire computation, kill an offending process/thread (*i.e.,* a part of the computation) or suppress the last action whenever observable events can be buffered [16]; altering this conclusion would compromise the validity of the remedial action taken by the monitor.

Proposition 4.7 (Width Irrevocability). For any histories H, H' and monitor m,

$$\text{if } \mathbf{rej}(H,m) \text{ then } \mathbf{rej}(H \cup H', m)$$

Proof Outline. The proof is straightforward by induction on the inference of $\mathbf{rej}(H,m)$. We only outline the case for rule ACT, *i.e.,* $\mathbf{rej}(H,\alpha.m)$ because $\mathbf{rej}(H'',m)$ where $H'' = \mathit{suffix}(H,\alpha)$.
Since $\mathbf{rej}(H'',m)$ is a sub-derivation of $\mathbf{rej}(H,\alpha.m)$, by the IH, we obtain that $\mathbf{rej}(H'' \cup H''', m)$ for any history H'''. Letting $H''' = \mathit{suffix}(H',\alpha)$, we know that $H'' \cup H''' = \mathit{suffix}(H,\alpha) \cup \mathit{suffix}(H',\alpha) = \mathit{suffix}(H \cup H', \alpha)$. Our result, $\mathbf{rej}(H \cup H', \alpha.m)$, follows by applying rule ACT. □

Proposition 4.8 (Length Irrevocability). For any history H, traces t,u and monitor m,

$$\text{if } \mathbf{rej}((H,t),m) \text{ then } \mathbf{rej}((H,tu),m)$$

Proof Outline. The proof is by induction on the inference of $\mathbf{rej}((H,t),m)$. We only outline the case for rule ACT, *i.e.,* $\mathbf{rej}((H,t),\alpha.m)$ because $\mathbf{rej}(H',m)$ where $H' = \mathit{suffix}((H,t),\alpha)$. There are two subcases:

- When $t = \alpha t'$, then $H' = (H'',t') = \mathit{suffix}((H,t),\alpha)$ for some H''. By the IH, we deduce $\mathbf{rej}((H'',t'u),m)$. But by definition, we also know $(H'',t'u) = \mathit{suffix}((H,tu),\alpha)$, meaning that $\mathbf{rej}(\mathit{suffix}((H,tu),\alpha),m)$. Our result, $\mathbf{rej}((H,tu),\alpha.m)$, follows by applying rule ACT.
- When $t = \beta t'$ where $\beta \neq \alpha$, we know $\mathit{suffix}((H,t),\alpha) = \mathit{suffix}(H,\alpha) = \mathit{suffix}((H,tu),\alpha)$. Our result follows by applying rule ACT.

The proof for the other cases is straightforward; for the complete proof, see Appendix B.2. □

For RV purposes, the least *correctness* requirement expected of our (irrevocable) *history analysis* is that any rejections imply property violations. Concretely, m monitors correctly for the (closed) formula φ only if, for *any* system p, whenever a monitor rejects this system, *i.e.,* $\mathbf{rej}(p,m)$, then it must be the case that p also violates the property, *i.e.,* $p \notin [\![\varphi]\!]$. This requirement is more commonly referred to as *(rejection) soundness* [75]. Correct monitors should observe soundness irrespective of the system they are instrumented with. This universal quantification over all systems (*i.e.,* for any system p) required by Definition 4.9 manifests our black-box treatment of the SUS.

Definition 4.9 (Soundness). Monitor m monitors *soundly* for the (closed) formula φ if for all $p \in \text{PRC}$,

$$\mathbf{rej}(p,m) \text{ implies } p \notin [\![\varphi]\!]$$ ■

Example 4.10. It can be shown that monitor $m_1 \stackrel{\text{def}}{=} \mathrm{rec}X.\big(r.s.X \otimes (a.\mathrm{no} \oplus c.\mathrm{no})\big)$ from Example 3.3 monitors soundly for violations of property $\varphi_4 \stackrel{\text{def}}{=} \max X.\big([r][s]X \wedge ([a]\mathrm{ff} \vee [c]\mathrm{ff})\big)$ from Example 2.4. For instance, recall system $p_1 \stackrel{\text{def}}{=} \mathrm{rec}X.\big(r.s.X + (a.X + c.\mathbf{0})\big)$ from Example 2.3 which produces the two trace prefixes $t_1 = rsa$ and $t_2 = rsc$ over two executions. Example 3.3 illustrates how these prefixes can be veraciously accumulated as a history $H = \{t_1, t_2\}$ once exhibited by p_1, and Example 3.8 shows that such a history is rejected, *i.e.,* $\mathbf{rej}(H, m_1)$. Since $H \subseteq T_p$, we conclude that m_1 rejects this system, *i.e.,* $\mathbf{rej}(p_1, m_1)$. Accordingly, system p_1 violates property φ_4, *i.e.,* $p_1 \notin [\![\varphi_4]\!]$.

By comparison, monitor $m_3 \stackrel{\text{def}}{=} r.s.a.\mathrm{no}$ is *not* sound for φ_4. Specifically, m_3 is capable of collecting and rejecting histories that contain trace t_1, which means it would reject systems such as $\mathrm{rec}X.r.s.a.X$ and $r.s.a.\mathbf{0}$ (since they can exhibit such a trace). However, these two systems do *not* actually violate φ_4. ■

A complementary requirement to soundness is its dual, *i.e.,* (rejection) completeness of Definition 4.11. A monitor m *monitors completely* for a property φ if it can reject *any* violating system based on some history it produces. Put otherwise, the monitor must be capable of identifying *all* violations of φ after having accumulated traces from a *finite* number of monitored executions.

Definition 4.11 (Completeness). Monitor m monitors *completely* for (closed) formula φ if for all $p \in \textsc{Prc}$,

$$p \notin [\![\varphi]\!] \text{ implies } \mathbf{rej}(p, m)$$ ■

Example 4.12. Using Definition 4.11, it can also be shown that m_1 monitors completely for violations of φ_4. Concretely, any system that violates φ_4 should be able to exhibit at least two traces of the form $(rs)^n a$ and $(rs)^n c$ for some $n \geq 0$. Once exhibited (and aggregated by m_1), one can show that m_1 rejects such a history, *i.e.,* $\mathbf{rej}(\{(rs)^n a, (rs)^n c, \ldots\}, m_1)$ using a proof derivation similar to that in Example 3.8.

We also note that not all sound monitors for a property are complete. For instance, although monitor $r.s.(a.\mathrm{no} \oplus c.\mathrm{no})$ is sound for φ_4, it cannot reject a violating system such as $r.s.r.s.(a.\mathbf{0} + c.\mathbf{0})$, which exhibits the traces $rsrsa$ and $rsrsc$ but *not* traces rsa and rsc. This makes the monitor *not* complete. ■

For monitors that are veracious and produce irrevocable verdicts—as those defined in Chapter 3—(rejection) soundness and completeness constitute the basis for our definition of monitor correctness.

Definition 4.13 (Correct Monitoring). A monitor m *monitors correctly* for the (closed) formula φ if it can do so *soundly* and *completely*. ■

Remark 4.14. Although Definitions 2.9 and 4.13 are analogous, their underlying notions of soundness and completeness are different; we opted to redefine the latter to eliminate any potential confusion. Crucially, in the classical RV setup, the meaning of *"m monitors correctly for φ"* was formalised as single-run monitoring (see Definition 2.9) whereas Definition 4.13 redefines it as multi-run monitoring. ■

Example 4.15. Using Examples 4.10 and 4.12, one can show that monitor m_1 monitors correctly for φ_4. The same cannot be said for monitor $r.s.(a.\mathrm{no} \oplus c.\mathrm{no})$ and property φ_4; although this monitor is sound for φ_4, Example 4.12 showed that it is not complete. ■

4.3 Summary

This chapter formalised what it means for a monitor to correctly analyse a property over multiple runs. We argued that a proper definition for monitor correctness needs to take into consideration both the history aggregation and the history analysis mechanisms of Chapter 3. For the former, we showed that executing-monitors are *veracious*, *i.e.,* traces collected by an executing-monitor are indeed traces of the SUS, and *deterministic*, *i.e.,* they always reach equivalent states when analysing the SUS same traces. For the latter, we showed that rejections flagged by monitors during analysis are *irrevocable*, *i.e.,* once a history is rejected by a monitor, further observations (in terms of additional traces, *width*, or longer traces, *length*) do *not* alter this conclusion. We expect two correctness guarantees from the multi-run monitors of Chapter 3, namely *soundness*, *i.e.,* rejections flagged by the verifying monitor do not contradict the semantic meaning of the property being monitored for, and *completeness*, *i.e.,* verifying monitors must reject all violating systems. These two requirements constitute the basis for our definition of monitor correctness and directly affect which properties can be monitored for at runtime and those that cannot, as will be demonstrated in the following chapter.

5 Monitorability

Monitorability [62, 76, 36, 12] delineates the properties that can be correctly monitored for at runtime and those that cannot. In our setting, this translates to a correspondence between the declarative logic semantics of Chapter 2 (*i.e.,* violations of logical formulae) and the operational monitor semantics of Chapter 3 (*i.e.,* monitor rejections). More specifically, monitorability is parametric w.r.t. the monitoring set-up assumed (*i.e.,* the operational semantics of Figure 2.1) and the definition of the statement *"monitor m monitors correctly for property φ"* from Chapter 4, formalised in Definition 4.13.

Apart from defining the notion of correct monitoring, monitorability plays a crucial in the development of RV tools by guiding the design of synthesis procedures that automatically generate (correct) monitors from logical formulae [32, 15, 30, 102, 37, 56, 108]. Distinguishing between properties that can and cannot be correctly monitored for has other practical advantages. For instance, synthesis procedures could be optimised to generate monitors *only* for monitorable properties [32, 30], avoiding unnecessary computation for those that cannot be verified at runtime. In certain settings, a *syntactic characterisation* for the *maximal* subset of formulae that are monitorable can also be identified [6, 10, 74, 76]. This, in turn, provides us with a simple yet effective syntactic check for determining whether a formula is worth runtime verifying. Concretely, if a formula falls within the scope of this fragment, we are assured that the synthesised monitor will *always* reach meaningful verdicts. Otherwise, an alternative verification technique, such as model checking, must be employed. This directly shapes the design of *hybrid* verification strategies that combine RV with other verification techniques [107].

This chapter investigates the increase in expressive power when moving from single-run monitoring to multi-run (using the operational model of Chapter 4). The approach we adopt follows closely that in [75] and applies to a variety of settings [115, 10, 86, 47, 17, 64, 28]. It fosters a separation of concerns between the specification semantics and the verification method employed. Crucially, such an approach ensures that the operational model can be adapted (*e.g.* by optimising it to aggregate the histories of Section 3.1 more efficiently), or even replaced, without affecting the set of monitorable properties, provided that the verifying monitors continue to satisfy the same correctness requirements.

Structure of the chapter. Section 5.1 formally defines what it means for a property to be *monitorable* over multiple runs. Based on this definition, we identify an *extended* logical fragment that is monitorable in our augmented monitoring setup; the proofs are given in the dedicated Section 5.2. Section 5.3 then establishes that the identified logical fragment is *maximally expressive*, *i.e.,* every property that can be monitored for correctly via the multi-run monitors of Chapter 3 can always be expressed as a formula in this fragment; the proofs are given in the dedicated Section 5.4. Section 5.5 concludes.

5.1 The Monitorable Logical Fragment

To investigate what properties can be verified at runtime using the multi-run monitors of Chapter 3, we start by formally defining what it means for a property to be monitorable. More specifically, Definition 5.1 below is *parametric* w.r.t. the definition of *"m monitors correctly for φ"*; prior work [76] formalised this as single-run monitoring (see Definition 2.9) whereas Definition 4.13 redefines it as multi-run monitoring.

Definition 5.1 (Monitorability [10, 76]). A formula $\varphi \in \text{RECHML}$ is *monitorable* iff there exists a monitor $m \in \text{MON}$ that *monitors correctly* for it, *i.e.,* soundly and completely. A (sub)logic $\mathcal{L} \subseteq \text{RECHML}$ is monitorable iff every formula $\varphi \in \mathcal{L}$ is monitorable. ■

Our multi-run monitoring setup allows us to extend the monitorability limits established in [10, 76]. Specifically, we identify $\text{sHML}^\vee$, a syntactic subset of RECHML that *extends* sHML (Theorem 2.12) with disjunctions, and show it is monitorable according to Definition 5.1.

Definition 5.2 (Disjunctive sHML). The RECHML safety fragment with disjunctions, $\text{sHML}^\vee$, is defined inductively as follows:

$$\varphi, \psi \in \text{sHML}^\vee \ ::= \ \mathsf{tt} \ \mid \ \mathsf{ff} \ \mid \ [\alpha]\varphi \ \mid \ \varphi \wedge \psi \ \mid \ \varphi \vee \psi \ \mid \ \max X.\varphi \ \mid \ X$$ ■

Showing that a logical fragment, $\mathcal{L} \subseteq \text{RECHML}$, is monitorable is non-trivial due to the various universal quantifications that need to be considered, *e.g.* all $\varphi \in \mathcal{L}$ from Definition 5.2 and all $p \in \text{PRC}$ from Definitions 4.9 and 4.11. We prove the monitorability of $\text{sHML}^\vee$ systematically, by concretising the existential quantification of a correct monitor for every formula in $\text{sHML}^\vee$ via the monitor synthesis function $(\!|-|\!)$ in Definition 5.3. We then prove that for any $\varphi \in \text{sHML}^\vee$, the generated monitor $(\!|\varphi|\!)$ monitors correctly for it, following Definition 4.13. A pleasant by-product of this proof strategy is that the synthesis function in Definition 5.3 can be used directly for tool construction to automatically generate (correct) witness monitors from specifications; see [32, 15].

Definition 5.3 (Monitor Synthesis). The function $(\!|-|\!) : \text{sHML}^\vee \to \text{MON}$ is defined as follows:

$$(\!|\mathsf{ff}|\!) \stackrel{\text{def}}{=} \mathsf{no} \qquad (\!|\varphi \wedge \varphi|\!) \stackrel{\text{def}}{=} (\!|\varphi|\!) \otimes (\!|\varphi|\!) \qquad (\!|[\alpha]\varphi|\!) \stackrel{\text{def}}{=} \alpha.(\!|\varphi|\!) \qquad (\!|X|\!) \stackrel{\text{def}}{=} X$$
$$(\!|\mathsf{tt}|\!) \stackrel{\text{def}}{=} \mathsf{end} \qquad (\!|\varphi \vee \varphi|\!) \stackrel{\text{def}}{=} (\!|\varphi|\!) \oplus (\!|\varphi|\!) \qquad (\!|\max X.\varphi|\!) \stackrel{\text{def}}{=} \mathsf{rec}X.(\!|\varphi|\!)$$ ■

Remark 5.4. The synthesis function $(\!|-|\!)$ can be optimised to remove redundant terms, *e.g.* $(\!|[r]\mathsf{ff} \wedge [r][s]\mathsf{ff}|\!)$ would return $r.\mathsf{no}$ instead of $r.\mathsf{no} \otimes r.s.\mathsf{no}$. For this version of the work, we opted for a simpler synthesis to better illustrate the correspondence between formulae and the monitors verifying them. ■

Theorem 5.5 (Monitorability). All (closed) formulae $\varphi \in \text{sHML}^\vee$ are monitorable. ■

The proof for Theorem 5.5 is given in the dedicated Section 5.2 which may be skipped upon first reading. This result equips us with a simple syntactic check for determining whether a formula is monitorable or not according to Definition 5.1 through a simple syntactic check.

Example 5.6. Since $\varphi_1, \varphi_2, \varphi_3, \varphi_4 \in \text{sHML}^\vee$ (Examples 1.1, 1.3 and 2.4), we know they are monitorable, following Theorem 5.5. Using Definition 5.3, we can also automatically synthesise a corresponding correct monitor, *e.g.* $(\!|\varphi_4|\!) = m_1$ (Example 3.3). ■

5.2 Proving Monitorability

Theorem 5.5 is proven in two steps; first, we show the monitors generated via the synthesis function $(\!|-|\!)$ are sound, Proposition 5.12, and then we show they are complete, Proposition 5.13. To prove these results, we give an alternative definition to that of Figure 2.1 that characterises the specific histories along which systems violate (closed) formulae in Definition 5.7 (a form of derivative semantics [124] adapted from [2]).

Definition 5.7 (Violation Relation). The *violation relation*, denoted as $\models_v$, is the least relation of the form $(\text{HST} \times \text{sHML}^{\vee})$ satisfying the following rules:

$$\text{vF}\ \frac{H \neq \emptyset}{H \models_v \mathtt{ff}} \qquad \text{vUM}\ \frac{\mathit{suffix}(H,\alpha) \models_v \varphi}{H \models_v [\alpha]\varphi} \qquad \text{vANDL}\ \frac{H \models_v \varphi}{H \models_v \varphi \wedge \psi} \qquad \text{vANDR}\ \frac{H \models_v \psi}{H \models_v \varphi \wedge \psi} \qquad \text{vOR}\ \frac{H \models_v \varphi \quad H \models_v \psi}{H \models_v \varphi \vee \psi}$$

$$\text{vMAX}\ \frac{H \models_v \varphi[\max X.\varphi/X]}{H \models_v \max X.\varphi}$$

We read "H violates φ" whenever $H \models_v \varphi$. ■

Example 5.8. System $p_1 \stackrel{\text{def}}{=} \text{rec}X.\big(r.s.X + (a.X + c.\mathbf{0})\big)$ from Example 2.3 can exhibit the history $\{rsa,rsc\}$, which violates formula φ_4, *i.e.,* $\{rsa,rsc\} \models_v \varphi_4$. This is justified by the left proof derivation below, which is very similar to that in Example 3.8.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\{\epsilon\} \models_v \mathtt{ff}}\text{vF}}{\{a,c\} \models_v [a]\mathtt{ff}}\text{vUM} \quad \dfrac{\dfrac{}{\{\epsilon\} \models_v \mathtt{ff}}\text{vF}}{\{a,c\} \models_v [c]\mathtt{ff}}\text{vUM}}{\{a,c\} \models_v [a]\mathtt{ff} \vee [c]\mathtt{ff}}\text{vOR}}{\{a,c\} \models_v [r][s]\varphi_4 \wedge ([a]\mathtt{ff} \vee [c]\mathtt{ff})}\text{vANDR}}{\{a,c\} \models_v \varphi_4}\text{vMAX}}{\{sa,sc\} \models_v [s]\varphi_4}\text{vUM}}{\{rsa,rsc\} \models_v [r][s]\varphi_4}\text{vUM}}{\{rsa,rsc\} \models_v [r][s]\varphi_4 \wedge ([a]\mathtt{ff} \vee [c]\mathtt{ff})}\text{vANDL}}{\{rsa,rsc\} \models_v \varphi_4}\text{vMAX}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\{\epsilon\} \models_v \mathtt{ff}}\text{vF}}{\{a\} \models_v [a]\mathtt{ff}}\text{vUM} \quad \dfrac{\emptyset \models_v \mathtt{ff} \quad (**)}{\{a\} \models_v [c]\mathtt{ff}}\text{vUM}}{\{a\} \models_v [a]\mathtt{ff} \vee [c]\mathtt{ff}}\text{vOR}}{\{a\} \models_v [r][s]\varphi_4 \wedge ([a]\mathtt{ff} \vee [c]\mathtt{ff})}\text{vANDR}}{\{a\} \models_v \varphi_4}\text{vMAX}}{\{sa\} \models_v [s]\varphi_4}\text{vUM}}{\{rsa\} \models_v [r][s]\varphi_4}\text{vUM}}{\{rsa\} \models_v [r][s]\varphi_4 \wedge ([a]\mathtt{ff} \vee [c]\mathtt{ff})}\text{vANDL}}{\{rsa\} \models_v \varphi_4}\text{vMAX}$$

However, formula φ_4 cannot be violated with a history that contains fewer traces, such as $\{rsa\}$, *i.e.,* $\{rsa\} \not\models_v \varphi_4$. This is depicted in the right proof derivation above, since no rule from Definition 5.7 can justify the judgement $\emptyset \models_v \mathtt{ff}$ at $(**)$. ■

Despite the technical discrepancies between the definitions for property violations in Figure 2.1 and Definition 5.7, *i.e.,* $p \notin [\![\varphi]\!]$ and $H \models_v \varphi$, we can show that the two definitions correspond. The proof for Theorem 5.9 is quite involved and can be found in Appendix A.

Theorem 5.9 (Correspondence). For all (closed) formulae $\varphi \in \text{sHML}^{\vee}$ and systems $p \in \text{PRC}$,

$$p \notin [\![\varphi]\!] \quad \text{iff} \quad (\exists H \subseteq T_p \text{ such that } H \models_v \varphi)$$

■

Soundness and completeness of the monitors generated by $(\!|-|\!)$, formalised in Propositions 5.12 and 5.13 below, rely on a number of additional results. In particular, Lemmas 10 and 11 show there is a tight correspondence between the rejected histories, $\mathbf{rej}(H,m)$, and the histories that violate formulae, $H \models_v \varphi$.

Lemma 10. For all (closed) formulae $\varphi \in \text{sHML}^\vee$ and histories $H \in \text{Hst}$, if $\mathbf{rej}(H,(\!|\varphi|\!))$ then $H \models_v \varphi$.

Proof. By induction on the derivation of $\mathbf{rej}(H,(\!|\varphi|\!))$. We outline the main cases:

- Case Act. We know $\mathbf{rej}(H,\alpha.m)$ because $\mathbf{rej}(\mathit{suffix}(H,\alpha),m)$ where $\varphi = [\alpha]\psi$ and $m = (\!|\psi|\!)$. By the IH, we deduce $\mathit{suffix}(H,\alpha) \models_v \psi$. Our result, $H \models_v [\alpha]\psi$, follows by applying rule vUm from Definition 5.7.
- Case Rec. We know $\mathbf{rej}(H,\text{rec}X.m)$ because $\mathbf{rej}(H,m[^{\text{rec}X.m}/_X])$ where $\varphi = \max X.\psi$ and $m = (\!|\psi|\!)$. We can also show that $m[^{\text{rec}X.m}/_X] = (\!|\psi|\!)[^{(\!|\max X.\psi|\!)}/_X] = (\!|\psi[^{\max X.\psi}/_X]|\!)$. Using the IH, we obtain $H \models_v \psi[^{\max X.\psi}/_X]$. Our result, $H \models_v \max X.\psi$, follows by rule vMax from Definition 5.7.

The remaining cases follow with similar reasoning. □

Lemma 11. For all (closed) formulae $\varphi \in \text{sHML}^\vee$ and histories $H \in \text{Hst}$, if $H \models_v \varphi$ then $\mathbf{rej}(H,(\!|\varphi|\!))$.

Proof. The proof is by rule induction on $H \models_v \varphi$, following an approach similar to that for Lemma 10. □

Based on these results, we are now in a position to prove the soundness and completeness of the monitors generated by $(\!|-|\!)$.

Proposition 5.12 (Soundness). For any formula $\varphi \in \text{sHML}^\vee$, monitor $(\!|\varphi|\!)$ monitors soundly for φ.

Proof. Expanding Definitions 3.5 and 4.9, we need to show that for all $\varphi \in \text{sHML}^\vee$ and all $p \in \text{Prc}$,

$$\text{if } \big(\exists H \subseteq T_p \text{ such that } \mathbf{rej}(H,(\!|\varphi|\!))\big) \text{ then } p \notin [\![\varphi]\!]$$

Suppose that $\exists H \subseteq T_p$ such that $\mathbf{rej}(H,(\!|\varphi|\!))$. By Lemma 10, we deduce $H \models_v \varphi$. Our result, $p \notin [\![\varphi]\!]$, then follows using Theorem 5.9. □

Proposition 5.13 (Completeness). For any formulae $\varphi \in \text{sHML}^\vee$, monitor $(\!|\varphi|\!)$ monitors completely for φ.

Proof. Expanding Definitions 3.5 and 4.11, we need to show that for all $\varphi \in \text{sHML}^\vee$ and all $p \in \text{Prc}$,

$$\text{if } p \notin [\![\varphi]\!] \text{ then } \big(\exists H \subseteq T_p \text{ such that } \mathbf{rej}(H,(\!|\varphi|\!))\big)$$

Suppose $p \notin [\![\varphi]\!]$. By Theorem 5.9, we know $\exists H \subseteq T_p$ such that $H \models_v \varphi$. Our result, $\mathbf{rej}(H,(\!|\varphi|\!))$, follows by Lemma 11. □

We are now in a position to prove Theorem 5.5, which we restate below.

Theorem 5.14 (Monitorability). All (closed) formulae $\varphi \in \text{sHML}^\vee$ are monitorable. ■

Proof. Follows from Propositions 5.12 and 5.13, with $(\!|\varphi|\!)$ as the witness correct monitor. □

5.3 Expressiveness of the Monitorable Logical Fragment

Although Theorem 5.5 showed that the monitorability limits identified in previous works [10, 76] can be extended with multiple executions, several formulae are still unmonitorable (for violations) according to Definition 5.1, particularly when they include existential modalities, $\langle\alpha\rangle\varphi$, and least fixed points, $\min X.\varphi$.

Intuitively, violations of formulae containing existential modalities are unmonitorable because they require monitoring to aggregate *all* the SUS traces from a certain point of the execution onwards. Since SUS traces can, in general, be infinite, this requires history analysis to consider histories with an *infinite number* of traces, which cannot be aggregated over unbounded (albeit finite) multiple runs.

Example 5.15. Consider the formula $\langle\alpha\rangle\mathsf{tt}$. Assume, towards a contradiction, that there exists a sound and complete monitor m for it. By the soundness and completeness requirements of Definitions 4.9 and 4.11, this means that for all systems $p \in \textsc{Prc}$, we have

$$p \notin [\![\varphi]\!] \quad \text{iff} \quad \mathbf{rej}(p,m) \tag{5.1}$$

Pick an arbitrary violating system $q \notin [\![\langle\alpha\rangle\mathsf{tt}]\!]$, *i.e.,* $q \overset{\alpha}{\not\longrightarrow}$. By eq. (5.1), monitor m must reject such a system, *i.e.,* $\mathbf{rej}(q,m)$. Expanding Definition 3.5, we this means that

$$\exists H \subseteq T_q \text{ such that } \mathbf{rej}(H,m)$$

However, from system q, we can build another (deterministic) system $q+\alpha.\mathbf{0}$ that *satisfies* the formula $\langle\alpha\rangle\mathsf{tt}$, *i.e.,* $q+\alpha.\mathbf{0} \in [\![\langle\alpha\rangle\mathsf{tt}]\!]$. We also know that H is a history of $q+\alpha.\mathbf{0}$ since $H \subseteq T_q \subseteq T_{q+\alpha.\mathbf{0}}$ which, by Definition 3.5, means that m *rejects* $q+\alpha.\mathbf{0}$, *i.e.,* $\mathbf{rej}(q+\alpha.\mathbf{0},m)$. This and $q+\alpha.\mathbf{0} \in [\![\langle\alpha\rangle\mathsf{tt}]\!]$ make monitor m unsound, contradicting our initial assumption that m is sound for $\langle\alpha\rangle\mathsf{tt}$ (apart from being complete). ■

Similarly, violations of formulae containing least fixed points are unmonitorable because they require monitors to analyse traces of *infinite length*, which cannot be aggregated by the operational semantics of Figure 3.1. This is better illustrated in Example 5.16 below.

Example 5.16. Using an argument similar to that in Example 5.15, we can show that the least fixed point formula $\min X.([\alpha]X \wedge [\beta]\mathsf{ff})$, describing system that eventually stop producing action α and never produce action β, is not monitorable according to Definition 5.1. Assume, towards a contradiction, that there exists a monitor m that monitors soundly and completely for this formula. Due to the soundness and completeness requirement of Definitions 3.5, 4.9 and 4.11, this means that for all $p \in \textsc{Prc}$, we have that

$$p \notin [\![\varphi]\!] \quad \text{iff} \quad \big(\exists H \subseteq T_p \text{ such that } \mathbf{rej}(H,m)\big) \tag{5.2}$$

The single-state system q with the sole transition $q \overset{\alpha}{\longrightarrow} q$ violates this formula, *i.e.,* $q \notin [\![\min X.([\alpha]X \wedge [\beta]\mathsf{ff})]\!]$. Using eq. (5.2), we must have $\mathbf{rej}(H,m)$ for some $H \subseteq T_q$. From Figure 3.1 and the structure of q, we also know H is a *finite* set of the form $\{\alpha^n \mid n \in \mathbb{N}\}$. Fix k to be the length of the *longest* trace in H and then consider the system r consisting of $k+1$ states that exclusively has the transitions $r = r_0 \overset{\alpha}{\longrightarrow} \ldots \overset{\alpha}{\longrightarrow} r_k$ (and nothing else). Clearly, system r satisfies $\min X.([\alpha]X \wedge [\beta]\mathsf{ff})$, *i.e.,* $r \in [\![\min X.([\alpha]X \wedge [\beta]\mathsf{ff})]\!]$. But since H is also a history for r, by Definition 3.5, monitor m must also reject r, *i.e.,* $\mathbf{rej}(r,m)$. This, however, contradicts the initial assumption that m is sound (and complete). ■

We can show that the syntactic subset sHML$^\vee$ of Definition 5.2 is the largest monitorable fragment of recHML w.r.t. the extended multi-run monitorability setup of Chapter 3. In particular, every property that can be monitored correctly according Definition 4.13 is always expressible as a formula in the sHML$^\vee$. However, we first define what we mean by language inclusion up to *semantic equivalence*.

Definition 5.17 (Language Inclusion). For all $\mathcal{L}_1, \mathcal{L}_2 \subseteq$ recHML,

$$\mathcal{L}_1 \sqsubseteq \mathcal{L}_2 \stackrel{\text{def}}{=} \text{For all } \varphi_1 \in \mathcal{L}_1, \text{ there exists } \varphi_2 \in \mathcal{L}_2 \text{ such that } [\![\varphi_1]\!] = [\![\varphi_2]\!] \qquad \blacksquare$$

[Maximality] If a language $\mathcal{L} \subseteq$ recHML is monitorable w.r.t. Mon, then every property $\varphi \in \mathcal{L}$ can be expressed as a property $\psi \in$ sHML$^\vee$ *i.e.,* $\mathcal{L} \sqsubseteq$ sHML$^\vee$. $\blacksquare$

The proof for Section 5.3 is given in the dedicated Section 5.4 which may be skipped upon first reading. This result ensures that restricting verification to the (sub)logic of Definition 5.2 does not impinge on the set of properties that can be monitored for at runtime (using the multi-run monitor of Chapter 3). Stated otherwise, one can determine if an arbitrary recHML formula is monitorable by reformulating it into a (semantically) equivalent formula from sHML$^\vee$. This would allow a verification framework to decide whether to runtime verify the property or employ alternative verification techniques, such as model checking. We note that even though the problem of checking for equivalence across formulae is non-trivial [61], it is arguably easier than using Definition 5.1; for the latter, numerous quantifications need to be considered, as already stated in Section 5.1. Section 5.3 is also useful for the construction of RV tools: the development of automated monitor synthesis can exclusively target the identified syntactic fragment sHML$^\vee$, assured that all monitorable properties are still covered.

Although Section 5.3 is defined in terms of the analytical powers of our extended multi-run setup, we can show that this result also holds for *arbitrary* monitoring setups, Theorem 5.21. This result relies on the notion of *bad histories*, where a history is said to be bad for a formula φ if all systems that produce that history also violate φ, Definition 5.18. In other words, that history demonstrates enough evidence to conclude with certainty the violation of the formula. Definition 5.20 then gives an alternative definition for monitorability based on the history evidenced, where a formula is *monitorable via bad histories* if all violating systems are capable of producing a bad history for it.

Definition 5.18 (Bad History). History H is *bad* for the (closed) formula φ if for all systems $p \in$ Prc such that $H \subseteq T_p$ then $p \notin [\![\varphi]\!]$. $\blacksquare$

Example 5.19. We can show that the history $H = \{rsc, rsa\}$ is bad for the formula φ_4 from Example 2.4. Concretely, Example 5.8 asserts that H violates φ_4, *i.e.,* $H \models_v \varphi_4$. Using Theorem 5.9, we can conclude that all systems p capable of producing H, *i.e.,* $H \subseteq T_p$, violate φ_4, *i.e.,* $p \notin [\![\varphi_4]\!]$. The same cannot be said for histories that contain fewer traces, such as $\{rsa\}$: system such as $\mathrm{rec}X.r.s.a.X$ and $r.s.a.\mathbf{0}$ can generate such a history but they do *not* violate φ_4. $\blacksquare$

Definition 5.20 (Monitorable via Bad Histories). A (closed) formula φ is *monitorable via bad histories* if for every $p \in$ Prc, $p \notin [\![\varphi]\!]$ iff $(\exists H \subseteq T_p$ such that H is bad for $\varphi)$. $\blacksquare$

Theorem 5.21 (General Maximality). If a language $\mathcal{L} \subseteq$ recHML is monitorable via bad histories, then $\mathcal{L}$ cannot (semantically) express more properties than sHML$^\vee$, *i.e.,* $\mathcal{L} \sqsubseteq$ sHML$^\vee$. $\blacksquare$

We highlight the fact that Theorem 5.21 is a stronger statement than Section 5.3, as it is independent of the monitoring setup used. At the same time, it follows from Theorem 5.21 that the extended multi-run monitoring setup of Chapter 3 is general enough to reject all violating systems since all properties that are monitorable via bad histories are also monitorable according to Definition 5.1, and vice versa.

5.4 Proving Maximal Expressiveness

The first step towards showing that sHML$^\vee$ is maximally expressive, formalised in Section 5.3, is to define expressive completeness w.r.t. the monitoring setup Mon of Chapter 3.

Definition 5.22 (Expressive-complete). A language $\mathcal{L} \subseteq$ recHML is *expressive-complete* if for all monitors $m \in$ Mon, there exists some (closed) formula $\varphi \in \mathcal{L}$ such that m monitors correctly for it. ■

We prove that the language sHML$^\vee$ is expressive-complete systematically, by concretising the existential quantification of a corresponding sHML$^\vee$ formula for every monitor in Mon via the function $\langle\langle - \rangle\rangle$ in Definition 5.23. We note that the function $\langle\langle - \rangle\rangle$ is the inverse of $(\!|-|\!)$ from Definition 5.3.

Definition 5.23 (Reverse Monitor Synthesis). The function $\langle\langle - \rangle\rangle : \text{Mon} \to \text{recHML}$ is defined inductively on the structure of m as follows:

$$\langle\langle \mathsf{no} \rangle\rangle \stackrel{\text{def}}{=} \mathsf{ff} \qquad \langle\langle m \otimes n \rangle\rangle \stackrel{\text{def}}{=} \langle\langle m \rangle\rangle \wedge \langle\langle n \rangle\rangle \qquad \langle\langle \alpha.m \rangle\rangle \stackrel{\text{def}}{=} [\alpha]\langle\langle m \rangle\rangle \qquad \langle\langle X \rangle\rangle \stackrel{\text{def}}{=} X$$
$$\langle\langle \mathsf{end} \rangle\rangle \stackrel{\text{def}}{=} \mathsf{tt} \qquad \langle\langle m \oplus n \rangle\rangle \stackrel{\text{def}}{=} \langle\langle m \rangle\rangle \vee \langle\langle n \rangle\rangle \qquad \langle\langle \mathsf{rec}X.m \rangle\rangle \stackrel{\text{def}}{=} \mathsf{max}X.\langle\langle m \rangle\rangle$$ ■

Lemma 24 below shows that there is a tight correspondence between the systems rejected by monitors, *i.e.,* $\mathbf{rej}(p,m)$, and the systems that violate the formulae generated by $\langle\langle - \rangle\rangle$, *i.e.,* $p \notin [\![\langle\langle m \rangle\rangle]\!]$. In particular, monitor m monitors correctly (*i.e.,* soundly and completely) for the formula $\langle\langle m \rangle\rangle$.

Lemma 24. For any system $p \in$ Prc and monitor $m \in$ Mon, $\mathbf{rej}(p,m)$ iff $p \notin [\![\langle\langle m \rangle\rangle]\!]$.

Proof. Since $m = (\!|\langle\langle m \rangle\rangle|\!)$, our result follows from Propositions 5.12 and 5.13 by letting $\varphi = \langle\langle m \rangle\rangle$. □

Theorem 5.25. The (sub)logic sHML$^\vee$ is Expressive-Complete w.r.t. Mon.

Proof. Follows by Lemma 24 since we can show $\langle\langle m \rangle\rangle \in$ sHML$^\vee$ for any $m \in$ Mon. □

We are now in a position to prove the main result of this section, namely that sHML$^\vee$ is the largest monitorable subset of recHML up to logical equivalence, Section 5.3 restated below.

[Maximality] If a language $\mathcal{L} \subseteq$ recHML is monitorable w.r.t. Mon, then every property $\varphi \in \mathcal{L}$ can be expressed as a property $\psi \in$ sHML$^\vee$ *i.e.,* $\mathcal{L} \sqsubseteq$ sHML$^\vee$. ■

Proof. Since $\mathcal{L}$ is monitorable w.r.t. Mon, by Definition 5.1, we have that for all $\varphi \in \mathcal{L}$,

$$\exists m \in \text{Mon} \text{ such that } m \text{ monitors correctly for } \varphi$$

Pick a formula $\varphi \in \mathcal{L}$ and assume there exists a monitor $m \in$ Mon such that m monitors correctly for it. Expanding Definition 4.13, followed by Definitions 4.9 and 4.11, we know that for all systems $p \in$ Prc,

$$\mathbf{rej}(p,m) \text{ iff } p \notin [\![\varphi]\!] \tag{5.3}$$

According to Definition 5.17, we need to show $\exists \psi \in \text{sHML}^{\vee}$ such that $[\![\varphi]\!] = [\![\psi]\!]$.

Using Theorem 5.25, for the monitor m used in eq. (5.3), we also know $\exists \psi \in \text{sHML}^{\vee}$ where $\psi = \langle\langle m \rangle\rangle$ and m monitors correctly for ψ. Expanding Definition 4.13, followed by Definitions 4.9 and 4.11, this means that for all systems $p \in \text{Prc}$, we have that

$$\mathbf{rej}(p, m) \text{ iff } p \notin [\![\psi]\!] \tag{5.4}$$

From eqs. (5.3) and (5.4), we immediately conclude that φ and ψ are equivalent, *i.e.,* $[\![\varphi]\!] = [\![\psi]\!]$. □

5.5 Summary

This chapter showed that the monitorability limits established in [10, 76] can be partly mitigated by observing multiple system executions. Our results demonstrate that monitors can extract sufficient information over multiple runs (of a deterministic SUS) to correctly detect the violation of a class of branching-time properties that may contain disjunctions (Theorem 5.5). We also proved that the identified monitorable fragment $\text{sHML}^{\vee}$ from Definition 5.2 is maximally expressive (Section 5.3 and Theorem 5.21). In particular, every property that can be monitored correctly using our monitoring framework can always be expressed as a formula in $\text{sHML}^{\vee}$. Such a syntactic characterisation of monitorable properties is useful for tool construction, as will be shown later in Part III of this thesis.

Part II

Repeated Monitoring for Non-Deterministic Systems

6 Revisiting the System Language

In this part of the thesis, we extend the work in Part I, which was limited to deterministic systems, to potentially non-deterministic systems. We start by redefining the system language presented in Chapter 2. In particular, we model system as Instrumentable Labelled Transition Systems, which extend standard LTSs with additional properties. These properties permit the instrumentation mechanism to extract more information about the SUS, which are then forwarded to the the monitor for analysis.

Structure of the chapter. Section 6.1 introduces the system language that will be used throughout this part of the thesis, building on that in Chapter 2. Section 6.2 argues that although an instrumented monitor is privy to more information about the SUS, the SUS is still treated as a black-box. In Section 6.3, we revisit the specification logic presented in Chapter 2 and discuss how it is affected by the reformulated system language of Section 6.1. Section 6.4 concludes.

6.1 The Model

We presuppose a set of actions, $\eta, \xi \in \textsc{Act} = \textsc{tAct} \uplus \{\tau\}$, with a distinguished *silent (untraceable) action* τ and a set of *traceable actions*, $\mu, \lambda \in \textsc{tAct} = \textsc{eAct} \uplus \textsc{iAct}$, that consists of two disjoint sets. *External actions*, $\alpha, \beta \in \textsc{eAct}$, describe the computation steps observable to an outside entity which are the subject of correctness specifications. *Internal actions*, $\gamma, \delta \in \textsc{iAct}$, describe some internal computation performed by the SUS. They are *not* of concern to correctness specifications but can still be discerned by a monitor with the appropriate instrumentation mechanism. Notably, silent actions (denoted by τ) *cannot* be traced by the instrumented monitors. This differs from the setup of Part I; there, we assumed that monitors could only observe external actions as all internal actions were represented by the silent action τ.

A SUS is modelled as an *Instrumentable Labelled Transition System* (ILTS), a septuple of the form

$$\langle\, \textsc{Prc}, \equiv, \textsc{eAct}, \textsc{iAct}, \{\tau\}, \rightarrow, \textsc{Det} \,\rangle$$

where SUS states are denoted by a set of processes, $p, q \in \textsc{Prc}$, with an associated equivalence relation denoted as $\equiv \subseteq \textsc{Prc} \times \textsc{Prc}$; two states p and q are said to be *equivalent* whenever $p \equiv q$.

The ternary transition relation, $\longrightarrow \subseteq (\textsc{Prc} \times \textsc{Act} \times \textsc{Prc})$, is defined over arbitrary actions (*i.e.,* silent, internal and external). We adopt a similar convention to that of Chapter 2 and write $p \xrightarrow{\eta} q$ to denote labelled state transitions, $(p, \eta, q) \in \longrightarrow$, and $p \xrightarrow{\eta}\!\!\!\!\!/\;\;$ whenever $\nexists q \in \textsc{Prc}$ such that $p \xrightarrow{\eta} q$. Transitions in this ILTS abstract over equivalent states, that is:

$$\text{for any } p \equiv q, \text{ if } p \xrightarrow{\eta} p' \text{ then there exists } q' \text{ such that } q \xrightarrow{\eta} q' \text{ where } p' \equiv q' \tag{6.1}$$

Remark 6.1. From Equation (6.1) it follows that the equivalence relation is a strong bisimulation [3]: if $p \equiv q$ then p and q are strongly bisimilar. Moreover, since equivalent systems, $p \equiv q$, exhibit identical behaviour, they produce exactly the same traces, $T_p = T_q$. ■

Instrumentation (to be later discussed in Chapter 7) can also abstract over (*non*-traceable) silent τ-transitions because they are *confluent* w.r.t. all other actions $\eta \in \text{Act}$, that is:

$$\begin{gathered}\text{for any } p, \text{ whenever } p \xrightarrow{\tau} p' \text{ and } p \xrightarrow{\eta} p'' \text{ then (i) either } \eta = \tau \text{ and } p' \equiv q', \text{ or} \\ \text{(ii) there exists a state } q \text{ and transitions } p' \xrightarrow{\eta} q \text{ and } p'' \xrightarrow{\tau} q \text{ joining the diamond.}\end{gathered}$$

Remark 6.2. Silent actions capture β-moves in the sense of [131, 83] and arise naturally in linearly-typed systems and as thread-local moves in concurrent systems. For instance, the system $\left(\text{let } x = v_1 \text{ in } e_1\right) \parallel \left(\text{let } y = v_2 \text{ in } e_2\right)$ can silently transition to either $e_1[^{v_1}/_x] \parallel \left(\text{let } y = v_2 \text{ in } e_2\right)$ or $\left(\text{let } x = v_1 \text{ in } e_1\right) \parallel e_2[^{v_2}/_y]$. In each case, system $e_1[^{v_1}/_x] \parallel e_2[^{v_2}/_y]$ can be reached in one further silent step. This means that τ-transitions are deterministic w.r.t. $\equiv$ (as defined in the next paragraph). ■

Finally, an ILTS partitions traceable actions via the predicate $\text{Det} : \text{tAct} \to \text{Bool}$ where all traceable actions $\mu \in \text{tAct}$ satisfying the predicate, $\text{Det}(\mu) = \text{true}$, must be *deterministic* (up to $\equiv$), that is:

$$\text{if } p \xrightarrow{\mu} p' \text{ and } p \xrightarrow{\mu} p'' \text{ then } p' \equiv p''$$

Remark 6.3. The outcomes of the predicate Det can be determined by analysing the actions in tAct. *E.g.* Part IV shows that many systems are deterministic w.r.t. a subset of actions, such as asynchronous systems and output actions. Crucially, Det treats actions as deterministic only when this is known with certainty. All other actions, which could be either deterministic or non-deterministic, are conservatively treated as non-deterministic. We also observe that ILTSs do not limit the range of systems modelled. In particular, deterministic systems (including those of Part I) can be described by requiring $\text{Det}(\mu) = \text{true}$ for all actions, whereas for general systems, we would always have $\text{Det}(\mu) = \text{false}$. ■

Weak transitions, $p \Longrightarrow q$, abstract over both silent and internal actions, whereas *weak traceable transitions*, $p \Rightarrow_T q$, abstract over silent actions only. Concretely, for external actions $\alpha \in \text{eAct}$ and traceable actions $\mu \in \text{tAct}$, we adopt the following conventions:

- $p \Longrightarrow q$ means that either $p = q$ or $\exists p' \in \text{Prc}$ and $\eta \in \left(\{\tau\} \cup \text{iAct}\right)$ such that $p \xrightarrow{\eta} p' \Rightarrow q$
- $p \Rightarrow_T q$ means that either $p = q$ or $\exists p' \in \text{Prc}$ such that $p \xrightarrow{\tau} p' \Rightarrow_T q$
- $p \overset{\alpha}{\Longrightarrow} q$ means $\exists p', p'' \in \text{Prc}$ such that $p \Rightarrow p' \xrightarrow{\alpha} p'' \Rightarrow q$
- $p \overset{\mu}{\Rightarrow}_T q$ means $\exists p', p'' \in \text{Prc}$ such that $p \Rightarrow_T p' \xrightarrow{\mu} p'' \Rightarrow_T q$

Traceable actions can be sequenced to form (augmented) traces, $t, u \in \text{Trc} = \text{tAct}^*$, representing (finite) prefixes of system runs. This is in contrast with the traces of Part I, which only consist of external actions. We adopt the same conventions as Chapter 2. For instance, for the (augmented) trace $t = \mu_1 \cdots \mu_n$, we write $p \overset{t}{\Rightarrow}_T q$ instead of the sequence of transitions $p \overset{\mu_1}{\Longrightarrow}_T \cdots \overset{\mu_n}{\Longrightarrow}_T q$.

(a) *SUS p as viewed by an external observer* **(b)** *SUS p as viewed by the instrumented monitor*

Figure 6.1. *Global views of the SUS*

6.2 A Black-box Approach

Most work on RV focuses on designing monitors that can analyse *any* system. This is typically done by treating the system as a *black-box* that emits the necessary events, without giving any information about its internal state, source code or implementation details. However, in practice, systems are rarely treated as complete black-boxes. More specifically, for an outline instrumentation to record enough relevant information about the SUS so that the monitor can check it against a correctness specification of interest, instrumentation must possess a certain degree of prior knowledge about the SUS itself. At the very least, it must know the type of systems being runtime verified (*e.g.* web services, embedded systems or message-passing systems) as it directly determines the kind of information that will be recorded by observing the computation of a running systems (*e.g.* API calls, read/write on files or send/receive messages). For instance, in Part I, as well as in other works on RV [10, 7, 13, 19, 5, 74], systems are modelled as LTS of the form $\langle \textsc{Prc}, \longrightarrow, \textsc{eAct} \cup \{\tau\}\rangle$. The only prior knowledge that instrumentation has about these systems is that they can either transition with external actions from $\textsc{eAct}$ or silent τ actions; instrumentation forwards the former to the monitor but keeps the latter hidden as they are not relevant for analysis.

By modelling systems as ILTSs (instead of LTSs), instrumentation gains the following two additional pieces of prior knowledge about the class of systems being monitored for:

(i) Systems can internally transition with actions from $\textsc{iAct}$.
(ii) The set of traceable actions $\textsc{tAct} = \textsc{iAct} \cup \textsc{eAct}$ can be partitioned into deterministic and non-deterministic actions using the predicate $\textsc{Det}$.

Instrumentation can thus extract certain internal aspects of system behaviour that were previously not observable and forward them to the monitor for analysis, together with information about whether the observed actions are deterministic or not. As a result, an ILTS provides two (global) views of the SUS: an external one, as viewed by an observer limited to $\textsc{eAct}$, and a lower-level view as seen by an instrumented monitor privy to $\textsc{tAct}$ together with $\textsc{Det}$, respectively depicted in Figures 6.1a and 6.1b.

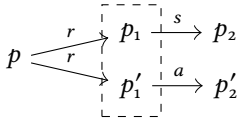
Our treatment of the SUS is still considered black-box for a number of reasons. First, monitoring is conducted w.r.t. a set of *prior* knowledge about the SUS that is *fixed* throughout the entire system computation. This means that the information on traceable actions $\textsc{tAct}$ (needed anyway for instrumentation reasons) together with their deterministic properties $\textsc{Det}$, characterises an *infinite* class of systems, not one specific SUS. For instance, when $\textsc{Det}(r) = \text{false}$, both the systems $r.s.\mathbf{0} + r.a.\mathbf{0}$ and $r.(s.\mathbf{0} + a.\mathbf{0})$ can arise. However, although the monitor is equipped with prior information about the system, it is still insufficient to uniquely identify the concrete system it is actually observing. Second, due to the determinism guarantees, a monitor can *at best* reason about SUS states within the same equivalence class (*i.e.,* $p \equiv q$), but *not* directly on specific states (see Proposition 6.5). For instance, if the monitor

is observes action r produced by $r.(s.\mathbf{0}+a.\mathbf{0})+r.(s.\mathbf{0}+a.\mathbf{0}+a.\mathbf{0})$, the system could have transitioned to either $s.\mathbf{0}+a.\mathbf{0}$ or $s.\mathbf{0}+a.\mathbf{0}+a.\mathbf{0}$, both of which belong to the same equivalence class.

6.3 Revisiting the Specification Logic

Properties are formulated for the external SUS view (*i.e.,* the SUS as viewed by an external observer) in terms of recHML formulae. This logic is defined by the negation free grammar in Figure 2.1, discussed in detail in Chapter 2. Crucially, existential modalities, $\langle\alpha\rangle\varphi$, and universal modalities, $[\alpha]\varphi$, in this logic only operate over *external* actions, *i.e.,* $\alpha \in \text{eAct}$. Put otherwise, recHML formulae do *not* reference internal actions, $\gamma \in \text{iAct}$; these are only used by the verifying monitors during analysis.

Example 6.4. Recall property $\varphi_2 \stackrel{\text{def}}{=} [r]([s]\mathsf{ff} \vee [a]\mathsf{ff})$ from Example 1.3 and suppose a monitor observes the two trace prefixes rs and ra along two different executions. Even though both executions start from the same SUS state, say p, distinct intermediate states could have been reached after exhibiting event r:



When $p_1 \neq p_1'$, the trace prefixes rs and ra only share the initial state p. However, if $\text{Det}(r) = \text{true}$, these two states must be equivalent *i.e.,* $p_1 \equiv p_1'$. Since transitions in an ILTS abstract over equivalent states (formalised in eq. (6.1)), a monitor can infer that the SUS state p_1 can also emit event a, and vice versa for the SUS state p_1' and event s. This means that the monitor can conclude that φ_2 is violated since the SUS state p_1 (resp. p_1') can exhibit both events s and a. ■

We recall that a state p *satisfies* formula φ if $p \in [\![\varphi]\!]$ and *violates* it if $p \notin [\![\varphi]\!]$. Proposition 6.5 shows that equivalent states, $p \equiv q$, satisfy (resp. violate) the same formulae. Crucially, it allows us to reason about system states in the same equivalence class.

Proposition 6.5 (Behavioural Equivalence). For all (closed) formulae $\varphi \in \text{recHML}$ and systems $p,q \in \text{Prc}$,

$$\text{if } p \in [\![\varphi]\!] \text{ and } p \equiv q \text{ then } q \in [\![\varphi]\!]$$

Proof Outline. Suppose $p \in [\![\varphi]\!]$ and $p \equiv q$. From Remark 6.1, p and q are strongly bisimilar. Our result, $q \in [\![\varphi]\!]$, follows by the well-known result that strong bisimulation preserves formula satisfactions. □

6.4 Summary

This chapter introduced the system language that will be used throughout this part of the thesis. Specifically, systems are modelled as ILTSs, where actions are categorised as either external, internal or silent, and traces are redefined as finite sequences of traceable (*i.e.,* internal and external) actions. An ILTS further partitions traceable actions into deterministic and non-deterministic ones via the predicate Det. These features are leveraged in the subsequent chapters to enhance the analytical capabilities of the monitors conducting runtime verification.

7 A Framework for Repeated Monitoring of Non-deterministic Systems

Chapter 3 formalised an online monitoring setup that gathers information over multiple monitored executions of the same *deterministic* SUS. In this chapter, we *extend* this multi-run RV setup to operate over (potentially) non-deterministic systems. Monitoring is still performed in two steps, namely *history aggregation* and *history analysis*. However, the SUS instrumentation we consider sits at a lower-level of abstraction to the external view used by recHML, where monitors are privy to traceable actions (instead of external actions only) and the associated action determinacy delineation defined by Det.

Structure of the chapter. Section 7.1 builds on Section 3.1. It first presents our *extended* monitor language, which remains largely unchanged from Section 3.1.1. This section then *extends* the instrumentation transition relation of Section 3.1.2 to make internal actions, iAct, visible to the instrumented monitor. Section 7.2 builds on Section 3.2, by redefining how a history consisting of traceable actions from tAct is rejected by a monitor privy to Det through a proof system. Based on this, we also redefine what it means for a monitor to reject a system that can produce such a history. Section 7.3 concludes.

7.1 History Aggregation

We extend the monitor language and the instrumentation transition relation of Section 3.1. In sharp contrast to the setup in Section 3.1, where all internal actions were represented by the silent τ action and thus kept hidden from the instrumented monitor, the extended framework presented in this part of the thesis allows monitors to observe (and aggregate) both internal and external actions.

7.1.1 Monitors

In Part I, runtime analysis was conducted by monitors from Mon (Figure 3.1), which record the external actions, $\alpha \in$ eAct, that lead to rejection monitor states. Monitors in this part of the thesis record the *traceable* actions, $\mu, \lambda \in$ tAct, that lead to rejection monitor states. Traceable actions can be sequenced to form (augmented) traces $t, u \in \text{eTrc} = \text{tAct}^*$. An *executing-monitor* state consists of a pair $(t, m) \in \text{eTrc} \times \text{Mon}$, where t is the (augmented) trace that some initial monitor has collected from the beginning of the run, up to the current execution point, and m is the current state of the monitor after analysing it.

The syntax of monitors remain unchanged from Part I. Concretely, in order to streamline monitor synthesis from formulae (which only mention external actions) the monitor syntax does *not* reference internal actions, *e.g.* $\alpha.m$ where $\alpha \in$ eAct in Figure 3.1. Accordingly, monitor semantics determines which *external* actions to record; see rules iMon and mAct in Figure 3.1, discussed in Section 3.1. Internal

$$\textsc{iAsI}\quad \frac{m \neq \mathsf{no} \qquad p \xrightarrow{\gamma} p'}{H \triangleright (t,m) \triangleleft p \xrightarrow{\tau} H \triangleright (t\gamma,m) \triangleleft p'}$$

Figure 7.1. *Instrumentation rule for internal actions*

actions, used to improve the precision of the history analysis, are recorded by the instrumentation semantics following rule iAsI, to be discussed later.

Executing-monitor transitions also carry over from Part I. They are defined w.r.t. a history H that stores the (augmented) trace prefixes accumulated in prior executions and take the form

$$(t,m) \xrightarrow{\eta}_H (t',m')$$

where $\eta \in \textsc{eAct} \cup \{\tau\}$. Such a transition denotes that the executing-monitor (t,m) transitions to (t',m') either by observing an external action *i.e.,* $\eta = \alpha$ where $\alpha \in \textsc{eAct}$, produced by the SUS or by evolving autonomously via the silent action, *i.e.,* $\eta = \tau$. Put otherwise, although trace t in (t,m) may contain internal actions, executing-monitors cannot transition with these actions, *i.e.,* for any $\gamma \in \textsc{iAct}$, we have $(t,m) \xrightarrow{\gamma}\!\!\!\!\!/$. For a detailed discussion of the rules defining the operational semantics of executing-monitors, refer to Section 3.1.1.

7.1.2 Instrumentation

We extend the instrumentation transition relation in Figure 3.1 by rule iAsI in Figure 7.1. This rule allows the SUS to transition with an internal action, *i.e.,* $p \xrightarrow{\gamma} p'$ where $\gamma \in \textsc{iAct}$. Instrumentation records action γ as part of the aggregated trace while concealing it as a τ action from observers, *i.e.,* $H \triangleright (t,m) \triangleleft p \xrightarrow{\tau} H \triangleright (t\gamma,m) \triangleleft p'$. The condition $m \neq \mathsf{no}$ in the premise ensures that the monitored system cannot transition with rule iNo; if the monitor is in a rejection state, the current trace t should be added to the history. The remaining rules remain unchanged (see Section 3.1.2 for details).

Example 7.1. Recall monitor m_1 below from Example 3.3, which reaches a rejection state no after observing actions a or c, following a sequence of serviced requests (*i.e.,* alternating r and s actions).

$$m_1 \stackrel{\text{def}}{=} \mathsf{rec}X.\big(r.s.X \otimes (a.\mathsf{no} \oplus c.\mathsf{no})\big)$$

Consider also system p_2 below, an extension on p_1 from Example 2.3, where the decision on whether to allocate memory or close depends on checking whether there is free memory or not, expressed as the internal actions δ_1 and δ_2 respectively.

$$p_2 \stackrel{\text{def}}{=} \mathsf{rec}X.\big(r.s.X + (\delta_1.a.X + \delta_2.c.\mathbf{0})\big)$$

When system p_2 is instrumented with the executing-monitor (ϵ,m_1) and history $H_0 = \emptyset$, it can reach a

$$\textsc{no}\ \frac{H \neq \emptyset}{\mathbf{rej}_{\textsc{Det}}(H,\mathsf{f},\mathsf{no})} \qquad \textsc{act}\ \frac{H' = \mathit{suffix}(H,\alpha) \quad \mathsf{f}' = \mathsf{f} \wedge \textsc{Det}(\alpha) \quad \mathbf{rej}_{\textsc{Det}}(H',\mathsf{f}',m)}{\mathbf{rej}_{\textsc{Det}}(H,\mathsf{f},\alpha.m)}$$

$$\textsc{actI}\ \frac{H' = \mathit{suffix}(H,\gamma) \quad \mathsf{f}' = \mathsf{f} \wedge \textsc{Det}(\gamma) \quad \mathbf{rej}_{\textsc{Det}}(H',\mathsf{f}',\alpha.m)}{\mathbf{rej}_{\textsc{Det}}(H,\mathsf{f},\alpha.m)} \qquad \textsc{parAL}\ \frac{\mathbf{rej}_{\textsc{Det}}(H,\mathsf{f},m)}{\mathbf{rej}_{\textsc{Det}}(H,\mathsf{f},m \otimes n)} \qquad \textsc{parAR}\ \frac{\mathbf{rej}_{\textsc{Det}}(H,\mathsf{f},n)}{\mathbf{rej}_{\textsc{Det}}(H,\mathsf{f},m \otimes n)}$$

$$\textsc{parO}\ \frac{\mathbf{rej}_{\textsc{Det}}(H,\mathsf{true},m) \quad \mathbf{rej}_{\textsc{Det}}(H,\mathsf{true},n)}{\mathbf{rej}_{\textsc{Det}}(H,\mathsf{true},m \oplus n)} \qquad \textsc{rec}\ \frac{\mathbf{rej}_{\textsc{Det}}(H,\mathsf{f},m[^{\mathsf{rec}X.m}/_X])}{\mathbf{rej}_{\textsc{Det}}(H,\mathsf{f},\mathsf{rec}X.m)}$$

Figure 7.2. *Proof System for non-deterministic SUSs*

rejection state no through the prefix $t_1 = rs\delta_1 a$ as follows:

$$\begin{aligned}
H_o \triangleright (\epsilon, m_1) \triangleleft p_2 &\xrightarrow{\tau} \cdot \xrightarrow{\tau} H_o \triangleright (\epsilon, r.s.m_1 \otimes (a.\mathsf{no} \oplus c.\mathsf{no})) \triangleleft r.s.p_2 + (\delta_1.a.p_2 + \delta_2.c.\mathbf{0}) && (\textsc{iAsP}, \textsc{iAsM})\\
&\xrightarrow{r} \cdot \xrightarrow{s} H_o \triangleright (rs, m_1) \triangleleft p_2 && (\textsc{iMon}, \textsc{iMon})\\
&\xrightarrow{\tau} \cdot \xrightarrow{\tau} H_o \triangleright (rs, r.s.m_1 \otimes (a.\mathsf{no} \oplus c.\mathsf{no})) \triangleleft r.s.p_2 + (\delta_1.a.p_2 + \delta_2.c.\mathbf{0}) && (\textsc{iAsP}, \textsc{iAsM})\\
&\xrightarrow{\tau} H_o \triangleright (rs\delta_1, r.s.m_1 \otimes (a.\mathsf{no} \oplus c.\mathsf{no})) \triangleleft a.p_2 && (\textsc{iAsI})\\
&\xrightarrow{a} H_o \triangleright (rs\delta_1 a, \mathsf{no}) \triangleleft p_2 && (\textsc{iMon})\\
&\xrightarrow{\tau} H_o \cup \{rs\delta_1 a\} \triangleright (rs\delta_1 a, \mathsf{end}) \triangleleft p_2 && (\textsc{iNo})
\end{aligned}$$

In a subsequent run with the augmented history $H_1 = \{t_1\}$, the monitored system $H_1 \triangleright (\epsilon, m_1) \triangleleft p_2$ can then aggregate the trace prefix $t_2 = rs\delta_2 c$ via the execution sequence

$$H_1 \triangleright (\epsilon, m_1) \triangleleft p_2 \overset{t_2}{\Longrightarrow} H_1 \triangleright (t_2, \mathsf{no}) \triangleleft p_2 \xrightarrow{\tau} H_2 \triangleright (t_2, \mathsf{end}) \triangleleft p_2 \overset{t_2}{\Longrightarrow} H_2 \triangleright (t_2, \mathsf{end}) \triangleleft p_2$$

where $H_2 = \{t_1, t_2\}$. ■

7.2 History Analysis

We formalise how a history (consisting of augmented traces) is rejected by a monitor through a proof system. Its main judgement is $\mathbf{rej}_{\textsc{Det}}(H,\mathsf{f},m)$ and denotes that monitor m *rejects* history H using Det with the boolean flag f. It uses internal actions and the predicate Det to calculate whether the traces are produced by the same SUS states (up to ≡), where the flag value true encodes that all the actions analysed up to this point were deterministic actions.

This analysis is the least relation defined by the rules in Figure 7.2, which relies on the helper function

$$\mathit{suffix}(H,\eta) = \{\, t \mid \eta t \in H \,\}$$

from Section 3.2, returning the continuation of all the traces in H prefixed by action η. Although several rules in Figure 7.2 are similar to ones in Figure 3.3, we give a full overview for the sake of completeness.

The axiom NO states that a no monitor rejects all *non-empty* histories with any flag f. This implicitly means that a monitor cannot reject a SUS outright, without any observation. A sequenced monitor $\alpha.m$ can reject a history H with flag f in either of two ways. Specifically, in rule ACT, monitor $\alpha.m$ rejects H with f if the (sub-)monitor m rejects the history returned by $\mathit{suffix}(H,\alpha)$ with updated flag $\mathsf{f}' = \big(\mathsf{f} \wedge \textsc{Det}(\alpha)\big)$. Alternatively, a sequenced monitor $\alpha.m$ can reject H with f following rule ACTI, by considering the suffixes of traces prefixed by an internal action γ, again updating the flag to $\mathsf{f}' = \big(\mathsf{f} \wedge \textsc{Det}(\gamma)\big)$, but leaving the monitor unchanged. Parallel conjunction monitors $m \otimes n$ reject H with f if either *one* of the constituent monitors m and n rejects H with f (rules PARAL and PARAR). Importantly, parallel disjunction monitors $m \oplus n$ reject H with only when the flag is true and *both* monitors reject it (rule PARO), ensuring that the trace prefix analysed consisted of deterministic actions. Finally, rule REC states that a recursive monitor rejects a history with some flag if its unfolding does. As a shorthand, we say that monitor m rejects history H whenever it rejects that history with flag true, formalised in Definition 7.2.

Definition 7.2. A monitor m rejects history H, denoted as $\mathbf{rej}_{\textsc{Det}}(H,m)$, whenever $\mathbf{rej}_{\textsc{Det}}(H,\mathsf{true},m)$. ■

Based on this definition for history rejections, Definition 7.3 below redefines what it means for a monitor m to reject a system p, denoted as $\mathbf{rej}_{\textsc{Det}}(p,m)$. Intuitively, if the system p is capable of producing a history H that the monitor rejects (using the rules in Figure 7.2), then that monitor also rejects p.

Definition 7.3 (System rejections). A monitor m rejects system p, denoted as $\mathbf{rej}_{\textsc{Det}}(p,m)$, whenever $\exists H \subseteq T_p$ such that $\mathbf{rej}_{\textsc{Det}}(H,m)$. ■

In the example below, we show how monitors privy to DET employ the proof system of Figure 7.2 to reject (potentially) non-deterministic systems by analysing the histories that they produce.

Example 7.4. Recall $m_1 \stackrel{\text{def}}{=} \mathrm{rec}X.\big(r.s.X \otimes (a.\mathsf{no} \oplus c.\mathsf{no})\big)$ and $p_2 \stackrel{\text{def}}{=} \mathrm{rec}X.\big(r.s.X + (\delta_1.a.X + \delta_2.c.\mathbf{0})\big)$ from Examples 7.1 and 3.3 respectively, and suppose that $\textsc{Det}(r) = \textsc{Det}(s) = \mathsf{true}$. Instrumentation can record the trace $t_1 = rs\delta_1 a$ during a first monitored execution, but m_1 fails to reject the recorded history, $\neg\mathbf{rej}_{\textsc{Det}}(\{t_1\},m_1)$. This is depicted in the proof derivation below (the flag is omitted from all judgements as it is always true), since no rule from Figure 7.2 can justify the judgement $\mathbf{rej}_{\textsc{Det}}(\emptyset,\mathsf{true},\mathsf{no})$ at $(**)$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\mathbf{rej}_{\textsc{Det}}(\{\epsilon\},\mathsf{no})}\,\textsc{no}}{\mathbf{rej}_{\textsc{Det}}(\{a\},a.\mathsf{no})}\,\textsc{act}}{\mathbf{rej}_{\textsc{Det}}(\{\delta_1 a\},a.\mathsf{no})}\,\textsc{actI} \qquad \dfrac{\mathbf{rej}_{\textsc{Det}}(\emptyset,\mathsf{no}) \quad (**)}{\mathbf{rej}_{\textsc{Det}}(\{\delta_1 a\},c.\mathsf{no})}\,\textsc{act}}{\mathbf{rej}_{\textsc{Det}}(\{\delta_1 a\},a.\mathsf{no}\oplus c.\mathsf{no})}\,\textsc{paro}}{\mathbf{rej}_{\textsc{Det}}(\{\delta_1 a\},r.s.m_1 \otimes (a.\mathsf{no}\oplus c.\mathsf{no}))}\,\textsc{parar}}{\mathbf{rej}_{\textsc{Det}}(\{\delta_1 a\},m_1)}\,\textsc{rec}}{\mathbf{rej}_{\textsc{Det}}(\{s\delta_1 a\},s.m_1)}\,\textsc{act}}{\mathbf{rej}_{\textsc{Det}}(\{rs\delta_1 a\},r.s.m_1)}\,\textsc{act}}{\mathbf{rej}_{\textsc{Det}}(\{rs\delta_1 a\},r.s.m_1 \otimes (a.\mathsf{no}\oplus c.\mathsf{no}))}\,\textsc{paral}}{\mathbf{rej}_{\textsc{Det}}(\{rs\delta_1 a\},m_1)}\,\textsc{rec}$$

When system p_2 is monitored again, the additional trace $t_2 = rs\delta_2 c$ can be aggregated, which monitor m_1

now rejects, $\mathbf{rej}_{\text{DET}}(\{t_1,t_2\},m_1)$. This is depicted by the following proof derivation (flag true omitted).

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\mathbf{rej}_{\text{DET}}(\{\epsilon\},\text{no})}\,\text{NO}}{\mathbf{rej}_{\text{DET}}(\{a\},a.\text{no})}\,\text{ACT}}{\mathbf{rej}_{\text{DET}}(\{\delta_1 a,\delta_2 c\},a.\text{no})}\,\text{ACTI} \quad \dfrac{\dfrac{\dfrac{}{\mathbf{rej}_{\text{DET}}(\{\epsilon\},\text{no})}\,\text{NO}}{\mathbf{rej}_{\text{DET}}(\{c\},c.\text{no})}\,\text{ACT}}{\mathbf{rej}_{\text{DET}}(\{\delta_1 a,\delta_2 c\},c.\text{no})}\,\text{ACTI}}{\mathbf{rej}_{\text{DET}}(\{\delta_1 a,\delta_2 c\},a.\text{no}\oplus c.\text{no})}\,\text{PAROR}}{\mathbf{rej}_{\text{DET}}(\{\delta_1 a,\delta_2 c\},r.s.m_1\otimes(a.\text{no}\oplus c.\text{no}))}\,\text{PARAL}}{\mathbf{rej}_{\text{DET}}(\{\delta_1 a,\delta_2 c\},m_1)}\,\text{REC}}{\mathbf{rej}_{\text{DET}}(\{s\delta_1 a,s\delta_2 c\},s.m_1)}\,\text{ACT}}{\mathbf{rej}_{\text{DET}}(\{rs\delta_1 a,rsc\},r.s.m_1)}\,\text{ACT}}{\mathbf{rej}_{\text{DET}}(\{rs\delta_1 a,rs\delta_2 c\},r.s.m_1\otimes(a.\text{no}\oplus c.\text{no}))}\,\text{PARAL}}{\mathbf{rej}_{\text{DET}}(\{rs\delta_1 a,rs\delta_2 c\},m_1)}\,\text{REC}$$

This means that m_1 can now reject p_2 via the witness history $\{t_1,t_2\}$, denoted as $\mathbf{rej}_{\text{DET}}(p_2,m_1)$. ■

Crucially, Example 7.5 below shows that parallel disjunction monitors $m\oplus n$ *cannot* reject a history if the trace prefix analysed contained non-deterministic actions; see rule PAROR.

Example 7.5. Consider monitor $m_4 \stackrel{\text{def}}{=} r.(s.\text{no}\oplus a.\text{no})$ and history $H=\{rs,rc\}$. Whenever $\text{DET}(r)=\text{false}$, we can show that m_4 cannot reject H, *i.e.,* $\neg\mathbf{rej}_{\text{DET}}(H,m_4)$. In particular, when m_4 analyses H using rule ACT from Figure 7.2, the premise flag can only be $\text{true}\wedge\text{DET}(r)=\text{false}$, which prohibits the analysis from using rule PAROR further up in the proof tree. As a result, there is no rule satisfying the judgement at $(*)$ in the left proof derivation below.

$$\dfrac{\mathbf{rej}_{\text{DET}}(\{s,a\},\text{false},s.\text{no}\oplus a.\text{no}) \quad (*)}{\mathbf{rej}_{\text{DET}}(\{rs,ra\},\text{true},m_4)}\,\text{ACT} \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\mathbf{rej}_{\text{DET}}(\{sa,sc\},\text{false},a.\text{no}\oplus c.\text{no}) \quad (***)}{\mathbf{rej}_{\text{DET}}(\{sa,sc\},\text{false},r.s.m_1\otimes(a.\text{no}\oplus c.\text{no}))}\,\text{PARAR}}{\mathbf{rej}_{\text{DET}}(\{sa,sc\},\text{false},m_1)}\,\text{REC}}{\mathbf{rej}_{\text{DET}}(\{sa,sc\},\text{false},s.m_1)}\,\text{ACT}}{\mathbf{rej}_{\text{DET}}(\{rsa,rsc\},\text{true},r.s.m_1) \quad (**)}\,\text{ACT}}{\mathbf{rej}_{\text{DET}}(\{rsa,rsc\},\text{true},r.s.m_1\otimes(a.\text{no}\oplus c.\text{no}))}\,\text{PARAL}}{\mathbf{rej}_{\text{DET}}(\{rsa,rsc\},\text{true},m_1)}\,\text{REC}$$

Using similar reasoning, one can also show that monitor m_1 fails to reject the history $\{rsa,rsc\}$ whenever $\text{DET}(r)=\text{false}$. Specifically, applying rule ACT at $(**)$ in the right proof derivation above turns the premise flag to $\text{true}\wedge\text{DET}(r)=\text{false}$, which prohibits the analysis from using rule PAROR at $(***)$. ■

7.3 Summary

In this chapter, we extended the multi-run monitoring setup from Part I to operate over (potentially) non-deterministic systems. While the monitor syntax remains unchanged to facilitate synthesis from high-level formulae, the underlying framework is moved to a lower level of abstraction. Specifically, we extended the instrumentation transition relation (via rule IASI) to allow monitors to observe and aggregate traceable actions TACT which encompass sequences of both external and internal actions. This allows internal actions to be recorded as part of the aggregated trace while remaining hidden as silent τ actions from external observers, thereby improving the precision of the history analysis.

The core of this chapter formalises a history analysis proof system based on the judgement $\mathbf{rej}_{\textsc{Det}}$. This enables monitors to use the predicate Det to distinguish between deterministic and non-deterministic actions. A central feature of this analysis is the use of a boolean flag f to track whether the actions analysed up to a certain point are deterministic. This flag is crucial for the parallel disjunction rule parO, which prevents monitors from rejecting a history if the trace prefix contains non-deterministic actions. By formalising system rejection $\mathbf{rej}_{\textsc{Det}}$ as the capacity of a system to produce a history that a monitor rejects, this framework establishes the rigorous foundation necessary to define monitor correctness and examine how multiple runs of a (potentially) non-deterministic SUS affects the monitorability limits identified in prior works [10, 76].

8 Monitor Correctness

Chapter 4 establishes what it means for a monitor m from the monitoring setup of Chapter 3 to *correctly monitor* for a property φ. This chapter revisits this definition in the context of the extended monitoring setup of Chapter 7, where monitors are capable of aggregating and analysing (augmented) traces exhibited by (potentially) non-deterministic systems over multiple executions.

Structure of the chapter. We adopt the same approach as in Chapter 4. More specifically, Section 8.1 builds on Section 4.1 and proves analogous correctness properties but in the context of the history aggregation mechanism of Section 7.1. Similarly, Section 8.2 builds on Section 4.2 and proves corresponding correctness properties w.r.t. the history analysis mechanism of Section 7.2. Section 8.2 then redefines the notions of soundness, completeness and correct monitoring. Section 8.3 concludes. We also note that this chapter restates several definitions and propositions from Chapter 4 to improve readability and make the presentation self-contained.

8.1 Correctness for History Aggregation

The first correctness result we show concerns the history aggregation mechanism of Chapter 7. Proposition 8.1 below corresponds to Proposition 4.1 and states that (augmented) traces collected by the monitor are indeed traces of the SUS with which it is instrumented. Following Proposition 8.1, we assume that all monitors considered in this part of the thesis are *veracious*.

Proposition 8.1 (Veracity). For any history H, monitor m, system p, and sequence of actions $\eta_1,\ldots,\eta_n$,

$$\text{if } \; H \triangleright (\epsilon, m) \triangleleft p \xrightarrow{\eta_1} \cdots \xrightarrow{\eta_n} H' \triangleright (t, m') \triangleleft p' \; \text{ then } \; p \overset{t}{\Longrightarrow}_T p'$$

Proof Outline. The proof is similar to that for Proposition 4.1. The main differences are that

- all weak transitions, $p \overset{\eta}{\Longrightarrow} p'$ for some p, p', in Proposition 4.1 are replaced by weak transitions of the form $p \overset{\eta}{\Longrightarrow}_T p'$.
- when analysing the rules that could have been used to derive $H' \triangleright (t, m') \triangleleft p' \xrightarrow{\eta_{k+1}} H'' \triangleright (t', m'') \triangleleft p''$ in the inductive case, rule iAsP must also be considered.

For the complete proof, refer to Appendix E.1. □

Another important correctness criteria for our multi-run monitoring setup is that executing-monitors *behave deterministically* [72, 11]. In our setting, this translates to Proposition 4.5 from Section 4.1, which

relies on Propositions 4.2 and 4.3. We restate them below for the sake of the reader and outline how the corresponding proofs change from Section 4.1.

Proposition 8.1 (τ-Race Absence). For all $\alpha \in$ EAct, if $(t,m) \xrightarrow{\tau}_H (t,n)$ then $(t,m) \xrightarrow{\alpha}\!\!\!\!\!\!\not\;\;\;_H$.

Proof Outline. Trace t may now contain internal actions, $\gamma \in$ IAct, but this does not affect the proof. □

Proposition 8.1 (τ-confluence). For monitors $m,m',m'' \in$ Mon, trace $t \in$ Trc and history $H \in$ Hst, if $(t,m) \xrightarrow{\tau}_H (t,m')$ and $(t,m) \xrightarrow{\tau}_H (t,m'')$, then there exist weak τ-moves $(t,m')\,(\xrightarrow{\tau}_H)^*\,(t,n)$ and $(t,m'')\,(\xrightarrow{\tau}_H)^*\,(t,n)$ for some monitor $n \in$ Mon.

Proof Outline. Trace t may now contain internal actions, $\gamma \in$ IAct, but this does not affect the proof. □

Proposition 8.1 (Monitor Determinism). For all traces u,t,t_1,t_2, history H, and monitors m,n_1,n_2,

$$\text{if } (t,m) \overset{u}{\Longrightarrow}_H (t_1,n_1) \text{ and } (t,m) \overset{u}{\Longrightarrow}_H (t_2,n_2) \text{ then } t_1 = t_2 \text{ and } (t_1,n_1) \approxeq_H (t_2,n_2)$$

Proof Outline. The proof is analogous to that in Section 4.1. For the *inductive case*, we can show that a third case $\eta = \gamma \in$ IAct never arises. To show this, assume, by way of contradiction, that $\eta = \gamma \in$ IAct, *i.e.,* $(t,m_1) \xrightarrow{\gamma}_H (v_1,n_1')$ and $u{=}\gamma u'$ for some $u' \in$ Trc. But by the rules in Figure 3.1, we know executing-monitors cannot analyse internal actions, *i.e.,* $(t,m_1) \xrightarrow{\gamma}\!\!\!\!\!\!\not\;\;\;_H$, contradicting our initial assumption. □

8.2 Correctness for History Analysis

Section 4.1 shows that the verdicts reached by the history analysis of Section 3.2 are irrevocable. We show that a similar correctness property also holds for the history analysis of Section 7.2. More specifically, Propositions 8.2 and 8.3 correspond to Propositions 4.7 and 4.8 respectively and state that, once the SUS is rejected by a monitor after exhibiting history H (using the history analysis of Section 7.2), further observations (in terms of additional traces, *width*, or longer traces, *length*) do *not* alter this conclusion.

Proposition 8.2 (Width Irrevocability). For any histories H, H' and monitor m,

$$\text{if } \mathbf{rej}_{\text{Det}}(H,m) \text{ then } \mathbf{rej}_{\text{Det}}(H \cup H', m)$$

Proof Outline. Expanding Definition 7.2, we have to show that if $\mathbf{rej}_{\text{Det}}(H,\mathsf{true},m)$ then $\mathbf{rej}(H \cup H',\mathsf{true},m)$. This follows from the statement in eq. (8.1) below, letting $\mathsf{f} = \mathsf{true}$:

$$\text{for any histories } H,H', \text{ flag } \mathsf{f} \text{ and monitor } m, \text{ if } \mathbf{rej}_{\text{Det}}(H,\mathsf{f},m) \text{ then } \mathbf{rej}_{\text{Det}}(H \cup H',\mathsf{f},m) \qquad (8.1)$$

The proof is by induction on the inference of $\mathbf{rej}_{\text{Det}}(H,m)$ and is very similar to that for Proposition 4.7.

For instance, for rule Act, we have that $\mathbf{rej}_{\text{Det}}(H,\mathsf{f},\alpha.m)$ because $\mathbf{rej}_{\text{Det}}(H'',\mathsf{f}',m)$ where $H'' = \mathit{suffix}(H,\alpha)$ and $\mathsf{f}' = \mathsf{f} \wedge \text{Det}(\alpha)$. Let $H''' = \mathit{suffix}(H',\alpha)$. By the IH, we obtain $\mathbf{rej}_{\text{Det}}(H'' \cup H''',\mathsf{f}',m)$. Our result, $\mathbf{rej}_{\text{Det}}(H \cup H',\mathsf{f},\alpha.m)$, then follows by applying rule Act.

We also have an additional case to consider, *i.e.,* rule ActI. Its proof is similar to that for rule Act. □

Proposition 8.3 (Length Irrevocability). For any history H, traces t,u and monitor m,

$$\text{if } \mathbf{rej}_{\text{Det}}((H,t),m) \text{ then } \mathbf{rej}_{\text{Det}}((H,tu),m)$$

Proof Outline. By Definition 7.2, we have to show that if $\mathbf{rej}_{\text{Det}}((H,t),\text{true},m)$ then $\mathbf{rej}((H,tu),\text{true},m)$. This follows from the stronger statement in eq. (8.2) below, letting $\mathsf{f} = \text{true}$.

$$\begin{gathered}\text{for any histories } H,H', \text{ traces } t,u, \text{ flag } \mathsf{f} \text{ and monitor } m,\\ \text{if } \mathbf{rej}_{\text{Det}}((H,t),\mathsf{f},m) \text{ then } \mathbf{rej}_{\text{Det}}((H,tu),\mathsf{f},m)\end{gathered} \tag{8.2}$$

The proof is straightforward by induction on the inference of $\mathbf{rej}_{\text{Det}}((H,t),\mathsf{f},m)$ and is very similar to that for Proposition 4.8. We also have an additional case to consider, *i.e.,* rule ActI. Its proof is similar to that for rule Act. For the full proof of eq. (8.2), see Lemma 1 in Appendix E.2. □

For RV purposes, the least *correctness* requirement expected of our (irrevocable) *history analysis* is that any rejections imply property violations, a requirement termed (rejection) *soundness* in Section 3.2. Definition 4.9 formalised what it means for monitors from Chapter 3 to soundly monitor for a formula (using the history analysis mechanism of Section 3.2). Definition 8.4 below redefines this notion for the extended history analysis of Section 7.2. Concretely, a monitor m (privy to tAct and Det) monitors soundly for the (closed) formula φ only if, for *any* system p, whenever m rejects this system, *i.e.,* $\mathbf{rej}_{\text{Det}}(p,m)$, then it must be the case that p also violates the property, *i.e.,* $p \notin [\![\varphi]\!]$.

Definition 8.4 (Soundness). Monitor m monitors *soundly* for the (closed) formula φ if for all $p \in \text{Prc}$,

$$\mathbf{rej}_{\text{Det}}(p,m) \text{ implies } p \notin [\![\varphi]\!]$$ ■

Example 8.5. It can be shown that monitor $m_1 \stackrel{\text{def}}{=} \text{rec}X.\big(r.s.X \otimes (a.\text{no} \oplus c.\text{no})\big)$ from Example 3.3 monitors soundly for violations of $\varphi_4 \stackrel{\text{def}}{=} \max X.\big([r][s]X \wedge ([a]\text{ff} \vee [c]\text{ff})\big)$ from Example 2.4. For instance, Example 7.1 illustrates how the trace prefixes $rs\delta_1 a$ and $rs\delta_2 c$ of p_2 can be veraciously accumulated as a history once exhibited by p_2, and Example 7.4 shows that such a history is rejected. Accordingly, p_2 violates φ_4.

Similarly, monitor $m_5 \stackrel{\text{def}}{=} s.\text{no} \otimes a.\text{no} \otimes c.\text{no}$ monitors soundly for property $\varphi_0 \stackrel{\text{def}}{=} [s]\text{ff} \wedge [a]\text{ff} \wedge [c]\text{ff}$ from Example 1.1; it rejects systems that are capable of producing any of the trace prefixes ts, ta and tc for some (internal) trace $t \in \text{iAct}^*$, all of which witness the violation of property φ_0.

By comparison, monitor $m_3 \stackrel{\text{def}}{=} r.s.a.\text{no}$ from Example 4.10 is *not* sound for property φ_4; it can collect and reject histories that contain the trace prefix $rs\delta_1 a$, but systems such as $\text{rec}X.r.s.\delta_1.a.X$ and $r.s.\delta_1.a.\mathbf{0}$ (which can exhibit such a trace) do *not* violate φ_4. ■

A complementary requirement to soundness is (rejection) completeness. Definition 4.11 formalised what it means for monitors from Chapter 3 to monitor completely for a formula (using the history analysis mechanism of Section 3.2). We redefine this notion for the history analysis of Section 7.2. Concretely, Definition 8.6 below states that a monitor m *monitors completely* for a property φ if it is capable of rejecting any violating system p based on some history it produces.

Definition 8.6 (Completeness). Monitor m monitors *completely* for (closed) formula φ if for all $p \in \text{Prc}$,

$$p \notin [\![\varphi]\!] \text{ implies } \mathbf{rej}_{\text{Det}}(p,m)$$ ■

Example 8.7. Using Definition 8.6, it can be shown that monitor $m_5 \stackrel{\text{def}}{=} s.\text{no} \otimes a.\text{no} \otimes c.\text{no}$ from Example 8.5 monitors completely for property $\varphi_0 \stackrel{\text{def}}{=} [s]\text{ff} \wedge [a]\text{ff} \wedge [c]\text{ff}$ from Example 1.1. Concretely, any violating system should be able to exhibit at least *one* trace prefix of the form ts, ta or tc for some (internal) trace $t \in \text{IAct}^*$. Once exhibited (and aggregated), one can show that m_5 rejects that history.

We also note that complete monitors for a property following Definition 4.11 do *not* necessarily satisfy the completeness requirement in Definition 8.6. For instance, Example 4.12 shows that m_1 monitors completely for φ_4 according to Definition 4.11, but it is not complete for φ_4 according to Definition 8.6; monitor m_1 *cannot* reject violating systems such as $r.(s.a.\mathbf{0} + s.c.\mathbf{0}) + r.s.\mathbf{0}$ whenever $\text{Det}(r) = \text{false}$, even though they might exhibit the (violating) trace prefixes rsa and rsc (see Example 7.5 for details). ■

For monitors that are veracious and produce irrevocable verdicts—as those defined in Chapter 7—soundness and completeness of Definitions 8.4 and 8.6 constitute the basis for monitor correctness, formalised in Definition 8.8. Although Definition 8.8 is analogous to Definition 4.13, the underlying notions of soundness and completeness are different; we opted to redefine the latter to eliminate any potential confusion.

Definition 8.8 (Correct Monitoring). A monitor m *monitors correctly* for the (closed) formula φ if it can do so *soundly* and *completely*. ■

Example 8.9. Using Examples 8.5 and 8.7, one can show that m_5 monitors correctly for φ_0. The same cannot be said for monitor m_1 and property φ_4; although this monitor is sound for φ_4, Example 8.7 showed it is not complete according to Definition 8.6. ■

8.3 Summary

This chapter formalised what it means for a monitor from Chapter 7 to correctly analyse a property over multiple runs of a (potentially) non-deterministic SUS. Similar to the earlier work in Part I, our notion of monitor correctness takes into consideration both the history aggregation and the history analysis mechanisms of Chapter 7. For the former, we showed that executing-monitors are *veracious*, *i.e.,* traces collected by an executing-monitor are indeed traces of the SUS, and *deterministic*, *i.e.,* they always reach equivalent states when analysing the SUS same traces. For the latter, we showed that rejections flagged by monitors during analysis are *irrevocable*, *i.e.,* once a history is rejected by a monitor, further observations (in terms of additional traces, *width*, or longer traces, *length*) do *not* alter this conclusion.

Since Chapter 7 reformulated what it means for a monitor to reject a system, we also revisited the notions of soundness and completeness. These two requirements constitute the basis for our definition of monitor correctness and directly affect which properties can be monitored for at runtime, as will be demonstrated in the following chapter.

9 Revisiting Monitorability

Monitorability establishes a correspondence between the declarative semantic of Chapter 2 and the operational semantics of the monitors conducting the runtime verification, as formalised in Definition 4.13. Chapter 5 defined monitorability w.r.t. the multi-run monitoring setup of Chapter 3, where monitoring was limited to deterministic systems, and identified the corresponding maximal monitorable subset. In this chapter, we revisit the definition of monitorability in the context of the augmented monitoring setup of Chapter 7, where monitors are capable of runtime verifying (potentially) non-deterministic systems. Accordingly, we investigate the increase in expressive power when moving from single-run monitoring to multi-run monitoring (using the monitor operational model of Chapter 7).

Structure of the chapter. We adopt an approach similar to that of Chapter 5. More specifically, Section 9.1 builds on Section 5.1. It identifies an *extended* logical fragment $\text{sHML}^{\vee}_{\text{DET}}$ that is monitorable in the augmented multi-run setup of Chapter 7; the proofs are given in the dedicated Section 9.2. Section 9.3 builds on Section 5.3. It proves that the monitorable fragment $\text{sHML}^{\vee}_{\text{DET}}$ is *maximally expressive*, *i.e.,* every property that can be monitored correctly using the multi-run setup of Chapter 7 can always be expressed as a formula in $\text{sHML}^{\vee}_{\text{DET}}$. Section 9.4 concludes. We also note that this chapter restates several definitions and propositions from Chapter 5 to improve readability and make the presentation self-contained.

9.1 The Monitorable Logical Fragment

To investigate what properties can be verified at runtime using the multi-run monitors of Chapter 7, we start by formalising what it means for a property to be monitorable. As already discussed in Chapter 5, this definition is *parametric* w.r.t. the meaning of the statement *"m monitors correctly for property φ"*; Chapter 4 specifies monitor correctness in terms of the predicate **rej**, whereas Chapter 8 redefines it in terms of the predicate $\textbf{rej}_{\text{DET}}$ where verifying monitors are privy to both tAct and Det. To avoid confusion and make it explicit that the notion of monitorability that we are considering in this chapter is w.r.t. the monitor correctness of Definition 8.8, we redefine this notion in Definition 9.1.

Definition 9.1 (Monitorability [10, 76]). A formula $\varphi \in \text{recHML}$ is *monitorable* iff there exists a monitor $m \in \text{Mon}$ that *monitors correctly* for it, *i.e.,* soundly and completely. A (sub)logic $\mathcal{L} \subseteq \text{recHML}$ is monitorable iff every formula $\varphi \in \mathcal{L}$ is monitorable. ■

Several formulae are not monitorable (for violations), particularly when they include existential modalities and least fixed points (see Examples 5.15 and 5.16, replacing the predicate **rej** with $\textbf{rej}_{\text{DET}}$; the rest of the argument is analogous). Apart from existential modalities and least fixed points, disjunctions

are the only other recHML logical constructs that are excluded from sHML, as restated in Theorem 2.12. However, formulae containing disjunctions can be monitorable with a few caveats.

Example 9.2. Recall $\varphi_2 \stackrel{\text{def}}{=} [r]([s]\mathsf{ff} \vee [a]\mathsf{ff})$ from Example 1.3. When $\textsc{Det}(r) = \mathsf{false}$, formula φ_2 is *not* monitorable, according to Definition 9.1. Arguing towards a contradiction, assume there exists a monitor m that monitors correctly for it, *i.e.,* soundly (Definition 8.4) and completely (Definition 8.6). Since

$$p_3 \stackrel{\text{def}}{=} r.(s.\mathbf{0} + a.\mathbf{0}) + r.s.\mathbf{0} \notin [\![\varphi_2]\!]$$

then we should have that $\mathbf{rej}_{\textsc{Det}}(p_3, m)$, which implies $\mathbf{rej}_{\textsc{Det}}(H, m)$ for some $H \subseteq T_{p_3}$. But $T_{p_3} = T_{p_4}$ where

$$p_4 \stackrel{\text{def}}{=} r.s.\mathbf{0} + r.a.\mathbf{0} \in [\![\varphi_2]\!]$$

This means that H is also a history of the *satisfying* system p_4, which, by Definition 7.3, implies $\mathbf{rej}_{\textsc{Det}}(p_4, m)$. This would make m unsound, contradicting our initial assumption.

However, when $\textsc{Det}(r) = \mathsf{true}$, property φ_2 is monitorable: an obvious correct monitor for it is $m_4 \stackrel{\text{def}}{=} r.(s.\mathsf{no} \oplus a.\mathsf{no})$ from Example 7.5. Although p_3 and p_4 would be ruled out by requiring $\textsc{Det}(r) = \mathsf{true}$, an ILTS would still allow systems such as

$$p_5 \stackrel{\text{def}}{=} r.(s.\mathbf{0} + a.\mathbf{0}) + r.(s.\mathbf{0} + a.\mathbf{0} + a.\mathbf{0})$$

that reaches the equivalent states $s.\mathbf{0} + a.\mathbf{0}$ and $s.\mathbf{0} + a.\mathbf{0} + a.\mathbf{0}$ after an r-transition. Thus, even if the history $H = \{ra, rs\}$ is aggregated by passing through *different* intermediary states, *i.e.,* $s.\mathbf{0} + a.\mathbf{0}$ and $s.\mathbf{0} + a.\mathbf{0} + a.\mathbf{0}$, the monitor analysis would still be sound in rejecting p_5 via H since the intermediary states are behaviourally equivalent; see Proposition 6.5.

A trickier formula is $\varphi_4 \stackrel{\text{def}}{=} \max X.\big([r][s]X \wedge ([a]\mathsf{ff} \vee [c]\mathsf{ff})\big)$ from Example 2.4. Although the disjunction is syntactically *not* prefixed by any universal modality, it can be reached after a single recursive unfolding, *i.e.,* $\varphi_4 \equiv [r][s]\varphi_4 \wedge ([a]\mathsf{ff} \vee [c]\mathsf{ff})$. Using similar reasoning to the one used for φ_2, formula φ_4 is monitorable whenever $\textsc{Det}(r) = \textsc{Det}(s) = \mathsf{true}$ but unmonitorable otherwise. ■

Definition 9.3 characterises the extended class of recHML monitorable formulae for multi-run monitoring, parametrised by the set of external actions eAct and the associated action determinacy delineation defined by Det. It employs a flag to calculate deterministic prefixes via rule cUm along the lines of Figure 7.2. This flag is then used by rule cOr, which is only defined when the flag is true, ensuring that the disjunction operator is only prefixed by deterministic actions.

Definition 9.3. The judgement $\mathsf{f} \vdash_{\textsc{Det}} \varphi$ for (closed) formulae $\varphi \in \text{recHML}$ and flag $\mathsf{f} \in \textsc{Bool}$ is defined coinductively as the *largest* relation satisfied by the following rules.

$$\textsc{cA}\ \frac{\varphi \in \{\mathsf{ff}, \mathsf{tt}, X\}}{\mathsf{f} \vdash_{\textsc{Det}} \varphi} \qquad \textsc{cUm}\ \frac{\mathsf{f} \wedge \textsc{Det}(\alpha) \vdash_{\textsc{Det}} \varphi}{\mathsf{f} \vdash_{\textsc{Det}} [\alpha]\varphi} \qquad \textsc{cAnd}\ \frac{\mathsf{f} \vdash_{\textsc{Det}} \varphi \quad \mathsf{f} \vdash_{\textsc{Det}} \psi}{\mathsf{f} \vdash_{\textsc{Det}} \varphi \wedge \psi}$$

$$\textsc{cOr}\ \frac{\mathsf{true} \vdash_{\textsc{Det}} \varphi \quad \mathsf{true} \vdash_{\textsc{Det}} \psi}{\mathsf{true} \vdash_{\textsc{Det}} \varphi \vee \psi} \qquad \textsc{cMax}\ \frac{\mathsf{f} \vdash_{\textsc{Det}} \varphi[^{\max X.\varphi}/_{X}]}{\mathsf{f} \vdash_{\textsc{Det}} \max X.\varphi}$$

The set $\text{sHML}^{\vee}_{\textsc{Det}} \stackrel{\text{def}}{=} \{\varphi \mid \mathsf{true} \vdash_{\textsc{Det}} \varphi\}$ defines the set of *extended* monitorable formulae. Specifically, it extends sHML (Theorem 2.12) with disjunctions as long as these are prefixed by universal modalities of deterministic external actions (up to largest fixed point unfolding). ■

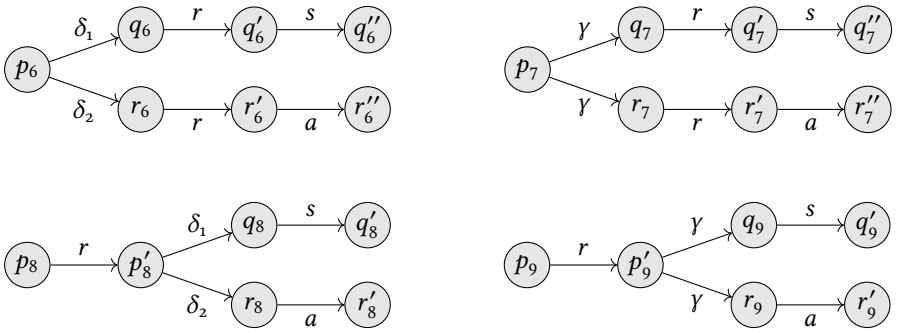


Figure 9.1. *A depiction of the systems in Example 9.6*

Remark 9.4. Following Definition 9.3, the greatest fixed point formula $\max X.[a]X$ belongs to the set of extended monitorable formulae, *i.e.,* $\max X.[a]X \in \text{sHML}^{\vee}_{\text{DET}}$, regardless of the value of $\text{DET}(a)$. However, although the formula $\max X.(X \vee X)$ is, like $\max X.[a]X$, a tautology and therefore equivalent to $\text{tt} \in \text{sHML}^{\vee}_{\text{DET}}$, it does not belong to the set of extended monitorable formulae when $\text{DET}(a) = \text{false}$. ■

Example 9.5. Assuming that $\text{DET}(r) = \text{DET}(s) = \text{true}$ and $\text{DET}(a) = \text{false}$, we can symbolically show that both formulae φ_2 and φ_4 are in $\text{sHML}^{\vee}_{\text{DET}}$. For instance, according to Definition 9.3, to justify the inclusion $\varphi_2 \in \text{sHML}^{\vee}_{\text{DET}}$ it suffices to prove the judgement $\text{true} \vdash_{\text{DET}} \varphi_2$. Now, the relation R below satisfies the coinductive rules of Definition 9.3 and includes the pair (true, φ_2), thereby proving that $\text{true} \vdash_{\text{DET}} \varphi_2$.

$$R = \left\{ \begin{array}{l} (\text{true},\ [r]([s]\text{ff} \vee [a]\text{ff})),\quad (\text{true},\ [s]\text{ff} \vee [a]\text{ff}), \\ (\text{true},\ [s]\text{ff}),\quad (\text{true},\ [a]\text{ff}), \\ (\text{true},\ \text{ff}),\quad (\text{false},\ \text{ff}) \end{array} \right\}$$

In contrast, when $\text{DET}(r) = \text{false}$, we can show that the formula φ_2 is not in $\text{sHML}^{\vee}_{\text{DET}}$. Assume, by way of contradiction, that $\varphi_2 \in \text{sHML}^{\vee}_{\text{DET}}$. According to Definition 9.3, there must exist a proof derivation justifying $\text{true} \vdash_{\text{DET}} \varphi_2$. However, there is no rule from Definition 9.3 that can satisfy the judgement $\text{false} \vdash_{\text{DET}} [s]\text{ff} \vee [a]\text{ff}$ at $(*)$ in the proof derivation below; the application of rule cUM forces the premise flag to be $\text{true} \wedge \text{DET}(r) = \text{false}$, which prohibits the analysis from using rule cOR.

$$\frac{\text{false} \vdash_{\text{DET}} [s]\text{ff} \vee [a]\text{ff} \qquad (*)}{\text{true} \vdash_{\text{DET}} [r]([s]\text{ff} \vee [a]\text{ff})}\ \text{cUM}$$

This means that $\neg(\text{true} \vdash_{\text{DET}} \varphi_2)$, which gives us a contradiction. Using a similar argument, one can also show that $\varphi_4 \notin \text{sHML}^{\vee}_{\text{DET}}$ whenever either $\text{DET}(r) = \text{false}$ or $\text{DET}(s) = \text{false}$. ■

Although the tracing of internal actions $\gamma \in \text{IACT}$ as part of the history helps with correct monitoring of (potentially) non-deterministic SUSs, multi-run RV requires us to limit systems to deterministic internal actions in order to attain violation completeness (Definition 8.6) for monitors MON of Chapter 7. This limitation is better illustrated by $p_{10} \notin [\![\varphi_2]\!]$ in Example 9.6 below.

Example 9.6. Consider again $\varphi_2 \stackrel{\text{def}}{=} [r]([s]\text{ff} \vee [a]\text{ff})$ from Example 1.3 with $\text{DET}(r) = \text{true}$. The two systems p_6 and p_7 below, depicted in Figure 9.1 where the arcs correspond to weak system transitions

$\overset{\mu}{\Longrightarrow}_T$, both *satisfy* property φ_2 when $\textsc{Det}(r) = \text{true}$.

$$p_6 \overset{\text{def}}{=} \delta_1.r.s.\mathbf{0} + \delta_2.r.a.\mathbf{0}$$
$$p_7 \overset{\text{def}}{=} \gamma.r.s.\mathbf{0} + \gamma.r.a.\mathbf{0}$$

In the case of p_6, the correct monitor $m_4 \overset{\text{def}}{=} r.(s.\text{no} \oplus a.\text{no})$ from Example 7.5 does *not* reject the history $\{\delta_1 rs, \delta_2 ra\}$ because the application of rule ActI of Figure 7.2 (for either δ_1 or δ_2) necessarily reduces the history size of the premise to *one* trace. In the case of system p_7, we must also have $\textsc{Det}(\gamma) = \text{false}$. As a result, monitor m_4 does *not* reject p_7 either; when it analyses the history $\{\gamma rs, \gamma ra\}$ using rule ActI, the premise flag can only be false, which prohibits the analysis from using parO.

In comparison, systems p_8 and p_9 below, again depicted in Figure 9.1, both violate φ_2. Accordingly, they are rejected by the correct monitor m_4 via the respective histories $\{r\delta_1 s, r\delta_2 a\}$ and $\{r\gamma s, r\gamma a\}$.

$$p_8 \overset{\text{def}}{=} r.(\delta_1.s.\mathbf{0} + \delta_2.a.\mathbf{0})$$
$$p_9 \overset{\text{def}}{=} r.(\gamma.s.\mathbf{0} + \gamma.a.\mathbf{0})$$

Nevertheless, non-deterministic internal actions pose a threat to (rejection) completeness. For instance, system $p_{10} \overset{\text{def}}{=} \gamma.p_8 + \gamma.\mathbf{0}$ violates φ_2 but m_4 cannot reject the history $\{\gamma r\delta_1 s, \gamma r\delta_2 a\}$: since $\textsc{Det}(\gamma) = \text{false}$, the initial application of ActI turns the flag to false, prohibiting subsequent uses of rule parO. ■

As already argued in Section 5.1, showing that a logical fragment, $\mathcal{L} \subseteq$ recHML, is monitorable according to Definition 9.1 can be onerous due to the various universal quantifications that need to be considered, *e.g.* all $\varphi \in \mathcal{L}$ and all $p \in \textsc{Prc}$ from Definitions 8.4 and 8.6. We prove the monitorability of $\text{sHML}^{\vee}_{\textsc{Det}}$ from Definition 9.3 systematically, using an approach similar to that used for Theorem 5.5. Concretely, we concretise the existential quantification of a correct monitor for every formula $\varphi \in \text{sHML}^{\vee}_{\textsc{Det}}$ via the monitor synthesis function $(\!|\varphi|\!)$ in Definition 5.3, restated below. We then prove that for any formula $\varphi \in \text{sHML}^{\vee}_{\textsc{Det}}$, the synthesised monitor $(\!|\varphi|\!)$ does monitor correctly for it according to Definition 9.1.

If we limit ILTSs to deterministic internal actions, *i.e.,* $\textsc{Det}(\gamma) = \text{true}$ for all $\gamma \in \textsc{iAct}$, we can show monitorability for arbitrary ILTSs and the fragment $\text{sHML}^{\vee}_{\textsc{Det}}$, formalised in Theorem 9.7.

Theorem 9.7 (Monitorability). When $\textsc{Det}(\gamma) = \text{true}$ for all internal actions $\gamma \in \textsc{iAct}$, all (closed) formulae $\varphi \in \text{sHML}^{\vee}_{\textsc{Det}}$ are monitorable.

Proof Outline. The proof for Theorem 9.7 is given in the dedicated Section 9.2 which may be skipped upon first reading. □

This result equips us with a syntactic check for determining whether a formula is monitorable or not w.r.t. the multi-run RV setup of Chapter 7.

9.2 Proving Monitorability

This section is dedicated to the proof of Theorem 9.7. The proof consists of two steps and follows an approach similar to that for Theorem 5.5. Concretely, we first show that the monitors generated via the synthesis function $(\!|-|\!)$ are sound (Definition 8.4), and then we show they are complete (Definition 8.6).

To facilitate the establishments of these results, we define an *explicit* witness-based violation relation $H \models_{\text{DET}} \varphi$ that characterises the specific histories along which arbitrary ILTSs (that include the function DET) violate (closed) formulas to avoid the existential quantifications over SUS histories of Definitions 8.4 and 8.6. The new judgement $H \models_{\text{DET}} \varphi$ corresponds to $p \notin [\![\varphi]\!]$ whenever $H \subseteq T_p$.

Definition 9.8 (Violation Relation). Given a predicate on TACT denoted as DET, the *violation relation*, denoted as $\models_{\text{DET}}$, is the least relation included in $(\text{HST} \times \text{BOOL} \times \text{sHML}^{\vee}_{\text{DET}})$ that satisfies the following rules:

$$\textsc{vF}\ \frac{H \neq \emptyset}{(H,\mathsf{f}) \models_{\text{DET}} \mathsf{ff}} \qquad \textsc{vMax}\ \frac{(H,\mathsf{f}) \models_{\text{DET}} \varphi[\max X.\varphi/X]}{(H,\mathsf{f}) \models_{\text{DET}} \max X.\varphi} \qquad \textsc{vAndL}\ \frac{(H,\mathsf{f}) \models_{\text{DET}} \varphi}{(H,\mathsf{f}) \models_{\text{DET}} \varphi \wedge \psi} \qquad \textsc{vAndR}\ \frac{(H,\mathsf{f}) \models_{\text{DET}} \psi}{(H,\mathsf{f}) \models_{\text{DET}} \varphi \wedge \psi}$$

$$\textsc{vOr}\ \frac{(H,\mathsf{true}) \models_{\text{DET}} \varphi \quad (H,\mathsf{true}) \models_{\text{DET}} \psi}{(H,\mathsf{true}) \models_{\text{DET}} \varphi \vee \psi} \qquad \textsc{vUmPre}\ \frac{H' = \mathit{suffix}(H,\gamma) \quad \mathsf{f}' = \mathsf{f} \wedge \text{DET}(\gamma) \quad (H',\mathsf{f}') \models_{\text{DET}} [\alpha]\varphi}{(H,\mathsf{f}) \models_{\text{DET}} [\alpha]\varphi}$$

$$\textsc{vUm}\ \frac{H' = \mathit{suffix}(H,\alpha) \quad \mathsf{f}' = \mathsf{f} \wedge \text{DET}(\alpha) \quad (H',\mathsf{f}') \models_{\text{DET}} \varphi}{(H,\mathsf{f}) \models_{\text{DET}} [\alpha]\varphi}$$

As a shorthand, we write $H \models_{\text{DET}} \varphi$ in lieu of $(H,\mathsf{true}) \models_{\text{DET}} \varphi$ and read it as "H violates φ". ■

Theorem 9.9 below shows that whenever a system p produces a history H that violates a formula φ, *i.e.,* $H \models_{\text{DET}} \varphi$, then it must be the case that p also violates it, *i.e.,* $p \notin [\![\varphi]\!]$ (for *arbitrary* ILTSs). To show correspondence in the other direction, Theorem 9.10, we need to limit ILTSs to deterministic internal actions. The reason for this is, once again, the set of systems such as p_{10} from Example 9.6 for which there is *no* history $H \subseteq T_{p_{10}}$ such that $H \models_{\text{DET}} \varphi_2$, even though $p_{10} \notin [\![\varphi_2]\!]$.

Theorem 9.9. For all (closed) formulae $\varphi \in \text{sHML}^{\vee}_{\text{DET}}$, if $\big(\exists H \subseteq T_p$ such that $H \models_{\text{DET}} \varphi\big)$ then $p \notin [\![\varphi]\!]$.

Proof Outline. Follows from the stronger statement below, by letting $\mathsf{f} = \mathsf{true}$.

For any formula $\varphi \in \text{sHML}^{\vee}_{\text{DET}}$, history $H \subseteq T_p$ and flag $\mathsf{f} \in \text{BOOL}$, if $(H,\mathsf{f}) \models_{\text{DET}} \varphi$ then $p \notin [\![\varphi]\!]$.

The complete proof is given in Appendix D.2. □

Theorem 9.10. Suppose $\text{DET}(\gamma) = \mathsf{true}$ for all internal actions $\gamma \in \text{IACT}$. For all (closed) formulae $\varphi \in \text{sHML}^{\vee}_{\text{DET}}$, if $p \notin [\![\varphi]\!]$ then $\big(\exists H \subseteq T_p$ such that $H \models_{\text{DET}} \varphi\big)$.

Proof Outline. Using Definition 9.3, our result follows from the statement below, by letting $\mathsf{f} = \mathsf{true}$.

Suppose $\text{DET}(\gamma) = \mathsf{true}$ for all $\gamma \in \text{IACT}$. For any formula $\varphi \in \text{RECHML}$ and flag $\mathsf{f} \in \text{BOOL}$,
if $\mathsf{f} \vdash_{\text{DET}} \varphi$ and $p \notin [\![\varphi]\!]$ then $\big(\exists H \subseteq T_p$ such that $(H,\mathsf{f}) \models_{\text{DET}} \varphi\big)$.

The complete proof is given in Appendix D.2. □

Soundness (Definition 8.4) and completeness (Definition 8.6) of the monitors generated by $(\!|-|\!)$, respectively formalised in Propositions 9.14 and 9.15 below, rely on a number of additional results. In particular,

Lemmas 12 and 13 show there is a tight correspondence between the rejected histories, $\mathbf{rej}_{\text{DET}}(H,\mathsf{f},m)$, and the histories violating formulae, $(H,\mathsf{f}) \models_{\text{DET}} \varphi$. Before giving the proofs for these two lemmata, we give an additional technical result, namely Lemma 11 below.

Lemma 11. For all (closed) formulae $\varphi,\psi \in \text{sHML}^{\vee}_{\text{DET}}$, $(\!|\varphi[\psi/X]|\!) = (\!|\varphi|\!)[(\!|\psi|\!)/X]$

Proof. By induction on the structure of φ, we can show that for all formulae $\varphi,\psi \in \text{sHML}^{\vee}$, we have $(\!|\varphi[\psi/X]|\!) = (\!|\varphi|\!)[(\!|\psi|\!)/X]$. Our result then follows using the fact that $\text{sHML}^{\vee}_{\text{DET}} \subseteq \text{sHML}^{\vee}$. □

Lemma 12. For all formulae $\varphi \in \text{sHML}^{\vee}_{\text{DET}}$ and histories H, if $\mathbf{rej}_{\text{DET}}(H,\mathsf{f},(\!|\varphi|\!))$ then $(H,\mathsf{f}) \models_{\text{DET}} \varphi$.

Proof. The proof proceeds by induction on $\mathbf{rej}_{\text{DET}}(H,\mathsf{f},(\!|\varphi|\!))$. We outline the main cases:

- Case ACT, *i.e.,* $\mathbf{rej}_{\text{DET}}(H,\mathsf{f},\alpha.m)$ because $\mathbf{rej}_{\text{DET}}(H',\mathsf{f}',m)$ where $H' = \mathit{suffix}(H,\alpha)$ and $\mathsf{f}' = \mathsf{f} \wedge \text{DET}(\alpha)$ and $\varphi = [\alpha]\psi$ and $m = (\!|\psi|\!)$. By the IH, we get $(H',\mathsf{f}') \models_{\text{DET}} \psi$. Our result, $(H,\mathsf{f}) \models_{\text{DET}} [\alpha]\psi$, follows by vUM.
- Case REC, *i.e.,* $\mathbf{rej}_{\text{DET}}(H,\mathsf{f},\text{rec}X.m)$ because $\mathbf{rej}_{\text{DET}}(H,\mathsf{f},m[^{\text{rec}X.m}/_X])$ where $\varphi = \max X.\psi$ and $m = (\!|\psi|\!)$. By Lemma 11, we also know that $m[^{\text{rec}X.m}/_X] = (\!|\psi|\!)[(\!|\max X.\psi|\!)/X] = (\!|\psi[^{\max X.\psi}/_X]|\!)$. Using the IH, we then obtain $(H,\mathsf{f}) \models_{\text{DET}} \psi[^{\max X.\psi}/_X]$. Our result, $(H,\mathsf{f}) \models_{\text{DET}} \max X.\psi$, follows by rule vMAX. □

Lemma 13. For all formulae $\varphi \in \text{sHML}^{\vee}_{\text{DET}}$, histories H and flag f, if $(H,\mathsf{f}) \models_{\text{DET}} \varphi$ then $\mathbf{rej}_{\text{DET}}(H,\mathsf{f},(\!|\varphi|\!))$.

Proof. Follows with a proof similar to that for Lemma 12. □

Based on these results, we are now in a position to prove the soundness and completeness of the monitors synthesised using the synthesis function $(\!|-|\!)$, according to Definitions 8.4 and 8.6.

Proposition 9.14 (Soundness). For any formula $\varphi \in \text{sHML}^{\vee}_{\text{DET}}$, monitor $(\!|\varphi|\!)$ monitors soundly for φ.

Proof. Expanding Definitions 7.3 and 8.4, we need to show that for all $\varphi \in \text{sHML}^{\vee}_{\text{DET}}$ and $p \in \text{PRC}$,

$$\text{if } \big(\exists H \subseteq T_p \text{ such that } \mathbf{rej}_{\text{DET}}(H,(\!|\varphi|\!))\big) \text{ then } p \notin [\![\varphi]\!].$$

Suppose $\exists H \subseteq T_p$ such that $\mathbf{rej}_{\text{DET}}(H,(\!|\varphi|\!))$. By Lemma 12, letting $\mathsf{f} = \text{true}$, we obtain $H \models_{\text{DET}} \varphi$. Our result, $p \notin [\![\varphi]\!]$, follows by Theorem 9.9. □

Proposition 9.15 (Completeness). Suppose $\text{DET}(\gamma) = \text{true}$ for all $\gamma \in \text{IACT}$. For any formula $\varphi \in \text{sHML}^{\vee}_{\text{DET}}$, monitor $(\!|\varphi|\!)$ monitors completely for φ.

Proof. Suppose that $\text{DET}(\gamma) = \text{true}$ for all internal actions $\gamma \in \text{IACT}$. Expanding Definitions 7.3 and 8.6, we need to show that for any formula $\varphi \in \text{sHML}^{\vee}_{\text{DET}}$ and system $p \in \text{PRC}$, we have that

$$\text{if } p \notin [\![\varphi]\!] \text{ then } \big(\exists H \subseteq T_p \text{ such that } \mathbf{rej}_{\text{DET}}(H,(\!|\varphi|\!))\big)$$

Suppose $p \notin [\![\varphi]\!]$. By the assumption that $\text{DET}(\gamma) = \text{true}$ for all $\gamma \in \text{IACT}$ and Theorem 9.9, we know $\exists H \subseteq T_p$ such that $H \models_{\text{DET}} \varphi$. Our result, $\mathbf{rej}_{\text{DET}}(H,(\!|\varphi|\!))$, follow by Lemma 13, letting $\mathsf{f} = \text{true}$. □

We can now prove the main result of this section, namely that the logical fragment $\text{sHML}^{\vee}_{\text{DET}}$ is monitorable, restated in Theorem 9.7 below.

Theorem 9.16 (Monitorability). When $\text{DET}(\gamma) = \text{true}$ for all internal actions $\gamma \in \text{IACT}$, all (closed) formulae $\varphi \in \text{sHML}^{\vee}_{\text{DET}}$ are monitorable.

Proof. Follows from Propositions 9.14 and 9.15, with $(\!|\varphi|\!)$ as the witness correct monitor. □

9.3 Expressiveness of the Monitorable Logical Fragment

Section 9.1 showed that the logical fragment $\text{sHML}^{\vee}_{\text{DET}}$ of Definition 9.3 is monitorable w.r.t. the multi-run RV setup of Chapter 7. In this section, we show an even stronger monitorability result called *maximality*, which ensures that restricting specifications to $\text{sHML}^{\vee}_{\text{DET}}$ does not impinge on the set of properties that can be monitored for at runtime, formalised in Theorem 9.29 below. Stated otherwise, all monitorable recHML formulae can be reformulated into a (semantically) equivalent formula from $\text{sHML}^{\vee}_{\text{DET}}$.

One way to prove the maximality of $\text{sHML}^{\vee}_{\text{DET}}$ is to show that every monitor in Mon corresponds to a formula in $\text{sHML}^{\vee}_{\text{DET}}$, according to Definition 8.8. Section 5.4 formalised this as expressive-completeness, which we restate below for the sake of the reader; we remind the readers that we use the notion of monitor correctness in Definition 8.8.

Definition 5.22 (Expressive-complete). A language $\mathcal{L} \subseteq \text{recHML}$ is *expressive-complete* if for all monitors $m \in \text{Mon}$, there exists some (closed) formula $\varphi \in \mathcal{L}$ such that m monitors correctly for it. ■

To show that the syntactic fragment $\text{sHML}^{\vee}_{\text{DET}}$ is expressive-complete, we typically rely on a reverse synthesis function $\langle\langle - \rangle\rangle$ that maps *any* monitor $m \in \text{Mon}$ to a characteristic formula $\langle\langle m \rangle\rangle \in \text{sHML}^{\vee}_{\text{DET}}$ that it monitors correctly for (as was done in Section 5.4). However, such a method is complicated by the occurrence of non-deterministic actions in the monitor description.

Example 9.17. Consider again formula $\varphi_2 \stackrel{\text{def}}{=} [r]([s]\text{ff} \vee [a]\text{ff})$ and monitor $m_4 \stackrel{\text{def}}{=} r.(s.\text{no} \oplus a.\text{no})$. Example 9.2 showed that for $\text{DET}(r) = \text{true}$, m_4 is correct for φ_2. However, when $\text{DET}(r) = \text{false}$, it does *not* correctly monitor for φ_2 as it *never* rejects any violating systems (see Example 7.5 for details). In order to overcome these anomalies and obtain our results, we first normalise the aforementioned monitor to $r.\text{end}$. ■

Definition 9.18 characterises the class of *normalised* monitors from Mon, parametrised by eAct and the associated action determinacy delineation defined by Det. Similar to Definition 9.3, it employs a flag to calculate deterministic prefixes via rule cAct along the lines of Figure 7.2. This is then used by rule cParO, which is only defined when the flag is true, ensuring that the parallel disjunction operator $\oplus$ is only prefixed by deterministic actions.

Definition 9.18. The judgement $\mathsf{f} \vdash_{\text{DET}} m$ for monitors $m \in \text{Mon}$ and flag $\mathsf{f} \in \text{Bool}$ is defined coinductively as the *largest* relation satisfying the following rules.

$$\text{cM}\ \frac{m \in \{\text{end}, \text{no}, X\}}{\mathsf{f} \vdash_{\text{DET}} m} \qquad \text{cAct}\ \frac{\mathsf{f} \wedge \text{DET}(\alpha) \vdash_{\text{DET}} m}{\mathsf{f} \vdash_{\text{DET}} \alpha.m} \qquad \text{cParA}\ \frac{\mathsf{f} \vdash_{\text{DET}} m \quad \mathsf{f} \vdash_{\text{DET}} n}{\mathsf{f} \vdash_{\text{DET}} m \otimes n} \qquad \text{cParO}\ \frac{\text{true} \vdash_{\text{DET}} m \quad \text{true} \vdash_{\text{DET}} n}{\text{true} \vdash_{\text{DET}} m \oplus n} \qquad \text{cRec}\ \frac{\mathsf{f} \vdash_{\text{DET}} m[\text{rec}X.m/X]}{\mathsf{f} \vdash_{\text{DET}} \text{rec}X.m}$$

The set $\text{Mon}_{\text{DET}} \stackrel{\text{def}}{=} \{m \mid \text{true} \vdash_{\text{DET}} m\}$ defines the set of *normalised* monitors. ■

To obtain an $\text{sHML}^{\vee}_{\text{DET}}$ formula for *any* monitor from Mon, we adopt the two-step approach illustrated in Figure 9.2. Specifically, we first use a normalisation function $N_{\text{DET}}(-)$ to map *arbitrary* monitors from Mon to monitors in Mon_{DET}, and then we use a reverse synthesis function $\langle\langle - \rangle\rangle$ to map Mon_{DET} monitors to $\text{sHML}^{\vee}_{\text{DET}}$ formulae. The normalisation function $N_{\text{DET}}(-)$ is formalised in Definition 9.19

Figure 9.2. *Monitors to Formulae*

below. It employs a flag to compute deterministic prefixes via rule nAct, which is then used by rule nParF to normalise parallel disjunction monitors to the inconclusive monitor end whenever the flag is false. This function also relies on a map $\sigma \in \text{Sub} : \text{TVars} \to \text{Mon} \times \text{Bool}$ that keeps track of which monitors were analysed with which flags. When $N_{\text{Det}}(-)$ analyses a recursion monitor $\text{rec}X.m$ with flag f following rule nRec, the entry $X \to \langle \text{rec}X.m, \mathsf{f} \rangle$ is added to σ; this ensures that the recursion variable X is unfolded only if the corresponding monitor $\text{rec}X.m$ was not already analysed with the current flag (rule nTVar3).

Definition 9.19 (Monitor Normalisation). Given a predicate on tAct denoted as Det, the normalisation function $N_{\text{Det}} : \text{Mon} \times \text{Bool} \times \text{Sub} \to \text{Mon}_{\text{Det}}$ is the smallest relation satisfying the following rules:

$$\text{nNo}\ \frac{}{N_{\text{Det}}(\mathsf{no},\mathsf{f},\sigma) = \mathsf{no}} \qquad \text{nEnd}\ \frac{}{N_{\text{Det}}(\mathsf{end},\mathsf{f},\sigma) = \mathsf{end}} \qquad \text{nTVar1}\ \frac{X \notin \mathbf{dom}(\sigma)}{N_{\text{Det}}(X,\mathsf{f},\sigma) = X} \qquad \text{nTVar2}\ \frac{\sigma(X) = \langle m,\mathsf{f} \rangle}{N_{\text{Det}}(X,\mathsf{f},\sigma) = X}$$

$$\text{nRec}\ \frac{N_{\text{Det}}(m,\mathsf{f},\sigma[X \mapsto \langle \text{rec}X.m,\mathsf{f} \rangle]) = n}{N_{\text{Det}}(\text{rec}X.m,\mathsf{f},\sigma) = \text{rec}X.n} \qquad \text{nTVar3}\ \frac{\sigma(X) = \langle m,\mathsf{f}' \rangle \quad \mathsf{f}' \neq \mathsf{f} \quad N_{\text{Det}}(m,\mathsf{f},\sigma) = n}{N_{\text{Det}}(X,\mathsf{f},\sigma) = n}$$

$$\text{nAct}\ \frac{N_{\text{Det}}(m,\mathsf{f} \wedge \text{Det}(\alpha),\sigma) = n}{N_{\text{Det}}(\alpha.m,\mathsf{f},\sigma) = \alpha.n} \qquad \text{nParA}\ \frac{N_{\text{Det}}(m,\mathsf{f},\sigma) = m' \quad N_{\text{Det}}(n,\mathsf{f},\sigma) = n'}{N_{\text{Det}}(m \otimes n,\mathsf{f},\sigma) = m' \otimes n'}$$

$$\text{nParO1}\ \frac{N_{\text{Det}}(m,\mathsf{true},\sigma) = m' \quad N_{\text{Det}}(n,\mathsf{true},\sigma) = n'}{N_{\text{Det}}(m \oplus n,\mathsf{true},\sigma) = m' \oplus n'} \qquad \text{nParO2}\ \frac{}{N_{\text{Det}}(m \oplus n,\mathsf{false},\sigma) = \mathsf{end}}$$

As a shorthand, we write $N_{\text{Det}}(m)$ to mean $N_{\text{Det}}(m,\mathsf{true},\emptyset)$. ■

Example 9.20. Using the rules in Definition 9.19, monitor $m_4 \stackrel{\text{def}}{=} r.(s.\mathsf{no} \oplus a.\mathsf{no})$ from Example 7.5 with $\text{Det}(r) = \mathsf{false}$ can be normalised to the monitor $r.\mathsf{end}$, as shown by the proof derivation below.

$$\dfrac{\dfrac{}{N_{\text{Det}}(s.\mathsf{no} \oplus a.\mathsf{no},\mathsf{false},\emptyset) = r.\mathsf{end}}\ \text{nParO2}}{N_{\text{Det}}(r.(s.\mathsf{no} \oplus a.\mathsf{no}),\mathsf{true},\emptyset) = r.\mathsf{end}}\ \text{nAct}$$

In contrast, when $\text{Det}(a) = \mathsf{true}$, the monitor remains unchanged, *i.e.,* $N_{\text{Det}}(m_4) = m_4$. ■

Although monitors m and $N_{\text{Det}}(m)$ might be syntactically different, Proposition 9.21 shows they correspond, in that they reject the same histories; see Example 9.23 below. Consequently, normalised monitors reject the same systems as their non-normalised counterparts, Corollary 9.22.

Proposition 9.21. For all $m \in \text{Mon}$ and $H \in \text{Hst}$, $\mathbf{rej}_{\text{Det}}(H,m)$ iff $\mathbf{rej}_{\text{Det}}(H,N_{\text{Det}}(m))$.

Proof Outline. The proof is quite involved and relies on several auxiliary results. See Appendix F. □

Corollary 9.22. For all monitors $m \in \text{Mon}$, $\mathbf{rej}_{\text{Det}}(p, m)$ iff $\mathbf{rej}_{\text{Det}}(p, N_{\text{Det}}(m))$. ■

Example 9.23. When $\text{Det}(r) = \text{false}$, Example 9.20 showed that $N_{\text{Det}}(m_4) = r.\text{end}$, and both m_4 and $r.\text{end}$ cannot reject (or aggregate) any histories. A trickier monitor is $m_2 \stackrel{\text{def}}{=} \text{rec}X.r.s.X \otimes a.X \otimes (a.\text{no} \oplus c.\text{no})$ from Example 3.4. When either $\text{Det}(r) = \text{false}$ or $\text{Det}(s) = \text{false}$, monitor m_2 can reject histories containing at least two trace prefixes of the form $a^n a$ and $a^n c$ for some $n \geq 0$, where the (sub-)trace a^n may be interleaved with finite sequences of deterministic internal actions. It cannot, however, reject histories such as $\{rsa^n a, rsa^n c\}$ where traces $a^n a$ and $a^n c$ are prefixed with actions rs. Accordingly, monitor m_2 is normalised to

$$m_6 \stackrel{\text{def}}{=} \text{rec}X.\big(r.s.(\text{rec}X.r.s.X \otimes a.X \otimes \text{end})\big) \otimes a.X \otimes (a.\text{no} \oplus c.\text{no})$$

where the sub-monitor $r.s.\big(\text{rec}X.r.s.X \otimes a.X \otimes \text{end}\big)$ can never reject (or aggregate) any history. This means that m_6 can only reject histories containing traces $a^n a$ and $a^n c$ for some $n \geq 0$ (modulo internal actions), which correspond to those rejected by m_2. ■

Remark 9.24. We note that $N_{\text{Det}}(-)$ could be optimised to eliminate redundant sub-monitors. For instance, m_2 would be normalised to $\text{rec}X.a.X \otimes (a.\text{no} \oplus c.\text{no})$ instead of m_6. However, for the proof of Theorem 9.29, the current, simpler version of the normalisation function in Definition 9.19 suffices. ■

For the second part of Figure 9.2, we recall the reverse synthesis function $\langle\langle - \rangle\rangle$ of Definition 5.23, mapping monitors from Mon to recHML formulae.

Definition 5.23 (Reverse Monitor Synthesis). The function $\langle\langle - \rangle\rangle : \text{Mon} \to \text{recHML}$ is defined inductively on the structure of m as follows:

$$\langle\langle \text{no} \rangle\rangle \stackrel{\text{def}}{=} \text{ff} \qquad \langle\langle m \otimes n \rangle\rangle \stackrel{\text{def}}{=} \langle\langle m \rangle\rangle \wedge \langle\langle n \rangle\rangle \qquad \langle\langle \alpha.m \rangle\rangle \stackrel{\text{def}}{=} [\alpha]\langle\langle m \rangle\rangle \qquad \langle\langle X \rangle\rangle \stackrel{\text{def}}{=} X$$
$$\langle\langle \text{end} \rangle\rangle \stackrel{\text{def}}{=} \text{tt} \qquad \langle\langle m \oplus n \rangle\rangle \stackrel{\text{def}}{=} \langle\langle m \rangle\rangle \vee \langle\langle n \rangle\rangle \qquad \langle\langle \text{rec}X.m \rangle\rangle \stackrel{\text{def}}{=} \max X.\langle\langle m \rangle\rangle$$ ■

Using the normalisation function $N_{\text{Det}}(-)$ in Definition 9.19 and the reverse monitor synthesis function $\langle\langle - \rangle\rangle$ in Definition 5.23, we can show that whenever we limit systems to deterministic internal actions (see Example 9.6 for details), the syntactic fragment $\text{sHML}^{\vee}_{\text{Det}}$ is expressive-complete, as formalised in Theorem 9.27. In particular, this result follows from Propositions 9.25 and 9.26 below, the proof of which may be skipped upon first reading.

Proposition 9.25. If $m \in \text{Mon}_{\text{Det}}$ then $\langle\langle m \rangle\rangle \in \text{sHML}^{\vee}_{\text{Det}}$. ■

Proof. Suppose $m \in \text{Mon}_{\text{Det}}$, *i.e.*, $\text{true} \vdash_{\text{Det}} m$ (Definition 9.18). We have to show $\langle\langle m \rangle\rangle \in \text{sHML}^{\vee}_{\text{Det}}$, *i.e.*, $\text{true} \vdash_{\text{Det}} \langle\langle m \rangle\rangle$ (Definition 9.3). This follows from the stronger statement below, letting $\mathsf{f} = \text{true}$.

$$\text{if } \mathsf{f} \vdash_{\text{Det}} m \text{ then } \mathsf{f} \vdash_{\text{Det}} \langle\langle m \rangle\rangle \tag{9.1}$$

To show eq. (9.1), we define the relation $\mathcal{R}$ below and show it is *closed* under the rules defining $\mathsf{f} \vdash_{\text{DET}} \varphi$ in Definition 9.3.

$$\mathcal{R} = \{\ (\mathsf{f}, \langle\langle m \rangle\rangle)\ \mid \mathsf{f} \vdash_{\text{DET}} m\}$$

Intuitively, we must show that for every pair $(\mathsf{f}, \varphi) \in \mathcal{R}$, there exists a rule in Definition 9.3 whose premise is satisfied by elements also in $\mathcal{R}$. We proceed by case analysis on the structure of the monitor m, outlining only one case as the others follow with similar reasoning:

- When $m = \alpha.n$, by rule cAct from Definition 9.18, we know that $(\mathsf{f} \wedge \text{DET}(\alpha)) \vdash_{\text{DET}} n$. By Definition 5.23, we also know that the synthesized formula is $\langle\langle m \rangle\rangle = \langle\langle \alpha.n \rangle\rangle = [\alpha]\langle\langle n \rangle\rangle$. But since $(\mathsf{f} \wedge \text{DET}(\alpha)) \vdash_{\text{DET}} n$, then the pair $((\mathsf{f} \wedge \text{DET}(\alpha)), \langle\langle n \rangle\rangle)$ is also in $\mathcal{R}$ by definition. Thus, the premise of rule cUm from Definition 9.3 is satisfied by an element that is already in $\mathcal{R}$.

Since we have shown that $\mathcal{R}$ is *closed* under the rules in Definition 9.3, then for any m such that $\mathsf{f} \vdash_{\text{DET}} m$, we have $\mathsf{f} \vdash_{\text{DET}} \langle\langle m \rangle\rangle$ for the corresponding synthesised formula $\langle\langle m \rangle\rangle$. □

Proposition 9.26. Suppose $\text{DET}(\gamma) = \text{true}$ for all $\gamma \in \text{iAct}$. For any $m \in \text{Mon}$, monitor m monitors correctly for formula $\langle\langle N_{\text{DET}}(m) \rangle\rangle$.

Proof. Assume $\text{DET}(\gamma) = \text{true}$ for all $\gamma \in \text{iAct}$. Pick $p \in \text{Prc}$. Let $n = N_{\text{DET}}(m)$ and $\varphi = \langle\langle n \rangle\rangle$. Since the codomain of $N_{\text{DET}}(-)$ is Mon_{DET}, we know $n \in \text{Mon}_{\text{DET}}$, and by Proposition 9.25, we get $\varphi \in \text{sHML}^{\vee}_{\text{DET}}$.

To show soundness, assume $\mathbf{rej}_{\text{DET}}(p,m)$. By Corollary 9.22, we know $\mathbf{rej}_{\text{DET}}(p,n)$ as well. Since $\varphi \in \text{sHML}^{\vee}_{\text{DET}}$, we can use Proposition 9.14, letting $\varphi = \langle\langle n \rangle\rangle$, to obtain $p \notin [\![\varphi]\!]$. We have thus shown that $\mathbf{rej}_{\text{DET}}(p,m)$ implies $p \notin [\![\varphi]\!]$, *i.e., m* monitors soundly for φ (Definition 8.4).

To show completeness, assume $p \notin [\![\varphi]\!]$. Since $\varphi \in \text{sHML}^{\vee}_{\text{DET}}$ and $\text{DET}(\gamma) = \text{true}$ for all $\gamma \in \text{iAct}$, we can use Proposition 9.15, letting $\varphi = \langle\langle n \rangle\rangle$, to obtain $\mathbf{rej}_{\text{DET}}(p,n)$. By Corollary 9.22, we conclude $\mathbf{rej}_{\text{DET}}(p,n)$. We have thus shown that $p \notin [\![\varphi]\!]$ implies $\mathbf{rej}_{\text{DET}}(p,m)$, *i.e., m* monitors completely for φ (Definition 8.6).

The required result, m monitors correctly for $\varphi = \langle\langle N_{\text{DET}}(m) \rangle\rangle$, follows by Definition 8.8. □

Equipped with these results, we can show that if we limit ILTSs to deterministic internal actions, *i.e.,* $\text{DET}(\gamma) = \text{true}$ for all $\gamma \in \text{iAct}$, any monitor in Mon can be mapped to a corresponding $\text{sHML}^{\vee}_{\text{DET}}$ formula.

Theorem 9.27. If $\text{DET}(\gamma) = \text{true}$ for all $\gamma \in \text{iAct}$, $\text{sHML}^{\vee}_{\text{DET}}$ is expressive-complete w.r.t. Mon.

Proof. Pick $m \in \text{Mon}$. By Proposition 9.26, we know monitor m monitors correctly for the formula $\langle\langle N_{\text{DET}}(m) \rangle\rangle$. Also, since the codomain of $N_{\text{DET}}(-)$ is Mon_{DET}, we know $N_{\text{DET}}(m) \in \text{Mon}_{\text{DET}}$, and by Proposition 9.25, we get $\langle\langle N_{\text{DET}}(m) \rangle\rangle \in \text{sHML}^{\vee}_{\text{DET}}$, as required. □

The expressive-completeness of the syntactic fragment $\text{sHML}^{\vee}_{\text{DET}}$ is better illustrated in Example 9.28, which shows how arbitrary monitors from Mon are mapped to corresponding $\text{sHML}^{\vee}_{\text{DET}}$ formulae following the two-step approach outlined in Figure 9.2.

Example 9.28. Consider again monitor $m_4 \stackrel{\text{def}}{=} r.(s.\text{no} \oplus a.\text{no})$ with $\text{DET}(r) = \text{false}$. Although $m_4 \notin \text{Mon}_{\text{DET}}$, Example 9.23 shows that it can be normalised to $N_{\text{DET}}(m_4) = r.\text{end}$ where $r.\text{end} \in \text{Mon}_{\text{DET}}$. A corresponding

formula can then be obtained by applying the reverse synthesis function on the normalised monitor, yielding $\langle\langle r.\mathsf{end}\rangle\rangle = [r]\mathsf{tt}$ where $[r]\mathsf{tt} \in \mathrm{sHML}^{\vee}_{\mathrm{DET}}$. Since formula $[r]\mathsf{tt} \equiv \mathsf{tt}$ cannot be violated and monitor $r.\mathsf{end}$ never rejects, it is not hard to see that $r.\mathsf{end}$ monitors correctly for $[r]\mathsf{tt}$.

Similarly, one can show that when $\textsc{Det}(r) = \mathsf{false}$, monitor m_2 monitors correctly for formula $\langle\langle N_{\mathrm{DET}}(m_2)\rangle\rangle$. Concretely, m_2 is normalised to $N_{\mathrm{DET}}(m_2) = m_6$ from Example 9.23, which is mapped to the $\mathrm{sHML}^{\vee}_{\mathrm{DET}}$ formula $\langle\langle m_6\rangle\rangle = \varphi_5$ where φ_5 is defined as follows:

$$\varphi_5 \stackrel{\text{def}}{=} \max X.\big([r][s](\max X.[r][s]X \wedge [a]X \wedge \mathsf{tt})\big) \wedge [a]X \wedge ([a]\mathsf{ff} \vee [c]\mathsf{ff})$$

Since $[r][s](\max X.[r][s]X \wedge [a]X \wedge \mathsf{tt}) \equiv \mathsf{tt}$, then $\varphi_5 \equiv \max X.[a]X \wedge ([a]\mathsf{ff} \vee [c]\mathsf{ff})$. This means that φ_5 is violated by systems that can produce at least two trace prefixes of the form $a^n a$ and $a^n c$ for some $a \geq \mathrm{o}$, where the (sub-)trace a^n may be interleaved with finite sequences of deterministic internal actions. Accordingly, monitor m_2 rejects any system capable of producing these two trace prefixes. ■

We are now in a position to prove the main result of this section, namely Theorem 9.29 below, stating that if we limit ILTSs to deterministic internal actions, *i.e.,* $\textsc{Det}(\gamma) = \mathsf{true}$ for all $\gamma \in \textsc{iAct}$, the syntactic fragment $\mathrm{sHML}^{\vee}_{\mathrm{DET}}$ is the largest monitorable subset of recHML up to logical equivalence. However, we first recall what we understand by language inclusion up to semantic equivalence from Section 5.4.

Definition 5.17 (Language Inclusion). For all $\mathcal{L}_1, \mathcal{L}_2 \subseteq \textsc{recHML}$,

$$\mathcal{L}_1 \sqsubseteq \mathcal{L}_2 \stackrel{\text{def}}{=} \text{For all } \varphi_1 \in \mathcal{L}_1, \text{ there exists } \varphi_2 \in \mathcal{L}_2 \text{ such that } [\![\varphi_1]\!] = [\![\varphi_2]\!]$$ ■

Theorem 9.29 (Maximality). If $\textsc{Det}(\gamma) = \mathsf{true}$ for all $\gamma \in \textsc{iAct}$ and language $\mathcal{L} \subseteq \textsc{recHML}$ is monitorable w.r.t. Mon, then $\mathcal{L}$ cannot (semantically) express more properties than $\mathrm{sHML}^{\vee}_{\mathrm{DET}}$, *i.e.,* $\mathcal{L} \sqsubseteq \mathrm{sHML}^{\vee}_{\mathrm{DET}}$.

Proof. Assume $\textsc{Det}(\gamma) = \mathsf{true}$ for all $\gamma \in \textsc{iAct}$. Assume also that $\mathcal{L} \subseteq \textsc{recHML}$ is monitorable w.r.t. Mon. By Definition 5.1, this means that for all $\varphi \in \mathcal{L}$,

$$\exists m \in \textsc{Mon} \text{ such that } m \text{ monitors correctly for } \varphi$$

Pick a formula $\varphi \in \mathcal{L}$ and assume $\exists m \in \textsc{Mon}$ such that m monitors correctly for it. Using Definition 8.8, followed by Definitions 8.4 and 8.6, this means that for any system $p \in \textsc{Prc}$, we have

$$p \notin [\![\varphi]\!] \text{ iff } \mathbf{rej}_{\mathrm{DET}}(p, m) \tag{9.2}$$

According to Definition 5.17, we need to show that $\exists \psi \in \mathrm{sHML}^{\vee}_{\mathrm{DET}}$ such that $[\![\varphi]\!] = [\![\psi]\!]$.

By the assumption that $\textsc{Det}(\gamma) = \mathsf{true}$ for all $\gamma \in \textsc{iAct}$ and Theorem 9.27, for the monitor m used in eq. (9.2), we also know $\exists \psi \in \mathrm{sHML}^{\vee}_{\mathrm{DET}}$ where $\psi = \langle\langle N_{\mathrm{DET}}(m)\rangle\rangle$ and m monitors correctly for ψ. Expanding Definition 8.8, followed by Definitions 8.4 and 8.6, this means that for all systems $p \in \textsc{Prc}$, we have

$$p \notin [\![\psi]\!] \text{ iff } \mathbf{rej}_{\mathrm{DET}}(p, m) \tag{9.3}$$

We prove $[\![\varphi]\!] = [\![\psi]\!]$ in two steps; first, we show $[\![\psi]\!] \subseteq [\![\varphi]\!]$ and then we show that $[\![\varphi]\!] \subseteq [\![\psi]\!]$.

For the former, assume an arbitrary system $p \notin [\![\varphi]\!]$. By eq. (9.2), we know $\mathbf{rej}_{\mathrm{DET}}(p, m)$, which by eq. (9.3) implies that $p \notin [\![\psi]\!]$. We thus have that

$$p \notin [\![\varphi]\!] \text{ implies } p \notin [\![\psi]\!] \tag{9.4}$$

By the contrapositive of eq. (9.4), we deduce that $p \in [\![\psi]\!]$ implies $p \in [\![\varphi]\!]$, *i.e.*, $[\![\psi]\!] \subseteq [\![\varphi]\!]$. Dually, we can show $[\![\varphi]\!] \subseteq [\![\psi]\!]$. Our result, $[\![\varphi]\!] = [\![\psi]\!]$, follows. □

Theorem 9.29 has several practical implications that are analogous to those for Theorem 9.29 (see Section 5.3 for details). For instance, maximality permits a verification framework to determine if a property is monitorable w.r.t. the setup of Chapter 7 via a simple syntactic check, thus deciding whether to runtime monitor for it or else employ alternative verification techniques, such as model checking.

9.4 Summary

This chapter showed that monitors can extract sufficient information over multiple runs of (potentially) non-deterministic systems to correctly detect the violation of a class of branching-time properties that may contain disjunctions, as long as these are *not* prefixed by non-deterministic actions (Theorem 9.7). We also proved that the identified monitorable fragment $\mathrm{sHML}^{\vee}_{\mathrm{DET}}$ from Definition 9.3 is maximally expressive (Theorem 9.29). In particular, every property that can be monitored correctly using the multi-run setup of Chapter 7 can always be expressed as a formula in $\mathrm{sHML}^{\vee}_{\mathrm{DET}}$. Such a syntactic characterisation of monitorable properties is useful for tool construction, as will be shown in Part III.

Part III

Practical Aspects

10 Implementability Concerns

Part II showed that previously established limits for branching-time properties can be extended by monitoring systems over multiple executions. Despite the merits of these theoretical results, they also raise several practical questions. The first, and arguably the most fundamental one, concerns the implementability of the verification technique presented in Chapter 7. Specifically, we would like to examine whether this technique lends itself well to the implementation of a tool that runtime verifies systems over multiple executions, possibly by extending existing RV tools. Second, we would also like to understand better the correlation between the structure of the specification formula expressing the property (*e.g.* the number of disjunctions used) and the number of SUS executions required to monitor adequately for that property. Such a measure is crucial for an *efficient* implementation of the operational semantics of the monitor (Figures 3.1, 7.1 and 7.2) where history analysis is not invoked unnecessarily whenever a new trace is aggregated to the history.

We note that although this part of the thesis focuses on the monitoring setup of Part II, the same arguments also apply to that of Part I.

Structure of the chapter. Section 10.1 shows that the verification technique in Chapter 7 lends itself well to the implementation of a tool that runtime verifies systems over multiple executions. Section 10.2 then presents a method for systematically determining lower bounds for the number of SUS executions required to conduct RV from the syntactic structure of the formula being verified. Section 10.3 concludes.

10.1 Towards Implementing the Multi-Run Monitoring Setup

In this section, we outline the steps towards a full automation of the verification technique in Chapter 7. Subsequently, we evaluate this technique by means of a complexity analysis.

The Algorithm The first step is to automatically generate executable monitors from properties expressed as $\text{sHML}^{\vee}_{\text{DET}}$ formulae, following the synthesis algorithm of Definition 5.3. These monitors must then be instrumented to execute alongside the SUS w.r.t. the history of traces observed thus far (initialised to the empty set). There are various instrumentation techniques that can be employed, but the one that suits our needs best are outlined [36]; as discussed in Chapter 3, this allows us to treat systems as black-boxes. Instrumentation then forwards the events generated by the SUS to the monitor, which aggregates traces according to the mechanism in Figures 3.1 and 7.1.

Prior work [15, 32] has shown that the synthesis and implementation of similar operational models is almost one-to-one. Aceto *et al.* [14] rigorously demonstrate their efficiency, which results in a stable tool for runtime verifying asynchronous component systems called `detectEr` [32, 15].

Whenever instrumentation aggregates a new trace to the history, the monitor is terminated (by transitioning to the inconclusive monitor end) and the history analysis in Section 7.2 is invoked; this can be automated following an approach similar to that in [19]. History aggregation and history analysis are then repeated until a permanent verdict is reached (Proposition 8.2).

Complexity Bounds The performance of the algorithm outlined above depends on three factors:

(i) The history aggregating mechanism of Section 7.1. Monitors analyse system events sequentially and transition accordingly, each monitor component incurring a linear complexity w.r.t. the length of the processed trace. The required number of monitor components and the cost of simulating these with a single monitor component has been studied extensively for similar monitoring systems in [8]. There, the authors prove that monitors without parallel components may require up to a doubly-exponential number of states w.r.t. the size of the formula that they monitor for. This means that it may be necessary to maintain an exponentially long description of the monitor configurations along a run. Under the assumption that formulae are generally significantly smaller than execution traces, or that monitors run asynchronously w.r.t. the SUS, the resulting overhead is acceptable.

(ii) The history analysis mechanism of Section 7.2. The complexity of derivations for $\mathbf{rej}_{\text{DET}}(H,m)$ is polynomial w.r.t. the size of m and the longest trace in H. Effectively, this amounts to μ-calculus model-checking on trees, *i.e.,* modal logic model-checking on acyclic graphs, which requires a bilinear time w.r.t. the size of the tree and the formula [82]. With the exception of rules parAL and parAR from Figure 7.2, derivations are mostly syntax-directed and monitors are guarded, *i.e.,* rule rec can only be applied a finite number of times before either rule act or rule actI is used. For a similar (but more complex) tableau format, [19, Section 5] showed that, in practice, the doubly-exponential worst-case complexity upper bound identified in [13] does not represent the average-case complexity.

(iii) The number of trace prefixes required by the monitor conducting the verification to reject the aggregated history. Theorem 10.10 (to be discussed in Section 10.2) provides partial insight into this quantity by relating rejections to structural properties of the monitored formula. However, obtaining a general upper bound is difficult, since for certain formulae, no finite upper bound exists. We revisit this aspect in Example 10.3 of Section 10.2.

10.2 Establishing Bounds

Despite the guarantees provided by Definition 8.8, the correct monitors underpinning monitorable formulae (Theorems 9.7 and 9.29) do not provide any indication of the *number of monitored runs needed* to reject a violating system. This measure is crucial for an *efficient* implementation of the operational semantics of the monitor (Figures 3.1, 7.1 and 7.2) where the history analysis of Figure 7.2 is not invoked unnecessarily whenever a new trace is aggregated to the history.

In view of this, we investigate whether there is a correlation between the syntactic structure of properties expressed in terms of $\text{sHML}^{\vee}_{\text{DET}}$ formulae and the number of partial traces required to conduct the verification. In particular, we study how this measure can be obtained through a *syntactic analysis*

of the *disjunction operators* used in the formula. Since we can only monitor for $\text{sHML}^{\vee}_{\text{DET}}$ formulae in a correct manner when the relevant internal actions are deterministic (see Example 9.6), internal actions are elided in the subsequent discussion as they only make examples more cumbersome.

Example 10.1. Assuming $\text{DET}(r) = \text{true}$, recall formula $\varphi_2 \stackrel{\text{def}}{=} [r]\big([s]\text{ff} \vee [a]\text{ff}\big)$ from Example 1.3 and its representative correct monitor $m_4 \stackrel{\text{def}}{=} r.\big(s.\text{no} \oplus a.\text{no}\big) = (\!|\varphi_2|\!)$ from Example 7.5. Violating systems can produce the history $H = \{rs, ra\}$ (modulo internal actions), which is enough for m_4 to reject. At the same time, no violating system for φ_2 can be rejected with fewer traces. Similarly, all violating systems for the formula $\varphi_6 \stackrel{\text{def}}{=} [r]\big([s]\text{ff} \vee [a]\text{ff}\big) \vee [a]\text{ff}$ can be rejected via the 3-size history $\{rs, ra, a\}$. ■

Although the evidence in Example 10.1 suggests that monitoring for a formula with n disjunctions requires $n+1$ executions to detect violations, this measure could be imprecise in general, for a number of reasons. First, as a consequence of monitor passivity, there is no guarantee that the SUS will only produce the trace prefixes required to reject as it might also exhibit other behaviour (recall that a monitor is passive and cannot steer the behaviour of the SUS). We thus reason about history bounds for the *best case scenario* where *every* monitored run produces a *relevant* trace prefix. Second, not all SUS violations are justified by the same number of (relevant) trace prefixes. For instance, formulae such as $\varphi_1 \wedge \varphi_2$ are violated by systems that either violate φ_1 or φ_2 (but not necessarily both), and thus the number of relevant trace prefixes required to violate each subformula φ_i for $i \in 1..2$ might differ. This means that lower and upper bounds may not necessarily coincide, as better illustrated in Example 10.2 below.

Example 10.2. Consider the formula $\varphi_7 \stackrel{\text{def}}{=} [r]\big([s]\text{ff} \vee [a]\text{ff}\big) \wedge [s]\text{ff}$, a slight modification on formula φ_2. A representative monitor for φ_7 can reject violating systems that exhibit both trace prefixes ra and rs, but it can also reject others exhibiting the single prefix s via the subformula $[s]\text{ff}$. This is problematic since our violating trace estimation needs to universally quantify over *all* systems, in order to adhere to a black-box treatment. ■

Recursive formulae pose further complications when attempting to calculate the executions required from the number of disjunctions present in a formula.

Example 10.3. Consider formula φ_8 below, a variation on formula φ_4 from Example 2.4, stating that *"if the system can allocate memory, then (i) it cannot also perform a close action and (ii) this property is invariant for all the states reached after servicing received queries."*

$$\varphi_8 \stackrel{\text{def}}{=} \max X.\big(\langle a\rangle\text{tt} \Longrightarrow ([c]\text{ff} \wedge [r][s]X)\big) \;\equiv\; \max X.\big([a]\text{ff} \vee ([c]\text{ff} \wedge [r][s]X)\big)$$

Formula φ_8 contains one disjunction, and its representative monitor

$$m_7 \stackrel{\text{def}}{=} \text{rec}X.\big(a.\text{no} \oplus (r.s.X \otimes c.\text{no})\big) = (\!|\varphi_8|\!)$$

can correctly monitor for it with no fewer than two trace prefixes. For instance, the system

$$p_1 \stackrel{\text{def}}{=} \text{rec}X.\big(r.s.X + (a.X + c.\mathbf{0})\big)$$

from Example 2.3 violates φ_8, and m_7 can detect this through the size-2 history $\{a, c\} \subseteq T_{p_1}$. But the same cannot be said for the violating system

$$p_{11} \stackrel{\text{def}}{=} a.\mathbf{0} + r.s.(a.\mathbf{0} + c.\mathbf{0})$$

Since $p_{11} \overset{c}{\not\Rightarrow}$, monitor m_7 cannot use the previous size-2 history $\{a,c\}$ and instead requires the size-3 history $\{a,rsa,rsc\} \subseteq T_{p_{11}}$. A similar argument can be made for the violating system

$$p_{12} \overset{\text{def}}{=} a.\mathbf{0} + r.s.(a.\mathbf{0} + r.s.(a.\mathbf{0} + c.\mathbf{0}))$$

which can only be detected via the size-4 history $\{a,rsa,rsrsa,rsrsc\} \subseteq T_{p_{12}}$. ■

Example 10.3 demonstrates how, when universally quantifying over all systems, execution upper bounds cannot be easily determined for *any* violating system from the structure of a formula. However, we show that the calculation of execution *lower* bounds from the formula structure is attainable. For instance, the lower bound for a conjunction $\varphi_1 \wedge \varphi_2$ would be the least value between the lower bounds of φ_1 and φ_2 respectively. Crucially, history lower bounds are invariant w.r.t. recursive formula unfolding of greatest fixed points.

Example 10.4. Recall formula φ_8 from Example 10.3, where it was (informally) established that the history lower bound required for a monitor to effect a rejection is that of with a history lower bound of size 2, which is equal to the number of disjunctions in φ_8 plus 1 (as argued in Example 10.1). By the semantics in Figure 2.1, the same violating systems should also violate the unfolding of φ_8, namely

$$\varphi_8' \overset{\text{def}}{=} [a]\mathsf{ff} \vee \big([c]\mathsf{ff} \wedge [r][s](\max X.([a]\mathsf{ff} \vee ([c]\mathsf{ff} \wedge [r][s]X)))\big) \;=\; [a]\mathsf{ff} \vee ([c]\mathsf{ff} \wedge [r][s]\varphi_8)$$

since the two formulae are *semantically* equivalent, *i.e.,* $\varphi_8 \equiv \varphi_8'$. A naive analysis would conclude that φ_8' contains 2 disjunctions, implying that violation detections would require histories of at least size $2+1=3$. However, a compositional approach for such a calculation, based on the reasoning followed in Example 10.2 would increase precision and allow us to conclude that history lower bounds of size 2 suffice. Concretely, to reject a violating SUS for φ_8', trace evidence is needed to determine violations for *both* sub-formulae $[a]\mathsf{ff}$ and $[c]\mathsf{ff} \wedge [r][s]\varphi_8$. Whereas one trace suffices to reject $[a]\mathsf{ff}$, determining the lower bounds for rejecting $[c]\mathsf{ff} \wedge [r][s]\varphi_8$ amounts to calculating the *least* lower bound required to reject either $[c]\mathsf{ff}$ or $[r][s]\varphi_8$. Since rejecting $[c]\mathsf{ff}$ requires only 1 trace, the total *lower* bound is that of $1+1=2$ traces, which is equal to that of φ_8. ■

Although recursive variables $X \in \text{TVAR}$ do not syntactically contain disjunctions, recursive unfoldings may introduce an unbounded number of them. Example 10.5 shows that our history lower bound calculation takes this into account.

Example 10.5. Recall $\varphi_4 \overset{\text{def}}{=} \max X.\big([r][s]X \wedge ([c]\mathsf{ff} \vee [a]\mathsf{ff})\big)$ from Example 2.4. The proposed syntactic analysis of this formula would determine that the history lower bound for φ_4 is 2. Concretely, the conjunction sub-formula $[r][s]X$ can potentially contain an unbounded number of disjunctions due to recursion, whereas the right sub-formula contains 1 disjunction, meaning that 2 traces are required; the lower bound across the conjunction is thus 2. The unfolding of φ_4 is:

$$\varphi_4' \overset{\text{def}}{=} \Big([r][s]\big(\max X.\big([r][s]X \wedge ([c]\mathsf{ff} \vee [a]\mathsf{ff})\big)\big)\Big) \wedge ([c]\mathsf{ff} \vee [a]\mathsf{ff})$$

where the history lower bound calculation is invariant at 2. ■

The function $lb(-)$ in Definition 10.6 below formalises the calculation of history lower bounds based on the (compositional) syntactical analysis of formulae.

Definition 10.6 (History Lower Bounds). The function $lb(-) : \text{sHML}^{\vee} \to \mathbb{N} \cup \{\infty\}$ is defined as follows:

$$lb(\mathsf{ff}) \stackrel{\text{def}}{=} 0 \qquad lb(\varphi \wedge \psi) \stackrel{\text{def}}{=} min\{lb(\varphi), lb(\psi)\} \qquad lb([\alpha]\varphi) \stackrel{\text{def}}{=} lb(\varphi)$$
$$lb(\mathsf{tt}) \stackrel{\text{def}}{=} \infty \qquad lb(\varphi \vee \psi) \stackrel{\text{def}}{=} lb(\varphi) + lb(\psi) + 1$$
$$lb(X) \stackrel{\text{def}}{=} \infty \qquad lb(\max X.\varphi) \stackrel{\text{def}}{=} lb(\varphi)$$

■

There is one further complication to consider when calculating the relationship between the syntactic structure of formulae and the number of trace prefixes required for rejections. Our implicit assumption has been that, for disjunctions $\varphi_1 \vee \varphi_2$, the incorrect system behaviour described by φ_1 and φ_2 is distinct. Whenever this is not the case, formulae do not observe the lower bound proposed above since φ_1 and φ_2 might be violated by common trace prefixes.

Example 10.7. Consider formula $\varphi_9 \stackrel{\text{def}}{=} [r]\mathsf{ff} \vee [r][s]\mathsf{ff}$ and its representative synthesised monitor $m_8 \stackrel{\text{def}}{=} r.\text{no} \oplus r.s.\text{no} = (\!|\varphi_9|\!)$. Although analysing φ_9 syntactically gives a history lower bound of 2, monitor m_8 rejects all violating systems with the single trace prefix rs. ■

We limit our calculations to a subset of recHML ruling out overlapping violating behaviour across disjunctions (Definition 10.8). The logical fragment $\text{sHML}^{\vee}_{\text{NF}}$ (below) combines universal modalities and disjunctions into one construct, $\bigvee_{i \in I}[\alpha_i]\varphi_i$, to represent the formula $[\alpha_1]\varphi_1 \vee \cdots \vee [\alpha_n]\varphi_n$ for the finite set index $I = \{1, \ldots, n\}$. When $I = \emptyset$, $\bigvee_{i \in I}[\alpha_i]\varphi_i$ denotes ff.

Definition 10.8. The logical fragment $\text{sHML}^{\vee}_{\text{NF}} \subseteq$ recHML is defined as:

$$\varphi, \psi \in \text{sHML}^{\vee}_{\text{NF}} ::= \mathsf{tt} \quad | \ \mathsf{ff} \quad | \ \varphi \wedge \psi \quad | \ \bigvee_{i \in I}[\alpha_i]\varphi_i \quad | \ \max X.\varphi \quad | \ X$$

where $\forall i, j \in I$, we have $i \neq j$ implies $\alpha_i \neq \alpha_j$. ■

Since disjunctions $\bigvee_{i \in I}[\alpha_i]\varphi_i$ in $\text{sHML}^{\vee}_{\text{NF}}$ require that all α_i actions are distinct, we can show that disjunction sub-formulae $[\alpha_i]\varphi_i$ must be violated by disjoint histories. Proposition 10.9 relies on the witness-based violation relation $H \models_{\text{DET}} \varphi$ of Definition 9.8, characterising the specific histories along which arbitrary ILTSs violate (closed) formulas.

Proposition 10.9 (Disjoint Violating Histories). For all (closed) formulae $\varphi \vee \psi \in \text{sHML}^{\vee}_{\text{NF}}$ and histories $H \in \text{Hst}$, if $H \models_{\text{DET}} \varphi \vee \psi$ then $H = H' \uplus H''$ such that $H' \models_{\text{DET}} \varphi$ and $H'' \models_{\text{DET}} \psi$.

Proof Outline. Assume $(H, \mathsf{f}) \models_{\text{DET}} \varphi \vee \psi$. Since $\varphi \vee \psi \in \text{sHML}^{\vee}_{\text{NF}}$, we know that

$$\varphi = \bigvee_{i \in I}[\alpha_i]\varphi_i' \qquad \text{and} \qquad \psi = \bigvee_{j \in J}[\alpha_j]\psi_j'$$

for some index set $I = \{1, \ldots, i\}$ and $J = \{i+1, \ldots, j\}$. We can further show that the violating history H can be decomposed into disjoint histories $H_1, \ldots, H_{i+j}$ such that, for any $k \in I \cup J$, all traces in H_k are prefixed

with an α_k action and $H_k \models_{\text{DET}} [\alpha_k]\varphi'_k$. Let $H' = H_1 \cup \cdots \cup H_i$ and $H'' = H_{i+1} \cup \cdots \cup H_j$. Repeatedly applying rule vOR from Definition 9.8, we obtain that $H' \models_{\text{DET}} \varphi$ and $H'' \models_{\text{DET}} \psi$.

We observe that the initial history H might also contain other traces that are not prefixed with any α_k action for $k \in I \cup J$, *i.e.*, $H = H' \uplus H'' \uplus H'''$. However, we can show that by adding more traces to H' (resp. H''), the resulting history $H' \cup H'''$ still violates the sub-formula φ (resp. ψ). This allows us to conclude that $H' \cup H''' \models_{\text{DET}} \varphi$ and $H'' \models_{\text{DET}} \psi$, as required.
For the complete proof, refer to Appendix G. □

Theorem 10.10 establishes a lower bound on the number of trace prefixes required to detect violations for formulae generated from $\text{sHML}^{\vee}_{\text{NF}}$.

Theorem 10.10 (History Lower Bounds). For all (closed) formulae $\varphi \in \text{sHML}^{\vee}_{\text{NF}}$ and histories $H \in \text{HST}$, if $H \models_{\text{DET}} \varphi$ then $|H| \geq lb(\varphi) + 1$.

Proof Outline. To prove this, we give an alternative definition $H \models^{s}_{\text{DET}} \varphi$ to the violation relation $H \models_{\text{DET}} \varphi$ of Definition 9.8 that is specific to $\text{sHML}^{\vee}_{\text{NF}}$ formulae (in contrast to $\models_{\text{DET}}$ which is defined over $\text{sHML}^{\vee}_{\text{DET}}$). In particular, we replace rule vOR in Definition 9.8 by the rule

$$\frac{H = H_1 \uplus H_2 \quad (H_1, \mathsf{true}) \models^{s}_{\text{DET}} \varphi \quad (H_2, \mathsf{true}) \models^{s}_{\text{DET}} \psi}{(H, \mathsf{true}) \models^{s}_{\text{DET}} \varphi \vee \psi}\ \text{sVOR}$$

where disjunction formulae are violated via disjoint histories. For the proof, refer to Appendix G. □

Example 10.11. Following Theorem 10.10, we can syntactically determine that $\varphi_1, \varphi_2, \varphi_4, \varphi_8 \in \text{sHML}^{\vee}_{\text{NF}}$ cannot be violated (by any system) with fewer than 2 trace prefixes since $lb(\varphi_1) = lb(\varphi_2) = lb(\varphi_4) = lb(\varphi_8) = 1$. In comparison, 1 trace prefix suffices to violate $\varphi_0, \varphi_3, \varphi_7 \in \text{sHML}^{\vee}_{\text{NF}}$ since $lb(\varphi_0) = lb(\varphi_3) = lb(\varphi_7) = 0$. ■

A pleasing side effect of Theorem 10.10 is that it also provides us with a simple syntactic check to determine whether formulae generated by $\text{sHML}^{\vee}_{\text{NF}}$ are worth monitoring for, according to Definition 8.8. Specifically, Corollary 10.12 below shows that whenever $lb(\varphi) = \infty$, the formula φ is always satisfied, meaning that violations for it can *never* be detected, regardless of the system being runtime verified.

Corollary 10.12. For all formulae $\varphi \in \text{sHML}^{\vee}_{\text{NF}}$ and histories $H \in \text{HST}$, if $lb(\varphi) = \infty$ then $H \not\models_{\text{DET}} \varphi$.

Proof. Suppose that $\varphi \in \text{sHML}^{\vee}_{\text{NF}}$ and $H \in \text{HST}$ such that $lb(\varphi) = \infty$. Since histories are finite, $|H| \leq lb(\varphi) < lb(\varphi) + 1 = \infty$. Our result, $H \not\models_{\text{DET}} \varphi$, follows by the contrapositive of Theorem 10.10. □

Example 10.13. The formula $\varphi_\infty \stackrel{\text{def}}{=} (\max X.[r][s]X) \vee [a][c]\mathsf{ff}$ turns out to be a tautology, *i.e.*, $\varphi_\infty \equiv \mathsf{tt}$, since $\max X.[r][s]X \equiv \mathsf{tt}$ and $\mathsf{tt} \vee [a][c]\mathsf{ff} \equiv \mathsf{tt}$. Accordingly, our lower bounds calculation gives us $lb(\varphi_\infty) = \infty$. ■

Finally, we note that there is a distinction between the number of SUS executions that need to be observed and the partial traces that need to be analysed: although a minimum of n trace prefixes might be required by Definition 10.6 for analysis, the SUS might need to be executed *more* than n times to obtain these prefixes, even in the best of cases. Intuitively, this is caused by redundancies in the

monitors (caused by the compositional method of monitor synthesis) and the incremental manner in which monitor instrumentation record trace prefixes, as illustrated in Example 10.14 below. When these redundancies are avoided, we end up with a one-to-one correspondence between the required number of SUS executions and partial traces.

Example 10.14. Assuming that $\textsc{Det}(a) = \mathsf{true}$, consider formula φ_{10} below, which describes the property *"after any number of serviced queries interspersed by arbitrary sequences of memory allocations, a system that can allocate memory cannot also perform a close action."*

$$\varphi_{10} \stackrel{\text{def}}{=} \max X.\big([r][s]X \wedge [a]X \wedge (\langle a\rangle \mathsf{tt} \Rightarrow [c]\mathsf{ff})\big) \;\equiv\; \max X.\big([r][s]X \wedge [a]X \wedge ([a]\mathsf{ff} \vee [c]\mathsf{ff})\big)$$

When synthesising the formula φ_{10}, we get the monitor $(\!|\varphi_{10}|\!) = m_2 \stackrel{\text{def}}{=} \mathrm{rec}X.\big(r.s.X \otimes a.X \otimes (a.\mathrm{no} \oplus c.\mathrm{no})\big)$ from Example 3.4. The system

$$p_{13} \stackrel{\text{def}}{=} \mathrm{rec}X.\big(r.s.X + a.X + a.c.\mathbf{0}\big)$$

violates φ_{10}, and monitor m_2 can detect this violation by rejecting the history $H = \{rsaa, rsac\} \subseteq T_{p_{13}}$. This is in line with Theorem 10.10 which states that all violating systems for φ_{10} cannot be rejected with fewer than $lb(\varphi_{10}) + 1 = 2$ trace prefixes. However, the incremental manner with which traces are aggregated (Chapter 7) requires that, whenever $rsaa \in H$, then $rsa \in H$ as well. This is due to the fact that for the execution trace $rsa\cdots$, we always have

$$\emptyset \triangleright (\epsilon, m_2) \triangleleft p_{13} \stackrel{rsa}{\Longrightarrow} \emptyset \triangleright (rsa, \mathrm{no}) \triangleleft p_{13'} \qquad \text{where} \qquad p'_{13} \stackrel{\text{def}}{=} r.s.p_{13} + a.p_{13} + a.c.\mathbf{0}$$

during the first monitored execution. Stated otherwise, although two prefixes are sufficient to detect a violation, the operational mechanism for aggregating the traces for analysis forces us to observe at least three SUS executions for the monitor to gather the necessary traces for analysis. ■

10.3 Summary

In this chapter, we outlined the steps towards a full automation of the multi-run monitoring technique proposed in Part II and gave a corresponding complexity analysis. We have also shown that although establishing execution upper bounds cannot be easily determined for any violating system, the least number of expected runs required to effect the runtime analysis can be calculated through a syntactic analysis of the disjunction operators used in the formula expressing the property being verified.

11 An Instantiation of the Multi-Run Framework

In this chapter, we validate the realisation of ILTSs from Chapter 6 and how realistic the constraints adopted in Chapter 9 are by considering an instantiation for actor-based systems [87, 21]. This concurrency model has been adopted by numerous programming languages, including Erlang [48], Akka [79], Elixir [90] and Swift [29], further highlighting the applicability and usability of our multi-run monitoring framework for real-world systems.

Structure of the chapter. In Section 11.1, we introduce the language for actor systems, whose operational semantics is specified in terms of an ILTS. In Section 11.2, we focus on two specific properties of the ILTS of Chapter 6, namely that transitions abstract over equivalent states and that silent (non-traceable) actions are confluent w.r.t. all other actions. We prove that transitions in our systems satisfy these two properties, reaffirming that our semantics is indeed an ILTS. In Section 11.3, we show which actions are treated as deterministic by our ILTS and prove results that justify this choice. Section 11.4 concludes.

11.1 Actor Systems

Actor systems are characterised by a set of concurrent processes called *actors* that interact with one another via *asynchronous message-passing*. Each actor is identified by its unique ID, which is used by other actors (possibly more than one) to send messages to it *i.e.,* the *single-receiver* property. Actors are also *persistently receptive* meaning that they are always able to receive messages addressed to them.

Figure 11.1 presents the syntax of our model actor language. This grammar assumes a set of *disjoint* actor names/addresses $i,j,h,k \in \text{Pid}$, atoms $a,b \in \text{Atom}$, expression variables $x,y \in \text{Vars}$, and term variables $X,Y \in \text{TVar}$. Values, $v \in \text{Val}$, range over $\text{Pid} \cup \text{Atom}$ and can be sent as messages. Identifiers, w, are syntactic entities that range over values and variables.

An actor system, $A,B \in \text{Actr}$, consists of multiple parallel actors $A \parallel B$, which can either be inactive, $\mathbf{0}$, or locally *scoped* to a subsystem of actors, $(\nu i)A$. A system may also have a number of asynchronous

$$
\begin{array}{rcl}
A,B \in \text{Actr} & ::= & \mathbf{0} \mid i[e \triangleleft q] \mid i\langle v \rangle \mid A \parallel B \mid (\nu i)A \\
e,d \in \text{Exp} & ::= & w_1!w_2.e \mid \mathtt{rcv}\{p_n \rightarrow e_n\}_{n\in I} \mid \mathtt{spwd\,as}\,x.e \mid \mathtt{self}\,x.e \mid \mathtt{rec}X.e \mid X \mid \mathbf{0} \\
p,o \in \text{Pat} & ::= & x \mid i \mid a \qquad\qquad q,r \in \text{MBox} ::= \epsilon \mid v{:}q
\end{array}
$$

Figure 11.1. *Erlang Syntax for Actor Systems*

messages in transit; a message in the ether [1] carrying value v addressed to i is denoted as $i\langle v\rangle$. Individual actors, $i[e \triangleleft q]$, are uniquely identifiable by their names, i, and consist of a running expression e and a mailbox q, *i.e.,* a list of values denoting a message queue. Incoming messages are added at the end of the queue, whereas pattern-matched messages are removed from the front of the queue. We use the list notation $q{:}r$ to denote queue concatenation, $v{:}q$ for the mailbox with v at the head and q at tail of the queue, and $q{:}v$ for the mailbox with v at the end of the queue preceded by q. When the mailbox is empty, ϵ, we often elide it from the individual actor and write $i[e]$ instead of $i[e \triangleleft \epsilon]$.

Actor expressions $e,d \in \textsc{Exp}$ can be outputs, $i!v.e$, or reading inputs from the mailbox through pattern-matching, $\mathtt{rcv}\{p_n \to e_n\}_{n\in I}$, where each expression e_n is guarded by a *disjoint* pattern p_n. We assume that patterns are disjoint, *i.e.,* if some v matches p_i, it does not match any other p_j for $i \neq j$ and $i,j \in I$. Expressions can also consist of self references (to the actor's own name), $\mathtt{self}\, x.e$, actor spawning, $\mathtt{spw}d\,\mathtt{as}\,x.e$, or recursion, $\mathrm{rec}X.e$. Receive patterns and spawn bind expression variables $x,y \in \textsc{Vars}$, whereas recursion binds term variables $X,Y \in \textsc{TVar}$. Similarly, $(\nu i)A$ binds the name ID i in A.

We assume the standard definitions $fn(A)$ and $fv(A)$ for the free names and variables of an actor system A and work up to α-conversion of bound names and variables. We also write $fId(A)$ for the free names i of the individual actors $i[e \triangleleft q]$ in A. *E.g.* for $A = i[e_1] \parallel j[e_2] \parallel (\nu h)h[e_3]$, we have $fId(A) = \{i,j\}$ and $fn(A) = \{i,j\} \cup fn(e_1) \cup fn(e_2) \cup (fn(e_1) \setminus \{h\})$. Running actor systems are closed, *i.e.,* $fv(A) = \emptyset$, and respect the single-receiver property, *i.e.,* whenever $A = B_1 \parallel B_2$ then $fId(B_1) \cap fId(B_2) = \emptyset$. For syntactic objects o,o', we write $o \sharp o'$ to mean that the free names of o and o' are disjoint, *e.g.* $A \sharp B$ denotes $fn(A) \cap fn(B) = \emptyset$. We also write K,d to mean $K \uplus \{d\}$ where $\uplus$ denotes disjoint union. Substitutions are partial maps from variables to values, $\sigma \in \textsc{Sub} : \textsc{Vars} \rightharpoonup \textsc{Val}$. The *subject* of an action $\eta \in \textsc{Act} \cup \{\tau\}$ is defined as $sbj(\tau) = sbj(com(i,v)) = sbj(ncom) = \emptyset$ and $sbj(i?v) = sbj(i!v) = \{i\}$.

The operational semantics of our language is given in terms of an ILTS. The implicit *observer* that A interacts with when running is represented by the set of IDs $O \subseteq \textsc{Pid}$; to model the single-receiver property, we have that $fId(A) \sharp O$. *Knowledge* $K \subseteq \textsc{Pid}$ denotes the set of IDs known by both an actor system A and the implicit observer with which it interacts. It is used by the rules in Figure 11.2 to keep track of bound/free names and abstract away from name bindings in actions [121, 83]; see [40].

Transitions are defined over system states of the form $K \mid O \triangleright A \in \textsc{Prc}$ where $fn(A) \subseteq K$, $O \subseteq K$ and $fId(A) \sharp O$. The ILTS transitions of the form $K \mid O \triangleright A \xrightarrow{\eta} K' \mid O' \triangleright B$ are governed by the judgement

$$K \mid O \triangleright A \xrightarrow{\eta} B \qquad (**)$$

defined by the rules in Figure 11.2. The evolution to K and O after η is left implicit in the judgement of $(**)$ since it is determined by the function $\mathbf{aft}(K \mid O,\eta)$ in Definition 11.1.

Definition 11.1 (Knowledge and Observer Evolution). The evolution of $K \mid O$ after action $\eta \in \textsc{Act}$, denoted as $\mathbf{aft}(K \mid O,\eta)$, is inductively defined as follows:

$$\mathbf{aft}(K \mid O, i?v) \stackrel{\text{def}}{=} K \cup fn(v) \mid O \cup (fn(v) \setminus K) \qquad \mathbf{aft}(K \mid O, com(i,v)) \stackrel{\text{def}}{=} K \mid O$$
$$\mathbf{aft}(K \mid O, i!v) \stackrel{\text{def}}{=} K \mid O \qquad \mathbf{aft}(K \mid O, ncom) \stackrel{\text{def}}{=} K \mid O$$
$$\mathbf{aft}(K \mid O, i\uparrow v) \stackrel{\text{def}}{=} K \cup fn(v) \mid O \qquad \mathbf{aft}(K \mid O, \tau) \stackrel{\text{def}}{=} K \mid O \qquad \blacksquare$$

[1] An ether is a communication medium.

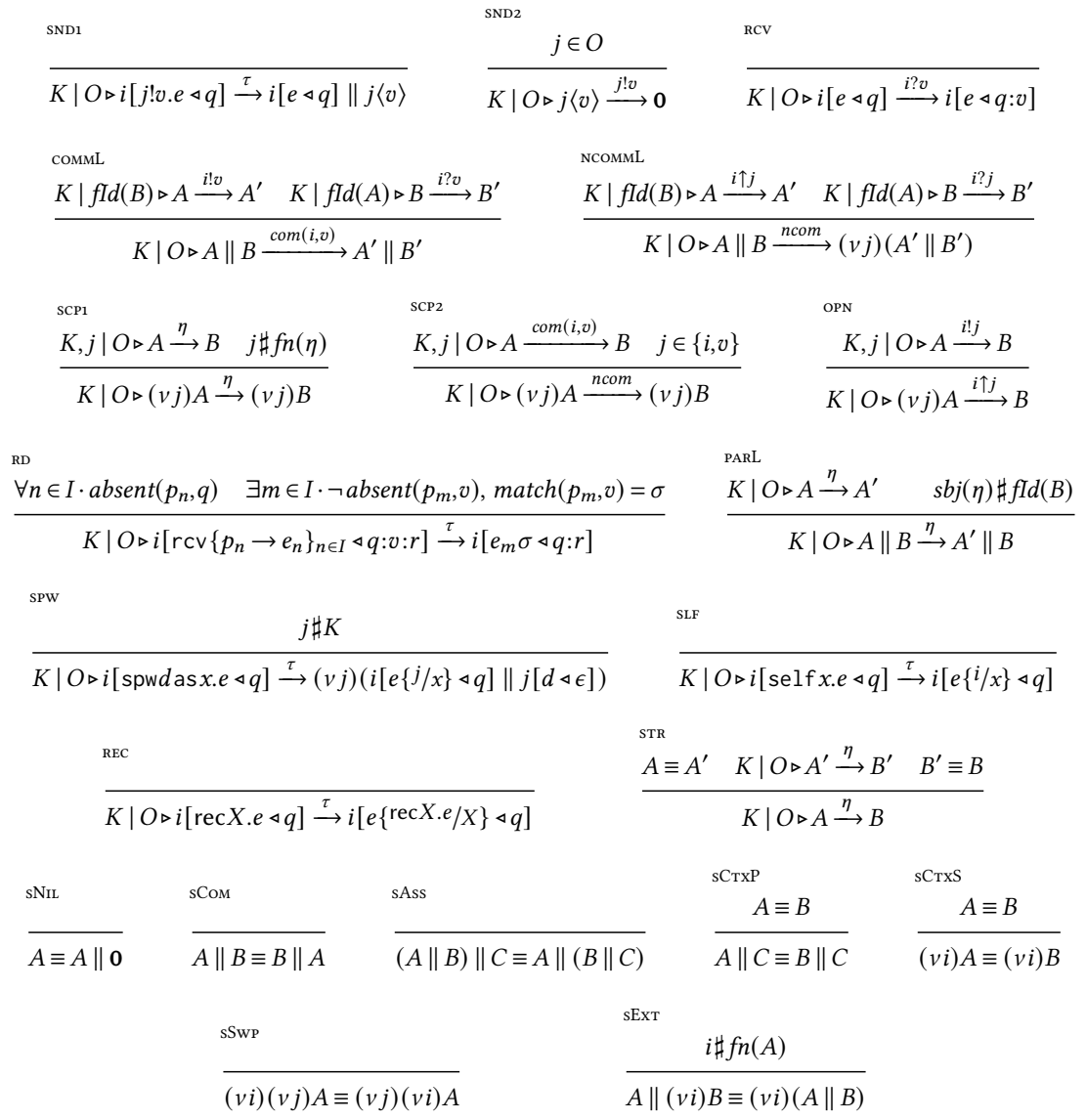


Figure 11.2. *Erlang Semantics for Actor Systems*

Actors communicate through asynchronous messages, which are sent in two stages: the actor first creates a message $j\langle v\rangle$ in the ether (rule SND1), and then the ether sends value v to actor j (rule SND2). Once received, messages are appended to the recipient's local mailbox (rule RCV) and *selectively* read following rule RD. This relies on the helper functions $absent(-)$ and $match(-)$ in Definition 11.2 below to find the first message v in the mailbox that matches one of the (pairwise-disjoint) patterns p_m in $\{p_n \to e_n\}_{n\in I}$. If a match is found, the actor branches to $e_m\sigma$, where e_m is the expression guarded by the matching pattern p_m and σ substitutes the free variables in e_m for the values resulting from the

pattern-match. Otherwise, reading blocks.

Definition 11.2. Functions $match : \textsc{Pat} \times \textsc{Val} \to \textsc{Sub} \cup \{\bot\}$ and $absent : \textsc{Pat} \times \textsc{MBox} \to \textsc{Bool}$ are defined as:

$$match(p,v) = \begin{cases} \emptyset & \text{if } p = v = i \text{ or } p = v = i = a \\ \{v/x\} & \text{if } p = x \\ \biguplus_{i=1}^{n} \sigma_i & \text{if } p = \{p_1,\ldots,p_n\} \text{ and } v = \{v_1,\ldots,v_n\}, \text{ and } \forall i \in \{1,\ldots,n\} \cdot match(p_i,v_i) = \sigma_i \\ \bot & \text{otherwise} \end{cases}$$

$$\sigma_1 \uplus \sigma_2 = \begin{cases} \sigma_1 \cup \sigma_2 & \text{if } \mathbf{dom}(\sigma_1) \cap \mathbf{dom}(\sigma_2) = \emptyset \\ \sigma_1 \cup \sigma_2 & \text{if } \forall v \in \mathbf{dom}(\sigma_1) \cap \mathbf{dom}(\sigma_2) \cdot \sigma_1(v) = \sigma_2(v) \\ \bot & \text{if } \sigma_1 = \bot \text{ or } \sigma_2 = \bot \\ \bot & \text{otherwise} \end{cases}$$

$$absent(p,\epsilon) = \text{true} \qquad absent(p,v{:}q) = \begin{cases} \text{false} & \text{if } match(p,v) = \bot \\ absent(p,q) & \text{otherwise} \end{cases}$$ ■

Parallel actors, $A \parallel B$, may *internally communicate* via rule NCOMML whenever A and B can respectively transition with dual output and input actions, $i!v$ and $i?v$, binding all names extruded by v in the process, or via rule COMML if all names in v are already known (symmetric rules NCOMMR and COMMR elided). Actors may also transition independently following rule PARL (symmetric rule PARR elided); the side-condition $sbj(\mu) \sharp fld(B)$ enforces the *single-receiver property* and checks that the message is not destined for some actor in B. An actor may also *scope extrude* names by communicating bound names to actors outside the scope (rule OPN). As will be discussed later, such a transition is represented by a specific form of output, $i{\uparrow}j$, denoting that the bound name j is scope extruded to actor i. Dually, when bound names are not mentioned in the action along which the transition occurs or the action denotes internal communication *ncom*, the names remain bound (rules SCP1 and SCP2). The remaining rules, SLF and SPW, are standard.

Our ILTS semantics assumes structural equivalence for actor systems $A \equiv B$ which is lifted as the process equivalence relation from Chapter 6, *i.e.,* $(K_1 \mid O_1 \triangleright A_1) \equiv (K_2 \mid O_2 \triangleright A_2)$ whenever $K_1 = K_2$, $O_1 = O_2$ and $A_1 \equiv A_2$; transitions abstract over such states via rule STR.

11.2 Actor Structural Equivalence and Silent Actions

In order to show that our semantics is indeed an ILTS, we need to prove a few additional properties. Proposition 11.3 below shows that transitions abstract over structurally-equivalent states.

Proposition 11.3 (Structural Invariance). For any actors $A, B \in \textsc{Actr}$ such that $A \equiv B$, whenever $K \mid O \triangleright A \xrightarrow{\eta} A'$ then there exists B' such that $K \mid O \triangleright B \xrightarrow{\eta} B'$ and $A' \equiv B'$.

Proof Outline. This required result follows from the following (stronger) statement since $A' \equiv A'$:

$$\text{for any } A,B \in \textsc{Actr} \text{ such that } A \equiv B, \text{ if } K \mid O \triangleright A \xrightarrow{\eta} A' \text{ then } K \mid O \triangleright B \xrightarrow{\eta} A'$$

The result is immediate from rule STR since $A' \equiv A'$. □

As a result of Proposition 11.4 below, we are guaranteed that any actor SUS instrumented via a mechanism that implements the semantics in Figures 3.1 and 7.1 can safely abstract over (non-traceable) silent transitions because they are confluent w.r.t. other actions.

Proposition 11.4 (τ-Confluence). If $K \mid O \triangleright A \xrightarrow{\tau} A'$ and $K \mid O \triangleright A \xrightarrow{\eta} A''$, then either $\eta = \tau$ and $A' \equiv A''$ or there exists an actor system B and moves $K \mid O \triangleright A' \xrightarrow{\eta} B$ and $\mathbf{aft}(K \mid O, \eta) \triangleright A'' \xrightarrow{\tau} B$.

Proof Outline. Intuitively, this is true because if $K \mid O \triangleright A$ does two *different* moves, then they must have occurred in different parallel components of A. The proof is by induction on the derivation of the first move, $K \mid O \triangleright A \xrightarrow{\tau} A'$, and is given in Appendix H.1. □

11.3 Actor Traceable Actions

Our actor semantics uses three forms of external actions:

$$\text{EACT} = \{ i?v,\ i!v,\ i{\uparrow}j \quad | \quad i,j \in \text{PID}, v \in \text{VAL} \}$$

Apart from input actions, $i?v$, and output actions, $i!v$, we identify a specific form of outputs, $i{\uparrow}j$, where the payload j is *scope-extruded* to the observer, which manifests itself as $j \notin K$ in our setting; see rule OPN in Figure 11.2. Our semantics also employs two forms of internal actions:

$$\text{IACT} = \{ com(i,v),\ ncom \quad | \quad i \in \text{PID},\ v \in \text{VAL} \}$$

We model actor communication via *internal communication actions*, $com(i,v)$, as opposed to using silent actions as is standard in [83, 121]. This permits the instrumented monitors to differentiate between different communication steps which can reach states that are not necessarily behaviourally equivalent. The exception to this strategy is internal communication involving scoped names, *ncom*; see rules NCOMML and SCP2 in in Figure 11.2. We still allow our monitor instrumentation to differentiate these transitions from silent actions, mainly because they do not satisfy properties such as Proposition 11.4, and thus treat them differently during runtime verification.

Remark 11.5. In the monitoring setup of Chapter 7 and the instantiation presented in this chapter, instrumentation records certain internal actions such as process communication, *ncom* and $com(a,b)$. We conjecture that this is a reasonable assumption since many programming language platforms come equipped with tracing mechanisms that are used for a variety of purposes such as debugging and telemetry. For instance, the Virtual Machine for Erlang and Elixir (two widely-used actor-based languages), called the EVM [48], records internal communication events as output events immediately followed by the corresponding input event. In existing tools, these two events are coalesced into silent (τ-)actions but they could be easily combined into internal communication events instead. ■

Our ILTS interpretation treats input actions, output actions and internal communication actions as deterministic, *i.e.,* for any $i \in \text{PID}$ and $v \in \text{VAL}$, we have $\text{DET}(i?v) = \text{DET}(i!v) = \text{DET}\big(com(i,v)\big) = \text{true}$. This treatment is justified by Propositions 11.6 to 11.8 below.

Proposition 11.6 (Input Determinacy). For all systems $K \mid O \triangleright A$ and input actions $i?v$, if $K \mid O \triangleright A \xrightarrow{i?v} A'$ and $K \mid O \triangleright A \xrightarrow{i?v} A''$ then $A' \equiv A''$.

Proof Outline. By induction on the sum of lengths of the proofs of the two transitions, relying also on the single-receiver property. The complete proof is given in Appendix H.2. □

Proposition 11.7 (Output Determinacy). For all systems $K \mid O \triangleright A$ and output actions $i!v$, if $K \mid O \triangleright A \xrightarrow{i!v} A'$ and $K \mid O \triangleright A \xrightarrow{i!v} A''$ then $A' \equiv A''$.

Proof. By induction on the sum of lengths of the proofs of the two transitions, relying also on $\equiv$ from Figure 11.2. The complete proof is given in Appendix H.2. □

Proposition 11.8 (Communication Determinacy). For all systems $K \mid O \triangleright A$ and internal communication actions $com(i,v)$, if $K \mid O \triangleright A \xrightarrow{com(i,v)} A'$ and $K \mid O \triangleright A \xrightarrow{com(i,v)} A''$ then $A' \equiv A''$.

Proof Outline. By induction on the sum of lengths of the proofs of the two transitions, relying also on Propositions 11.6 and 11.7. The complete proof is given in Appendix H.2. □

In contrast, scope-extruding output (*i.e.,* output that communicates bound names to actors outside the scope) and internal communication involving scoped names are *not* considered to be deterministic, *i.e.,* for all $i, j \in \textsc{Pid}$, we have $\textsc{Det}(i\uparrow j) = \textsc{Det}(ncom) = \text{false}$. Examples 11.9 and 11.10 illustrate why they are treated differently from other traceable actions.

Example 11.9 (Non-deterministic bound communication). Consider the actor state $K \mid O \triangleright A_1$ where $j \in O$ and the running actor is defined as

$$A_1 \stackrel{\text{def}}{=} (\nu i)(i[\mathtt{rcv}\, x \to j!x.\mathbf{0}] \parallel i\langle v_1\rangle \parallel i\langle v_2\rangle)$$

where $v_1 \neq v_2$. The actor identified by i, which is scoped by the outer construct (νi), can internally receive either value v_1 or v_1 via rules SCP2 and COMMR as follows:

$$K \mid O \triangleright A_1 \xrightarrow{ncom} K \mid O \triangleright (\nu i)(i[\mathtt{rcv}\, x \to j!x.\mathbf{0} \triangleleft v_1] \parallel \mathbf{0} \parallel i\langle v_2\rangle) \tag{11.1}$$

$$K \mid O \triangleright A_1 \xrightarrow{ncom} K \mid O \triangleright (\nu i)(i[\mathtt{rcv}\, x \to j!x.\mathbf{0} \triangleleft v_2] \parallel i\langle v_1\rangle \parallel \mathbf{0}) \tag{11.2}$$

In particular, since $v_1 \neq v_2$, the systems reached in (11.1) and (11.2) are *not* structurally equivalent. More importantly, these two states exhibit a different observational behaviour by sending different payloads to the observer actor at j; see rules RD, SND1 and SND2. ■

Example 11.10 (Non-determinisic scope-extruding outputs). Consider the actor system $K \mid O \triangleright A_2$ where $h \in O$. The running actor A_2 is defined below, where name i is locally scoped twice and e_1 and e_2 exhibit different behaviour.

$$A_2 \stackrel{\text{def}}{=} (\nu i)\big(i[e_1] \parallel h\langle i\rangle\big) \parallel (\nu i)\big(i[e_2] \parallel h\langle i\rangle\big)$$

The actor system $K \mid O \triangleright A_2$ can scope extrude name i by delivering the message $h\langle i\rangle$ in two possible ways using rules PARL, PARR and OPN as follows.

$$K \mid O \triangleright A_2 \xrightarrow{h\uparrow i} K \cup \{i\} \mid O \triangleright \big(i[e_1] \parallel \mathbf{0}\big) \parallel (\nu i)\big(i[e_2] \parallel h\langle i\rangle\big) \tag{11.3}$$

$$K \mid O \triangleright A_2 \xrightarrow{h\uparrow i} K \cup \{i\} \mid O \triangleright (\nu i)\big(i[e_1] \parallel h\langle i\rangle\big) \parallel (i[e_2] \parallel \mathbf{0}) \tag{11.4}$$

Since the systems reached in (11.3) and (11.4) above are *not* structurally equivalent, they are possibly not behaviourally equivalent either. Particularly, once an observer learns of the new actor address i, it could interact with it by sending messages and subsequently observe different behaviour through the different actor expressions e_1 and e_2. ■

The following example showcases how the properties introduced in Examples 1.1 and 1.3 to 2.4 can be adapted to monitor for actor systems.

Example 11.11. With the values *req*, *ans*, *all*, *cls*, *init* ∈ Atom, a server, expressed as actor i, can receive queries, $i?req$, reply to an observer client located at address j, $j!ans$, and send messages to a resource manager, abstracted as an observer actor at address h, to either allocate more memory, $h!all$, or close a connection, $h!cls$. We can reformulate φ_4 from Example 2.4 as formula φ_{11} below:

$$\varphi_{11} \stackrel{\text{def}}{=} \max X.\big([i?req][j!ans]X \wedge ([h!cls]\mathsf{ff} \vee [h!all]\mathsf{ff})\big)$$

Assuming $\{i, j, h, k_1, k_2\} \subseteq K$ and $\{j,h\} \subseteq O$, consider the server implementation $K \mid O \triangleright A_{\text{srv}}$ (defined below) that violates formula φ_{11}.

$$A_{\text{srv}} \stackrel{\text{def}}{=} i[\mathtt{rcv}\, req \to (k_1!init.k_2!init.j!ans)] \parallel k_1[\mathtt{rcv}\, init \to h!all] \parallel k_2[\mathtt{rcv}\, init \to h!cls.\mathbf{0}]$$

This implementation can produce the history $\{t_1,t_2\}$ where the traces t_1 and t_2 are defined as follows:

$$t_1 = (i?req).com(k_1,init).com(k_2,init).(j!ans).(h!all)$$
$$t_2 = (i?req).com(k_1,init).com(k_2,init).(j!ans).(h!cls)$$

Since, by Propositions 11.6 and 11.7, we know that $\textsc{Det}(i?req) = \textsc{Det}(i!ans) = \text{true}$, the visibility of the internal actions $com(k_1,init)$ and $com(k_2,init)$ in the trace prefixes t_1 and t_2 suffices for the representative monitor $m_8 \stackrel{\text{def}}{=} (\!|\varphi_{11}|\!)$ to reject A_{srv}.

This changes for the server implementation $K' \mid O \triangleright (\nu k_1,k_2)(A_{\text{srv}})$ where $K' = K \setminus \{k_1,k_2\}$. The aforementioned traces would change to t_3 and t_4 below:

$$t_3 = (i?req).ncom.ncom.(j!ans).(h!all)$$
$$t_4 = (i?req).ncom.ncom.(j!ans).(h!cls)$$

The obscured *ncom* events prohibit monitoring from determining whether behaviourally equivalent SUS states are reached after these transitions, thus soundly relate t_3 with t_4 in history $\{t_3,t_4\}$. As a result, the representative monitor m_8 cannot reject $K' \mid O \triangleright (\nu k_1,k_2)(A_{\text{srv}})$, even though it violates φ_{11}. ■

11.4 Summary

In this chapter, we outlined a possible instantiation of the multi-run monitoring RV framework proposed in Part II to actor-based systems, which, in turn, validates its applicability and usability. We have also shown, through a pathological example, how previously discussed recHML properties can be adapted to monitor for them in actor-based systems.

Part IV

Confluence and Determinism in Concurrent Systems

12 Systems as Process Calculi

The multi-run RV setup proposed in Part II is underpinned by an instrumentation mechanism that largely relies on the notions of *determinism* and *confluence*. In particular, we recall that our instrumentation *(i)* makes certain internal computation of system behaviour visible to the monitor to allow it to differentiate between *confluent* internal moves and moves that lead to non-deterministic branching, and *(ii)* provides the monitor with information about whether the observed actions are *deterministic*.

This part of the thesis presents a general study of these two notions in a wider context. In particular, confluence and determinism have been studied extensively for the π-calculus [78, 105, 106, 80, 113, 126, 116, 111, 81, 103, 130], the touchstone model for expressing concurrent systems. For instance, Philippou *et al.* [113] study these two properties for value-passing CCS agents expressed in the π-calculus and show they are preserved by certain system-building constructs. Although the π-calculus is highly general and capable of describing a wide range of computational patterns, concurrency in modern systems is often more structured. For instance, another prominent concurrent paradigm is Hewitt's actor model [87], later formalised by Agha in [22]; we have already shown that our multi-run RV framework can be instantiated to actor-based systems in Part III. Despite its conceptual similarities with the π-calculus, the actor model has several key characteristics that make it easier to distribute and implement in real-world systems [69]. In particular, we recall that actor-based systems consist of a set of processes called *actors* that observe the following properties:

(Prop. 1) Actors interact with one another through *asynchronous message-passing*.

(Prop. 2) Every actor is identified by a unique ID, which is used by other actors to send messages to it *i.e.,* the *single-receiver property*.

(Prop. 3) Actors are *persistently receptive*, *i.e.,* that can always receive messages addressed to them.

We systematically investigate how each of these three defining characteristics of the actor model impact confluence and determinism of concurrent systems. Concretely, we depart from the π-calculus and incrementally introduce *semantic* constraints that capture Prop. 1 to Prop. 3 for specific actor IDs; these constraints can be extended to all actors IDs as needed. At each step, we also assess whether the introduced semantic constraints provide any confluence or determinism guarantees. Since the monitoring setup in Part II considers classes of systems that are deterministic w.r.t. a *subset* of actions, we restrict our study of confluence and determinism to a subset of actions; although this study could be extended to a wider range of actions, we leave this for future work. Moreover, since semantic conditions are, in general, undecidable, we also identify *syntactic* conditions that guarantee their semantic counterparts.

Structure of the chapter. In Section 12.1, we present the system language that will be used throughout this part of this thesis, a standard version of the synchronous π-calculus with name matching. Its operational semantics is specified in terms of an LTS, based on the one in [71]. Section 12.2 then formally defines the notions of determinism and confluence, adapted from Philippou *et al.*'s work in [113]. It also establishes a well-known confluence result, asserting that certain internal actions do not affect the observable behaviour of the system. Section 12.3 concludes.

12.1 The Process Language

Figure 2.1 presents our process language, a standard version of the synchronous π-calculus with name matching. It assumes a finite set of disjoint channel names $a,b,c,d \in \textsc{Chans}$, name variables $x,y,z \in \textsc{Vars}$, process variables $X,Y \in \textsc{PVars}$, and values, $u,v \in \textsc{Val}$, which range over $\textsc{Chans} \cup \textsc{Vars}$.

A process, $P,Q \in \textsc{Prc}$, consists of multiple parallel processes $P \parallel Q$, which can either be inactive, $\mathbf{0}$, or locally *scoped* to a subsystem of processes, $(\nu c)P$, denoting that channel c is hidden from the environment and is thus private to process P. Processes interact by sending and receiving messages over channels using the standard constructs for input prefixing, $a?x.P$, and output prefixing, $a!b.P$. The name matching construct, $\mathsf{if}\ a=b\ \mathsf{then}\ P\ \mathsf{else}\ Q$, expresses conditional behaviour based on the equality of a and b. Finally, a process can be recursive, $\mathrm{rec}X.P$.

The input construct, $a?x.P$, the recursion construct, $\mathrm{rec}X.P$, and the scoping construct, $(\nu c)P$ respectively bind the free instances of variable x, process variable X and channel name c in P. We assume standard definitions for the free channel names, name variables and process variables in P, respectively denoted as $fn(P)$, $fv(P)$ and $fV(P)$. We also work up to α-conversion of bound channel names and process variables, assuming processes are closed, *i.e.,* $fv(P)=\emptyset$, and guarded, unless otherwise stated. The function $fIn(-)$ in Definition 12.1 below returns all the channel names that are free in an input position, *e.g.* $fIn(a?x.\mathbf{0})=\{a\}$ and $fIn((\nu a)(a?x.\mathbf{0}))=\emptyset$. General syntactic objects are ranged over by o,o', and we write $o \sharp o'$ to mean that the free names of o and o' are disjoint, *e.g.* $P \sharp Q$ denotes $fn(P) \cap fn(Q) = \emptyset$. We use K,c as shorthand for $K \uplus \{c\}$ where $\uplus$ denotes disjoint union. We also assume two substitutions, $\sigma \in \textsc{Sub} : \textsc{Vars} \rightharpoonup \textsc{Chans}$ and $\sigma \in \textsc{Sub} : \textsc{TVars} \rightharpoonup \textsc{Prc}$, which respectively map variable names to channel names and process variables to processes. This abuse of notation is justified by the fact that it is always clear from the context which substitution is being used, thereby avoiding any potential ambiguity.

Definition 12.1 (Free Input Channels). The function $fIn : \textsc{Prc} \mapsto \mathcal{P}(\textsc{Chans})$ is defined as follows:

$$fIn(\mathbf{0}) \stackrel{\text{def}}{=} \emptyset \qquad fIn(X) \stackrel{\text{def}}{=} \emptyset$$

$$fIn(u!v.P) \stackrel{\text{def}}{=} fIn(P) \qquad fIn(u?x.P) \stackrel{\text{def}}{=} \begin{cases} fIn(P) \cup \{a\} & \text{if } v = a \text{ for some } a \in \textsc{Chans} \\ \text{false} & \text{otherwise} \end{cases}$$

$$fIn(P \parallel Q) \stackrel{\text{def}}{=} fIn(P) \cup fIn(Q) \qquad fIn((\nu c)P) \stackrel{\text{def}}{=} fIn(P) \setminus \{c\}$$

$$fIn(\mathrm{rec}X.P) \stackrel{\text{def}}{=} fIn(P) \qquad fIn(\mathsf{if}\ u=v\ \mathsf{then}\ P\ \mathsf{else}\ Q) \stackrel{\text{def}}{=} fIn(P) \cup fIn(Q)$$ ■

The operational semantics of our language is given in terms of an LTS, defined by the rules in Figure 12.1. Interfaces $K \subseteq \textsc{Chans}$ denote the set of channel names *known* by the process and an implicit external observer with which it interacts; these are used by the rules in Figure 12.1 to keep track of bound/free

π-Calculus Syntax

$$
\begin{array}{llllll}
P,Q,\ldots \in \textsc{Prc} ::= & \mathbf{0} & \text{(inactive)} & \mid X & \text{(recursion variable)} \\
\mid u?x.P & \text{(input)} & \mid u!v.P & \text{(output)} \\
\mid P \parallel Q & \text{(parallel)} & \mid (\nu c)P & \text{(scoping)} \\
\mid \mathrm{rec}X.P & \text{(recursion)} & \mid \text{if } u=v \text{ then } P \text{ else } Q & \text{(conditional)}
\end{array}
$$

π-Calculus Semantics

$$
\textsc{pOut}\ \frac{}{K \triangleright a!b.P \xrightarrow{a!b} P} \qquad
\textsc{pIn}\ \frac{}{K \triangleright a?x.P \xrightarrow{a?b} P[b/x]} \qquad
\textsc{pRec}\ \frac{}{K \triangleright \mathrm{rec}X.P \xrightarrow{\tau} P[\mathrm{rec}X.P/X]} \qquad
\textsc{pScp1}\ \frac{K,b \triangleright P \xrightarrow{\eta} Q \quad b \sharp \eta}{K \triangleright (\nu b)P \xrightarrow{\eta} (\nu b)Q}
$$

$$
\textsc{pScp2}\ \frac{K,c \triangleright P \xrightarrow{com(a,b)} Q \quad c \in \{a,b\}}{K \triangleright (\nu c)P \xrightarrow{ncom} (\nu c)Q} \qquad
\textsc{pComm}\ \frac{K \triangleright P \xrightarrow{a!b} P' \quad K \triangleright Q \xrightarrow{a?b} Q'}{K \triangleright P \parallel Q \xrightarrow{com(a,b)} P' \parallel Q'} \qquad
\textsc{pNcomm}\ \frac{K \triangleright P \xrightarrow{a\uparrow b} P' \quad K \triangleright Q \xrightarrow{a?b} Q'}{K \triangleright P \parallel Q \xrightarrow{ncom} (\nu b)(P' \parallel Q')}
$$

$$
\textsc{pOpn}\ \frac{K,b \triangleright P \xrightarrow{a!b} Q}{K \triangleright (\nu b)P \xrightarrow{a\uparrow b} Q} \qquad
\textsc{pPar}\ \frac{K \triangleright P \xrightarrow{\mu} P'}{K \triangleright P \parallel Q \xrightarrow{\mu} P' \parallel Q} \qquad
\textsc{pThn}\ \frac{}{K \triangleright \text{if } c=c \text{ then } P \text{ else } Q \xrightarrow{\tau} P}
$$

$$
\textsc{pEls}\ \frac{c \sharp d}{K \triangleright \text{if } c=d \text{ then } P \text{ else } Q \xrightarrow{\tau} Q} \qquad
\textsc{pStr}\ \frac{P \equiv P' \quad K \triangleright P' \xrightarrow{\eta} Q' \quad Q' \equiv Q}{K \triangleright P \xrightarrow{\eta} Q}
$$

Figure 12.1. *π-Calculus Syntax and Semantics*

names and abstract away from channel name bindings in actions [121, 83] accordingly (see [40] for more details). System states are of the form $K \triangleright P$, where $fn(P) \subseteq K$ and P is closed, *i.e.,* $fv(P) = \emptyset$. An interface K is said to be *valid* for process P if $fn(P) \subseteq K$. Transitions in our LTS are defined over system states, $K \triangleright P \xrightarrow{\eta} K' \triangleright Q$, and are governed by judgements of the form

$$K \triangleright P \xrightarrow{\eta} Q \tag{$*$}$$

denoting that process P performs interaction η with the observer represented by interface K and transitions to process Q. The evolution of interface K after action η is left implicit in ($*$) as it can be recovered as a function of K and η, to be discussed in Definition 12.7. Interactions η,ξ range over

$$\textsc{Act} = \textsc{eAct} \cup \textsc{iAct} \cup \{\tau\}$$

where eAct is the set of *external* actions, iAct is the set of *internal* actions, and τ is a distinguished silent action.

Remark 12.2. Internal actions $\gamma,\delta \in \textsc{iAct}$ are used to model process communication, as opposed to using silent actions as is standard [83, 121, 113]. We differentiate between these two kinds of actions because,

as we will show later, process communication might lead to different observational behaviour, whereas silent actions capture β-moves in the sense of [131, 83]. ■

The set EAct is defined below and consists of three forms of *external* actions.

$$\alpha,\beta \in \text{EAct} = \{\ a?b,\ a!b,\ a\uparrow b \quad | \quad a,b \in \text{Chans}\ \}$$

Apart from input actions, $a?b$, and output actions, $a!b$, we identify a specific form of outputs, $a\uparrow b$, where channel name b is *scope extruded* to the observer, *i.e.,* channel name b is communicated as a payload to a processes outside the scope. In our setting, this manifests itself as $b \notin K$ (rule pOpn to be discussed later).

Remark 12.3. In standard text [121, 83], the action $a\uparrow b$ is expressed as $(\nu b)a!b$. However, the latter notation may lead to confusion about the scope of b, especially in the process reached after the interaction: the construct (νb) denotes a binder, but when it is used in an interaction, $(\nu b)a!b$, its meaning differs as channel b becomes fixed in the resulting process. We thus abstract away from name bindings in actions, and write $a\uparrow b$ to make it explicit that b is free in the resulting process; see [40] for details. ■

The set IAct contains two forms of *internal communication* actions that model process communication.

$$\gamma,\delta \in \text{IAct} = \{\ com(a,b),\ ncom \quad | \quad a,b \in \text{Chans}\ \}$$

Internal actions of the form $com(a,b)$ denote internal communication between two processes on channel name a carrying payload b (rule pComm to be discussed later). In contrast, when internal communication involves scoped names, internal actions take the form *ncom* (rules pNComm and pScp2, discussed later).

Remark 12.4. In actions $a?b$, $a!b$, $a\uparrow b$, and $com(a,b)$, channel name a is the *subject*, whereas channel name b is the *object*. Occasionally, we also use the suggestive notation $a?\_$ to denote that the object of that input action ranges over all channel names $b \in \text{Chans}$. ■

The judgement in $(*)$ is defined by the reduction rules in Figure 12.1. Rules pOut and pIn respectively detail how messages are sent and received. Parallel processes, $P \,\|\, Q$, may *internally* communicate via rule pNComm whenever P and Q can respectively transition with dual input and output actions, $a?b$ and $a!b$, binding all names extruded by b in the process, or via rule pComm if all names in b are already known. Alternatively, processes P and Q in $P \,\|\, Q$ may transition independently from one another following rule pPar. A process may also perform *scope-extrusion* by communicating bound names b to names outside the scope, *i.e.,* $b \notin K$, via rule pOpn. Dually, when bound channel names are not mentioned in the action along which the transition occurs or the action denotes internal communication *ncom*, the names remain bound (rules pScp1 and pScp2 respectively). Rules pThn and pEls describe standard conditional branching based on name matching. Our semantics assumes standard structural equivalence; transitions abstract over structurally equivalent states via rule pStr, thus allowing for a more concise presentation of the reduction rules.

Structural equivalence permits the syntactic rearrangement of processes and is defined as the least relation on processes that satisfy the rules in Definition 12.5 below. The axioms sCom, sAss and sNil respectively state that parallel composition is commutative and associative, and that it contains the inactive process, **0**. Axiom sSwp allows for the reordering of channel name bindings, whereas axiom

sExt allows the a-binder to be extended to a new process P or retracted, as long as channel name a is not free in that process, *i.e.*, $a \sharp fn(P)$. Finally, rules sCtxP and sCtxS respectively state that structural equivalence is closed under parallel composition and scoping.

Definition 12.5. *Structural equivalence*, denoted as $\equiv$, is the *least* relation of the form $(\text{Prc} \times \text{Prc})$ satisfying the following rules:

$$\text{sCom}\ \frac{}{P \parallel Q \equiv Q \parallel P} \qquad \text{sAss}\ \frac{}{(P \parallel Q) \parallel R \equiv P \parallel (Q \parallel R)} \qquad \text{sNil}\ \frac{}{P \equiv P \parallel \mathbf{0}} \qquad \text{sCtxP}\ \frac{P \equiv Q}{P \parallel R \equiv Q \parallel R} \qquad \text{sCtxS}\ \frac{P \equiv Q}{(\nu a)P \equiv (\nu a)Q}$$

$$\text{sSwp}\ \frac{}{(\nu a)(\nu b)P \equiv (\nu b)(\nu a)P} \qquad \text{sExt}\ \frac{a \sharp fn(P)}{P \parallel (\nu a)Q \equiv (\nu a)(P \parallel Q)}$$

Two processes, P and Q, are said to be structurally equivalent whenever $P \equiv Q$. ■

Remark 12.6. Rules sCom and sAss, together with rule pStr, inherently capture the symmetries of rules pPar, pComm and pNcomm in Figure 12.1, thereby facilitating the presentation of the reduction rules. ■

Definition 12.7 below formally defines how the evolution of interface K after action η can be retrieved through the function $\mathbf{aft}(K,\eta)$, where both the observer and the process may extend K through input and output actions respectively.

Definition 12.7 (Interface Evolution). The function $\mathbf{aft} : \mathcal{P}(\text{Chans}) \times \text{Act} \mapsto \mathcal{P}(\text{Chans})$ is inductively defined as follows:

$$\mathbf{aft}(K, a?b) \stackrel{\text{def}}{=} K \cup \{b\} \qquad \mathbf{aft}(K, com(a,b)) \stackrel{\text{def}}{=} K \qquad \mathbf{aft}(K, ncom) \stackrel{\text{def}}{=} K$$
$$\mathbf{aft}(K, a!b) \stackrel{\text{def}}{=} K \qquad \mathbf{aft}(K, a\uparrow b) \stackrel{\text{def}}{=} K \cup \{b\} \qquad \mathbf{aft}(K, \tau) \stackrel{\text{def}}{=} K$$ ■

Example 12.8. Consider process $P_1 \stackrel{\text{def}}{=} (\nu\, \text{sec})(\text{inv}?x.x!\text{sec}.\mathbf{0})$, modelling a toy server that waits for inputs on an invocation channel, inv, and replies with a secure channel name, sec. For interface $K = \{\text{inv}\}$, denoting that both P_1 and the observer know of channel inv and can communicate over it, the system $K \triangleright P_1$ can perform the transition

$$K \triangleright P_1 \xrightarrow{\text{inv?ret}} \big((\nu\, \text{sec})(x!\text{sec}.\mathbf{0})\big)[^{\text{ret}}/_x] \qquad \text{(using rules pScp1 and pIn)}$$

where $\big((\nu\text{sec})(x!\text{sec}.\mathbf{0})\big)[^{\text{ret}}/_x] = (\nu\text{sec})(\text{ret}!\text{sec}.\mathbf{0})$. The interface after the transition is $K' = \mathbf{aft}(K, \text{inv?ret}) = K \cup \{\text{ret}\}$, denoting that the process is now aware of the return channel name, ret, which it can use to communicate with the observer by transitioning w.r.t. the extended interface K' as follows:

$$K' \triangleright (\nu\, \text{sec})(\text{ret}!\text{sec}.\mathbf{0}) \xrightarrow{\text{ret}\uparrow\text{sec}} \mathbf{0} \qquad \text{(using rules pOpn and pOut)}$$

Following this transition, the observer becomes aware of the previously scoped channel name sec. Consequently, the interface is extended to $\mathbf{aft}(K', \text{ret}\uparrow\text{sec}) = K' \cup \{\text{sec}\}$. ■

Traceable actions μ, λ range over both external and internal actions, $\eta, \xi \in \text{tAct} = \text{eAct} \cup \text{iAct}$. Since the monitors of Chapter 7 can observe any traceable action, and our goal is to examine what makes

systems well-behaved in terms of terms of determinism and confluence—as utilised by the monitors of Chapter 7—we adopt a finer-grained notion of observational behaviour than is standard in the π-calculus [121, 83, 113] where only external actions are observable. Specifically, our semantics treats traceable actions as observable, and weak transitions $K \triangleright P \Rightarrow Q$ only abstract over silent actions. Given $\mu,\lambda \in \textsc{tAct}$ and $\eta \in \textsc{Act}$, we write:

- $K \triangleright P \xrightarrow{\mu}\!\!\!\!\!\!/\;\;$ to mean that $\nexists Q$ such that $K \triangleright P \xrightarrow{\mu} Q$
- $K \triangleright P \xrightarrow{\mu} \mathbf{aft}(K,\mu) \triangleright P' \xrightarrow{\lambda} P''$ to mean that $K \triangleright P \xrightarrow{\mu} P'$ and $\mathbf{aft}(K,\mu_1) \triangleright P' \xrightarrow{\lambda} P''$
- $K \triangleright P \Longrightarrow Q$ to mean that $\exists P_1,\ldots,P_n$ such that $K \triangleright P_0 \xrightarrow{\tau} K \triangleright P_1 \xrightarrow{\tau} \cdots \xrightarrow{\tau} P_n$ where $P_0 = P$ and $P_n = Q$
- $K \triangleright P \overset{\mu}{\Longrightarrow} Q$ to mean that $\exists P',P''$ such that $K \triangleright P \Rightarrow K \triangleright P' \xrightarrow{\mu} \mathbf{aft}(K,\mu) \triangleright P'' \Rightarrow Q$.
- $K \triangleright P \overset{\hat{\eta}}{\Longrightarrow} Q$ to mean that $K \triangleright P \Rightarrow Q$ when $\eta = \tau$ and $K \triangleright P \overset{\eta}{\Longrightarrow} Q$ when $\eta \neq \tau$

Traceable actions can be sequenced to form *traces*, $t,u \in \textsc{Trc} = \textsc{tAct}^*$, representing finite prefixes of system runs. For $t = \mu_1 \cdots \mu_n$, we write $P \overset{t}{\Longrightarrow} Q$ to mean $K \triangleright P \overset{\mu_1}{\Longrightarrow} \cdots \overset{\mu_n}{\Longrightarrow} Q$ and lift the function in Definition 12.7 to traces, $\mathbf{aft}(K,t)$, in the obvious manner.

Definition 12.9 (System Derivative). A system $K' \triangleright Q$ is a *derivative* of system $K \triangleright P$ iff there exists a trace $t \in \textsc{Trc}$ such that $K \triangleright P \overset{t}{\Longrightarrow} Q$ and $\mathbf{aft}(K,t) = K'$. ■

Two systems are *observationally equivalent* if each time one system performs an action η, then the other system can match that move through a sequence of silent actions, followed by the same action η, and possibly further silent actions. This is formalised as (weak) bisimilarity in Definition 12.10 below.

Definition 12.10 (Bisimilarity). A binary relation $\mathcal{R}$ is a (weak) bisimulation iff for each $(K \triangleright P, K' \triangleright Q) \in \mathcal{R}$ and action $\eta \in \textsc{Act}$,

(i) $K = K'$;
(ii) if $K \triangleright P \xrightarrow{\eta} P'$ then $K' \triangleright Q \overset{\hat{\eta}}{\Longrightarrow} Q'$ for some Q' such that $\left(\mathbf{aft}(K,\eta) \triangleright P', \mathbf{aft}(K',\eta) \triangleright Q'\right) \in \mathcal{R}$;
(iii) if $K' \triangleright Q \xrightarrow{\eta} Q'$ then $K \triangleright P \overset{\hat{\eta}}{\Longrightarrow} P'$ for some P' such that $\left(\mathbf{aft}(K',\eta) \triangleright P', \mathbf{aft}(K,\eta) \triangleright Q'\right) \in \mathcal{R}$.

Two systems, $K \triangleright P$ and $K' \triangleright Q$, are *(weakly) bisimilar*, denoted as $K \triangleright P \approx K' \triangleright Q$, iff there is a (weak) bisimulation that relates them. As a shorthand, we often write $K \models P \approx Q$ to mean $K \triangleright P \approx K \triangleright Q$. ■

Since two systems are bisimilar if there is a bisimulation that relates them, to prove they are bisimilar, it only suffices to exhibit a bisimulation that relates them. This is better illustrated in the proof for Theorem 12.11 below. For the sake of clarity, we occasionally describe transitions as relations over system states and write $K \triangleright P \xrightarrow{\eta} \mathbf{aft}(K,\eta) \triangleright Q$ instead of $K \triangleright P \xrightarrow{\eta} Q$.

Theorem 12.11. For all processes $P,Q \in \textsc{Prc}$, if $P \equiv Q$ then $K \models P \approx Q$.

Proof. Suppose that $P \equiv Q$. We need to show $K \models P \approx Q$. To do so, we need to find a bisimulation $\mathcal{R}$ such that $(K \triangleright P, K \triangleright Q) \in \mathcal{R}$. Let us define it as follows:

$$\mathcal{R} = \{\, (K \triangleright P, K \triangleright Q) \quad | \quad P \equiv Q \,\}$$

We have to show that $\mathcal{R}$ is a bisimulation, *i.e.,* it meets the requirements in Definition 12.10.
Pick a pair $(K \triangleright P, K \triangleright Q) \in \mathcal{R}$ and suppose $K \triangleright P \xrightarrow{\eta} \mathbf{aft}(K,\eta) \triangleright P'$. By definition of $\mathcal{R}$, we know $P \equiv Q$.

Moreover, since $\equiv$ is an equivalence relation, $P' \equiv P'$. Applying rule pStr from Figure 12.1, we thus obtain $K \triangleright Q \xrightarrow{\eta} P'$, which by definition implies $K \triangleright Q \stackrel{\hat{\eta}}{\Longrightarrow} P'$. Since $P' \equiv P'$, then, by definition of $\mathcal{R}$ again, we know $\left(\mathbf{aft}(K,\eta) \triangleright P', \mathbf{aft}(K,\eta) \triangleright P'\right) \in \mathcal{R}$ as well. This satisfies condition (ii) of Definition 12.10.
Proving condition (iii) of Definition 12.10 follows with analogous reasoning. □

12.2 What Makes a System Confluent or Deterministic?

In this section, we formally define what we understand by *determinism* and *confluence* for the systems of Section 12.1, which are adapted from Philippou *et al.*'s work in [113]. Since our aim is to identify a subset of actions that are confluent or deterministic w.r.t. the class of systems that satisfy Prop. 1 to Prop. 3, without requiring the systems to be confluent/deterministic themselves, our definitions are parametric w.r.t. the set of actions Act. Concretely, Definition 12.12 asserts that a system is *deterministic* w.r.t. a specific action if all states reached after transitioning with that action exhibit the same behaviour.

Definition 12.12 (Determinism). A system $K \triangleright P$ is *deterministic* w.r.t. action $\eta \in \text{Act}$ if for all processes $Q, Q' \in \text{Prc}$, $K \triangleright P \xrightarrow{\eta} Q$ and $K \triangleright P \xrightarrow{\eta} Q'$ implies $\mathbf{aft}(K,\eta) \models Q \approx Q'$. ■

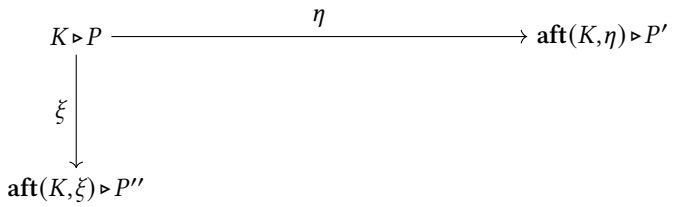
The notion of confluence is formalised in Definition 12.13.

Definition 12.13 (Confluence). A system $K \triangleright P$ is *confluent* w.r.t. actions η and ξ if whenever $K \triangleright P \xrightarrow{\eta} P'$ and $K \triangleright P \xrightarrow{\xi} P''$ then either

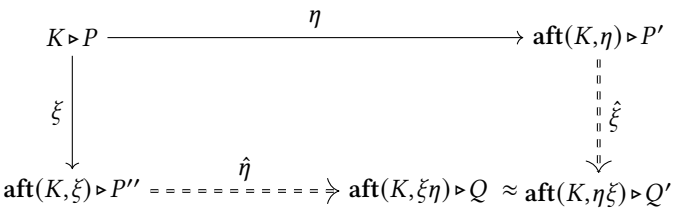
(i) $\eta = \xi$ and $\mathbf{aft}(K,\eta) \models P' \approx P''$,

(ii) or there exist processes Q, Q' and moves $\mathbf{aft}(K,\xi) \triangleright P'' \stackrel{\hat{\eta}}{\Longrightarrow} \mathbf{aft}(K,\xi\eta) \triangleright Q$ and $\mathbf{aft}(K,\eta) \triangleright P' \stackrel{\hat{\xi}}{\Longrightarrow} \mathbf{aft}(K,\eta\xi) \triangleright Q'$ where $\mathbf{aft}(K,\xi\eta) \triangleright Q \approx \mathbf{aft}(K,\eta\xi) \triangleright Q'$.

Diagrammatically, we can say that given



then either $\eta = \xi$ and $\mathbf{aft}(K,\eta) \triangleright P' \approx \mathbf{aft}(K,\eta) \triangleright P''$, or there exist transitions



In such diagrams, solid lines indicate the given moves and dashed lines indicate those required. ■

Remark 12.14. A major difference from Definitions 12.12 and 12.13 and Philippou *et al.*'s [113] notions of determinism and confluence is that, in the latter, two states are observationally equivalent if they

can produce the same sequence of external actions $\alpha \in \text{EACT}$. In contrast, our notion of observational equivalence is more stringent. Notably, two states are observationally equivalent if they can produce the same sequence of traceable actions, $\mu \in \text{TACT} = \text{EACT} \cup \text{IACT}$. The reason for this is, once again, that the monitors of Chapter 7 are privy to TACT. ■

As already previewed in Remark 12.2, Theorem 12.15 below shows that silent actions capture β-moves in the sense of [131, 83], where τ-moves do *not* affect the possible occurrence of other moves, *i.e.,* they do not affect the branching structure of the ILTS w.r.t. any other actions in ACT. This result can then be used to show that systems generated by the grammar in Figure 2.1 are confluent w.r.t. τ actions and any $\eta \in \text{ACT}$, formalised in Corollary 12.16 below. Importantly, this allows us to safely abstract away from silent actions when reasoning about the system's behaviour.

Theorem 12.15. If $K \triangleright P \xrightarrow{\eta} P'$ and $K \triangleright P \xrightarrow{\tau} P''$ then either

(i) $\eta = \tau$ and $P' \equiv P''$,
(ii) or there exists a process Q and moves $\mathbf{aft}(K,\eta) \triangleright P' \xrightarrow{\tau} Q$ and $K \triangleright P'' \xrightarrow{\eta} Q$.

Proof Outline. The proof is similar to that in [83, Lemma 6.6]. Intuitively, it holds because if $K \triangleright P$ does two *different* moves $K \triangleright P \xrightarrow{\tau} P'$ and $K \triangleright P \xrightarrow{\eta} P''$, then both moves must have occurred in different (parallel) components of P. The proof is by rule on the two transitions $K \triangleright P \xrightarrow{\eta} P'$ and $K \triangleright P \xrightarrow{\tau} P''$. □

Corollary 12.16 (Confluence w.r.t. τ-moves). All derivatives of system $K \triangleright P$ are confluent w.r.t. silent actions and any $\eta \in \text{ACT}$.

Proof Outline. Follows from Theorems 12.11 and 12.15. □

Remark 12.17. The semantics in Figure 12.1 can be seen as an ILTS (see Chapter 6). Indeed, it satisfies the ILTS property stating that silent τ transitions are confluent w.r.t. other actions (Theorem 12.15). For systems expressed using the standard π-calculus, we have $\text{DET}(\eta) = \text{false}$ for all actions $\eta \in \text{ACT}$. However, the output of this predicate changes throughout this part of the thesis, depending on the determinacy results we obtain by imposing semantic constraints that guarantee Prop. 1 to Prop. 3. ■

12.3 Conclusion

This chapter presented the process language, a standard version of the synchronous π-calculus with name matching, that will be used throughout this part of the thesis. It also formally established what it means for a system to be deterministic or confluent w.r.t. specific actions.

13 Actor Properties and their Behavioural Implications

This chapter investigates how the three defining characteristic of actor systems, namely Prop. 1 to Prop. 3 from Chapter 12, effect determinism and confluence of concurrent systems expressed using the π-calculus. We also recall that we only examine these properties for *specific* channel names, but they can be extended to all (free) channel names as needed.

We depart from the π-calculus and introduce *semantic* constraints that capture Prop. 1 to Prop. 3. Based on these, we consider a *subset* of system actions and examine whether they give any determinism or confluence guarantee. While our main focus are the semantic constraints and the corresponding implications for determinism and confluence, we show that these semantic constraints can be enforced via syntactic restrictions on the structure of the process itself. The syntactic constraints be seen as an "approximation" of their semantic counterpart, which offer a conservative but lightweight and static means of ensuring that the system behaviour aligns with that specified by the corresponding properties.

Structure of the chapter. Section 13.1 formalises semantic constraints that capture Prop. 1 and prove the corresponding determinism/confluence results. Section 13.1.1 then identifies syntactic conditions that soundly approximates these semantic constraints. Sections 13.2 and 13.2.1 and Sections 13.3 and 13.3.1 correspond to Sections 13.1 and 13.1.1 but in the context of Prop. 2 and Prop. 3 respectively. Section 13.4 summarises the obtained results.

13.1 The Asynchronous Message-Passing Property

We start by formalising a semantic condition that captures Prop. 1 for specific channel names. Definition 13.1 below asserts that a system $K \triangleright P$ outputs asynchronously on channel a via action $a!b$, evolving to Q in the process, if $K \triangleright P$ and $K \triangleright Q \parallel a!b.\mathbf{0}$ exhibit the same behaviour. This ensures that the system $K \triangleright P$ does *not* need to wait for a matching receiver for $a!b$ before continuing execution. Instead, the message is sent as soon as a corresponding receiver becomes available. Similarly, when $K \triangleright P$ may transition to Q after scope extruding channel name b via action $a\uparrow b$, it must behave the same as $K \triangleright (\nu b)(Q \parallel a!b.\mathbf{0})$. These restrictions on outputs correspond to the properties identified by Amadio *et al.* for the asynchronous π-calculus [26].

Definition 13.1 (Asynchrony). A system $K \triangleright P$ *outputs asynchronously* on channel name a whenever

(i) if $K \triangleright P \xrightarrow{a!b} Q$ then $K \models P \approx Q \parallel a!b.\mathbf{0}$; and
(ii) if $K \triangleright P \xrightarrow{a\uparrow b} Q$ then $K \models P \approx (\nu b)(Q \parallel a!b.\mathbf{0})$.

■

A system $K \triangleright P$ then *persistently* outputs asynchronously on channel name a if the property in Definition 13.1 is preserved by all system transitions. This is formalised Definition 13.2, which relies on the notion of system derivatives in Definition 12.9.

Definition 13.2 (Persistent Asynchrony). A system $K \triangleright P$ *persistently outputs asynchronously* on channel name a whenever each derivative $K' \triangleright Q$ of $K \triangleright P$ outputs asynchronously on channel name a. ■

Example 13.3. Using Definitions 13.1 and 13.2, we can show that for $K = \{a, b\}$ and $P_2 \stackrel{\text{def}}{=} \text{rec}X.a!b.X$, the system $K \triangleright P_2$ persistently outputs asynchronously on channel name a. Concretely, although $K \triangleright P_2$ cannot immediately output on channel a (which vacuously satisfies the conditions in Definition 13.1), it can do so after one recursive unfolding:

$$K \triangleright P_2 \xrightarrow{\tau} K \triangleright a!b.P_2 \xrightarrow{a!b} P_2$$

It is not hard to see that $K \triangleright a!b.P_2$ and $K \triangleright P_2 \parallel a!b.\mathbf{0}$ exhibit the same behaviour as they both repeatedly output b on a. In particular, we can show that $K \models a!b.P_2 \approx P_2 \parallel a!b.\mathbf{0}$ via the following bisimulation:

$$\mathcal{R} \stackrel{\text{def}}{=} \left\{ \left(K \triangleright P_2, K \triangleright P_2\right), \left(K \triangleright a!b.P_2, K \triangleright P_2 \parallel a!b.\mathbf{0}\right) \right\}$$

By Definition 13.1, this means that system $K \triangleright a!b.P_2$ outputs asynchronously on channel name a. Moreover, the system $K \triangleright a!b.P_2$ can transition with $a!b$ and return to the initial state $K \triangleright P_2$, which we have already shown to asynchronously output on a.

In contrast, the system $K \triangleright P_3$ where $P_3 \stackrel{\text{def}}{=} a!b.a?x.\mathbf{0}$ can transition with action $a!b$ and reach the state $K \triangleright a?x.\mathbf{0}$. However, $K \triangleright P_3$ and $K \triangleright a?x.\mathbf{0} \parallel a!b.\mathbf{0}$ exhibit different behaviours: the latter can immediately receive on a, whereas the former can only do so after an $a!b$ transition. Hence, these two systems are *not* bisimilar, which violates condition (i) of Definition 13.1. ■

Since Prop. 1 implicitly requires all outputs to be asynchronous, regardless of the name for which that message is destined, we lift Definition 13.2 to sets of names, formalised in Definition 13.4. Intuitively, Definition 13.4 asserts that as soon as previously-scoped names become available for communication, any outputs on those names must always occur asynchronously.

Definition 13.4 (Full Asynchrony). A system $K \triangleright P$ is *fully asynchronous* whenever each derivative $K' \triangleright Q$ of $K \triangleright P$ outputs asynchronously on all $a \in K'$. ■

Our first key result, formalised in Theorem 13.5 below, asserts that if a system satisfies Prop. 1—captured by the semantic constraint in Definition 13.4—then all output actions, $a!b$, lead to bisimilar states.

Theorem 13.5 (Output Determinacy). A fully asynchronous system $K \triangleright P$ is deterministic w.r.t. any output action $a!b$.

Proof Outline. Suppose $K \triangleright P$ has asynchronous outputs on channel name a and $K \triangleright P \xrightarrow{a!b} K' \triangleright Q$ and $K \triangleright P \xrightarrow{a!b} K' \triangleright Q'$. We have to show $K' \models Q \approx Q'$.

From these two transitions, we can show there exist $\vec{d} \subseteq \text{Chans}$ and $P_1, P_2, P_3, P_4 \in \text{Prc}$ such that

$$\begin{aligned} P &\equiv (\nu \vec{d})(P_1 \parallel a!b.P_2) \qquad \text{and} \qquad Q \equiv (\nu \vec{d})(P_1 \parallel P_2) \\ P &\equiv (\nu \vec{d})(P_3 \parallel a!b.P_4) \qquad \text{and} \qquad Q' \equiv (\nu \vec{d})(P_3 \parallel P_4) \end{aligned} \tag{13.1}$$

By transitivity/symmetry, we obtain $(\nu \vec{d})(P_1 \parallel a!b.P_2) \equiv (\nu \vec{d})(P_3 \parallel a!b.P_4)$. Since this could have only been derived using rule sCtxS, we also know $P_1 \parallel a!b.P_2 \equiv P_3 \parallel a!b.P_4$. This gives us a number of sub-cases, but their corresponding proofs are long and tedious. We focus on the main ones:

- When $a!b.P_2 = a!b.P_4$ and $P_1 \equiv P_3$, by the statements in eq. (13.1) and structural equivalence rules, we obtain that $Q \equiv Q'$. Our result, $K' \models Q \approx Q'$, follows by Theorem 12.11.
- When $P_1 \equiv R \parallel R'$ and $P_3 \equiv R \parallel a!b.P_2$ and $a!b.P_4 \equiv R'$, using the statements in eq. (13.1), the structural equivalence rules and transitivity, we can show that

  $$Q \equiv (\nu \vec{d})\big((R \parallel a!b.P_4) \parallel P_2\big) \quad \text{and} \quad Q' \equiv (\nu \vec{d})\big((R \parallel a!b.P_2) \parallel P_4\big) \tag{13.2}$$

  Using the the reduction rules in Figure 2.1, we can also show that these two systems can output channel b on channel a as follows:

  $$\begin{aligned} K' \triangleright (\nu \vec{d})\big((R \parallel a!b.P_4) \parallel P_2\big) &\xrightarrow{a!b} (\nu \vec{d})\big((R \parallel P_4) \parallel P_2\big) \quad \text{and} \\ K' \triangleright (\nu \vec{d})\big((R \parallel a!b.P_2) \parallel P_4\big) &\xrightarrow{a!b} (\nu \vec{d})\big((R \parallel P_2) \parallel P_4\big) \end{aligned}$$

  But by the assumption that $K \triangleright P$ is fully asynchronous and Definition 13.4, we also know that all its derivatives, including $K' \triangleright (\nu \vec{d})\big((R \parallel a!b.P_4) \parallel P_2\big)$ and $K' \triangleright (\nu \vec{d})\big((R \parallel a!b.P_2) \parallel P_4\big)$, asynchronously output on any $a \in K'$. Thus, by Definition 13.1 and the two transitions above, we also have that:

  $$\begin{aligned} K' \models (\nu \vec{d})\big((R \parallel a!b.P_4) \parallel P_2\big) &\approx (\nu \vec{d})\big((R \parallel P_4) \parallel P_2\big) \parallel a!b.\mathbf{0} \quad \text{and} \\ K' \models (\nu \vec{d})\big((R \parallel a!b.P_2) \parallel P_4\big) &\approx (\nu \vec{d})\big((R \parallel P_2) \parallel P_4\big) \parallel a!b.\mathbf{0} \end{aligned}$$

  By transitivity/symmetry of $\approx$, we obtain

  $$K' \models (\nu \vec{d})\big((R \parallel a!b.P_4) \parallel P_2\big) \approx (\nu \vec{d})\big((R \parallel a!b.P_2) \parallel P_4\big)$$

  Our result, $K' \models Q \approx Q'$, follows using eq. (13.2) and Theorem 12.11.

For a more detailed proof, see Appendix J.1. □

In contrast, scope-extruding outputs, $a\uparrow b$, give neither determinism nor confluence guarantees. This is better illustrated in Example 13.6 below.

Example 13.6. Consider the fully asynchronous system $K \triangleright P_4$ where $K = \{a\}$ and channel name c is locally scoped twice in the running process P_4, defined below.

$$P_4 \stackrel{\text{def}}{=} (\nu c)(a!c.\mathbf{0} \parallel c?x.\mathbf{0}) \;\parallel\; (\nu c)(a!c.\mathbf{0} \parallel c?x.a!x.\mathbf{0})$$

The system $K \triangleright P_4$ can scope extrude name c by outputting it to channel name a in two possible ways using rules pPar and pOpn as follows:

$$K \triangleright P_4 \xrightarrow{a\uparrow c} K \cup \{c\} \triangleright (\mathbf{0} \parallel c?x.\mathbf{0}) \;\parallel\; (\nu c)(a!c.\mathbf{0} \parallel c?x.a!x.\mathbf{0}) \tag{13.3}$$

$$K \triangleright P_4 \xrightarrow{a\uparrow c} K \cup \{c\} \triangleright (\nu c)(a!c.\mathbf{0} \parallel c?x.\mathbf{0}) \;\parallel\; (\mathbf{0} \parallel c?x.a!x.\mathbf{0}) \tag{13.4}$$

The systems reached in (13.3) and (13.4) above are *not* bisimilar; that in (13.4) can produce the trace $c?b \cdot a!b$ but the system in (13.3) cannot. Particularly, once an external process learns of the new channel name c, it could interact with it by sending messages and subsequently observe different behaviour. In turn, this implies that $K \triangleright P_4$ is not deterministic w.r.t. action $a\uparrow c$. Moreover, we can show that this system is not confluent w.r.t. $a\uparrow c$ either; since the transitions in (13.3) and (13.4) scope extruded name c, the resulting systems cannot transition with $a\uparrow c$ again. ■

13.1.1 Syntactic Condition

Despite the merits of Definitions 13.1, 13.2 and 13.4, these are *semantic conditions*, which are, in general, undecidable. One way to guarantee that these conditions are satisfied is to restrict the systems in Chapter 12 to exclude output prefixing. This syntactic restriction is characterised by the predicate asy:Prc$\to$Bool in Definition 13.1, where $asy(P)$ returns true if every output construct in P is immediately followed by the inactive process, *i.e.,* $u!v.P' = u!v.\mathbf{0}$.

Definition 13.7. The predicate $asy(P) : \text{Prc} \mapsto \text{Bool}$ is inductively defined as follows:

$$asy(\mathbf{0}) = asy(X) \stackrel{\text{def}}{=} \text{true} \qquad asy(\text{if } u=v \text{ then } P \text{ else } Q) = asy(P \parallel Q) \stackrel{\text{def}}{=} asy(P) \wedge asy(Q)$$

$$asy(u!v.P) \stackrel{\text{def}}{=} \begin{cases} \text{true} & \text{if } P = \mathbf{0} \\ \text{false} & \text{otherwise} \end{cases} \qquad asy(a?x.P) = asy((\nu c)P) = asy(\text{rec}X.P) \stackrel{\text{def}}{=} asy(P)$$

■

Remark 13.8. The restriction on outputs described by Definition 13.7 is precisely that captured by the asynchronous variant of the π-calculus, which Honda *et al.*and Boudol showed to be still quite expressive [89, 43]. Our work differs from that the asynchronous π-calculus in that $asy(-)$ does not restrict the π-calculus at the grammar level. For instance, Boudol [43] defines the asynchronous π-calculus using the grammar

$$P,Q \in \text{Prc} ::= x(y).P \;\mid \bar{x}y \;\mid P \parallel Q \;\mid (\nu x)P \;\mid !P$$

where the constructs $x(y).P$, $\bar{x}y$ and $!P$ respectively denote input prefixing, output and process replication. Crucially, the output construct $\bar{x}y.P$ in the standard π-calculus (which corresponds to $u!v.P$ in Figure 12.1) is replaced by the construct $\bar{x}y$ to restrict outputs from having a continuation. In contrast, we retain the full π-calculus and identify a syntactic subset where outputs must be of the form $u!v.\mathbf{0}$. ■

Proposition 13.9 below shows that by removing output prefixing, we can guarantee asynchronous outputs on any free channel name.

Proposition 13.9. If $asy(P)$ then $K \triangleright P$ outputs asynchronously on any channel name $a \in K$.

Proof. Fix an interface $K \subseteq \text{Chans}$ and pick $a \in K$. Suppose also $asy(P)$. To prove that $K \triangleright P$ outputs asynchronous on channel a, we have to show that both conditions in Definition 13.1 are satisfied:

(i) if $K \triangleright P \xrightarrow{a!b} Q$ then $K \models P \approx Q \parallel a!b.\mathbf{0}$
(ii) if $K \triangleright P \xrightarrow{a\uparrow b} Q$ then $K \models P \approx (\nu b)(Q \parallel a!b.\mathbf{0})$

We only give the proof for (i); that for (ii) follows with a similar argument.
Suppose $K \triangleright P \xrightarrow{a!b} Q$. We can show that there exists $\vec{d} \subseteq \textsc{Chans}$ and $P_1, P_2 \in \textsc{Prc}$ such that $\vec{d} \sharp \{a,b\}$ and $P \equiv (\nu \vec{d})(P_1 \parallel a!b.P_2)$ and $Q \equiv (\nu \vec{d})(P_1 \parallel P_2)$. Since $asy(P)$, we can also show that $asy((\nu \vec{d})(P_1 \parallel a!b.P_2))$, which by Definition 13.7 implies that $P_2 = \mathbf{0}$. Using the facts that $P_2 = \mathbf{0}$ and $\vec{d} \sharp \{a,b\}$ and the structural equivalence rules in Definition 12.5, we obtain that

$$P \equiv \big((\nu \vec{d})P_1\big) \parallel a!b.\mathbf{0} \qquad \text{and} \qquad Q \equiv \big((\nu \vec{d})P_1\big) \parallel \mathbf{0}$$

Again using the structural equivalence rules in Definition 12.5 and symmetry, we thus get that

$$Q \parallel a!b.\mathbf{0} \;\equiv\; \big((\nu \vec{d})P_1\big) \parallel \mathbf{0} \parallel a!b.\mathbf{0} \;\equiv\; \big((\nu \vec{d})P_1\big) \parallel a!b.\mathbf{0} \;\equiv\; P$$

By transitivity/symmetry, this implies $P \equiv Q \parallel a!b.\mathbf{0}$. Our result, $K \models P \approx Q \parallel a!b.\mathbf{0}$, follows by Theorem 12.11. The omitted lemmata for this proof are given in Appendix J.1. □

Proposition 13.10 below shows that when the initial process does not contain any output prefixing, then this property is preserved by all system transitions.

Proposition 13.10 (Preservation). If $asy(P)$ and $K \triangleright P \xrightarrow{\eta} Q$ then $asy(Q)$.

Proof. Straightforward by rule induction on $K \triangleright P \xrightarrow{\eta} Q$. □

Proposition 13.11 below confirms that requiring $asy(P)$ suffices to guarantee full asynchrony.

Proposition 13.11. For all valid interfaces K, if $asy(P)$ then system $K \triangleright P$ is fully asynchronous.

Proof. Suppose that $asy(P)$. Following Definition 13.4, we show that for any trace $t \in \textsc{Act}^*$, if $K \triangleright P \overset{t}{\Longrightarrow} K' \triangleright Q$ then $K' \triangleright Q$ outputs asynchronously on any channel $a \in K'$.
The transition $K \triangleright P \overset{t}{\Longrightarrow} K' \triangleright Q$ can be expanded as follows, where $K_1 = K$, $K_n = K'$, $P_1 = P$ and $P_n = Q$.

$$K_1 \triangleright P_1 \xrightarrow{\eta_1} K_2 \triangleright P_2 \xrightarrow{\eta_2} \cdots \xrightarrow{\eta_n} K_n \triangleright P_n$$

The proof is by numerical induction on n.

- When $n = 0$, then $t = \epsilon$ and $P = P_1 = P_n = Q$. By $asy(P)$ and Proposition 13.9, we immediately conclude that $K \triangleright P$ outputs asynchronously on any $a \in K$.
- When $n = k+1$, then $K_1 \triangleright P_1 \xrightarrow{\eta_1} K_2 \triangleright P_2 \overset{t'}{\Longrightarrow} Q_n$ where t' has length k. By $asy(P_1)$ and Proposition 13.10, we deduce $asy(P_2)$. From $K_2 \triangleright P_2 \overset{t'}{\Longrightarrow} Q$ and the IH, we get that $K_n \triangleright P_n$ is fully asynchronous. Our result, $K' \triangleright Q$ outputs asynchronously on any $a \in K'$, follows by the facts that $K_n = K'$ and $P_n = Q$. □

The converse of Proposition 13.11 does not hold; this is better illustrated in the following example.

Example 13.12. The system $K \triangleright P_2$ from Example 13.3 is fully asynchronous but $\neg\, asy(P_2)$. Another example is $K' \triangleright P_5$ where $K' = \{a,b,d\}$ and $P_5 \overset{\text{def}}{=} a!b.\big((\nu c)c!d.\mathbf{0}\big)$. This system can output a on b via the transition $K' \triangleright P_5 \xrightarrow{a!b} (\nu c)c!d.\mathbf{0}$. Since the resulting system is blocked, *i.e.,* $K' \triangleright (\nu c)c!d.\mathbf{0} \overset{\eta}{\not\longrightarrow}$ for any $\eta \in \textsc{Act}$, it is bisimilar to the inactive system $K' \triangleright \mathbf{0}$, *i.e.,* $K' \models (\nu c)c!d.\mathbf{0} \approx \mathbf{0}$. In turn, this implies that $K' \models P_5 \approx (\nu c)c!d.\mathbf{0} \parallel a!b.\mathbf{0}$, which means that $K' \triangleright P_5$ is fully asynchronous, even though $\neg\, asy(P_5)$. ■

Since the asynchronous π-calculus has been extensively studied, for the remainder of this part of the thesis, we only consider processes that do not contain any output prefixing, *i.e.,* $asy(P)$ for all $P \in \textsc{Prc}$.

13.2 The Single Receiver Property

Prop. 2 implicitly states that, in actor-based systems, each message may be received by at *most* one actor. In our setting, this translates to requiring that each message destined to some channel a can be received by at most one (matching) input-prefixed process, $a?x.P$. However, in general, the LTSs of Chapter 12 do not observe this restriction: *any* process offering a matching input can receive the message, which may lead to multiple potential receivers. This is better illustrated by the following example.

Example 13.13. Consider process $P_6 \stackrel{\text{def}}{=} a?x.x!b.\mathbf{0} \parallel (\nu c)(a?y.y!c.\mathbf{0})$, which extends P_1 from Example 12.8. Given interface $K = \{a,b\}$, system $K \triangleright P_6$ can input some channel name d on a in two possible ways:

$$K \triangleright P_6 \xrightarrow{a?d} d!b.\mathbf{0} \parallel (\nu c)(a?x.x!c.\mathbf{0}) \qquad \text{(using rules pPar and pIn)}$$

$$K \triangleright P_6 \xrightarrow{a?d} a?x.x!b.\mathbf{0} \parallel (\nu c)(d!c.\mathbf{0}) \qquad \text{(using rules pStr, pPar and pIn)}$$

This violates Prop. 2, as multiple parallel processes, namely $a?x.x!b.\mathbf{0}$ and $(\nu c)(a?x.x!c.\mathbf{0})$, are simultaneously capable of accepting an input on channel name a. ■

We model Prop. 2 for systems $K \triangleright P$ from Figure 2.1 in two steps, namely Definitions 13.14 and 13.17. Concretely, Definition 13.14 below states that a system $K \triangleright P$ has *single receivers* on a *specific* channel a if *at most* one of its parallel components is capable of accepting an input on it.

Definition 13.14 (Single Receiver). A system $K \triangleright P$ has *single receivers* on channel name a whenever for all $P', P'' \in \text{Prc}$ and $\vec{d} \subseteq \text{Chans}$ such that $P \equiv (\nu \vec{d})(P' \parallel P'')$ and $\vec{d} \sharp a$,

$$\text{if } K \triangleright P' \xrightarrow{a?\_} Q \text{ then } K \triangleright P'' \xrightarrow{a?} \not\longrightarrow$$ ■

Remark 13.15. We observe that system $K \triangleright P$ can input on a channel name a if and only if that channel name occurs free in an input position, *i.e.*, $a \in \mathit{fIn}(P)$. As a result, $K \triangleright P$ vacuously has single receivers on channel a whenever $a \notin \mathit{fIn}(P)$ since $K \triangleright P' \xrightarrow{a?} \not\longrightarrow Q$. We also observe that whenever $P \equiv (\nu \vec{d})(P' \parallel P'')$ and $a \in \mathit{fIn}(P)$, then it implicitly follows that $\vec{d} \sharp a$. Otherwise, if $a \in \vec{d}$ then a, is by definition, bound in P, contradicting $a \in \mathit{fIn}(P)$. This means that the condition $\vec{d} \sharp a$ in Definition 13.14 could technically be dropped, but we opt to retain it for added clarity. ■

Example 13.16. Recall process $P_1 \stackrel{\text{def}}{=} (\nu c)(a?x.x!c.\mathbf{0})$ from Example 12.8. Given $K = \{a\}$, system $K \triangleright P_1$ trivially has single receivers on a. Concretely, the only way how process P_1 can be decomposed into two parallel sub-processes is when $P_1 \equiv (\nu c)\big(a?x.x!c.\mathbf{0} \parallel P_1'\big)$ where process P_1' is structurally equivalent to the inactive process, *i.e.*, $P_1' \equiv \mathbf{0}$. The first parallel sub-process $a?x.x!c.\mathbf{0}$ can input on a, but the second one cannot (since $P_1' \equiv \mathbf{0}$ and $\mathbf{0} \stackrel{\eta}{\not\longrightarrow}$ for any action η). We can also show that for $K' = \{a,b\}$ and process P_7 below, system $K' \triangleright P_7$ has single receivers on a.

$$P_7 \stackrel{\text{def}}{=} b?x.x?y.\mathbf{0} \parallel a?x.\mathbf{0}$$

Intuitively, this is because only the second parallel component of P_7 can input on channel name a, *i.e.*, $K \triangleright a?x.\mathbf{0} \xrightarrow{a?b} \mathbf{0}$ but $K \triangleright b?y.x?y.\mathbf{0} \xrightarrow{a?b} \not\longrightarrow$.

The same cannot be said for $K \triangleright P_6$ from Example 13.13; process P_6 contains two parallel sub-processes, namely $a?x.x!b.\mathbf{0}$ and $(\nu c)(a?y.y!c.\mathbf{0})$, that can both input on channel a. ■

Since Prop. 2 holds throughout the entire system execution, Definition 13.17 states that a system has *persistent single receivers* on channel name a if it satisfies the single receiver condition w.r.t. a (Definition 13.14) not just in the current state, but also in every state reachable from it. Put otherwise, at no point during execution can two parallel processes be simultaneously capable of inputting on a.

Definition 13.17 (Persistent Single Receiver). A system $K \triangleright P$ has *persistent single receivers* on channel name a whenever each derivative $K' \triangleright Q$ of $K \triangleright P$ has single receivers on channel name a. ■

Example 13.18. Using Definition 13.17, we can show $K \triangleright P_1$, $K \triangleright P_2$ and $K \triangleright P_4$ from Examples 13.3, 13.6 and 12.8 (restated below) have persistent single receivers on a, whereas $K \triangleright P_6$ from Example 13.13 does not.

$$P_1 \stackrel{\text{def}}{=} (\nu c)(a?x.x!c.\mathbf{0}) \qquad P_2 \stackrel{\text{def}}{=} \text{rec}X.a!b.X$$
$$P_4 \stackrel{\text{def}}{=} (\nu c)(a!c.\mathbf{0} \parallel c?x.\mathbf{0}) \;\parallel\; (\nu c)(a!c.\mathbf{0} \parallel c?x.a!x.\mathbf{0})$$

A trickier system is $K' \triangleright P_7$ from Example 13.16. Although we already showed that it has single receivers on channel a, they are *not* persistent. In particular, $K' \triangleright P_7$ can input a on b by transitioning as follows:

$$K' \triangleright P_7 \xrightarrow{b?a} \big(x?y.\mathbf{0} \parallel a?x.\mathbf{0}\big)[^a/_x] \;=\; a?y.\mathbf{0} \parallel a?x.\mathbf{0} \qquad \text{(using rules pIn and pParL)}$$

The resulting system contains two parallel processes, $a?y.\mathbf{0}$ and $a?x.\mathbf{0}$, that are capable of inputting on channel a, thereby violating the condition required for $K' \triangleright a?y.\mathbf{0} \parallel a?x.\mathbf{0}$ to have single receivers on a.

Another tricky system is $K \triangleright P_8$ where process P_8 below behaves like an identity that repeatedly waits for inputs on channel a and sends them back.

$$P_8 \stackrel{\text{def}}{=} \text{rec}X.(a?x.a!x.\mathbf{0} \parallel X)$$

Although only one of the parallel components of P_8, namely $a?x.a!x.\mathbf{0}$, is syntactically prefixed with an input on channel a, the sub-process X can also input on a after two recursive unfoldings. Concretely, system $K \triangleright P_8$ can transition with the execution sequence

$$K \triangleright P_8 \xrightarrow{\tau} K \triangleright a?x.a!x.\mathbf{0} \parallel P_8 \xrightarrow{\tau} a?x.a!x.\mathbf{0} \parallel (a?x.a!x.\mathbf{0} \parallel P_8) \qquad \text{(using rules pRec and pPar)}$$

and reach a system state that does *not* have single receivers on a, as both $a?x.a!x.\mathbf{0}$ and $a?x.a!x.\mathbf{0} \parallel P_8$ can immediately input on a, *i.e.*, $K \triangleright a?x.a!x.\mathbf{0} \xrightarrow{a?c} a!c.\mathbf{0}$ and $K \triangleright a?x.a!x.\mathbf{0} \parallel P_8 \xrightarrow{a?c} a!c.\mathbf{0} \parallel P_8$ for some channel c. In turn, this implies that $K \triangleright P_8$ does *not* have persistent single receivers on a. ■

For LTSs that satisfy the criteria in Definitions 13.14 and 13.17, we can establish several key determinism results. We start with Theorem 13.20 below, which states that having single receivers on a channel a ensures determinism w.r.t. inputs on that name. Its proof, which may be skipped upon first reading, relies on the stronger result in Proposition 13.19, showing that for such systems, all states reachable via an inputs on channel a are structurally equivalent.

Proposition 13.19 (Input Determinacy). Suppose $K \triangleright P$ has single receivers on a. If $K \triangleright P \xrightarrow{a?b} Q$ and $K \triangleright P' \xrightarrow{a?b} Q'$ where $P \equiv P'$ then $Q \equiv Q'$.

Proof Outline. Intuitively, this is because by Definition 13.14, exactly one parallel component of P is capable of inputting on a. The proof is by rule induction on the first move, namely $K \triangleright P \xrightarrow{a?b} Q$. The most interesting case is for rule pPar, outlined below:

- Case PPAR, *i.e.,* $P = P_1 \parallel P_2$ and $K \triangleright P_1 \xrightarrow{a?b} P_1'$ and $Q = P_1' \parallel P_2$. By the assumption that $K \triangleright P$ has single receivers on a, we know $K \triangleright P_2 \not\xrightarrow{a?b}$. For the second move, we can show that

$$\text{either } K \triangleright P_1 \xrightarrow{a?b} Q_1' \text{ and } Q' \equiv Q_1' \parallel P_2 \tag{13.5}$$

$$\text{or } K \triangleright P_2 \xrightarrow{a?b} Q_2 \text{ and } Q' \equiv P_1 \parallel Q_2 \tag{13.6}$$

  However, the move in (13.6) contradicts $K \triangleright P_2 \not\xrightarrow{a?b}$, which implies that (13.5) must hold. Thus, by the IH, we obtain $Q_1 \equiv Q_1'$. Our result, $Q \equiv Q'$, then follows by applying rule SCTXP from Definition 12.5.

For the complete proof, see Appendix J.1. □

Using Proposition 13.19, we can prove Theorem 13.20 below.

Theorem 13.20 (Input Determinacy). If a system $K \triangleright P$ has single receivers on channel name a, then $K \triangleright P$ is deterministic w.r.t. all input actions $a?b$.

Proof. Assume $K \triangleright P \xrightarrow{a?b} Q$ and $K \triangleright P \xrightarrow{a?b} Q'$. According to Definition 12.12, we must show that $\mathbf{aft}(K, a?b) \models Q \approx Q'$. This follows by Proposition 13.19, using also Theorem 12.11 and the fact that $P \equiv P$. □

This leads to the following corollary, which generalises the result to all reachable states, assuming that the initial system has persistent single receivers on a given channel name.

Corollary 13.21. If $K \triangleright P$ has persistent single receivers on channel name a, then each derivative of $K \triangleright P$ is deterministic w.r.t. all input actions $a?b$. ■

Building on Theorem 13.20, we can also show that whenever a system has single receivers on channel a, all states that can be reached after internally communicating some channel b to channel a, described by the (internal) action $com(a,b)$, are also bisimilar.

Theorem 13.22 (Internal Communication Determinacy). If $K \triangleright P$ has single receivers on channel name a, then $K \triangleright P$ is deterministic w.r.t. all internal communication actions $com(a,b)$.

Proof Outline. Suppose $K \triangleright P \xrightarrow{com(a,b)} Q$ and $K \triangleright P \xrightarrow{com(a,b)} Q'$. We must show $K \models Q \approx Q'$.
From the first move, $K \triangleright P \xrightarrow{com(a,b)} Q$, and our assumption that $asy(P)$ (see Section 13.1.1), we can show that there exist $R \in \text{PRC}$ such that

$$P \equiv R \parallel a!b.\mathbf{0} \quad \text{and} \quad K \triangleright R \xrightarrow{a?b} R' \quad \text{and} \quad Q \equiv R'$$

Similarly, from the second move, $K \triangleright P \xrightarrow{com(a,b)} Q'$, and $asy(P)$, we can show there exist $S \in \text{PRC}$ such that

$$P \equiv S \parallel a!b.\mathbf{0} \quad \text{and} \quad K \triangleright S \xrightarrow{a?b} S' \quad \text{and} \quad Q' \equiv S'$$

From $P \equiv R \parallel a!b.\mathbf{0}$ and $P \equiv S \parallel a!b.\mathbf{0}$ and transitivity, we know $R \parallel a!b.\mathbf{0} \equiv S \parallel a!b.\mathbf{0}$, from which we can show that $R \equiv S$. Thus, by rule STRN and $K \triangleright S \xrightarrow{a?b} S'$, we deduce $K \triangleright R \xrightarrow{a?b} S'$. Since $K \triangleright P$ has single receivers on a and $P \equiv S \parallel a!b.\mathbf{0}$, we can show $K \triangleright S$ also has single receivers on a. Using $K \triangleright R \xrightarrow{a?b} S'$

and $K \triangleright R \xrightarrow{a?b} R'$ and Theorem 13.20, we thus get $K \models R' \approx S'$. Since $Q \equiv R'$ and $Q' \equiv S'$, we conclude $K \models Q \approx Q'$.
For the detailed proof, refer to Appendix J.1. □

Theorem 13.22 can be generalised to all reachable states, assuming that the initial system has persistent single receivers on a given channel.

Corollary 13.23. If $K \triangleright P$ has persistent single receivers on channel name a, then each derivative of $K \triangleright P$ is deterministic w.r.t. all internal communication actions $com(a,b)$. ■

Although a system capable of inputting on a channel a can transition with input action $a?b$, the payload b depends on the observer's interaction with the system and *cannot* be controlled or predicted in advance. Example 13.24 below shows that requiring single receivers (resp. persistent single receivers) on a given channel does not give any determinism or confluence guarantees when input actions carry different payloads.

Example 13.24. Consider process P_9 below, which repeatedly waits for two consecutive inputs on channel name a and then replies by outputting the first channel name it received.

$$P_9 \stackrel{\text{def}}{=} \text{rec}X.a?x.a?y.(a!x.\mathbf{0} \parallel X)$$

Given $K = \{a\}$, we can show that although $K \triangleright P_9$ has persistent single receivers on channel a, the system reached after a recursive unfolding, *i.e.,* $K \triangleright P_9 \xrightarrow{\tau} a?x.a?y.(a!x.\mathbf{0} \parallel P_9)$, is not confluent w.r.t. input actions $a?b$ and $a?c$, with $b \neq c$. Concretely, $K \triangleright a?x.a?y.(a!x.\mathbf{0} \parallel P_9)$ can input either channel b or channel c on channel a using rules pStr, pPar and pIn, as follows:

$$K \triangleright a?x.a?y.(a!x.\mathbf{0} \parallel P_9) \xrightarrow{a?b} a?y.(a!b.\mathbf{0} \parallel P_9) \tag{13.7}$$

$$K \triangleright a?x.a?y.(a!x.\mathbf{0} \parallel P_9) \xrightarrow{a?c} a?y.(a!c.\mathbf{0} \parallel P_9) \tag{13.8}$$

When $b \neq c$, the resulting systems in (13.7) and (13.8) are clearly not bisimilar: they exhibit a different observational behaviour by outputting different payloads to channel a. Moreover, even though

$$\mathbf{aft}(K, a?b) \triangleright a?y.(a!b.\mathbf{0} \parallel P_9) \xrightarrow{a?c} a!b.\mathbf{0} \parallel P_9 \quad \text{and} \tag{13.9}$$

$$\mathbf{aft}(K, a?c) \triangleright a?y.(a!c.\mathbf{0} \parallel P_9) \xrightarrow{a?b} a!c.\mathbf{0} \parallel P_9 \tag{13.10}$$

the resulting systems in (13.9) and (13.10) are not bisimilar; again, they output different payloads to channel a. This means that the system $K \triangleright a?x.a?y.(a!x.\mathbf{0} \parallel P_9)$ is not confluent w.r.t. input actions $a?b$ and $a?c$, as neither of the conditions in Definition 12.13 are satisfied. ■

Similarly, requiring single receivers (resp. persistent single receivers) on a given channel does give any determinism or confluence guarantees for internal communication actions with the same subject but different objects, *i.e.,* actions such as $com(a,b)$ and $com(a,c)$ where $b \neq c$.

Example 13.25. Consider system $K \triangleright P_{10}$ where $K = \{a\}$ and process P_{10} is defined as follows, with $b \neq c$.

$$P_{10} \stackrel{\text{def}}{=} P_9 \parallel a!b.\mathbf{0} \parallel a!c.\mathbf{0} \qquad \text{where} \qquad P_9 \stackrel{\text{def}}{=} \text{rec}X.a?x.a?y.(a!x.\mathbf{0} \parallel X)$$

The system reached after a recursive unfolding, *i.e.,* $K \triangleright P_{10} \xrightarrow{\tau} P'_{10}$ where

$$P'_{10} \stackrel{\text{def}}{=} \big(a?x.a?y.(a!x.\mathbf{0} \parallel P_9)\big) \;\parallel\; a!b.\mathbf{0} \;\parallel\; a!c.\mathbf{0}$$

has persistent single receivers on a, but it is not confluent w.r.t. communication actions $com(a,b)$ and $com(a,c)$. In particular, the system $K \triangleright P'_{10}$ can *internally* receive either channel b or c on channel a via rules pIn and pComm as follows:

$$K \triangleright P'_{10} \xrightarrow{com(a,b)} a?y.(a!b.\mathbf{0} \parallel P_9) \;\parallel\; \mathbf{0} \;\parallel\; a!c.\mathbf{0} \tag{13.11}$$

$$K \triangleright P'_{10} \xrightarrow{com(a,b)} a?y.(a!c.\mathbf{0} \parallel P_9) \;\parallel\; a!c.\mathbf{0} \;\parallel\; \mathbf{0} \tag{13.12}$$

Since $b \neq c$, the systems reached in (13.11) and (13.12) exhibit different observational behaviour by outputting different payloads on channel a; the former outputs b, whereas the latter outputs c. More importantly, they can resp. internally receive channels c and b on channel a by transitioning as follows:

$$K \triangleright a?y.(a!b.\mathbf{0} \parallel P_9) \;\parallel\; \mathbf{0} \;\parallel\; a!c.\mathbf{0} \xrightarrow{com(a,c)} (a!b.\mathbf{0} \parallel P_9) \;\parallel\; \mathbf{0} \;\parallel\; \mathbf{0} \tag{13.13}$$

$$K \triangleright a?y.(a!c.\mathbf{0} \parallel P_9) \;\parallel\; a!c.\mathbf{0} \;\parallel\; \mathbf{0} \xrightarrow{com(a,b)} (a!c.\mathbf{0} \parallel P_9) \;\parallel\; \mathbf{0} \;\parallel\; \mathbf{0} \tag{13.14}$$

However, the states reached in (13.13) and (13.14) are clearly not bisimilar, as they output different payloads to channel a. This implies that $K \triangleright P'_{10}$ is not confluent w.r.t. the actions $com(a,b)$ and $com(a,c)$. ■

13.2.1 Syntactic Condition

To guarantee Definitions 13.14 and 13.17, we impose several *syntactic* restrictions on the processes of Figure 12.1. We start by formalising how a process is (statically) analysed for *disjoint receivers* on a given channel through a proof system. Its main judgement is $dr(P,a,V)$, *i.e.,* process P has disjoint receivers on channel a where V keeps track of the recursion variables that were unfolded during the analysis. This analysis is the least relation defined by the rules in Definition 13.26 below. Rules sRec1 and sRec2 use the set of visited recursion variables V to unfold recursive processes $\mathrm{rec}X.P$ only if this process was not already analysed before, *i.e.,* $X \notin V$; this is justified by Proposition 13.28, to be discussed later. The only other non-straightforward rule is sPar. It states that parallel processes $P \parallel Q$ have disjoint receivers on a, *i.e.,* $dr(P \parallel Q,a,V)$, whenever channel a is not free in an input position in both P and Q, *i.e.,* $a \in fIn(P) \cap fIn(Q)$, and both P and Q have disjoint receivers on a, *i.e.,* $dr(P,a,V)$ and $dr(Q,a,V)$.

Definition 13.26 (Disjoint Receivers). The judgement $dr(P,a,V)$ is defined inductively as the least relation included in $(\text{Prc} \times \text{Chans} \times \mathcal{P}(\text{TVars}))$ satisfying the rules

$$\text{sNil}\ \frac{}{dr(\mathbf{0},a,V)} \qquad \text{sTVar}\ \frac{}{dr(X,a,V)} \qquad \text{sOut}\ \frac{dr(P,a,V)}{dr(u!v.P,a,V)} \qquad \text{sIn}\ \frac{dr(P,a,V)}{dr(u?v.P,a,V)} \qquad \text{sScp}\ \frac{dr(P,a,V) \quad a \neq c}{dr((\nu c)P,a,V)}$$

$$\text{sCmp}\ \frac{dr(P,a,V) \quad dr(P,a,V)}{dr(\text{if } u=v \text{ then } P \text{ else } Q,a,V)} \qquad \text{sPar}\ \frac{a \notin fIn(P) \cap fIn(Q) \quad dr(P,a,V) \quad dr(Q,a,V)}{dr(P \parallel Q,a,V)} \qquad \text{sRec1}\ \frac{X \in V \quad dr(P,a,V)}{dr(\mathrm{rec}X.P,a,V)}$$

$$\text{sRec2}\ \frac{X \notin V \quad dr(P[\mathrm{rec}X.P/X],a,V \cup \{X\})}{dr(\mathrm{rec}X.P,a,V)}$$

As a shorthand, we write $dr(P,a)$ to denote $dr(P,a,\emptyset)$ and read it "*P has disjoint receivers on a*". ■

Example 13.27. Consider again process $P_9 \stackrel{\text{def}}{=} \text{rec}X.a?x.a?y.(a!x.\mathbf{0} \,\|\, X)$ from Example 13.24. The derivation below shows that P_9 has disjoint receivers on channel name a, *i.e.,* $dr(P_9,a)$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{dr(\mathbf{0},a,\{X\})}\,\text{sNL}}{dr(a!x.\mathbf{0},a,\{X\})}\,\text{sOut} \quad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{dr(\mathbf{0},a,\{X\})}\,\text{sNL}}{dr(a!x.\mathbf{0},a,\{X\})}\,\text{sOut} \quad \dfrac{}{dr(X,a,\{X\})}\,\text{sTVar}}{dr(a!x.\mathbf{0} \,\|\, X,a,\{X\})}\,\text{sPar}}{dr(a?y.(a!x.\mathbf{0} \,\|\, X),a,\{X\})}\,\text{sIn}}{dr(a?x.a?y.(a!x.\mathbf{0} \,\|\, X),a,\{X\})}\,\text{sIn}}{dr(P_9,a,\{X\})}\,\text{sRec1}}{dr(a!x.\mathbf{0} \,\|\, P_9,a,\{X\})}\,\text{sPar}}{dr(a?y.(a!x.\mathbf{0} \,\|\, P_9),a,\{X\})}\,\text{sIn}}{dr(a?x.a?y.(a!x.\mathbf{0} \,\|\, P_9),a,\{X\})}\,\text{sIn}}{dr(P_9,a,\emptyset)}\,\text{sRec2}$$

The same cannot be said for process $P_7 \stackrel{\text{def}}{=} a?x.x!b.\mathbf{0} \,\|\, (\nu c)(a?x.x!c.\mathbf{0})$ from Example 13.13. Concretely, rule sPar from Definition 13.26 cannot be used to justify $dr(P_7,a)$ since we have that $a \in fIn(a?x.x!b.\mathbf{0})$ and $a \in fIn((\nu c)(a?x.x!c.\mathbf{0}))$. This means that P_7 does *not* have disjoint receivers on a, *i.e.,* $\neg\, dr(P_7,a)$. ■

Proposition 13.28 below, previewed earlier, asserts that analysing a single unfolding of a recursive process $\text{rec}X.P$ is sufficient to determine whether it has disjoint receivers on a specific channel. The underlying idea is that after unfolding once, we can safely exclude the recursion variable X from the set of recursion variables that were analysed. Although further unfoldings may occur, they do not introduce new structural information that could violate the disjoint receptiveness of process $\text{rec}X.P$ on channel a.

Proposition 13.28. If $dr(P[^{\text{rec}X.P}/_X],a,V \uplus \{X\})$ then $dr(P[^{\text{rec}X.P}/_X],a,V)$.

Proof Outline. By rule induction on $dr(P[^{\text{rec}X.P}/_X],a,V \cup \{X\})$. For the full proof, see Appendix J.2. □

For a process P with disjoint receivers on some channel a, *i.e.,* $dr(P,a)$, we can show that for any valid interface K, the system $K \triangleright P$ has single receivers on that channel.

Proposition 13.29. For any valid interface $K \subseteq \text{Chans}$, if $dr(P,a)$ then $K \triangleright P$ has single receivers on a.

Proof Outline. The proof is by rule induction on $dr(P,a)$. The intuition is that if $P \equiv (\nu \vec{d})(P' \,\|\, P'')$ and $K \triangleright P' \xrightarrow{a?\_} Q$, channel a must occur free in an input position, *i.e.,* $a \in fIn(P')$. By the assumptions $P \equiv (\nu \vec{d})(P' \,\|\, P'')$ and $dr(P,a)$, we can also show $dr((\nu \vec{d})(P' \,\|\, P''),a)$. Using rule sPar, this gives us that $a \notin fIn(P') \cap fIn(P'')$. But since $a \in fIn(P')$ we must have $a \notin fIn(P'')$, which implies $K \triangleright P'' \not\xrightarrow{a?\_}$. This means $K \triangleright P$ has single receivers on a. □

Definition 13.26 and Proposition 13.29 can be better understood through the example below.

Example 13.30. Example 13.27 showed that $dr(P_9,a)$. Accordingly, system $K \triangleright P_9$ with $K = \{a\}$ has single receivers on a. In comparison, the system $K' \triangleright P_7$ with $K' = \{a,b\}$ does *not* have single receivers on a (see Example 13.16). Accordingly, we have that $\neg\, dr(P_7,a)$ as shown in Example 13.27. ■

The converse of Proposition 13.29 does not hold. Intuitively, this happens because the judgement $dr(-)$ rules out parallel processes that have common free names in input positions, even if those names are inactive, *i.e.,* not ready to receive inputs, as illustrated in Example 13.31. While requiring disjoint receivers might appear too restrictive, it aligns with the usage of names in actor system, where each (parallel) actor is identified by a unique ID, which is used by other actors to send messages to it, formalised as Prop. 2.

Example 13.31. Recall $P_8 \stackrel{\text{def}}{=} \mathrm{rec}X.(a?x.a!x.\mathbf{0} \parallel X)$ from Example 13.18. When $K = \{a\}$, system $K \triangleright P_8$ has single receivers on a, as it can only input on a after a recursive unfolding. However, it does *not* have disjoint receivers on that channel, *i.e.,* $\neg\, dr(P_8, a)$. Concretely, after applying rule sRec2, there is no rule from Definition 13.26 to justify $dr(a?x.a!x.\mathbf{0} \parallel P_8, a, \{X\})$ since $a \in fIn(a?x.a!x.\mathbf{0})$ but also $a \in fIn(P_8)$. ■

In order to guarantee persistent single receivers, Definition 13.26 alone does not suffice. In particular, as already alluded to in Example 13.18, our LTSs also permit transmitted channel names to be later used by the recipient in an input position. While this mechanism is harmless when ensuring single receivers on specific channels, it should not be used indiscriminately when the goal is *persistent* single receivers.

Example 13.32. Recall process $P_7 \stackrel{\text{def}}{=} b?x.x?y.\mathbf{0} \parallel a?x.\mathbf{0}$ from Example 13.16. Example 13.18 showed that for $K' = \{a, b\}$, system $K' \triangleright P_7$ does not have persistent single receivers on a; after inputting a on b, the resulting state, $K' \triangleright a?y.\mathbf{0} \parallel a?x.\mathbf{0}$, violates the single receiver condition for a. Yet, a static analysis of P_7 indicates that $dr(P_7, a)$. This shows that disjoint receivers of the initial process alone do not ensure persistent single receivers, as Definition 13.26 does not account for how the system evolves with input actions, particularly when the received channel name is later used in an input position. ■

We circumvent the complications described in Example 13.32 by also demanding that received channel names can be used in output positions but *not* in input positions. To enforce this *locality* constraint, we rely on the predicate $loc(-)$ in Definition 13.33, which characterises the *local* (asynchronous) π-calculus of [104]. Notably, requiring locality rules out problematic processes like P_7 from Example 13.16; see Example 13.34 below.

Definition 13.33 (Locality). The predicate $loc(P, V) : \textsc{Prc} \times \mathcal{P}(\textsc{PVars}) \mapsto \textsc{Bool}$ is defined as follows:

$$
\begin{aligned}
&loc(\mathbf{0}, V) = loc(X, V) \stackrel{\text{def}}{=} \mathsf{true} \\
&loc(u?x.P, V) \stackrel{\text{def}}{=} u \notin V \wedge loc(P, V \cup \{x\}) \\
&loc(\mathsf{if}\, u = v\, \mathsf{then}\, P\, \mathsf{else}\, Q, V) = loc(P \parallel Q, V) \stackrel{\text{def}}{=} loc(P, V) \wedge loc(Q, V) \\
&loc(u!v.P, V) = loc((\nu c)P, V) = loc(\mathrm{rec}X.P, V) \stackrel{\text{def}}{=} loc(P, V)
\end{aligned}
$$

As a shorthand, we write $loc(P)$ whenever $loc(P, \emptyset)$. ■

Example 13.34. Recall process $P_7 \stackrel{\text{def}}{=} b?x.x?y.\mathbf{0} \parallel a?x.\mathbf{0}$ from Example 13.16. Using Definition 13.33, we can show that $\neg\, loc(P_7)$. Assume, by way of contradiction, that $loc(P_7)$. By definition, then we must also have $loc(b?x.x?y.\mathbf{0})$ and $loc(a?x.\mathbf{0})$. The former implies that $loc(x?y.\mathbf{0}, \{x\})$ but by definition we know $\neg\, loc(x?y.\mathbf{0}, \{x\})$ since $x \in \{x\}$. This gives us a contradiction, which means that $\neg\, loc(P_7)$. ■

Proposition 13.35 below shows that when the initial process is both local, *i.e.,* $loc(P)$, and has disjoint receivers on a, *i.e.,* $dr(P,a)$, then these two properties are preserved under all system transitions.

Proposition 13.35 (Preservation). Suppose $loc(P)$ and $dr(P,a)$. If $K \triangleright P \xrightarrow{\eta} Q$ then $loc(Q)$ and $dr(Q,a)$.

Proof Outline. By induction on the inference of $K \triangleright P \xrightarrow{\eta} Q$. We only outline the proof for rule PIN.

- Case PIN, *i.e.,* $K \triangleright b?x.P \xrightarrow{b?c} P[^c/_x]$ for some $b,c \in$ CHANS, possibly equal to a.
  From the assumption $loc(b?x.P)$, we can show $P[^c/_x]$ satisfies the locality constraint, *i.e.,* $loc(P[^c/_x])$, since substitution in *local* processes does *not* affect input positions.
  Showing $dr(P[^c/_x],a)$ is slightly more involved. From the assumption $dr(b?x.P,a)$ and rule SIN, we deduce $dr(P,a)$. But since $loc(P)$, process P does not contain any (sub-)processes of the form $x?a.P'$. This means that substituting the free occurrences of x by c does *not* affect the set of (sub-)processes in P that can input on a. Thus, we can conclude $dr(P[^b/_x],a)$.

For the complete proof, see Appendix J.2. □

Proposition 13.36 below shows that if a process P has disjoint receivers on a specific channel name a (Definition 13.26) and satisfies the locality constraint (Definition 13.33), then for a valid interface K, systems $K \triangleright P$ have persistent single receivers on that channel.

Proposition 13.36 (Persistent Single Receiver). For all valid interfaces $K \subseteq$ CHANS, if $loc(P)$ and $dr(P,a)$ then system $K \triangleright P$ has persistent single receivers on a.

Proof. Similar to that for Proposition 13.11, using Proposition 13.29, 13.35 instead of Proposition 13.9, 13.10. □

13.3 The Persistent Receptiveness Property

The *persistent-receptiveness property* in Prop. 3 asserts that actors are always available for communication. We model this property by specifying the conditions under which certain channel names are ready to input at any time during the system execution. We start with Definition 13.37 below, stating that a system is *receptive* on a given channel name a if it is capable of inputting on that channel after *any* number of silent transitions. This ensures that any message sent at a can be immediately processed.

Definition 13.37 (Receptiveness). A system $K \triangleright P$ is *receptive* on channel name a if there exists processes $Q,Q' \in$ PRC such that $K \triangleright P \Rightarrow K \triangleright Q' \xrightarrow{a?\_} Q$. ■

Intuitively, a system $K \triangleright P$ is receptive on channel name a, according to Definition 13.37, if *at least one* of the parallel components of process P is capable of inputting of a. Note that, this does not imply single receivers on a (Definition 13.14) as a receptive system may contain multiple processes ready to input on it; this is better illustrated by $K' \triangleright P_6$ in Example 13.38 below.

Example 13.38. Recall processes P_1, P_6, P_7 and P_9 below from Examples 12.8, 13.13, 13.16 and 13.24 respectively.

$$P_1 \stackrel{\text{def}}{=} (\nu c)(a?x.x!c.\mathbf{0}) \qquad P_6 \stackrel{\text{def}}{=} a?x.x!b.\mathbf{0} \parallel (\nu c)(a?y.y!c.\mathbf{0})$$

$$P_7 \stackrel{\text{def}}{=} b?x.x?y.\mathbf{0} \parallel a?x.\mathbf{0} \qquad P_9 \stackrel{\text{def}}{=} \text{rec}X.a?x.a?y.(a!x.\mathbf{0} \parallel X)$$

For $K = \{a\}$ and $K' = \{a,b\}$, we can show that systems $K \triangleright P_1$ and $K' \triangleright P_7$ are receptive on channel a as they can immediately input on that channel, without requiring any silent transitions:

$$K \triangleright P_1 \xrightarrow{a?b} (\nu c)(b!c.\mathbf{0}) \qquad \text{(using rule pIn)}$$

$$K' \triangleright P_7 \xrightarrow{a?b} b?x.x?y.\mathbf{0} \parallel \mathbf{0} \qquad \text{(using rules pIn, pPar)}$$

Similarly, we can show $K' \triangleright P_6$ is receptive on a as it can input on that channel in either of two ways (see Example 13.13). Although $K \triangleright P_9$ cannot immediately input on a, it can do so after a recursive unfolding:

$$K \triangleright P_9 \xrightarrow{\tau} K \triangleright a?x.a?y.(a!x.\mathbf{0} \parallel P_9) \xrightarrow{a?b} a?y.(a!b.\mathbf{0} \parallel P_9) \qquad \text{(using rules pRec, pIn)}$$

This means $K \triangleright P_9$ is also receptive on channel a. The same cannot be said for the system $K' \triangleright P_{11}$ where $P_{11} \stackrel{\text{def}}{=} b?x.a?y.\mathbf{0}$ since it is blocked from inputting on channel a until it first inputs on channel b. ■

Building on Definition 13.37, we formalise what it means for receptiveness on a given channel name to be *persistent* throughout the entire system execution. This persistence is crucial whenever *unboundedly many* messages could be sent to that channel. Definition 13.39 follows closely Sangiorgi's terminology [120] for ω-receptiveness and states a channel name is *persistently receptive* if it is *always* ready to input on that channel. However, unlike the notion of ω-receptiveness in [120], our definition for persistent receptiveness is not of the "uniform" kind; it does not require all inputs on a given channel a to have the same continuation (*i.e.,* the input end of a is "functional"). Instead, Definition 13.39 allows for varying continuations that depend on the current system state; see $K \triangleright P_9$ in Example 13.40 below.

Definition 13.39 (Persistent Receptiveness). A system $K \triangleright P$ is *persistently receptive* on channel name a whenever each derivative $K' \triangleright Q$ of $K \triangleright P$ is receptive on channel name a. ■

Example 13.40. Consider again P_9 from Example 13.24. To show that $K \triangleright P_9$ is persistently receptive on channel a according to Definition 13.39, we must show that all its derivatives are receptive on that channel. Example 13.38 already showed that this is indeed the case for the initial system, $K \triangleright P_9$, and its τ-derivative, $K \triangleright a?x.a?y.(a!x.\mathbf{0} \parallel P_9)$. The latter system can input any $b \in \text{Chans}$ on channel a, and evolve to $K' \triangleright a?y.(a!b.\mathbf{0} \parallel P_9)$ where $K' = K \cup \{b\}$, which is clearly receptive on a. Similarly, the resulting system can input any $c \in \text{Chans}$ on channel a and reach $K'' \triangleright a!b.\mathbf{0} \parallel P_9$ where $K'' = K' \cup \{c\}$. Continuing in this manner, it is not hard to see that there is *always* one process available for input on channel a.

In comparison, although Example 13.38 showed that system $K \triangleright P_1$ is receptive on channel a, this receptiveness is *not* persistent. Concretely, after inputting some channel b on channel a via the transition $K \triangleright P_1 \xrightarrow{a?b} (\nu c)(b!c.\mathbf{0})$, the system loses the ability to accept inputs on channel a, *i.e.,* $\mathbf{aft}(K, a?b) \triangleright (\nu c)(b!c.\mathbf{0}) \xrightarrow{a?} \not\longrightarrow$ and $\mathbf{aft}(K, a?b) \triangleright (\nu c)(b!c.\mathbf{0}) \stackrel{\tau}{\not\longrightarrow}$. A similar argument applies for systems $K' \triangleright P_6$ and $K' \triangleright P_7$ where $K' = \{a,b\}$, which were shown to be receptive on channel a in Example 13.38. ■

Since we have already shown that Definitions 13.4 and 13.17 guarantee determinism w.r.t. input actions, output actions that do not involve bound names, and internal communication actions, we do not revisit these actions. Instead, we only focus on input and internal communication actions with the *same* recipient but a *different* payload, *i.e.,* input actions $a?b$ and $a?c$ and communication actions $com(a,c)$ and

$com(a,c)$ where $b \neq c$. It turns out while Definitions 13.37 and 13.39 guarantee Prop. 3, they are too weak to provide any determinism or confluence guarantees for these kinds of actions.

Example 13.41. Recall processes P_8 and P_{10} from Examples 13.18 and 13.25, restated below.

$$P_8 \stackrel{\text{def}}{=} \text{rec}X.(a?x.a!x.\mathbf{0} \parallel X)$$
$$P_{10} \stackrel{\text{def}}{=} \left(\text{rec}X.a?x.a?y.(a!x.\mathbf{0} \parallel X)\right) \parallel a!b.\mathbf{0} \parallel a!c.\mathbf{0}$$

Although $K \triangleright P_8$ with $K = \{a\}$ is persistently receptive on channel name a (Example 13.38), it is not confluent w.r.t. input actions $a?b$ and $a?c$, where $b \neq c$ (Example 13.18). Similarly, Example 13.25 demonstrated that $K \triangleright P_{10}$, which is persistently receptive on channel name a, is not confluent w.r.t. communication actions $com(a,b)$ and $com(a,c)$ with $b \neq c$. ■

We strengthen Definition 13.37 by not only requiring receptiveness on given channel names a, but also some form of structured response to inputs on those channels. One approach would be to follow Sangiorgi's notion of uniformity [120] and require all inputs on a given channel name to have the *same* continuation. Another way would be relax to this requirement by only demanding that a portion of the resulting system remains behaviourally equivalent (up to bisimilarity) to the initial one; this allows for more flexibility in how the system evolves, while ensuring that the core behaviour of the system is preserved. Definition 13.42 follows the latter approach. Concretely, the first part of (ii), namely $P \equiv (\nu \vec{d})P'$, states that process P may contain a number of restricted channel names $\vec{d}$. The additional requirement, $P' \not\equiv (\nu c)P''$ for all $P'' \in \text{Prc}$, ensures that there are no additional top-level restrictions hidden inside the internal structure of P'. This allows us to reason directly about the behaviour of P' inside the current scope, $(\nu \vec{d})P'$. The final part of (ii) asserts that whenever P' inputs on channel a, *i.e.,* $K' \triangleright P' \Longrightarrow K' \triangleright Q \xrightarrow{a?b} Q'$ where $K' = K \cup \{\vec{d}\}$, the resulting process Q' can be decomposed into two parallel components, *i.e.,* $Q' \equiv Q_1 \parallel Q_2$. Crucially, one of these components, say Q_1, must behave in the same way as the original process before the input *i.e.,* $\mathbf{aft}(K', a?b) \models Q_1 \approx P'$. This means that even though the system has processed an input on channel a, a part of it continues behaving as if nothing has changed.

Definition 13.42 (Strong Receptiveness). A system $K \triangleright P$ is *strongly receptive* on channel name a whenever the following conditions hold:

(i) System $K \triangleright P$ is receptive on a.
(ii) Suppose $P \equiv (\nu \vec{d})P'$ such that $P' \not\equiv (\nu c)P''$ for all $P'' \in \text{Prc}$. If $K' \triangleright P' \Longrightarrow K' \triangleright Q \xrightarrow{a?b} Q'$ where $K' = K, \vec{d}$ then $\exists Q_1, Q_2 \in \text{Prc}$ such that $Q' \equiv Q_1 \parallel Q_2$ and $\mathbf{aft}(K', a?b) \models Q_1 \approx P'$. ■

Definition 13.42 is better illustrated in the following example.

Example 13.43. Consider process P_{12} below, a slight modification of P_8 from Example 13.18.

$$P_{12} \stackrel{\text{def}}{=} \text{rec}X.a?x.(a!x.\mathbf{0} \parallel X)$$

When $K = \{a\}$, we can show that $K \triangleright P_{12}$ is strongly receptive on channel a. Concretely, it is not hard to see that this system is receptive on a, satisfying (i) of Definition 13.42. After inputting some channel b on channel a, system $K \triangleright P_{12}$ evolves to $a!b.\mathbf{0} \parallel P_{12}$, where the second parallel component, P_{12}, is clearly

bisimilar to the initial process. This means that $K \triangleright P_{12}$ also satisfies (ii) of Definition 13.42, which allows us to conclude that it is strongly receptive on a.

In comparison, system $K \triangleright P_9$ where $P_9 \stackrel{\text{def}}{=} \text{rec}X.a?x.a?y.(a!x.\mathbf{0} \parallel X)$ (from Example 13.24) is *not* strongly receptive on channel name a; although it is receptive on channel a, it does not satisfy condition (ii) of Definition 13.42. Specifically, after inputting some channel b on a, it evolves to $\mathbf{aft}(K,a?b) \triangleright a?y.(a!b.\mathbf{0} \parallel P_9)$. This system only contains one parallel process, which does not behave the same as the original one; the process $a?y.(a!b.\mathbf{0} \parallel P_9)$ can output b on a after receiving another input on a, but P_9 cannot. ■

Definition 13.44 builds on Definition 13.42. It states that a system is *persistently strongly receptive* on channel name a if it is satisfies the strong receptiveness property w.r.t. that channel name not just in the current state, but also in every state reachable from it.

Definition 13.44 (Persistent Strong Receptiveness). A system $K \triangleright P$ is *persistently strongly receptive* on channel name a whenever each derivative of $K \triangleright P$ is strongly receptive on channel name a. ■

Example 13.45. Using Definition 13.44, we can show that for $K = \{a\}$, system $K \triangleright P_{12}$ from Example 13.43 is persistently strongly receptive on channel a. The same cannot be said for $K \triangleright P_{13}$ where

$$P_{13} \stackrel{\text{def}}{=} \text{rec}X.a?x.\big(x?y.a!y.\mathbf{0} \parallel X\big)$$

This system can input channel a by transitioning with rules pRec and pIn as follows:

$$\begin{aligned} K \triangleright \text{rec}X.a?x.\big(x?y.a!y.\mathbf{0} \parallel X\big) &\xrightarrow{\tau} K \triangleright a?x.\big(x?y.a!y.\mathbf{0} \parallel P_{13}\big) \\ &\xrightarrow{a?a} a?y.a!y.\mathbf{0} \parallel P_{13} \end{aligned} \tag{13.15}$$

The system in (13.15) is *not* strongly receptive on channel a. In particular, it can input channel b on channel a using rules pIn and pPar as follows:

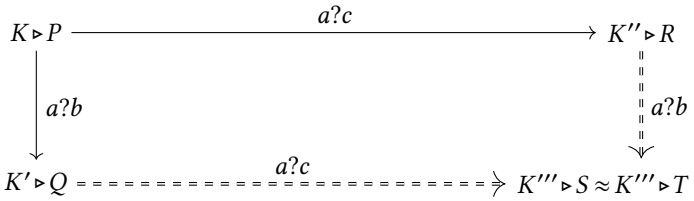
$$K \triangleright a?y.a!y.\mathbf{0} \parallel P_{13} \xrightarrow{a?b} a!b.\mathbf{0} \parallel P_{13}$$

The resulting system does *not* contain a sub-process that is bisimilar to the system before accepting the input; the latter, $a?y.a!y.\mathbf{0} \parallel P_{13}$, can input and then immedaitely output on a, whereas neither $a!b.\mathbf{0}$, P_{13} nor $a!b.\mathbf{0} \parallel P_{13}$ can exhbit such behaviour. This violates condition (ii) of Definition 13.42, which, in turn, means that the intial system $K \triangleright P_{13}$ is not persistently strongly receptive on channel a. ■

For LTSs that satisfy the properties in Definitions 13.42 and 13.44, we can establish several key confluence results. We start with Theorem 13.46, stating that strong receptiveness on a given channel name guarantees confluence for input actions with the same subject but a different object.

Theorem 13.46 (Input Confluence). If a system $K \triangleright P$ is *strongly receptive* on channel a, then that system is confluent w.r.t. input actions $a?b$ and $a?c$ where $b \neq c$.

Proof Outline. Suppose $K \triangleright P$ is strongly receptive on channel a and $K \triangleright P \xrightarrow{a?b} K' \triangleright Q$ and $K \triangleright P \xrightarrow{a?c} K'' \triangleright R$. We have to complete the following diagram, where $K''' = K \cup \{b,c\}$, the solid lines indicate the moves that are given and the dotted lines are those that are required:



Assume $P \equiv (\nu \vec{d})P'$ and $P' \not\equiv (\nu c)P''$ for any c and P''. Since P is strongly receptive on a, then we know that both $K' \triangleright Q$ and $K'' \triangleright R$ contain sub-processes that are bisimilar to the initial process P', that is:

$$Q \equiv (\nu \vec{d})(Q_1 \parallel Q_2) \quad \text{and} \quad K', \vec{d} \models Q_1 \approx P' \quad \text{and} \quad R \equiv (\nu \vec{d})(R_1 \parallel R_2) \quad \text{and} \quad K'', \vec{d} \models R_1 \approx P'$$

From this, we can show that $K' \triangleright Q$ and $K'' \triangleright R$ can input the channel name that the other inputted, *i.e.,* the former inputs c on a whereas the latter inputs b on a, as follows:

$$K' \triangleright Q \overset{a?b}{\Longrightarrow} K''' \triangleright (\nu \vec{d})(Q_1' \parallel Q_2) \quad \text{where} \quad K''', \vec{d} \models Q_1' \approx R_1 \parallel R_2 \tag{13.16}$$

$$K'' \triangleright R \overset{a?b}{\Longrightarrow} K''' \triangleright (\nu \vec{d})(R_1' \parallel R_2) \quad \text{where} \quad K''', \vec{d} \models R_1' \approx Q_1 \parallel Q_2 \tag{13.17}$$

There only remains to show that systems reached in eqs. (13.16) and (13.17) are bisimilar. From the results above, we know that

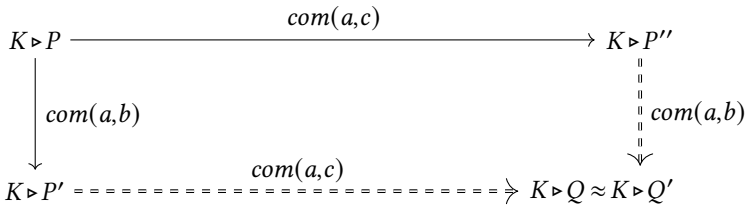
$$K''', \vec{d} \models Q_1' \parallel Q_2 \approx (R_1 \parallel R_2) \parallel Q_2 \approx (P' \parallel R_2) \parallel Q_2$$
$$K''', \vec{d} \models R_1' \parallel R_2 \approx (Q_1 \parallel Q_2) \parallel R_2 \approx (P' \parallel Q_2) \parallel R_2$$

which, by transitivity and symmetry, implies that $K''', \vec{d} \models Q_1' \parallel Q_2 \approx R_1' \parallel R_2$. From this, we can then show that $K''' \models (\nu \vec{d})(Q_1' \parallel Q_2) \approx (\nu \vec{d})(R_1' \parallel R_2)$ which gives us the required result.
The interested reader should consult Appendix J.3 for the complete proof. □

The second main result of this section, formalised in Theorem 13.47 below, builds on Theorem 13.46 and states that strong receptiveness on a given channel also guarantees confluence for communication actions with the same subject but a different object.

Theorem 13.47 (Communication Confluence). If a system $K \triangleright P$ is *strongly receptive* on channel a, then that system is confluent w.r.t. communication actions $com(a,b)$ and $com(a,c)$ where $b \neq c$.

Proof Outline. Suppose $K \triangleright P$ is strongly receptive on channel a and $K \triangleright P \xrightarrow{com(a,b)} P'$ and $K \triangleright P \xrightarrow{com(a,b)} P''$. We have to complete the following diagram, where the solid lines indicate the moves that are given and the dotted lines are those that are required.



From the given moves and the underlying assumption that $asy(P)$ in Section 13.1.1, we can show that there exists a process R such that

$$P \equiv R \parallel a!b.\mathbf{0} \parallel a!c.\mathbf{0} \text{ and } K \triangleright R \xrightarrow{a?b} R' \text{ and } K \triangleright R \xrightarrow{a?c} R'' \\ \text{where } P' \equiv R' \parallel a!c.\mathbf{0} \text{ and } P'' \equiv R'' \parallel a!b.\mathbf{0} \tag{13.18}$$

Using the rules in Figure 12.1, we can obtain the moves

$$K \triangleright P \xrightarrow{a?b} R' \parallel a!b.\mathbf{0} \parallel a!c.\mathbf{0} \quad \text{and} \quad K \triangleright P \xrightarrow{a?c} R'' \parallel a!b.\mathbf{0} \parallel a!c.\mathbf{0}$$

From these moves, together with the fact that $K \triangleright P$ is strongly receptive on channel name a and Theorem 13.46, we know there exist processes S, S' such that

$$\begin{array}{ccc} K \triangleright P & \xrightarrow{\quad a?c \quad} & K \triangleright R'' \parallel a!b.\mathbf{0} \parallel a!c.\mathbf{0} \\ \Big\downarrow a?b & & \Big\downarrow a?b \\ K \triangleright R' \parallel a!b.\mathbf{0} \parallel a!c.\mathbf{0} & \xRightarrow{\quad a?c \quad} & K \triangleright S \approx K \triangleright S' \end{array}$$

From these moves, we show there exists a process Q such that $K \triangleright R' \overset{a?c}{\Longrightarrow} Q$ and $S = Q \parallel a!b.\mathbf{0} \parallel a!c.\mathbf{0}$. From $K \triangleright R' \overset{a?c}{\Longrightarrow} Q$ we can also show $K \triangleright R' \parallel a!c.\mathbf{0} \overset{com(a,c)}{\Longrightarrow} Q$. Using the fact that $P' \equiv R' \parallel a!c.\mathbf{0}$ in eq. (13.18) and rule PSTR gives us the first required move, $K \triangleright P' \overset{com(a,c)}{\Longrightarrow} Q$.
Similarly, we can obtain the second required move, $K \triangleright P'' \overset{com(a,b)}{\Longrightarrow} Q'$ where $S' = Q' \parallel a!b.\mathbf{0} \parallel a!c.\mathbf{0}$.
To show that $K \triangleright Q \approx K \triangleright Q'$, we use $K \triangleright S \approx K \triangleright S'$ where $S = Q \parallel a!b.\mathbf{0} \parallel a!c.\mathbf{0}$ and $S' = Q' \parallel a!b.\mathbf{0} \parallel a!c.\mathbf{0}$.
For the complete proof, see Appendix J.3. □

Theorems 13.46 and 13.47 can be generalised to all reachable states, assuming that the initial system is persistently strongly receptive on a given channel name.

Corollary 13.48. If $K \triangleright P$ is persistently strongly receptive on channel name a, then each derivative of $K \triangleright P$ is confluent w.r.t. input actions $a?b$ and $a?c$ where $b \neq c$. ■

Corollary 13.49. If $K \triangleright P$ is persistently strongly receptive on channel name a, then each derivative of $K \triangleright P$ is confluent w.r.t. internal communication actions $com(a,b)$ and $com(a,c)$ where $b \neq c$. ■

The only system action that we have not yet considered is internal communication actions that involve scoped names, *ncom*. Example 13.50 shows that neither single receivers (resp. persistent single receivers) nor strong receptiveness (resp. persistent strong receptiveness) on a given channel give any determinism or confluence guarantees for it.

Example 13.50. Consider the system $K \triangleright P_{14}$ where $K = \{a,c,d\}$ and the running process P_{14} is defined below. Using Definitions 13.17 and 13.44, we can show that this system has both persistent single receivers and is persistently strongly receptive on channel name a.

$$P_{14} \overset{\text{def}}{=} P_{12} \parallel (\nu b)\big(b?x.x!b.\mathbf{0} \parallel b!c.\mathbf{0} \parallel b!d.\mathbf{0}\big) \quad \text{where} \quad P_{12} \overset{\text{def}}{=} \text{rec}X.a?x.(a!x.\mathbf{0} \parallel X)$$

System $K \triangleright P_{14}$ can internally receive either c or d on channel b via rules pPar and pNComm as follows:

$$K \triangleright P_{14} \xrightarrow{ncom} P_{12} \parallel (\nu b)\big(c!b.\mathbf{0} \parallel \mathbf{0} \parallel b!d.\mathbf{0}\big) \tag{13.19}$$

$$K \triangleright P_{14} \xrightarrow{ncom} P_{12} \parallel (\nu b)\big(d!b.\mathbf{0} \parallel b!c.\mathbf{0} \parallel \mathbf{0}\big) \tag{13.20}$$

The systems reached in (13.19) and (13.20) are clearly not bisimilar as they exhibit a different observational behaviour; the former scope extrudes b by outputting it on c, whereas the latter outputs it on d. Moreover, neither of these systems can transition with action *ncom*. This means $K \triangleright P_{14}$ is neither deterministic nor confluent w.r.t. *ncom* actions. ■

13.3.1 Syntactic Condition

To ensure that a system $K \triangleright P$ satisfies the *semantic* properties in Definitions 13.37, 13.39 and 13.42, we impose some restrictions on the structure of the running process P. Our first restriction relies on the function $iac(-)$ in Definition 13.51, returning the set of *input-active channels* of a process, *i.e.,* all channel names that are ready to accept an input after any number of τ-transitions. *E.g.* $iac(a?x.b?x.\mathbf{0} \| c?x.\mathbf{0}) = \{a,c\}$ and $iac((\nu a)(a?x.x!a.\mathbf{0})) = \emptyset$. Requiring $a \in iac(P)$ thus essentially reduces to syntactically checking that a process has at least one unguarded input on channel a.

Definition 13.51 (Input-active channels). The function $iac : \text{Prc} \mapsto \mathcal{P}(\text{Chans})$ is defined as follows:

$$iac(\mathbf{0}) \stackrel{\text{def}}{=} \emptyset \quad iac(X) \stackrel{\text{def}}{=} \emptyset \qquad iac(a?x.P) \stackrel{\text{def}}{=} \{a\}$$

$$iac(a!b.\mathbf{0}) \stackrel{\text{def}}{=} \emptyset \qquad iac(\text{if } b=c \text{ then } P \text{ else } Q) \stackrel{\text{def}}{=} iac(Q)$$

$$iac(P \parallel Q) \stackrel{\text{def}}{=} iac(P) \cup iac(Q) \qquad iac(\text{if } b=b \text{ then } P \text{ else } Q) \stackrel{\text{def}}{=} iac(P)$$

$$iac((\nu c)P) \stackrel{\text{def}}{=} iac(P) \setminus \{c\} \qquad iac(\text{rec}X.P) \stackrel{\text{def}}{=} iac(P)$$ ■

We observe that the function $iac(-)$ is total. Specifically, our initial assumption that all processes are closed indicates that processes cannot be of the form $P = \text{if } x = y \text{ then } P' \text{ else } P''$ or $Q = x!y.P'$, where variables x and y are not bound by some input prefix. Moreover, since we are assuming that $asy(P)$ for any process P, then output processes are always of the form $a!b.\mathbf{0}$ for some channel names a,b.

Proposition 13.52 below assures us that whenever a given channel a is input-active in a process P, then all systems $K \triangleright P$ must also be receptive on that channel.

Proposition 13.52. For any valid interface $K \subseteq \text{Chans}$, if $a \in iac(P)$ then $K \triangleright P$ is receptive on a.

Proof Outline. Assume that a is an input-active channel of P, *i.e.,* $a \in iac(P)$. The proof is by induction on the structure of P. We outline the main cases:

- When $P = b?x.Q$, by the assumption that $a \in iac(P)$, we must have $b = a$. Our result follows by rule pIn since $b?x.Q \xrightarrow{b?c} Q[c/x]$ for any $c \in \text{Chans}$.
- When $P = P_1 \parallel P_2$, then either $a \in P_1$ or $a \in P_2$. *W.l.o.g.*, suppose the former. By the IH, we obtain that $K \triangleright P_1$ is receptive on a, *i.e.,* $K \triangleright P_1 \Rightarrow \cdot \xrightarrow{a?\_} Q_1$ for some Q_1. Repeatedly applying rule pPar, we obtain $K \triangleright P_1 \parallel P_2 \Rightarrow \cdot \xrightarrow{a?\_} Q_1 \parallel P_2$, meaning that $K \triangleright P_1 \parallel P_2$ is receptive on a.

For the complete proof, refer to Appendix J.3. □

In order to guarantee that a process is strongly receptive on channel name a, requiring that a is input-active does not suffice. For instance, although $a \in iac(P_{12})$, Example 13.43 showed that $K \triangleright P_{12}$ is not strongly receptive on that channel. We thus add a second constraint, namely that channel a is always available in *input-replicated* form, $*a?x.P$, where

$$*a?x.P \stackrel{\text{def}}{=} \mathrm{rec}X.a?x.(P \parallel X) \quad \text{with} \quad X \notin fV(P) \tag{13.21}$$

The notation in eq. (13.21) is only used to facilitate readability; in what follows, we occasionally revert to describing input-replicated processes using the recursion construct. We also note that since we only want to guarantee strong receptiveness on specific channel names, we still allow basic inputs on other channel names; see rules rIn2 to be discussed later.

Remark 13.53. The notation in eq. (13.21) is closely related to the replicated input construct $!a(x).P$ found in certain versions of the asynchronous π-calculus, *e.g.* [104, 120]. Indeed, both constructs express persistent input behaviour. The difference lies in how this behaviour is represented within the calculus. For instance, in the calculus of Sangiorgi *et al.* [120], replicated input is a primitive syntactic construct whose behaviour is specified directly by the operational rule

$$\frac{}{!a(x).P \xrightarrow{a\langle b\rangle} P[^b/_x] \parallel !a(x).P}\ \text{REP}$$

In contrast, our calculus includes recursion, allowing persistent input processes to be expressed as recursive processes. Consequently, $*a?x.P$ is introduced as convenient notation that can be expanded into a recursive process that can transition using rule pRec, without requiring a dedicated operational rule for replication. ■

We statically check if channel name a is only available in input-replicated form in process P through a proof system. Its main judgement is $r(P,a)$, *i.e., a is input-replicated in* P, and it is the least relation defined by the rules in Definition 13.54. The main rules are rRec1, rRec2, rIn1 and rIn2. Rule rRec1 states that channel a is input-replicated in $\mathrm{rec}X.P$ if *(i)* process P in $\mathrm{rec}X.P$ is precisely of the form $a?x.(P' \parallel X)$ for some P', *i.e.,* $\mathrm{rec}X.P = *a?x.P'$, and *(ii)* channel a is input-replicated in P'. Alternatively, channel a is input-replicated in $\mathrm{rec}X.P$ if P is not prefixed with an input on that channel, and $r(P,a)$ (rule rRec2). Since an input-replicated process $*a?x.P'$ is behaviourally equivalent to its unfolding, namely $a?x.(P' \parallel *a?x.P')$, rule rIn1 states that channel a is input-replicated in $a?x.P$ if P is of the form $P' \parallel *a?x.P'$ for some P', and a is input-replicated in P'. In contrast, if a process is prefixed by an input on a channel different from a, *i.e.,* $u?x.P$ with $a \neq u$, then a is input-replicated in that process if it is input-replicated in P (rule pIn2). The remaining rules are straightforward.

Definition 13.54 (Input-replication). The judgement $r(P,a)$ is defined as the least relation of the form (Prc × Chans) satisfying the following rules:

$$\frac{}{r(\mathbf{0},a)}\,\textsc{rNil} \qquad \frac{}{r(X,a)}\,\textsc{rTVar} \qquad \frac{r(P,a)}{r(u!v.P,a)}\,\textsc{rOut} \qquad \frac{r(P,a)\quad a\neq c}{r((\nu c)P,a)}\,\textsc{rScp}$$

$$\frac{r(P,a)\quad r(Q,a)}{r(\text{if } u=v \text{ then } P \text{ else } Q,a)}\,\textsc{rCmp} \qquad \frac{P=P' \parallel *a?x.P'\quad r(P',a)}{r(a?x.P,a)}\,\textsc{rIn1}$$

$$\frac{r(P,a)\quad r(Q,a)}{r(P\parallel Q,a)}\,\textsc{rPar} \qquad \frac{P=a?x.(P'\parallel X)\quad X\notin fV(P')\quad r(P',a)}{r(\mathrm{rec}X.P,a)}\,\textsc{rRec1}$$

$$\frac{P\neq a?x.P'\quad r(P,a)}{r(\mathrm{rec}X.P,a)}\,\textsc{rRec2} \qquad \frac{u\neq a\quad r(P,a)}{r(u?x.P,a)}\,\textsc{rIn2}$$

■

Example 13.55. Consider again process $P_{12} \stackrel{\text{def}}{=} \mathrm{rec}X.a?x.(a!x.\mathbf{0} \parallel X)$ from Example 13.43. According to Definition 13.54, channel a is input-replicated in P_{12} as shown by the left proof derivation. Indeed, using the notation in (13.21), this process can be rewritten as $*a?x.a!x.\mathbf{0}$.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{r(\mathbf{0},\,a)}\,\textsc{rNil}}{r(a!x.\mathbf{0},\,a)}\,\textsc{rOut} \qquad \dfrac{}{r(X,\,a)}\,\textsc{rTVar}}{r(a!x.\mathbf{0} \parallel X,\,a)}\,\textsc{rPar}}{r(\mathrm{rec}X.a?x.(a!x.\mathbf{0} \parallel X),\,a)}\,\textsc{rRec1}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{r(\mathbf{0},\,b)}\,\textsc{rNil}}{r(a!x.\mathbf{0},\,b)}\,\textsc{rOut} \qquad \dfrac{}{r(X,\,b)}\,\textsc{rTVar}}{r(a?y.(a!x.\mathbf{0} \parallel X),\,b)}\,\textsc{rPar}}{r(a?x.a?y.(a!x.\mathbf{0} \parallel X),\,b)}\,\textsc{rRec2}}{r(\mathrm{rec}X.a?x.a?y.(a!x.\mathbf{0} \parallel X),\,b)}\,\textsc{rRec2}$$

Similarly, the right proof derivation above shows that channel name b is input-replicated in process $P_9 \stackrel{\text{def}}{=} \mathrm{rec}X.a?x.a?y.(a!x.\mathbf{0} \parallel X)$ from Example 13.24. We observe that, since we are checking for input-replication on channel name a, this derivation uses rule rRec2 instead of rRec1. This is because although the body of the recursive process P_9 is prefixed with an input, this input occurs on a different channel, $b \neq a$, and therefore its form is not relevant, *i.e.,* it need not be input-replicated. In contrast, channel name a is not input-replicated in P_9: there is no process P' that satisfies the equality $a?x.a?y.a!x.X = a?x.(P' \parallel X)$ required by the premise of rule rRec1. ■

Proposition 13.56 below states that requiring channel a to be both *(i)* input-active and *(ii)* input-replicated in process P gurantees that all systems $K \triangleright P$ are strongly receptive on channel a.

Proposition 13.56. For any valid interface $K \subseteq \textsc{Chans}$, if $a \in iac(P)$ and $r(P,a)$ then system $K \triangleright P$ is strongly receptive on channel name a.

Proof Outline. Intuitively, this result holds for the following reasons:

- Proposition 13.52 and the assumption that $a \in iac(P)$ guarantees that $K \triangleright P$ is receptive on channel a, satisfying the first condition of Definition 13.42.
- Showing that $K \triangleright P$ satisfies the second condition of Definition 13.42 is more involved. The intuition is that whenever a process P accepts an input on an input-replicated channel a, a copy of that process immediately becomes available for input. *E.g.* if $K \triangleright a?x.P \xrightarrow{a?b} Q$, then using rule rIn1, we can infer that $P = P' \parallel *a?x.P'$ and $Q = P'[^x/_b] \parallel *a?x.P'$ where the sub-process $*a?x.P'$ can input on a and is bisimilar to the initial one.

The complete proof is given in Appendix J.3. □

Example 13.57 shows that the converse of Proposition 13.56 does not hold.

Example 13.57. Consider the basic system $K \triangleright P_{13}$ where $K = \{a\}$ and the running process is defined as $P_{13} \stackrel{\text{def}}{=} \text{rec}X.a?x.X$. Clearly, $K \triangleright P_{13}$ is strongly receptive on channel a, as $K \triangleright P_{13} \xrightarrow{\tau} \cdot \xrightarrow{a?b} P_{13}$ and $\mathbf{aft}(K, a?b) \models P_{13} \approx P_{13}$ for any inputted channel b. However, this system is ruled out by our analysis as process P_{13} does not satisfy the second syntactic condition stating that channel a must be available in input-replicated form, *i.e.,* $\neg\, r(P_{13}, a)$. ■

Ensuring that strong receptiveness on a given channel is persistent throughout the entire system execution is complicated by the system's capability of using received channel names in input positions, discussed earlier in Section 13.2.1. This is better illustrated in the example below.

Example 13.58. Consider again process P_{13} from Example 13.45, rewritten using the notation in (13.21).

$$P_{13} \stackrel{\text{def}}{=} *a?x.x?y.a!y.\mathbf{0}$$

Example 13.45 showed that for $K = \{a\}$, system $K \triangleright P_{13}$ is not persistently strongly receptive on channel a. However, statically analysing P_{13} establishes that it is both input-active and input-replicated on channel a, *i.e.,* $a \in iac(P_{13})$ and $r(P_{13}, a)$. This means that requiring that the initial process satisfies these two conditions alone does ensure persistent strong receptiveness. This happens because Definition 13.54 only imposes constraints on the initial structure of the system—namely inputs on a specific channel—but does not account for how it evolves with input actions. In particular, when the received channel name a is used in an input position,

$$K \triangleright P_{13} \xrightarrow{\tau} \cdot \xrightarrow{a?a} a?y.a!y.\mathbf{0} \parallel *a?x.x?y.a!y.\mathbf{0}$$

a new input on that channel is created, but it does *not* occur in input-replicated form. ■

Following the approach in Section 13.2.1, we guarantee persistent strong receptiveness by demanding that all processes are local, *i.e.,* only the output capabilities of channel names can be transmitted, using the predicate $loc(-)$ in Definition 13.33. Whenever channel name a is input-active and input-replicated in process P and P is local, then these three properties are preserved under all system transitions.

Proposition 13.59 (Preservation). Suppose that $a \in iac(P)$ and $r(P, a)$ and $loc(P)$. If $K \triangleright P \xrightarrow{\eta} Q$ then $a \in iac(P)$ and $r(Q, a)$ and $loc(Q)$.

Proof Outline. The proof is by induction on the derivation of $r(Q, a)$. Intuitively, it holds because the locality constraint ensures that received inputs do not introduce new inputs on channel a, which ensures that $r(-)$ is satisfied by all reachable systems. Conversely, receptiveness ensures that whenever the process inputs on channel name a, a new input on that channel name immediately becomes available, meaning that there is always at least one process that is input-active on a.
For the full proof, refer to Appendix J.3. □

When used in conjunction, these three conditions guarantee that the strong receptiveness property is persistent, as formalised in Proposition 13.60 below.

Proposition 13.60 (Persistent Strong Receptiveness). For all valid interfaces $K \subseteq \textsc{Chans}$, if $a \in iac(P)$ and $r(P, a)$ and $loc(P)$ then system $K \triangleright P$ is persistently strongly receptive on channel name a.

Proof. Similar to that for Proposition 13.11, using Propositions 13.56,13.59 instead of Propositions 13.9,13.10
□

13.4 Summary

This chapter showed, through a series of pathological examples, that in general, processes expressed using the standard π-calculus are neither confluent nor deterministic w.r.t. any actions. We systematically showed that certain degrees of determinism and confluence can be recovered by imposing semantic constraints that capture the defining characteristic of the actor model, described by Prop. 1 to Prop. 3.

To guarantee Prop. 1, we introduced the notions of asynchrony and persistent asynchrony for specific channel names and proved that the class of LTSs that satisfy these properties are deterministic w.r.t. output actions $a!b$ (Theorem 13.5). In Proposition 13.11, we proved that we can guarantee asynchrony and persistent asynchrony for all channel names by removing output prefixing.

To guarantee Prop. 2, we presented the notions of single receivers and persistent single receivers. Theorems 13.5 and 13.22 respectively proved that the class of LTSs that satisfy these two properties w.r.t. channel a are deterministic for input actions $a?b$ and internal communication actions $com(a,b)$. We proved that we can guarantee single receivers on specific channel names by demanding process to have disjoint receivers on that channel name (Proposition 13.29) and that this property is persistent if received channel names cannot be used in input positions (Proposition 13.36).

To guarantee Prop. 3, we presented the notions of strong receptiveness and persistent strong receptiveness. Theorems 13.46 and 13.47 respectively proved that the class of LTSs that satisfy these two properties w.r.t. a given channel a are confluent for both input and internal communication actions carrying different payloads, *i.e.,* input actions $a?b$ and $a?c$, and communication actions $com(a,b)$ and $com(a,b)$ with $b \neq c$. Finally, we showed that we can guarantee strong receptiveness on specific channel names by requiring that channel to be active in an input position and that all inputs on it are only available in a specific form (Proposition 13.56). This property is then persistent throughout the entire system execution if received channel names cannot be used in input positions (Proposition 13.60).

14 Related Work

We divide this chapter into two sections: the first discusses the existing literature related to Parts I to III, whereas the second reviews the literature relevant to Part IV.

14.1 Multiple Traces for Runtime Verification

Linear-time properties This thesis showed that the RV technique can be extended to meaningfully verify a wider range of branching-time properties by considering multiple system executions. This is in sharp contrast to most research on RV, which centres around monitoring linear-time properties [36, 100]. In particular, RV might seem to be an inherently linear-time exercise: a monitor in the classical RV setup can, after all, only observe a single execution. However, the key difference in monitoring in the linear- and branching-time settings is not the monitor performing the verification, but rather, the correctness property it is monitoring for. More specifically, in linear-time monitoring, we are interested in a property of the *current* execution, rather than the system as a whole. This is particularly useful for checking in deployed systems whether the output of a third-party component is safe to use in a critical component, for example. In contrast, when RV is used as a best-effort alternative to model checking, we are trying to determine out whether the entire *system*, rather than the current execution, is, in some sense, correct. Since monitors still only observe one execution, the class of monitorable branching-time properties is, unavoidably, syntactically smaller than that of monitorable linear-time properties [10], explaining, in part, why the linear-time setting appears less restrictive when being runtime verified. For instance, linear-time properties that are monitorable for violations are closed under disjunctions, $\varphi \vee \phi$, and existential modalities, $\langle \alpha \rangle \varphi$, as in that setting, these operations can be eliminated by encoding them in an effective, if not efficient, manner [10, 8] in terms of other recHML constructs. In contrast, disjunctions and existential modalities in a branching-time setting cannot be encoded in terms of other recHML constructs. As a result, the benefit of using multiple runs is even greater, as our result show that they expand the realm of monitorable branching-time properties.

Hyperproperties Various bodies of work employ monitors over multiple runs for the purpose of RV. The most prominent efforts target *Hyperproperties*, *i.e.,* properties describing sets of traces called *hypertraces*, used to describe safety and privacy requirements [52].

Finkbeiner *et al.* [67] investigate the monitorability of hyperproperties expressed in HyperLTL [53] and identify three classes for monitoring hypertraces: the bounded sequential, the unbounded sequential and the parallel classes. They also develop a monitoring tool called RVHyper [66] that analyses hypertraces sequentially by converting an alternation-free HyperLTL formula into an alternating automaton, which is then executed over different permutations of the observed traces. The authors show that deciding

monitorability for alternation-free HyperLTL formulae in this class is PSPACE-complete but undecidable in general. Our monitor setup fits their unbounded sequential class because monitors receive each trace in sequence, and a SUS may exhibit an unbounded number of traces.

Agrawal *et al.* [23] give a semantic characterisation for monitorable HyperLTL hyperproperties called k-safety. They also identify syntactic HyperLTL fragments and show they are k-safety properties, backed up by a monitor synthesis algorithm that generates a combination of Petri nets [112] and LTL_3 monitors [38], whereby monitors are generated on-the-fly.

Stucki *et al.* [125] show that many properties in HyperLTL involving quantifier alternation cannot be monitored for. They also present a monitoring methodology for properties with one alternation by combining static verification and RV: the static part extracts information about the set of traces that the SUS can produce (*i.e.,* branching information about the number of traces in the SUS, expressed as a symbolic execution tree) that is used by monitors to convert quantifications into k-(trace)-quantifications.

Despite the similarities of using multi-run monitoring, the works in [67, 23, 125] differ from ours in a number of ways. For instance, the methods used are very different. Our monitor synthesis algorithm is directly based on the formula syntax and does not rely on auxiliary models such as alternating automata or petri nets, which facilitates syntax-driven proofs. The results they present are also substantially different from ours. Although [23, 125] prove that their monitor synthesis algorithm is sound, neither work considers completeness results, maximality or execution lower bound estimation. More importantly, our target logic, recHML, is intrinsically different from (linear-time) hyperlogics since it (and other branching-time logics) is interpreted over LTSs (or more concretely, system states), whereas hyperlogics are defined over sets of traces. The latter model of describing a system is inherently coarser than an LTSs. For instance, the systems $a.b.\mathbf{0}+a.c.\mathbf{0}$ and $a.(b.\mathbf{0}+c.\mathbf{0})$ are described by different LTSs but have an identical trace-based model, namely $\{ab,ac\}$. Even though this is not an issue when monitoring for hyperproperties, it was a major source of complication for our technical development. However, even if we were to restrict our analysis to the *deterministic* LTS settings, where system such as $a.b.\mathbf{0}+a.c.\mathbf{0}$ are disallowed, it remains unclear how the two types of logics correspond. For one, hyperlogics employ existential and universal quantifications over traces, which are absent from our logic. If we had to normalise these differences (*i.e.,* by removing trace quantifications), a reasonable mapping would be to take a a linear-time interpretation $[\![\varphi]\!]_{\text{LT}}$ [10, 12] for every recHML branching-time property φ, and require it to hold for all of its traces. For all branching-time properties φ and deterministic systems p, we would then expect that p satisfies φ if and only if the set of all traces produced by p satisfies the linear-time interpretation of φ, that is:

$$p \in [\![\varphi]\!] \text{ iff } T_p \in [\![\varphi]\!]_{\text{LT}}$$

But even this correspondence fails. For instance, the branching-time property $[a]\mathsf{ff} \vee [b]\mathsf{ff}$ describes systems that *cannot* perform both a and b actions, and the system $a.\mathbf{0}+b.\mathbf{0}$ clearly violates it. However, with a linear-time interpretation, this formula denotes a *tautology, i.e.,* $[a]\mathsf{ff} \vee [b]\mathsf{ff} \equiv \mathsf{tt}$: it is satisfied by *all* traces since they are necessarily either not prefixed with an a action or with a b action. There are, however, notable similarities between our history evaluation (Figures 3.3 and 7.2) and team semantics for temporal logics [95, 127], and this relationship is worth further investigation.

Distributed Monitoring Multiple execution traces are also considered in distributed monitoring as a means of fault tolerance through the replication of components that perform specific tasks, including

monitors. Fraigniaud *et al.* [70] and Bonakdarpour *et al.* [42] explore scenarios where events can be observed by multiple monitors that may potentially crash (*i.e.,* halt and never recover). Surviving monitors then emit verdicts based on their respective partial views of the execution and communicate it to the other monitors to collectively arrive at a global verdict. Our approach could, in principle, adopt fault-tolerant techniques from this work to mitigate history corruption and execution crashes across the various monitored executions of the SUS.

Parametric Trace Slicing Other approaches use multiple traces to runtime verify traces with imprecise event ordering [129, 49, 35, 31] due to interleaved executions of components architectures, which might hinder the monitor's detection capability. Parametric trace slicing [49, 35] infers additional traces from a trace with interleaved events by traversing the original trace and dispatching events to the corresponding slice. Attard *et al.* [31] partition the observed trace at the instrumentation level by synthesising monitors and attaching them to specific system components; they hint at how this could enhance the monitoring expressive power for certain properties but do not prove any monitorability results. Despite their relevance, all traces considered in [129, 49, 35, 31] are extracted from a *single* execution as a means to recover missing or imprecise data in the observed trace.

Testing Abramsky [1] studies testing on multiple, yet finite, copies of the same system, combining the information from multiple runs. Our approach differs in three key aspects. Firstly, our multiple executions correspond to creating multiple copies of the system from its initial state; Abramsky allows copies to be created at *any point* of the execution. Secondly, tests are composed using parallel composition, can steer the execution of the SUS and can detect refusals. In contrast, our monitors are composed using an instrumentation relation: they are passive and their verdicts are evidence-based (*i.e.,* what happened, not what could *not* have happened). Third, the visibility afforded by monitor instrumentation considered in this work is larger than that obtained via parallel composition.

Silva *et al.* [55] investigate combining traces produced by the same system in subsequent runs to create temporal models that approximates the SUS's behaviour which can then be used to model check for branching-time properties. However, this approach is *not* sound as the generated model may violate properties that are not violated by the actual system. Because of this, the authors advise using their approach as a formal complement to software testing to suggest possible problems.

Monitoring with Prior Knowledge The closest work to ours is [7], where Aceto *et al.* give a framework to extend the capabilities of monitors. They study monitorability under a grey-box assumption where, at runtime, a monitor has access to additional SUS information, linked to the system's states, in the form of decorated states. The additional state information is parametrised by a class of *conditions* that represent different situations, such as access to information about that state gathered from previous system executions. Other works have also examined how to use prior knowledge about the SUS to extend monitorability both in the linear-time setting [85, 50, 99] and the branching-time setting [13]. For instance, Aceto *et al.* [13] incorporate prior knowledge about the SUS, such as independence between individual components or state-reachability (expressed as branching-time properties K) during monitor synthesis: instead of generating monitors from correctness properties P, they generate monitors based on the conjunction $K \wedge P$. In contrast, our work makes no prior assumptions about the SUS and treats it

as a black box; while our augmented instrumentation exposes additional information to the monitor, this information remains fixed throughout the system execution and characterises an infinite class of systems, rather than a specific SUS (see Section 6.2 for details).

14.2 A study of Actor Model Properties in Process Calculi

Confluence and Determinism Confluence was first introduced in [119, 105] and later examined in [106] by Milner for CCS processes, where it was shown that confluence implies determinacy under silent τ actions, and that it is preserved by certain system-building constructs. A similar study was later conducted by Philippou *et al.* [113] for a richer calculus with name passing, namely the π-calculus. Although our definitions for confluence and determinism are grounded in those of [113], our objectives are significantly different. Whereas Philippou's work confines the use of certain system-building operators to construct systems that are guaranteed to be confluent, our work takes a more analytic approach: we identify a subset of actions that are confluent or deterministic within a system, without requiring the entire system to be confluent/deterministic itself.

Our work also shares similarities with that of Aceto *et al.* [4] who address the undecidability of semantic determinism through syntactic criteria. However, the role of syntax differs substantially between the two approaches. The authors propose rule formats that serve as templates for defining languages whose processes satisfy determinism-related properties by construction. In contrast, we use syntactic conditions as auxiliary criteria for detecting pre-defined semantic notions of confluence and determinism. Moreover, whereas their meta-theory establishes these properties for entire language fragments or classes of operators, our approach identifies deterministic and confluent subsets of actions within a given system.

Process Calculi inspired by the Actor Model The work in [20] presents a typed variant of the π-calculus called A_π, designed to represent the actor model. Agha *et al.* also present an actor language called SAL and show how actor systems expressed in SAL can be translated into A_π to help transfer results from one theory to the other. Actor properties, such as uniqueness of actor names, persistence (actors do not disappear after processing a message) and freshness (actors cannot be created with names received in a message), are enforced via a type system. Agha *et al.*'s notions of uniqueness and freshness imply our single-receiver property: the former prevents multiple (parallel) processes from inputting on the same channel name, whereas the latter ensures that this holds throughout the entire system execution. Although their persistence property is related to our notion of persistent receptiveness, the proposed type system relaxes the persistence requirement by allowing channel names to disappear after accepting an input: the authors interprets this as actors with a "sink" behaviour that consume all the message they receive without performing any further actions.

Sangiorgi [120] presents two candidate definitions for receptiveness in the π-calculus, namely linear receptiveness and ω-receptiveness, and shows how they can be captured through type systems. Our notion of persistent receptiveness is closely related to the latter since we require persistently receptive channel names to *always* be ready to accept an input. However, our definition is not of the *uniform* kind: it does not require all inputs on a persistently receptive channel to have the same continuation. Other languages and calculi based on the π-calculus, such as PICT [114], the join calculus [68], and the

blue calculus [44], also make use of specific input constructs that satisfy properties corresponding to Sangiori's notion for ω-receptiveness. Similarly, inputs on these channels are treated uniformly.

Amadio *et al.* [27] present a variant of the distributed π-calculus [84] called the *receptive π-calculus*. In contrast to Sangiorgi's definition [120], their notion of receptiveness is not uniform: they require receptive inputs to be involved in a recursive process that may reincarnate itself, possibly in a *different* state, after accepting an input. They show that the receptiveness of all private channel names can be ensured through a static analysis that, when used in tandem with a type system, guarantees that messages always will find a matching receiver.

Despite the similarities of capturing certain actor properties in languages or calculi based on the π-calculus, the works in [120, 114, 68, 44, 27] differ from ours in a number of ways. For one, their notions of receptiveness applies to *private* channel names (within their scope), whereas our definition is concerned with *free* channel names. Since we want to study whether the persistent receptiveness property (and other actor properties) yields any confluence or determinism guarantees, it makes little sense to ascribe such properties to bound channel names, since bound names are local can thus be freely α-renamed. Another key difference is that all the work in [120, 114, 68, 44, 27] capture receptiveness by restricting their calculus at the syntactic level, *e.g.* through type systems. In contrast, we define persistent receptiveness in terms of semantic properties and derive our confluence and determinism results directly from them. While we also introduce syntactic conditions, their role is auxiliary: they only serve as practical checks that ensure the corresponding semantic counterparts are realisable.

Several other calculi [77, 89, 117, 57, 69] were inspired by the actor model and the π-calculus, but they are either not entirely faithful to the actor model or make use of primitives that are not intrinsic to either of the models, making them difficult to compare with our work. Indeed, the aim of these works were significantly different from ours: these calculi were primarily designed to model interaction in concurrent object-oriented programming and examine their behaviour. For instance, Gaspari and Zavattaro [77] present a process algebra based on the actor model. Objects in their calculus observe certain behavioural properties akin to those in the actor model, such as asynchronous message-passing and the single-receiver property, but ignore the persistent-receptiveness property. They also have constructs that are found in neither the π-calculus nor the actor model. Honda *et al.* [89] introduce an asynchronous variant of the π-calculus that restricts the syntax to only allow output prefixes that are succeeded by the inactive process, effectively capturing asynchronous message-passing. Ravara *et al.* [117] present an object-based extension of the asynchronous π-calculus [89] where each object has a unique mailbox that it can read from and received channel names can only be used in output positions. De' Liguoro *et al.* [57] propose the mailbox calculus, an extension of the asynchronous π-calculus with first-class mailboxes, selective reads and non-deterministic choices. In contrast to the pure actor model, processes in this calculus may read from *multiple* mailboxes. The authors also present a mailbox type system, guaranteeing that processes can only receive messages that they are capable of processing, while ruling out configurations where processes are blocked. Despite the similarities of [117, 89, 57] with our work, they only capture the asynchronous message-passing property of the actor model but do not consider the single receivers and persistent-receptiveness properties.

15 Conclusion

This thesis proposed a monitoring framework to systematically extend the RV technique to meaningfully verify a wider range of branching-time properties by observing multiple system executions. As a first step towards a comprehensive study, we focused on deterministic systems. Our results demonstrate that monitors can extract sufficient information over multiple runs of an arbitrary deterministic SUS to correctly detect the violation of *any* branching-time properties that may contain disjunctions (Theorem 5.5). We also identified a monitorable fragment $\text{sHML}^{\vee}$ (Definition 5.2) and showed that it is maximally expressive (Section 5.3). In particular, every property that can be monitored for correctly using the proposed multi-run monitoring framework can always be expressed as a formula in $\text{sHML}^{\vee}$. Such a syntactic characterisation of monitorable properties is useful for the construction of a tool based on the proposed method: the development of an automated monitor synthesis could exclusively target the identified syntactic fragment $\text{sHML}^{\vee}$, assured that all monitorable properties are still covered.

Since the ultimate goal of this thesis was to extend the existing monitorability limitations for *any* system, even if it might exhibit certain non-deterministic behaviour, the second step was to extend our study to potentially non-deterministic systems. Concretely, we presented an augmented instrumentation that makes certain hidden (τ-)actions visible to the monitor (to allow it to differentiate between confluent internal moves and moves that lead to non-deterministic branching), as well as provides the monitor with information about whether the observed actions are deterministic. Equipped with this additional SUS information, our results show that monitors can extract sufficient information over multiple runs to correctly detect the violation of a class of branching-time properties that may contain disjunctions (Theorem 9.7). We also identified a monitorable fragment $\text{sHML}^{\vee}_{\text{DET}}$ (Definition 9.3) and showed that it is maximally expressive (Theorem 9.29). It is worth pointing out that an implementation based on our theoretical framework could relax some of the assumptions that are only used only to attain completeness and maximality results. For instance, instead of assuming that all internal actions of a SUS are deterministic, a tool based on our theory could adopt a pragmatic stance and simply stop monitoring a formula that contains disjunctions as soon as a non-deterministic internal action is encountered, which would still yield a sound (but incomplete) monitor.

To show that the proposed technique lends itself well to the implementation of a tool, we outlined the steps towards a full automation and gave a corresponding complexity analysis. In order to attain an efficient implementation, we demonstrated that the (least) number of expected runs required to effect the runtime analysis can be calculated from the structure of the formula expressing the property being verified (as opposed to obtaining this information by statically analysis the SUS [125]); see Theorem 10.10. We are unaware of similar results in the RV literature and do not rule out the possibility that our techniques could be adapted to carry out similar minimal-run calculations for hyperproperties. We also

validated the realisability of our (extended) multi-run monitoring RV framework by outlining a possible instantiation to (concurrent) actor-based systems. Since such system are inherently non-deterministic, we also presented a general study of how enforcing certain characteristics of the actor model in a highly non-deterministic calculus allows us to recovery a certain degree of determinism (Theorems 13.5, 13.20, 13.22, 13.46 and 13.47). Finally, we showed that these semantic properties can be guaranteed through syntactic conditions on the processes themselves.

15.1 Future Work

This thesis is part of an ongoing effort to extend RV to a wider class of properties. While we demonstrated how some of the inherent limitations of the technique can be partially overcome by considering multiple runs, there are still several possible avenues for future research.

We plan to investigate how our results can be extended further by considering more of a grey-box view of the system, in order to combine our machinery with techniques from existing work. For instance, a possible direction would be following the approach of Aceto *et al.* [7] where, at runtime, instrumentation provides the monitor with additional (branching) information about *specific* system states, such as computation steps that could have happened or could not have happened, gathered from previous system executions. We will also study strategies to optimise the collection of relevant SUS traces. Depending on the application, one might seek to either maximise the information collected from every execution (*e.g.* by continuing to monitor the same execution even after a trace prefix is added to the history) or else minimise the runtime during which the monitor is active. Moreover, we plan to study whether we can establish any upper bounds for the number of SUS executions required for RV, and complement the lower bounds established in Theorem 10.10.

Another avenue for future work is the construction of a tool that runtime verifies systems over multiple executions. Chapter 10 outlined an algorithm for a full automation of the proposed verification technique. The next step is to automate this algorithm, possibly by extending existing (single-run) open-source monitoring tools for ReCHML such as detectEr [32, 15] that already target actor systems. In the multi-run monitoring setup of Chapter 7 and the instantiation of Chapter 11, instrumentation records certain internal actions such as process communication. We conjecture that this is a reasonable assumption since many actor-based programming language platforms come ready equipped with tracing mechanisms that are used for a variety of purposes such as debugging and telemetry. For instance, the Virtual Machine for Erlang and Elixir (two widely-used actor-based languages), called the EVM, records internal communication events as output events immediately followed by the corresponding input event. In existing tools, these two events are coalesced into silent (τ-)actions but they could be easily combined into the internal communication events of Chapter 11 instead.

Appendix I

Repeated Monitoring for Deterministic Systems Results

A General Results

In this section, we prove some properties about the violation relation $\models_v$ of Section 5.2, including Theorem 5.9, since this alternative definition for property violations will be used in several proofs.

Lemma 1. If $\mathbf{rej}(H,m)$ then $H \neq \emptyset$.

Proof. By rule induction on $\mathbf{rej}(H,m)$. □

Lemma 2. If $H \models_v \varphi$ then $H \neq \emptyset$.

Proof. By rule induction on $H \models_v \varphi$. □

The first property that we prove is about the violation relation $\models_v$ of Definition 5.7 is that it observes sanity checks akin to those for the history analysis of Section 3.2. In particular, Propositions A.3 and A.4 below guarantee that once a system violates a formula via a history, it will persistently violate that formula, regardless of any other behaviour it might exhibit (described in terms of additional traces added to the history, *width*, or longer trace prefixes, *length*).

Proposition A.3 (Width Violation Idempotency). For any histories H,H', if $H \models_v \varphi$ then $H \cup H' \models_v \varphi$.

Proof. Straightforward by rule induction on $H \models_v \varphi$. □

Proposition A.4 (Length Violation Idempotency). For any history H and traces t,u, if $H,t \models_v \varphi$ then $H,tu \models_v \varphi$.

Proof. By rule induction on $H,t \models_v \varphi$ using a proof similar to that for Proposition 4.8. □

To prove that despite the technical discrepancies between the definitions for property violations in Figure 2.1 and Definition 5.7, *i.e.*, $p \notin [\![\varphi]\!]$ and $H \models_v \varphi$ for $H \subseteq T_p$, the two definitions correspond, formalised as Theorem 5.9, we first need to prove several additional results.

Lemma 5. For any formula $\varphi \in \text{sHML}^\vee$ and history $H \subseteq T_p$, if $H \models_v \varphi$ then $p \notin [\![\varphi]\!]$.

Proof. Proof is by induction on the derivation of $H \models_v \varphi$.

- Case vF. We know $H \models_v \mathsf{ff}$. Our result, $p \notin [\![\mathsf{ff}]\!]$, is immediate since $[\![\mathsf{ff}]\!] = \emptyset$.
- Case vUm. We know $H \models_v [\alpha]\psi$ because $\mathit{suffix}(H,\alpha) \models_v \psi$. By Lemma 2, we also know $H' \neq \emptyset$. This means that $p \overset{\alpha}{\Longrightarrow} q$ for some q and $\mathit{suffix}(H,\alpha) \subseteq T_q$. By the IH, we thus obtain that $q \notin [\![\psi]\!]$. Using this and the fact that $\mathit{suffix}(H,\alpha) \models_v \psi$, we can conclude $p \notin \{q \mid q \overset{\alpha}{\Longrightarrow} q' \text{ implies } q' \in [\![\psi]\!]\} = [\![[\alpha]\psi]\!]$.

- Case vAndL. We know $H \models_v \psi \wedge \chi$ because $H \models_v \psi$. By the IH, we obtain $p \notin [\![\psi]\!]$. This implies $p \notin [\![\psi]\!] \cap [\![\chi]\!] = [\![\psi \wedge \chi]\!]$.
- Case vAndR. Proof is analogous to that for vAndL.
- Case vOr. We know $H \models_v \psi \vee \chi$ because $H \models_v \psi$ and $H \models_v \chi$. By the IH, we obtain $p \notin [\![\psi]\!]$ and $p \notin [\![\chi]\!]$, which implies $p \notin [\![\psi]\!] \cup [\![\chi]\!] = [\![\psi \vee \chi]\!]$.
- Case vMax. We know that $H \models_v \max X.\psi$ because $H \models_v \psi[^{\max X.\psi}/_X]$. By the IH, we obtain that $p \notin [\![\psi[^{\max X.\psi}/_X]]\!] = [\![\max X.\psi]\!]$.

This completes our proof. □

Corollary A.6. For all (closed) formulae $\varphi \in \text{sHML}^\vee$, if $\left(\exists H \subseteq T_p \text{ such that } H \models_v \varphi\right)$ then $p \notin [\![\varphi]\!]$.

Proof. Suppose $\exists H \subseteq T_p$ such that $H \models_v \varphi$. Our result follows by Lemma 5. □

Lemma 7. For all (closed) formulae $\varphi \in \text{sHML}^\vee$, if $p \notin [\![\varphi]\!]$ then $\left(\exists H \subseteq T_p \text{ such that } H \models_v \varphi\right)$.

Proof. To prove this lemma, we must be able to reason about potentially open formulae. For this, we need a *semantic* operation mapping the free variables of φ to a set of systems in the antecedent of the statement; we do this using environments ρ. Conversely, for the consequent, we need a *syntactic* operation mapping the free variables of φ to closed formulae; we handle this via substitutions $\sigma \in \text{Sub}$, Definition A.9. This results in the statement below (restated in Lemma 13):

$$\textit{For all } \varphi \in \text{sHML}^\vee,\ p \notin [\![\varphi, \rho]\!] \textit{ implies } \exists H \subseteq T_p \text{ such that } H \models_v \varphi\sigma \tag{A.1}$$

Since both ρ and σ operate on φ to deal with the *same* free variables, albeit in a different manner, $\rho(X)$ describes how the two are related: this is formalised in Definition A.8 below. Intuitively, ρX denotes the set of systems that don't violate $X\sigma$.
We also note that, when φ is closed, $[\![\varphi, \rho]\!] = [\![\varphi]\!]$ and $\varphi\sigma = \varphi$. Our result follows from eq. (A.1). □

Definition A.8. For all $\sigma \in \text{Sub}$ and $X \in \text{TVars}$,

$$\rho(X) = \{p \mid H \not\models_v X\sigma \text{ for all } H \subseteq T_p\}$$ ■

Definition A.9. Substitutions are total maps from recursion variables to *closed* formulae, $\sigma \in \text{Sub} : \text{TVars} \to \text{recHML}$. We use $[\psi_1/X_1, \psi_1/X_2, \ldots]$ to denote the substitution mapping each X_i to ψ_i for $i \in \{1, \ldots, n\}$. Substitution application, $\varphi\langle\sigma, V\rangle$, is defined w.r.t. a set of recursion variables $V \subseteq \text{TVars}$ as follows:

$$\mathsf{ff}\langle\sigma,V\rangle \stackrel{\text{def}}{=} \mathsf{ff} \qquad \mathsf{tt}\langle\sigma,V\rangle \stackrel{\text{def}}{=} \mathsf{tt} \qquad X\langle\sigma,V\rangle \stackrel{\text{def}}{=} \begin{cases} \sigma(X) & X \notin V \\ X & \text{otherwise} \end{cases}$$

$$([\alpha]\varphi)\langle\sigma,V\rangle \stackrel{\text{def}}{=} [\alpha]\varphi\langle\sigma,V\rangle \qquad (\varphi \wedge \psi)\langle\sigma,V\rangle \stackrel{\text{def}}{=} \varphi\langle\sigma,V\rangle \wedge \psi\langle\sigma,V\rangle$$

$$(\varphi \vee \psi)\langle\sigma,V\rangle \stackrel{\text{def}}{=} \varphi\langle\sigma,V\rangle \vee \psi\langle\sigma,V\rangle \qquad (\max X.\varphi)\langle\sigma,V\rangle \stackrel{\text{def}}{=} \max X.\varphi\langle\sigma, V \uplus \{X\}\rangle$$

We use σ_V as a shorthand for $\langle\sigma, V\rangle$. When $V = \emptyset$, we occasionally write $\varphi\sigma$ to mean $\varphi\sigma_V$. We also use $\sigma[\varphi/X]$ when the mapping for X in σ is overridden by φ. ■

We highlight a few aspects about Definition A.9. First, V in σ_V keeps track of the recursion variables bound by some outer greatest fixed point so as not to substitute them with the formulae mapped to in σ. Concretely, the substitution application in $\max X.\big(\varphi\sigma_{\{X\}}\big)$ leaves all occurrences of X in φ unchanged since they are bound by the outer max operator. Second, $\varphi\sigma$ always returns a *closed* formula since each free recursion variables X is substituted by the *closed* formula $\sigma(X)$.

Example A.10. Consider the formula $\max Z.\big([a]X \wedge Y \wedge [b]Z\big)$ and the substitution $\sigma = [\varphi/X, \psi/Y, \chi/Z, \ldots]$ where $\varphi, \psi, \chi, \ldots$ are closed formulae. Following Definition A.9, the substitution application $\varphi\sigma$ returns $\max Z.\big([a]\varphi \wedge \psi \wedge [b]Z\big)$ since X and Y are free. ■

Lemma 11. For any closed $\psi \in \text{sHML}^\vee$, $V \subseteq \text{TVars}$ and $\sigma' = \sigma[\psi/X]$, then $(\varphi\sigma_{V \uplus \{X\}})[\psi/X] = \varphi\sigma'_V$

Proof. By structural induction on $\varphi \in \text{sHML}^\vee$.

- Case $\varphi = \mathsf{ff}$. Trivial because $(\mathsf{ff}\sigma_{V \uplus \{X\}})[\psi/X] = \mathsf{ff} = \mathsf{ff}\sigma'_V$.
- Case $\varphi = Y$. There are two subcases to consider.
  - When $Y \notin V \uplus \{X\}$, then $X \neq Y$ and $(Y\sigma_)V \uplus \{X\}[\psi/X] = \sigma(Y)[\psi/X] = \sigma(Y)$ since $\sigma(Y)$ is closed by Definition A.9. Since the only difference between σ and $\sigma' = \sigma[\psi/X]$ is the entry for X but $X \neq Y$, we deduce that $\sigma(Y) = \sigma'(Y)$. Also, since $Y \notin V$, we know $\sigma'(Y) = Y\sigma'_V$. Our result, $(Y\sigma_{V \uplus \{X\}})[\psi/X] = Y\sigma'_V$, thus follows.
  - When $Y \in V \uplus \{X\}$, then $(Y\sigma_{V \uplus \{X\}})[\psi/X] = Y[\psi/X]$. If $Y = X$, we know $Y[\psi/X] = X[\psi/X] = \psi = X\sigma'_V = Y\sigma'_V$ since $X \notin V$ and $\sigma'(X) = \psi$. Otherwise, if $Y \neq X$, then $Y[\psi/X] = Y = Y\sigma'_V$ since $Y \notin V$. Our result follows.
- Case $\varphi = \varphi_1 \wedge \varphi_2$. By Definition A.9 and the IH, we have that $\big((\varphi_1 \wedge \varphi_2)\sigma_{V \uplus \{X\}}\big)[\psi/X] = (\varphi_1\sigma_{V \uplus \{X\}})[\psi/X] \wedge (\varphi_2\sigma_{V \uplus \{X\}})[\psi/X] = (\varphi_1\sigma'_V) \wedge (\varphi_2\sigma'_V) = (\varphi_1 \wedge \varphi_2)\sigma'_V$ as required.
- Case $\varphi = \varphi_1 \vee \varphi_2$. Analogous to the previous case.
- Case $\varphi = \max Y.\varphi'$. Assuming, by α-equivalence, that $X \neq Y$ we have $(\max Y.\varphi')\sigma_{V \uplus \{X\}}[\psi/X] = \max Y.(\varphi'\sigma_{V \uplus \{X,Y\}}[\psi/X])$. By the IH, this is equal to $\max Y.(\varphi'\sigma'_{V \uplus \{Y\}}) = (\max Y.\varphi'\sigma'_V)$, as required. □

Lemma 12. For any (closed) formulae $\varphi, \varphi_1, \varphi_2 \in \text{sHML}^\vee$ and history $H \in \text{Hst}$, we have that:

(i) $H \not\models_v [\alpha]\varphi$ iff $\mathit{suffix}(H, \alpha) \not\models_v \varphi$

(ii) $H \not\models_v \varphi_1 \wedge \varphi_2$ iff $H \not\models_v \varphi_1$ and $H \not\models_v \varphi_2$

(iii) $H \not\models_v \varphi_1 \vee \varphi_2$ iff $H \not\models_v \varphi_1$ or $H \not\models_v \varphi_2$

(iv) $H \not\models_v \max X.\varphi$ iff $H \not\models_v \varphi[^{\max X.\varphi}/_X]$

Proof. We only give the proof for (2); the others follow with a similar but more straightforward argument. *Proof of (2).* For the *if* direction, assume $H \not\models_v \varphi_1 \wedge \varphi_2$. Suppose, by way of contradiction, that either $H \models_v \varphi_1$ or $H \models_v \varphi_2$. In both cases, we obtain $H \models_v \varphi_1 \wedge \varphi_2$ by respectively applying rules vAndL and vAndR, both of which contradict the initial assumption.

For the *only if* direction, assume $H \not\models_v \varphi_1$ and $H \not\models_v \varphi_2$. Suppose, by way of contradiction, that $H \models_v \varphi_1 \wedge \varphi_2$. Since this could have been derived using either rule vAndL or vAndR, we infer that $H \models_v \varphi_1$ or $H \models_v \varphi_2$ holds, a contradiction. □

Lemma 13. Suppose ρ is defined as in Definition A.8. For all $\varphi \in \text{sHML}^\vee$, if $p \notin [\![\varphi, \rho]\!]$ then $\big(\exists H \subseteq T_p$ such that $H \models_v \varphi\sigma\big)$.

Proof. We prove the contrapositive, *i.e.,* for all $\varphi \in \text{sHML}^\vee$, we have that

$$\big(H \not\models_v \varphi\sigma \text{ for any } H \subseteq T_p\big) \textit{ implies } p \in [\![\varphi, \rho]\!]$$

Assume $H \not\models_v \varphi$ for all $H \subseteq T_p$. The proof proceeds by induction on the structure of φ.

- Case $\varphi = \mathsf{ff}$. We know $\mathsf{ff}\sigma = \mathsf{ff}$. Since $\{\epsilon\} \subseteq T_q$ for all $q \in \text{Prc}$, there exists some $H \neq \emptyset$ such that $H \models_v \mathsf{ff}$ by rule vF, which contradicts the initial assumption that $H \not\models_v \mathsf{ff}$ for all $H \subseteq T_p$. The statement is thus vacuously true.
- Case $\varphi = X$. Since $H \not\models_v X\sigma$ and $\rho(X) = \{q \mid H \not\models_v X\sigma \text{ for all } H \subseteq T_q\}$, then $p \in \rho(X) = [\![X, \rho]\!]$ as required.
- Case $\varphi = [\alpha]\psi$. Since $H \not\models_v ([\alpha]\psi)\sigma$ then $H \not\models_v [\alpha]\big(\psi\sigma\big)$. By Lemma 12(i), we know that $\textit{suffix}(H, \alpha) \not\models_v \psi\sigma$. There are two cases how this could happen. The first case is when $p \overset{\alpha}{\not\Rightarrow}$ and $H = \emptyset$. When that happens, we immediately get $p \in \{p \mid p \overset{\alpha}{\Longrightarrow} q \textit{ implies } q \in [\![\psi\sigma, \rho]\!]\} = [\![[\alpha]\psi]\!]$. The second case is when $p \overset{\alpha}{\Longrightarrow} q$ and $\textit{suffix}(H, \alpha) \subseteq T_q$. When that happens, by the IH, we obtain that $q \in [\![\psi\sigma, \rho]\!]$, which implies $p \in [\![[\alpha]\psi, \rho]\!]$.
- Case $\varphi = \psi_1 \wedge \psi_2$. Since $H \not\models_v (\psi_1 \wedge \psi_2)\sigma$, then $H \not\models_v \psi_1\sigma \wedge \psi_2\sigma$. From Lemma 12(ii), we know $H \not\models_v \psi_1\sigma$ and $H \not\models_v \psi_2\sigma$. By the IH, we deduce $p \in [\![\psi_1\sigma, \rho]\!]$ and $p \in [\![\psi_2\sigma, \rho]\!]$, which implies $p \in [\![\psi_1\sigma, \rho]\!] \cap [\![\psi_2\sigma, \rho]\!] = [\![\psi_1\sigma \wedge \psi_2\sigma, \rho]\!] = [\![(\psi_1 \wedge \psi_2)\sigma, \rho]\!]$.
- Case $\varphi = \psi_1 \vee \psi_2$. Since $H \not\models_v (\psi_1 \vee \psi_2)\sigma$, then $H \not\models_v \psi_1\sigma \vee \psi_2\sigma$. By Lemma 12(iii), we know that either $H \not\models_v \psi_1\sigma$ or $H \not\models_v \psi_2\sigma$. Without loss of generality, suppose the former. By the IH, we obtain $p \in [\![\psi_1\sigma, \rho]\!]$, which implies $p \in [\![\psi_1\sigma, \rho]\!] \cup [\![\psi_2\sigma, \rho]\!] = [\![\psi_2\sigma \vee \psi_2\sigma, \rho]\!] = [\![(\psi_1 \vee \psi_2)\sigma, \rho]\!]$.
- Case $\varphi = \max X.\psi$. We know $H \not\models_v (\max X.\psi)\sigma$ where $(\max X.\psi)\sigma$ is a closed formula. By Definition A.9, we also know $(\max X.\psi)\sigma = \max X.\psi\sigma_{\{X\}}$, meaning that

  $$H \not\models_v \max X.\psi\sigma_{\{X\}} \tag{A.2}$$

  By Lemma 12(iv), we obtain

  $$H \not\models_v (\psi\sigma_{\{X\}})[^{\max X.\psi\sigma_{\{X\}}}/_X] \tag{A.3}$$

  By lemma 11 and the fact that $\max X.\psi\sigma_{\{X\}}$ is closed, we know that for $\sigma' = \sigma[^{\max X.\psi\sigma_{\{X\}}}/_X]$,

  $$(\psi\sigma_{\{X\}})[^{\max X.\psi\sigma_{\{X\}}}/_X] = \psi\sigma' \tag{A.4}$$

  From eqs. (A.3) and (A.4) we obtain that $H \not\models_v \psi\sigma'$.
  Let $S = \{q \mid H' \not\models_v \psi\sigma' \text{ for all } H' \subseteq T_q\}$. We show S is equal to $\{q \mid H' \not\models_v X\sigma' \text{ for all } H' \subseteq T_q\}$ as follows:

Pick $q \in S$ and $H' \subseteq T_q$. Then, we have

$$
\begin{aligned}
& H' \not\models_v \psi\sigma' \text{ by definition of S} \\
& \textit{iff } H' \not\models_v (\psi\sigma_{\{X\}})[^{\max X.\psi\sigma_{\{X\}}}/X] \text{ by } eq.\ (\text{A.4}) \\
& \textit{iff } H' \not\models_v \max X.\psi\sigma_{\{X\}} \text{ by Lemma 12(iv)} \\
& \textit{iff } H' \not\models_v X\sigma' \text{ because } X\sigma' = \sigma'(X) = \max X.\psi\sigma_{\{X\}}
\end{aligned}
$$

By the IH, we know that for all $q \in \textsc{Prc}$,

$$q \in S \text{ implies } q \in [\![\psi, \rho[X \mapsto S]]\!] \tag{A.5}$$

Since $p \in S$, we obtain $p \in [\![\psi, \rho[X \mapsto S]]\!]$, which implies that $p \in \bigcup\{S \mid S \subseteq [\![\psi, \rho[X \mapsto S]]\!]\} = [\![\max X.\psi, \rho]\!]$.

This completes our proof. □

We are now in a position to prove Theorem 5.9 from Section 5.2, which we restate below.

Theorem A.14 (Correspondence). For all (closed) formulae $\varphi \in \text{sHML}^{\vee}$ and systems $p \in \textsc{Prc}$,

$$p \notin [\![\varphi]\!] \quad \text{iff} \quad (\exists H \subseteq T_p \text{ such that } H \models_v \varphi) \qquad \blacksquare$$

Proof. Follows from Corollary A.6 and Lemma 7 below. □

B Proving Monitor Correctness

In this section, we prove the instrumentation and monitor properties from Chapter 4.

B.1 Instrumentation Properties

The proof of Proposition 4.1 relies on several technical lemmas that help us reason about the structure of the traces t in executing-monitors (t,m).

Lemma 1. If $(t,m) \xrightarrow{\alpha}_H (t',m')$ then $t' = t\alpha$.

Proof. By rule induction on $(t,m) \xrightarrow{\alpha}_H (t',m')$. □

Lemma 2. If $(t,m) \xrightarrow{\tau}_H (t',m')$ then $t = t'$.

Proof. By case analysis of the rules that that could have been used to derive $(t,m) \xrightarrow{\tau}_H (t',m')$. □

Lemma 3. If $(t,m)\ (\xrightarrow{\tau}_H)^* (t',m')$ then $t = t'$.

Proof. Follows from Lemma 2. □

Proposition B.3 (Veracity). For any history H, monitor m, system p, and sequence of actions $\eta_1,\ldots,\eta_n$,

$$\text{if } H \triangleright (\epsilon,m) \triangleleft p \xrightarrow{\eta_1} \cdots \xrightarrow{\eta_n} H' \triangleright (t,m') \triangleleft p' \text{ then } p \overset{t}{\Longrightarrow} p'$$

Proof. The proof proceeds by induction on n.

For the base case, when $n = 0$, the result is immediate.

For the inductive case, when $n = k+1$, the execution sequence can be expanded as follows:

$$H \triangleright (\epsilon,m) \triangleleft p \xrightarrow{\mu_1} \cdots \xrightarrow{\mu_k} H' \triangleright (t,m') \triangleleft p' \xrightarrow{\mu_{k+1}} H'' \triangleright (t',m'') \triangleleft p''$$

We show that $p \overset{t'}{\Longrightarrow} p''$.
By the IH and $H \triangleright (\epsilon,m) \triangleleft p \xrightarrow{\mu_1} \cdots \xrightarrow{\mu_k} H' \triangleright (t,m') \triangleleft p'$, we obtain that

$$p \overset{t}{\Longrightarrow} p' \tag{B.1}$$

By case analysis, $H' \triangleright (t,m') \triangleleft p' \xrightarrow{\mu_{k+1}} H'' \triangleright (t',m'') \triangleleft p''$ could have been derived via several rules:

- Using rule INo, then $p'' = p'$ and $t = t'$, which implies that $p' \overset{\epsilon}{\Longrightarrow} p''$. By eq. (B.1), we conclude that $p \overset{t}{\Longrightarrow} p' \overset{\epsilon}{\Longrightarrow} p''$, *i.e.*, $p \overset{t}{\Longrightarrow} p''$.

- Using rule iTer, then $t' = t\alpha$ and $p' \xrightarrow{\alpha} p''$ for some $\alpha \in \text{Act}$, which implies that $p' \overset{\alpha}{\Rightarrow} p''$. By eq. (B.1), we conclude that $p \overset{t}{\Rightarrow} p' \overset{\alpha}{\Rightarrow} p''$, *i.e.,* $p \overset{t\alpha}{\Longrightarrow} p''$.
- Using rule iAsS, then $t = t'$ and $p' \xrightarrow{\tau} p''$, which implies that $p' \overset{\epsilon}{\Rightarrow} p''$. By eq. (B.1), we conclude that $p \overset{t}{\Rightarrow} p' \overset{\epsilon}{\Rightarrow} p''$, *i.e.,* $p \overset{t}{\Rightarrow} p''$.
- Using rule iAsM, then $p' = p''$ and $(t,m') \xrightarrow{\tau}_H (t',m'')$. By Lemma 2, we obtain $t = t'$, and since $p' = p''$, we obtain $p' \overset{\epsilon}{\Rightarrow} p''$. Using eq. (B.1), we conclude $p \overset{t}{\Rightarrow} p' \overset{\epsilon}{\Rightarrow} p''$, *i.e.,* $p \overset{t}{\Rightarrow} p''$.
- Using rule iMon, then $p' \xrightarrow{\alpha} p''$ and $(t,m') \xrightarrow{\alpha}_H (t',m'')$ for some $\alpha \in \text{Act}$. By Lemma 1, we obtain that $t' = t\alpha$, and since $p' \xrightarrow{\alpha} p''$ we know that $p' \overset{\alpha}{\Rightarrow} p''$. Using eq. (B.1), we conclude $p \overset{t}{\Rightarrow} p' \overset{\alpha}{\Rightarrow} p''$, *i.e.,* $p \overset{t\alpha}{\Longrightarrow} p''$. □

B.2 Monitor Properties

In this section, we give the complete proofs for Propositions 4.2, 4.3, 4.5 and 4.8 from Chapter 4. However, we first give a few useful technical results about the executing-monitors of Figure 2.1.

Lemma 4. For any monitor $m,m',n \in \text{Mon}$, if $(t,m)\,(\xrightarrow{\tau}_H)^*\,(t,m')$, then $(t,m \odot n)\,(\xrightarrow{\tau}_H)^*\,(t,m' \odot n)$.

Proof. By induction on the number of τ-transitions. □

For the sake of completeness, we restate Proposition 4.2 below. This result asserts that a monitor that τ-transition cannot transition along other actions.

Proposition B.4 (τ-Race Absence). For all $\alpha \in \text{eAct}$, if $(t,m) \xrightarrow{\tau}_H (t,n)$ then $(t,m) \xrightarrow{\alpha}\!\!\!\!\!/\;_H$.

Proof. Proof is straightforward by case analysis. □

Proposition 4.3 below assures us that monitor behaviour is confluent w.r.t. τ-moves.

Proposition B.4 (τ-confluence). For monitors $m,m',m'' \in \text{Mon}$, trace $t \in \text{Trc}$ and history $H \in \text{Hst}$, if $(t,m) \xrightarrow{\tau}_H (t,m')$ and $(t,m) \xrightarrow{\tau}_H (t,m'')$, then there exist weak τ-moves $(t,m')\,(\xrightarrow{\tau}_H)^*\,(t,n)$ and $(t,m'')\,(\xrightarrow{\tau}_H)^*\,(t,n)$ for some monitor $n \in \text{Mon}$.

Proof. The proof proceeds by induction on $(t,m) \xrightarrow{\tau}_H (t,m')$.

- Case mVrP1. We have $(t,\text{no} \odot n) \xrightarrow{\tau}_H (t,n)$ where $t \in H$. The second transition $(t,\text{no} \odot n) \xrightarrow{\tau}_H (t,m'')$ could have been derived in two ways:
  - Using rule mVrP1, *i.e.,* $(t,\text{no} \odot n) \xrightarrow{\tau}_H (t,n)$, which requires o matching moves.
  - Using rule mTauR, *i.e.,* $(t,\text{no} \odot n) \xrightarrow{\tau}_H (t,\text{no} \odot n')$ and $(t,n) \xrightarrow{\tau}_H (t,n')$. Since $t \in H$, we know $(t,\text{no} \odot n') \xrightarrow{\tau}_H (t,n')$ by rule mVrP1. This and $(t,n) \xrightarrow{\tau}_H (t,n')$ give the required matching moves.
- Case mTauL. We have $(t,n_1 \odot n_2) \xrightarrow{\tau}_H (t,n_1' \odot n_2)$ because $(t,n_1) \xrightarrow{\tau}_H (t,n_1')$, which implies $n_1 \neq \text{no}$. The second transition $(t,n_1 \odot n_2) \xrightarrow{\tau}_H (t,m'')$ could have been derived using either of the following rules:
  - Rule mVrP1R, *i.e.,* $(t,n_1 \odot n_2) \xrightarrow{\tau}_H (t,n_1)$ where $n_2 = \text{no}$ and $t \in H$. By mVrP1R, we deduce $(t,n_1' \odot n_2) \xrightarrow{\tau}_H (t,n_1')$. This and $(t,n_1) \xrightarrow{\tau}_H (t,n_1')$ are the matching moves.

– Rule MVRP2R, *i.e.,* $(t,n_1 \odot n_2) \xrightarrow{\tau}_H (t,\text{no})$ where $n_2 = \text{no}$ and $t \notin H$. By rule MVRPR2, we deduce $(t,n_1' \odot n_2) \xrightarrow{\tau}_H (t,\text{no})$. This and $(t,\text{no})(\xrightarrow{\tau}_H)^{\text{o}}(t,\text{no})$ give the required matching moves.

– Rule MTAUL, *i.e.,* we have $(t,n_1 \odot n_2) \xrightarrow{\tau}_H (t,n_1'' \odot n_2)$ and $(t,n_1) \xrightarrow{\tau}_H (t,n_1'')$. By the IH, there exist moves $(t,n_1')(\xrightarrow{\tau}_H)^*(t,n)$ and $(t,n_1'')(\xrightarrow{\tau}_H)^*(t,n)$ for $n \in \text{MON}$. We obtain the matching moves, $(t,n_1' \odot n_2)(\xrightarrow{\tau}_H)^*(t,n \odot n_2)$ and $(t,n_1'' \odot n_2)(\xrightarrow{\tau}_H)^*(t,n \odot n_2)$, from Lemma 4.

– Rule MTAUR, *i.e.,* we have $(t,n_1 \odot n_2) \xrightarrow{\tau}_H (t,n_1 \odot n_2')$ and $(t,n_2) \xrightarrow{\tau}_H (t,n_1')$. The required matching moves, $(t,n_1' \odot n_2) \xrightarrow{\tau}_H (t,n_1' \odot n_2')$ and $(t,n_1 \odot n_2') \xrightarrow{\tau}_H (t,n_1' \odot n_2')$, follow by rules MTAUL and MTAUR.

This completes our proof. □

Corollary B.5. If $(t,m)\,(\xrightarrow{\tau}_H)^*\,(t,m')$ and $(t,m)\,(\xrightarrow{\tau}_H)^*\,(t,m'')$, then there must exist weak moves $(t,m')\,(\xrightarrow{\tau}_H)^*\,(t,n)$ and $(t,m'')\,(\xrightarrow{\tau}_H)^*\,(t,n)$ for some $n \in \text{MON}$.

Proof. Follows by repeatedly applying Proposition 4.3. □

For completeness's sake, we recall what it means for monitors to be equivalent up to τ-moves, Definition 4.4, and prove some properties about it.

Definition 4.4 (Monitor τ-Equivalence). The executing-monitors (t,m_1) and (t,m_2) are τ-equivalent w.r.t. history H, denoted as $(t,m_1) \approx_H (t,m_2)$, if there exists a common monitor $n \in \text{MON}$ such that $(t,m_1)\,(\xrightarrow{\tau}_H)^*\,(t,n)$ and $(t,m_2)\,(\xrightarrow{\tau}_H)^*\,(t,n)$. ■

Lemma 6. $\approx_H$ is an equivalence relation.

Proof. Proving $\approx_H$ is symmetric and reflexive is straightforward. To prove that $\approx_H$ is transitive, suppose that $(t,m_1) \approx_H (t,m_2) \approx_H (t,m_3)$. By Definition 4.4, we know there exist monitors n_1 and n_2 such that:

$$(t,m_1)\,(\xrightarrow{\tau}_H)^*\,(t,n_1) \text{ and } (t,m_2)\,(\xrightarrow{\tau}_H)^*\,(t,n_1)$$
$$(t,m_2)\,(\xrightarrow{\tau}_H)^*\,(t,n_2) \text{ and } (t,m_3)\,(\xrightarrow{\tau}_H)^*\,(t,n_2)$$

By Corollary B.5, we know that there also exists some monitor n such that $(t,n_1)\,(\xrightarrow{\tau}_H)^*\,(t,n)$ and $(t,n_2)\,(\xrightarrow{\tau}_H)^*\,(t,n)$, which implies that $(t,m_1)(\xrightarrow{\tau}_H)^*(t,n)$ and $(t,m_3)(\xrightarrow{\tau}_H)^*(t,n)$. Our result, namely $(t,m_1) \approx_H (t,m_3)$, follows by Definition 4.4. □

Lemma 7 below shows that two τ-equivalent monitors must be equal if they can transition along the same actions $\alpha \in \text{ACT}$. Moreover, the executing-monitors reached after performing that transition are also equal.

Lemma 7. If $(t,m) \xrightarrow{\alpha}_H (t',m')$ and $(t,n) \xrightarrow{\alpha}_H (t'',n')$ where $(t,m) \approx_H (t,n)$, then $m = n$ and $m' = n'$ and $t' = t''$.

Proof. Assume $(t,m) \xrightarrow{\alpha}_H (t',m')$ and $(t,n) \xrightarrow{\alpha}_H (t'',n')$ where $(t,m) \approx_H (t,n)$. By Definition 4.4, there exists some n'' such that $(t,m)\,(\xrightarrow{\tau}_H)^*\,(t,n'')$ and $(t,n)\,(\xrightarrow{\tau}_H)^*\,(t,n'')$. But by Proposition 4.2, we also know $(t,m) \xrightarrow{\tau}\!\!\!\!\!\!/\;_H$ and $(t,n) \xrightarrow{\tau}\!\!\!\!\!\!/\;_H$, which implies that $(t,m)\,(\xrightarrow{\tau}_H)^{\text{o}}\,(t,n'')$ and $(t,n)\,(\xrightarrow{\tau}_H)^{\text{o}}\,(t,n'')$, and thus $m = n' = n$.

To show that if $(t,m) \xrightarrow{\alpha}_H (t',m')$ and $(t,m) \xrightarrow{\alpha}_H (t'',n')$ then $t' = t''$ and $(t',m') \approx_H (t'',n')$, we use rule induction on $(t,m) \xrightarrow{\alpha}_H (t',m')$. We outline the main cases:

- Case mEnd. We have $(t,\mathsf{end}) \xrightarrow{\alpha}_H (t,\mathsf{end})$ where $m = \mathsf{end}$. Result follows immediately since the second transition $(t,\mathsf{end}) \xrightarrow{\alpha}_H (t'',n')$ could have only been derived using the rule mEnd, which implies $t'' = t$ and $m'' = \mathsf{end}$.
- Case mPar1. We have $(t,m_1 \odot m_2) \xrightarrow{\alpha}_H (t',m_1' \odot m_2')$ where $m = m_1 \odot m_2$ because $(t,m_1) \xrightarrow{\alpha}_H (t',m_1')$ and $(t,m_2) \xrightarrow{\alpha}_H (t',m_2')$. By Proposition 4.2, we know $(t,m_1) \xrightarrow{\tau}\!\!\!\!\!/\;_H$ and $(t,m_2) \xrightarrow{\tau}\!\!\!\!\!/\;_H$, which implies $m_1 \neq \mathsf{no}$ and $m_2 \neq \mathsf{no}$. This means that the second transition $(t,m_1 \odot m_2) \xrightarrow{\alpha}_H (t'',n')$ could have only been derived by mPar1. Thus, we infer that $n' = n_1 \odot n_2$, $(t,m_1) \xrightarrow{\alpha}_H (t'',n_1)$ and $(t,m_2) \xrightarrow{\alpha}_H (t'',n_2)$. Our result, $t' = t''$ and $m' = m''$, follows by the IH.

The remaining cases follow with analogous arguments. □

Similarly, τ-equivalent monitors must be equal if they can (weakly) transition with the same trace $u \in \mathrm{Trc}$, in which case the executing-monitors reached are also equal.

Lemma 8. For all $u \in \mathrm{Trc}$, if $(t,m_1) \overset{u}{\Longrightarrow}_H (t_1,n_1)$ and $(t,m_2) \overset{u}{\Longrightarrow}_H (t_2,n_2)$ where $(t,m_1) \approx_H (t,m_2)$, then $t_1 = t_2$ and $(t_1,n_1) \approx_H (t_2,n_2)$.

Proof. The proof proceeds by induction on the length l of transitions in $(t,m_1) \overset{u}{\Longrightarrow}_H (t_1,n_1)$.

- For the *base case*, suppose $l = 0$. Then $u = \epsilon$, $m_1 = n_1$ and $(t,m_2)(\xrightarrow{\tau}_H)^*(t_2,n_2)$. By Lemma 3, we know $t = t_2$. By this and Definition 4.4, we also know $(t,m_2) \approx_H (t_2,n_2) = (t,n_2)$. Since $(t,m_1) \approx_H (t,m_2)$, our result, $(t,m_1) \approx_H (t_2,n_2)$, follows via Lemma 6 (transitivity).
- For the *inductive case*, suppose $l = k+1$. The transition sequence $(t,m_1) \overset{u}{\Longrightarrow}_H (t_1,n_1)$ can be expanded as

$$(t,m_1) \xrightarrow{\mu}_H (v_1,n_1') \overset{u'}{\Longrightarrow}_H (t_1,n_1)$$

  where $u', v_1 \in \mathrm{Trc}$, $n_1' \in \mathrm{Mon}$ and $\mu \in \mathrm{Act} \cup \{\tau\}$. There are two subcases to consider:

  – When $\mu = \tau$, we have $(t,m_1) \xrightarrow{\tau}_H (v_1,n_1')$ and $u = u'$, which implies $t = v_1$ by Lemma 3 and $(t,m_1) \approx_H (v_1,n_1')$ by Definition 4.4. By $(t,m_1) \approx_H (t,m_2)$ and $(t,m_1) \approx_H (v_1,n_1')$ and Lemma 6, we obtain $(v_1,n_1') \approx_H (t,m_2)$. By $(t,n_1') \overset{u}{\Longrightarrow}_H (t_1,n_1)$, the original assumption $(t,m_2) \overset{u}{\Longrightarrow}_H (t_2,n_2)$ and IH, we conclude $t_1 = t_2$ and $(t_1,n_1) \approx_H (t_2,n_2)$.

  – When $\mu = \alpha \in \mathrm{Act}$, we have $(t,m_1) \xrightarrow{\alpha} (v_1,n_1')$ and $u = \alpha u'$ for some $u' \in \mathrm{Trc}$. The second sequence $(t,m_2) \overset{u}{\Longrightarrow}_H (t_2,n_2)$ can be expanded as

$$(t,m_2)\,(\xrightarrow{\tau}_H)^*\,(t,n_2') \xrightarrow{\alpha}_H (v_2,n_2'') \overset{u'}{\Longrightarrow}_H (t_2,n_2)$$

  where $v_2 \in \mathrm{Trc}$. By Definition 4.4, we also know $(t,m_2) \approx_H (t,n_2')$. From this, the original assumption that $(t,m_1) \approx_H (t,m_2)$ and Lemma 6, we deduce $(t,m_1) \approx_H (t,n_2')$. Since $(t,m_1) \xrightarrow{\alpha}_H (v_1,n_1')$ and $(t,n_2') \xrightarrow{\alpha}_H (v_2,n_2'')$ where $(t,m_1) \approx_H (t,n_2')$, we obtain that $m_1 = n_2'$ and $n_1' = n_2''$ and $v_1 = v_2$ by Lemma 7. Our result, $t_1 = t_2$ and $(t_1,n_1) \approx_H (t_2,n_2)$, follows by the IH.

This completes our proof. □

We can now prove Proposition 4.5 from Chapter 4, restated below.

Proposition B.8 (Monitor Determinism). For all traces u,t,t_1,t_2, history H, and monitors m,n_1,n_2,

$$\text{if } (t,m) \overset{u}{\Longrightarrow}_H (t_1,n_1) \text{ and } (t,m) \overset{u}{\Longrightarrow}_H (t_2,n_2) \text{ then } t_1 = t_2 \text{ and } (t_1,n_1) \approx_H (t_2,n_2)$$

Proof. Assume that $(t,m) \overset{u}{\Rightarrow}_H (t',m')$ and $(t,m) \overset{u}{\Rightarrow}_H (t'',m'')$. By Lemma 6, we know that $(t,m) \approx_H (t,m)$. By Lemma 8, we obtain that $t' = t''$ and $(t',m') \approx_H (t'',m'')$. Our result follows by Definition 4.4. □

We now show monitor rejections are irrevocable in terms of both additional traces, *width*, and longer traces, *length*, formalised in Propositions 4.7 and 4.8, which are restated below.

Proposition B.8 (Length Irrevocability). For any history H, traces t,u and monitor m,

$$\text{if } \mathbf{rej}((H,t),m) \text{ then } \mathbf{rej}((H,tu),m)$$

Proof. The proof proceeds by induction on $\mathbf{rej}((H,t),m)$.

- Case NO. Follows immediately because $\mathbf{rej}(H',\text{no})$ for all $H' \neq \emptyset$.
- Case ACT. We know $\mathbf{rej}((H,t),\alpha.m)$ because $\mathbf{rej}(H',m)$ where $H' = \mathit{suffix}((H,t),\alpha)$. There are two subcases to consider:
  - When $t = \alpha t'$, then $H' = (H'',t') = \mathit{suffix}((H,t),\alpha)$ for some H''. By the IH, we deduce $\mathbf{rej}((H'',t'u),m)$. But by definition, we also know $(H'',t'u) = \mathit{suffix}((H,tu),\alpha)$, meaning that $\mathbf{rej}(\mathit{suffix}((H,tu),\alpha),m)$. Our result, $\mathbf{rej}((H,tu),\alpha.m)$, follows by rule ACT.
  - When $t = \beta t'$ where $\beta \neq \alpha$, we know by definition that $\mathit{suffix}((H,t),\alpha) = \mathit{suffix}(H,\alpha) = \mathit{suffix}((H,tu),\alpha)$. Our result, $\mathbf{rej}((H,tu),\alpha.m)$, follows immediately by applying rule ACT.
- Case PARAL. We know that $\mathbf{rej}((H,t),m' \otimes m'')$ because of $\mathbf{rej}((H,t),m')$. By the IH, we obtain $\mathbf{rej}((H,tu),m')$. Using rule PARAL, we can conclude $\mathbf{rej}((H,tu),m' \otimes m'')$.
- Case PARAR. Proof is analogous to that for PARAR.
- Case PARO. We know $\mathbf{rej}((H,t),m' \oplus m'')$ because $\mathbf{rej}((H,t),\text{true},m')$ and $\mathbf{rej}((H,t),m'')$. By the IH, $\mathbf{rej}((H,tu),m')$ and $\mathbf{rej}((H,tu),m'')$. Applying rule PARO, we obtain $\mathbf{rej}((H,tu),m' \oplus m'')$.
- Case REC. We know $\mathbf{rej}((H,t),\text{rec}\,X.m)$ because $\mathbf{rej}((H,t),m[^{\text{rec}X.m}/_X])$. By the IH, we obtain $\mathbf{rej}((H,tu),m[^{\text{rec}X.m}/_X])$. Our result, $\mathbf{rej}((H,tu),\text{rec}\,X.m)$, follows by rule REC. □

C Proving Maximal Expressiveness w.r.t. any monitoring setup

In this section, we prove Theorem 5.21 from Chapter 5. Before we embark on this endeavour, we first show how to synthesize monitors for recHML formulas (in contrast to Definition 5.3, which only generates monitors for formulae in sHML$^\vee$). Not all these formulas from recHML are monitorable, according to Definition 5.1. As a result, our synthesis does not guarantee monitors that monitor completely for the corresponding formulas. Instead, our synthesis uses ideas and techniques from [13] to generate *optimal* monitors. For completeness's sake, we also recall Definitions 5.18 and 5.20 from Chapter 5, restated below.

Definition 5.18 (Bad History). History H is *bad* for the (closed) formula φ if for all systems $p \in \text{Prc}$ such that $H \subseteq T_p$ then $p \notin [\![\varphi]\!]$. ■

Definition 5.20 (Monitorable via Bad Histories). A (closed) formula φ is *monitorable via bad histories* if for every $p \in \text{Prc}$, $p \notin [\![\varphi]\!]$ iff $(\exists H \subseteq T_p$ such that H is bad for $\varphi)$. ■

Definition C.1. Monitor m *monitors optimally* for the (closed) formula $\varphi \in$ recHML if it can do so soundly and, additionally, $\mathbf{rej}(H,m)$ for every bad H for φ. ■

Assumptions about our formulas. We can assume that recursion is guarded in every formula φ [96], in that for every subformula $\mathsf{min}X.\psi$ or $\mathsf{max}X.\psi$ of φ, variable X is in the scope of a modal operator. We assume a reasonable closure operator $\mathsf{cl}(-)$ on the subformulas of φ, such that $\mathsf{cl}(X) = \mathsf{cl}(\mathsf{min}X.\psi)$ or $\mathsf{cl}(X) = \mathsf{cl}(\mathsf{max}X.\psi)$, if $\mathsf{min}X.\psi$ or, respectively, $\mathsf{max}X.\psi$ is a subformula of φ.

Since recHML-satisfiability is decidable (in exponential time) [94], we also assume that for every subformula ψ of φ, either $\mathsf{cl}(\psi)$ is satisfiable or $\psi = \mathsf{ff}$.
Finally, we also assume that in φ, conjunctions and diamonds only appear in the context of subformulas of the following form, where $A \subseteq \text{Act}$ and B_α is a set of formulas indexed by α:

$$\bigwedge_{\alpha \in A} \left(\Big(\bigwedge_{\psi \in B_\alpha} \langle \alpha \rangle \psi \Big) \wedge [\alpha] \psi'_\alpha \right)$$

This is a more relaxed condition than the disjunctive form from [128], where Walukiewicz gives a method to turn every recHML formula to an equivalent one in disjunctive form. On deterministic systems, we have that $\langle \alpha \rangle \varphi \equiv \langle \alpha \rangle \mathsf{tt} \wedge [\alpha] \varphi$, resulting in formulae of the form

$$\bigwedge_{\alpha \in A} \left(\Big(\bigwedge_{\psi \in B_\alpha} (\langle \alpha \rangle \mathsf{tt} \wedge [\alpha] \psi) \Big) \wedge [\alpha] \psi'_\alpha \right)$$

This can be rewritten in the more general form below:

$$\bigwedge_{\alpha\in A}\langle\alpha\rangle\mathsf{tt}\wedge\bigwedge_{\beta\in B}[\beta]\psi_\beta \tag{C.1}$$

Therefore, with the assumption that conjunctions and diamonds only appear in the context of subformulas of the form in (C.1), we all recHML formulas can be expressed as formulas in $\text{recHML}_{\text{NF}}$, Definition C.2 and Proposition C.3.

Definition C.2. The logical fragment $\text{recHML}_{\text{NF}}$ is defined inductively as follows:

$$\varphi,\psi\in\text{recHML}_{\text{NF}} ::= \mathsf{tt}\mid\mathsf{ff}\mid[\alpha]\varphi\mid\varphi\vee\psi\mid\bigwedge_{\alpha\in A}\langle\alpha\rangle\mathsf{tt}\wedge\bigwedge_{\beta\in B}[\beta]\psi_\beta\mid\min X.\varphi\mid\max X.\varphi\mid X \qquad \blacksquare$$

Proposition C.3. For all $\varphi\in\text{recHML}$, $\exists\varphi'\in\text{recHML}_{\text{NF}}$ such that $[\![\varphi]\!]=[\![\varphi']\!]$. $\blacksquare$

We define $\mathsf{s}(H)$ as the sum of the lengths of the traces in H. Formally: $\mathsf{s}(H)=\sum_{t\in H}|t|$ where $|\epsilon|=0$ and $|\alpha t'|=1+|t'|$. In particular, $\mathsf{s}(\emptyset)=0$, but in what follows, this does not arise since bad histories for formulae cannot be empty (later shown in Lemma 6).

We also define $k(\psi)$ to be the length of the longest path in the syntax tree of $\mathsf{cl}(\psi)$ until we find a modal operator. Formally: $k(\mathsf{ff})=k(\mathsf{tt})=k(\langle\alpha\rangle\psi)=k([\alpha]\psi)=0$; $k(\max X.\psi)=1+k(\psi)$; $k(X)=1+k(\max X.\psi)$ or $k(X)=1+k(\min X.\psi)$; and $k(\psi_1\wedge\psi_2)=k(\psi_1\vee\psi_2)=1+\max\{k(\psi_1),k(\psi_2)\}$. We know that $k(\psi)$ is always finite, as we have assumed that recursion is guarded in φ.

Synthesis of optimal monitors We define a new monitor synthesis function, $(\!|-|\!)^e:\text{recHML}_{\text{NF}}\to\textsc{Mon}$, by extending the synthesis function $(\!|-|\!)$ in Definition 5.3 to all $\text{recHML}_{\text{NF}}$formulas such that

$$(\!|\min X.\varphi|\!)^e\stackrel{\text{def}}{=}\mathsf{rec}X.(\!|\varphi|\!)^e\quad\text{and}\quad(\!|\langle\alpha\rangle\mathsf{tt}|\!)^e\stackrel{\text{def}}{=}\mathsf{end}$$

We also define a new violation relation, $\models_v^e$, by extending the rules of the violation relation in Definition 5.7 to treat the case of $\min X.\varphi$ as $\max X.\varphi$.

We show that for any $\varphi\in\text{recHML}_{\text{NF}}$, the generated monitor $(\!|\varphi|\!)^e$ monitors optimally for it, according to Definition C.1, formalised in Theorem C.8. This relies on two important technical results. Specifically, Lemma 4 shows the rejected histories, $\mathbf{rej}(H,m)$, and that the histories along which systems violate formulae, $(p,H)\models_v^e\psi$, coincide. Lemma 7 then further shows that there is also a tight correspondence between bad histories, H bad for φ, and rejected histories, $\mathbf{rej}(H,m)$.

Lemma 4. For any $\varphi\in\text{recHML}_{\text{NF}}$, $p\in\textsc{Prc}$ and $H\subseteq T_p$, we have $\mathbf{rej}((\!|\varphi|\!)^e,H)$ iff $H\models_v^e\varphi$.

Proof. We prove this in two steps. For the *if* direction, assume $H\models_v^e\varphi$. Note that, when $\varphi=\langle\alpha\rangle\mathsf{tt}$, then $H\not\models_v^e\varphi$ since there is no rule justifying $H\models_v^e\langle\varphi\rangle\mathsf{tt}$. The proof proceeds by rule induction on $H\models_v^e\varphi$ as follows:

- When $H\models_v^e\min X.\varphi$ because $H\models_v^e\varphi[^{\min X.\varphi}/_X]$. The proof is similar to that for vMax in Lemma 11.
- The remaining cases are identical to those in Lemma 11.

For the *only if* direction, assume that $\mathbf{rej}(\langle\!|\varphi|\!\rangle^e, H)$. Note that, when $\varphi = \langle\alpha\rangle\mathsf{tt}$, then $\langle\!|\varphi|\!\rangle^e = \mathsf{end}$ and $\neg\mathbf{rej}(H,\mathsf{end})$ since there there is no rule in Figure 3.3 justifying $\mathbf{rej}(H,\mathsf{end})$. The proof proceeds by rule induction on $\mathbf{rej}(H, \langle\!|\varphi|\!\rangle^e)$ and is identical to that in Lemma 10. □

Before proving Lemma 7 (below), we give two helpful technical results, namely Lemma 5 and Lemma 6.

Lemma 5. For any $H \subseteq \textsc{Hst}$ and $\alpha \in \textsc{Act}$ such that $\mathsf{s}(H) \neq \mathsf{o}$, then $\mathsf{s}(\mathit{suffix}(H,\alpha)) < \mathsf{s}(H)$.

Proof. Proof is straightforward. □

Lemma 6. If H is bad for φ then $H \neq \emptyset$.

Proof. Suppose H is bad for φ. Expanding Definition 5.18, we have that $\forall q.\ (H \subseteq T_q \implies q \notin [\![\varphi]\!])$ Suppose, for contradiction, that $H = \emptyset$. Since $\emptyset \subseteq T_q$ for all systems q, from our assumption, we get that $q \notin [\![\varphi]\!]$ for all systems q, a contradiction. Thus, we can conclude $H \neq \emptyset$. □

Lemma 7. For every formula $\varphi \in \textsc{recHML}_{\textsc{nf}}$, system $p \in \textsc{Prc}$ and history $H \subseteq T_p$, we have that H is bad for φ iff $H \models^e_v \varphi$.

Proof. We prove this in two steps:

For the "only if"case, we prove that for every subformula ψ of φ, system p, and $H \subseteq T_p$, if $\forall q.\ (H \subseteq T_q \implies q \notin [\![\mathsf{cl}(\psi)]\!])$ then $(p,H) \models^e_v \mathsf{cl}(\psi)$.

Pick a subformula ψ of φ, system $p \in \textsc{Prc}$ and history $H \subseteq T_p$, and assume that

$$\forall q.\ (H \subseteq T_q \implies q \notin [\![\mathsf{cl}(\psi)]\!]) \tag{C.2}$$

Since $p \in \textsc{Prc}$ and $H \subseteq T_p$, then by eq. (C.2), we also know that

$$p \notin [\![\psi]\!] \tag{C.3}$$

We want to show that $H \models^e_v \mathsf{cl}(\psi)$.
We proceed by lexicographic induction on $(\mathsf{s}(H), k(\psi))$.

- For the *base case*, when $\mathsf{s}(H) = \mathsf{o}$, we have that $H = \{\epsilon\}$ (since $H \neq \emptyset$ by Lemma 6). By definition of histories, then $H \subseteq T_q$ for every system q, and by eq. (C.2), we obtain $q \notin [\![\mathsf{cl}(\psi)]\!]$. This means that $\mathsf{cl}(\psi)$ is not satisfiable. Thus, due to our assumptions about φ, we have $\psi = \mathsf{ff}$, which implies $H \models^e_v \mathsf{cl}(\psi)$.
- For the *inductive case*, when $\mathsf{s}(H) = k{+}1$, we take cases for ψ.
  - If $\psi = \langle\alpha\rangle\mathsf{tt}$, then from (C.3), it is not hard to see that for $q = \alpha.\mathsf{nil}$, the system $p + q$ is a deterministic, $p + q \in [\![\psi]\!]$, and $H \subseteq T_p \subseteq T_{p+q}$, contradicting the assumption in (C.2). Thus, $\psi \neq \langle\alpha\rangle\mathsf{tt}$.
  - If $\psi = [\alpha]\psi'$, then it is not hard to see that $\forall q$ such that $p \stackrel{\alpha}{\Longrightarrow} q$, we have that $(\mathit{suffix}(H,\alpha) \subseteq T_q \implies q \notin [\![\mathsf{cl}(\psi')]\!])$. Using Lemma 5, we also know that $\mathsf{s}(\mathit{suffix}(H,\alpha)) < \mathsf{s}(H)$. Thus, by the IH, we obtain that $\mathit{suffix}(H,\alpha) \models^e_v \mathsf{cl}(\psi')$. Our result, $H \models^e_v \mathsf{cl}([\alpha]\psi')$, follows by rule vUm.

- If $\psi = \bigwedge_{\alpha\in A}\langle\alpha\rangle \mathrm{tt} \wedge \bigwedge_{\alpha\in B}[\alpha]\psi_\beta$, then it suffices to prove that $\forall q$ such that $p \overset{\alpha}{\Longrightarrow} q$, we have that $(\mathit{suffix}(H,\alpha) \subseteq T_q \implies q \notin [\![\mathrm{cl}(\psi_\alpha)]\!])$ for some $\alpha \in B$.
  If that is not the case, then for every $\alpha \in B$, there exists some $q_\alpha \in [\![\mathrm{cl}(\psi_\alpha)]\!]$ such that $\mathit{suffix}(H,\alpha) \subseteq T_{q_\alpha}$. Let $A' = A \setminus B$ and $q = \sum_{\alpha\in A'}\alpha.\mathrm{nil} + \sum_{\alpha\in B}\alpha.q_\alpha$. Then q is a deterministic system, $H \subseteq T_q$ and $q \in [\![\mathrm{cl}(\psi)]\!]$, which contradicts the assumption in (C.2).
- The remaining cases are straightforward.

For the "if" case, assume $H \models^e_v \varphi$. Then we can prove that for every $H \subseteq T_p$, we have $p \notin [\![\varphi]\!]$. The proof is straightforward by induction on the derivation of $H \models^e_v \varphi$. □

We are now in a position to prove that the generated monitor $(\!|\varphi|\!)^e$ monitors optimally for φ, according to Definition C.1.

Theorem C.8. For every (closed) formula $\varphi \in \textsc{recHML}_{\textsc{nf}}$, $(\!|\varphi|\!)^e$ monitors optimally for φ.

Proof. Suppose $\varphi \in \textsc{recHML}$. Expanding Definition C.1, we have to show that $(\!|\varphi|\!)^e$ monitors soundly for φ and that $\mathbf{rej}(H,(\!|\varphi|\!)^e)$ for every bad H for φ.

To show soundness, suppose $p \in \textsc{Prc}$ and $H \subseteq T_p$ such that $\mathbf{rej}(H,(\!|\varphi|\!)^e)$. By Lemma 4, we obtain $H \models^e_v \varphi$, and by Lemma 7, we know that H is bad for φ. But $H \subseteq T_p$, which implies that $p \notin [\![\varphi]\!]$ (since H is bad for φ).

To show $\mathbf{rej}(H,(\!|\varphi|\!)^e)$ for every bad H for φ, suppose $p \in \textsc{Prc}$ and $H \subseteq T_p$ such that H is a bad history for φ. By Lemma 7, we deduce that $H \models^e_v \varphi$, and by Lemma 4, we conclude $\mathbf{rej}(H,(\!|\varphi|\!)^e)$. □

As a consequence, we see that $\mathrm{sHML}^\vee$ characterizes the class of all $\textsc{recHML}_{\textsc{nf}}$ formulas—and, by extension, recHML formulae—that are monitorable via bad histories. Put differently, every property that is monitorable via bad histories and can be described by a recHML formula, can also be described with an $\mathrm{sHML}^\vee$ formula. This is a stronger statement than Section 5.3, as it does not depend on the monitor model.

Theorem C.9 (General Maximality). If a language $\mathcal{L} \subseteq \textsc{recHML}$ is monitorable via bad histories, then $\mathcal{L}$ cannot (semantically) express more properties than $\mathrm{sHML}^\vee$, *i.e.*, $\mathcal{L} \sqsubseteq \mathrm{sHML}^\vee$. ■

Proof. Suppose that $\varphi \in \textsc{recHML}$ is monitorable via bad histories and let $p \notin [\![\varphi]\!]$. By Proposition C.3, we know there exists $\psi \in \textsc{recHML}_{\textsc{nf}}$ such that $[\![\varphi]\!] = [\![\psi]\!]$ and $p \notin [\![\psi]\!]$ and ψ is monitorable via bad histories. Our result follows by Section 5.3 if we can show that ψ is monitorable (Definition 5.1); that is, there exists a monitor $m \in \textsc{Mon}$ that monitors soundly and completely for ψ.

From Definition 5.20, we know that there exists $H \subseteq T_p$ such that H is bad for ψ. Using Theorem C.8, we obtain that $(\!|\psi|\!)^e$ is an optimal monitor for ψ. Expanding Definition C.1, this means that

$$(\!|\psi|\!)^e \text{ monitors soundly for } \psi \tag{C.4}$$

$$\mathbf{rej}(H,(\!|\psi|\!)^e) \text{ for every bad } H \text{ for } \psi \tag{C.5}$$

Since $p \notin [\![\psi]\!]$ and H is bad for ψ, then by (C.5), we obtain $\mathbf{rej}(H,(\!|\psi|\!)^e)$, which implies $\mathbf{rej}(p,(\!|\psi|\!)^e)$. Thus, $(\!|\psi|\!)^e$ also monitors completely for ψ. □

Appendix II

Repeated Monitoring for Non-Deterministic Systems Results

D General Results

In this chapter, we prove some general results that will be used throughout the proofs in this part of the appendix. Concretely, Appendix D.1 focuses on the ILTSs of Chapter 6 whereas Appendix D.2 focuses on the violation relation $\models_{\text{DET}}$ of Definition 9.8 from Section 9.2.

D.1 LTS Properties

We prove some general results about the ILTS of Chapter 6.

Lemma 1. Whenever $p \xrightarrow{\tau} p'$ and $p \overset{t}{\Rightarrow}_T p''$ where $t \in \text{TACT}^*$ then either

- $\tau = \epsilon$ and $p' \equiv q'$; or
- there exist moves $p' \overset{t}{\Rightarrow}_T q$ and $p'' \xrightarrow{\tau} q$.

Proof. Follows from the confluence property of our ILTSes: silent (τ)-transitions are confluent w.r.t. other actions (Chapter 6). □

Lemma 2. If $p \xrightarrow{\tau} q$ then $T_p = T_q$.

Proof. Let $p \xrightarrow{\tau} q$. We show that $T_p = T_q$ in two parts.

- To show that $T_p \subseteq T_q$, suppose $t \in T_p$, *i.e.,* $p \overset{t}{\Rightarrow}_T p'$ for some p'. We show $t \in T_q$, that is $q \overset{t}{\Rightarrow}_T q'$ for some q'. This required matching move follows from Lemma 1.
- To show that $T_q \subseteq T_p$, suppose $t \in T_q$. We show $t \in T_p$, *i.e.,* $p \overset{t}{\Rightarrow}_T p'$ for some p'. The required matching move is $p \xrightarrow{\tau} q \overset{t}{\Rightarrow}_T q'$. □

Corollary D.3. If $p \Rightarrow_T q$ then $T_p = T_q$. ■

Lemma 4. If $p \equiv q$ then $T_p = T_q$.

Proof. Suppose $p \equiv q$. We show that $T_p = T_q$ in two parts.

- To show $T_p \subseteq T_q$, suppose $t \in T_p$, *i.e.,* $p \overset{t}{\Rightarrow}_T p'$ for some p'. By definition of $\equiv$, we know $\exists q'$ such that $q \overset{t}{\Rightarrow}_T q'$ and $p' \equiv q'$, which means $t \in T_q$.
- To show $T_q \subseteq T_p$, suppose $t \in T_q$. The proof for showing $t \in T_p$ is analogous.

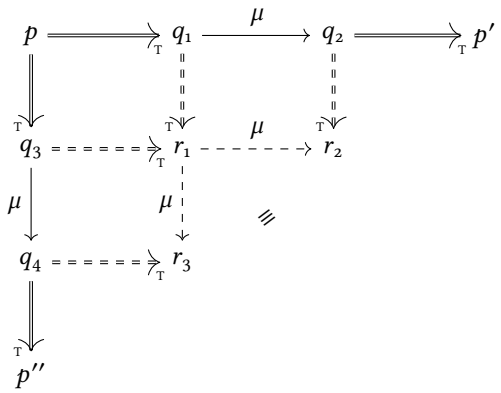
□

Lemma 5. For all $\mu \in \text{TAct}$, if $p \overset{\mu}{\Rightarrow}_T p'$ and $p \overset{\mu}{\Rightarrow}_T p''$ and $\text{Det}(\mu) = \text{true}$ then $T_{p'} = T_{p''}$.

Proof. Suppose that $p \overset{\mu}{\Rightarrow}_T p'$ and $p \overset{\mu}{\Rightarrow}_T p''$. By definition, $\exists q_1, q_2, q_3, q_4$ such that

$$p \Rightarrow_T q_1 \xrightarrow{\mu} q_2 \Rightarrow_T p' \quad \text{and} \quad p \Rightarrow_T q_3 \xrightarrow{\mu} q_4 \Rightarrow_T p''$$

We have to show that $T_{p'} = T_{p''}$. Repeatedly using the property of our ILTS that silent actions are confluent w.r.t. other actions (Chapter 6) and the assumption that $\text{Det}(\mu) = \text{true}$, we obtain the dashed transitions in the diagram below.



By Lemma 4, we know $T_{r_2} = T_{r_3}$. By Corollary D.3, we also know $T_{p'} = T_{q_2} = T_{r_2}$ and $T_{p''} = T_{q_4} = T_{r_3}$. We can thus conclude that $T_{p'} = T_{p''}$. □

Proposition D.6 shows the relation between the two forms of weak transitions, namely $\Rightarrow$ and $\Rightarrow_T$.

Proposition D.6. For all systems $p, q \in \text{Prc}$, external actions $\alpha \in \text{EAct}$ and internal actions $\gamma \in \text{IAct}$,

(i) if $p \Rightarrow_T q$ then $p \Rightarrow q$;
(ii) if $p \overset{\gamma}{\Rightarrow}_T q$ then $p \Rightarrow q$;
(iii) if $p \Rightarrow q$ then $p \overset{t}{\Rightarrow}_T q$ for some $t \in \text{IAct}^*$;
(iv) if $p \overset{\alpha}{\Rightarrow} q$ then $p \overset{t\alpha t'}{\Longrightarrow}_T q$ for some $t, t' \in \text{IAct}^*$;
(v) if $p \overset{\alpha}{\Rightarrow}_T q$ then $p \overset{\alpha}{\Rightarrow} q$.

Proof. We prove the above as follows:

To prove (1) straightforward by definition.

To prove (2) suppose $p \overset{\gamma}{\Rightarrow}_T q$. By definition, $\exists p', p''$ such that $p \Rightarrow_T p' \xrightarrow{\gamma} p'' \Rightarrow_T q$. By (1), we obtain $p \Rightarrow p' \xrightarrow{\gamma} p'' \Rightarrow q$, *i.e.*, $p \Rightarrow q$.

To prove (3) suppose $p \Rightarrow q$. The proof proceeds by induction on the number of (strong) transitions n. For the base case (*i.e.*, $n = 0$), then $p = q$ and $p \overset{\epsilon}{\Rightarrow}_T q$. For the inductive case (*i.e.*, $n = k + 1$), then either $\exists p'$ such that $p \xrightarrow{\tau} p' \Rightarrow q$ or $\exists p', \gamma$ such that $p \xrightarrow{\gamma} p' \Rightarrow q$. For the first subcase, by the IH, we obtain $p' \overset{t}{\Rightarrow}_T q$ for some $t \in \text{IAct}^*$, which implies $p' \overset{t}{\Rightarrow}_T q$. For the second subcase, by the IH, we obtain $p' \overset{t}{\Rightarrow}_T q$, which implies $p \overset{\gamma t}{\Rightarrow}_T q$.

To prove (4) suppose $p \overset{\alpha}{\Rightarrow} q$. By definition, $\exists p', p''$ such that $p \Rightarrow p' \xrightarrow{\alpha} p'' \Rightarrow q$. By (3), we obtain $p \overset{t}{\Rightarrow}_T p' \xrightarrow{\alpha} p'' \overset{t'}{\Rightarrow}_T q$ for some $t, t' \in \text{IAct}^*$, which means $p \overset{t\alpha t'}{\Longrightarrow}_T q$, as required.

To prove (5) straightforward by definition and (1). □

We also prove Proposition 6.5 restated below, stating that equivalent systems satisfy the same formulae.

Proposition D.6 (Behavioural Equivalence). For all (closed) formulae $\varphi \in \text{recHML}$ and systems $p, q \in \text{Prc}$,

$$\text{if } p \in [\![\varphi]\!] \text{ and } p \equiv q \text{ then } q \in [\![\varphi]\!]$$

Proof. Suppose $p \in [\![\varphi]\!]$ and $p \equiv q$. By definition, p and q are also strongly bisimilar [3]. Our result, $q \in [\![\varphi]\!]$, then follows by the well-known result that strong bisimulation preserves formula satisfactions. □

D.2 Properties of the Violation Relation

We prove some properties of the violation relation $\models_{\text{Det}}$ of Definition 9.8, including Theorems 9.9 and 9.10 from Section 9.2. We start with some general results.

Lemma 7. If $(H, \mathrm{f}) \models_{\text{Det}} \varphi$ then $H \neq \emptyset$.

Proof. Straightforward by rule induction. □

We show that the violation relation $\models_v$ observes sanity checks akin to those for the history analysis of Figure 3.3. In particular, Propositions D.8 and D.9 below guarantee that once a system violates a formula via a history, it will persistently violate that formula, regardless of any other behaviour it might exhibit (described in terms of additional traces added to the history, width, or longer trace prefixes, length).

Proposition D.8 (Width Irrevocability). If $(H, \mathrm{f}) \models_{\text{Det}} \varphi$ then $(H \cup H', \mathrm{f}) \models_{\text{Det}} \varphi$.

Proof. The proof is similar to that for Proposition 8.2. □

Proposition D.9 (Length Irrevocability). If $(H \cup t, \mathrm{f}) \models_{\text{Det}} \varphi$ then $(H \cup \{tu\}, \mathrm{f}) \models_{\text{Det}} \varphi$.

Proof. The proof is similar to that for Proposition 8.3. □

Corollary D.10. If $(H \cup H', \mathrm{f}) \not\models_v \varphi$ then $(H, \mathrm{f}) \not\models_v \varphi$.

We also lift the function $\mathit{suffix}(-)$ to traces as follows: $\mathit{suffix}(H, \epsilon) = H$ and $\mathit{suffix}(H, \mu t) = \mathit{suffix}(\mathit{suffix}(H, \mu), t)$. Similarly, $\text{Det}(\epsilon) = \text{true}$ and $\text{Det}(\mu t) = \text{Det}(\mu) \wedge \text{Det}(t)$.

Lemma 11. For all $\varphi \in \text{sHML}^{\vee}_{\text{Det}}$ and $t \in \text{IAct}^*$, if $H' = \mathit{suffix}(H, t\alpha)$ and $\mathrm{f}' = \mathrm{f} \wedge \text{Det}(t\alpha)$ and $(H', \mathrm{f}') \models_v \varphi$ then $(H, \mathrm{f}) \models_{\text{Det}} [\alpha]\varphi$.

Proof. The proof proceeds by induction on the length of t, *i.e.*, $n = |t|$.

- When $n = 0$, then $t = \epsilon$, $H' = \mathit{suffix}(H,\alpha)$, $\mathsf{f}' = \mathsf{f} \wedge \textsc{Det}(\alpha)$ and $(H',\mathsf{f}') \models_v \varphi$. Our result, $(H,\mathsf{f}) \models_v [\alpha]\varphi$, follows immediately by rule vUm.
- When $n = k+1$, then $t = \gamma_1 \cdots \gamma_n \in \textsc{lAct}^*$ and $H' = \mathit{suffix}(H,t\alpha)$ and $\mathsf{f}' = \mathsf{f} \wedge \textsc{Det}(t\alpha)$ and $(H',\mathsf{f}') \models_v \varphi$. By definition of $\mathit{suffix}(-)$, we know there exists some H'' such that

$$H'' = \mathit{suffix}(H,\gamma_1) \quad \text{and} \quad H' = \mathit{suffix}(H',\gamma_2 \cdots \gamma_n \alpha) \tag{D.1}$$

  By definition of $\textsc{Det}(-)$, we also know there exists some f'' such that

$$\mathsf{f}'' = \mathsf{f} \wedge \textsc{Det}(\gamma_1) \quad \text{and} \quad \mathsf{f}' = \mathsf{f}'' \wedge \textsc{Det}(\gamma_2 \cdots \gamma_n) \tag{D.2}$$

  Using (D.1), (D.2) and the IH, we obtain $(H'',\mathsf{f}'') \models_v [\alpha]\varphi$. Our result, $(H,\mathsf{f}) \models_v [\alpha]\varphi$, follows by applying rule vUmPre. □

We prove that whenever a system p produces a history H that violates a formula φ, *i.e.,* $H \models_{\textsc{Det}} \varphi$, then p must also violate it, *i.e.,* $p \notin [\![\varphi]\!]$, namely Theorem 9.9 from Section 9.2. This proof relies on an additional result. Specifically, Lemma 12 below states that if a history violates a formula with the flag set to false, then a single trace t suffices to violate that formula.

Lemma 12. If $(H,\text{false}) \models_{\textsc{Det}} \varphi$ then $\exists t \in H$ such that $(\{t\},\text{false}) \models_{\textsc{Det}} \varphi$.

Proof. Straightforward by rule induction. □

Lemma 13 below then states that whenever a single trace violates a formula, then the system producing that trace also violates the formula.

Lemma 13. For all $t \in T_p$, if $(\{t\},\mathsf{f}) \models_{\textsc{Det}} \varphi$ then $p \notin [\![\varphi]\!]$.

Proof. Straightforward by rule induction. □

We are now in a position to prove Theorem 9.9, restated below.

Theorem D.14. For all (closed) formulae $\varphi \in \text{sHML}^{\vee}_{\textsc{Det}}$, if $\left(\exists H \subseteq T_p \text{ such that } H \models_{\textsc{Det}} \varphi\right)$ then $p \notin [\![\varphi]\!]$.

Proof. Suppose that $\exists H \subseteq T_p$ such that $H \models_{\textsc{Det}} \varphi$. Our result, $p \notin [\![\varphi]\!]$, follows from Lemma 15 below, by letting $\mathsf{f} = \text{true}$. □

Lemma 15. For all $\varphi \in \text{sHML}^{\vee}_{\textsc{Det}}$ and $H \subseteq T_p$, if $(H,\mathsf{f}) \models_{\textsc{Det}} \varphi$ then $p \notin [\![\varphi]\!]$.

Proof. The proof proceeds by induction on $(H,\mathsf{f}) \models_v \varphi$ where $H \subseteq T_p$.

- Case vF, *i.e.,* $(H,\mathsf{f}) \models_{\textsc{Det}} \mathsf{ff}$ where $H \neq \emptyset$. Our result, $p \notin [\![\mathsf{ff}]\!]$, is immediate since $[\![\mathsf{ff}]\!] = \emptyset$.
- Case vUm, *i.e.,* $(H,\mathsf{f}) \models_{\textsc{Det}} [\alpha]\varphi$ because $(H',\mathsf{f}') \models_{\textsc{Det}} \varphi$ where $H' = \mathit{suffix}(H,\alpha)$ and $\mathsf{f}' = \mathsf{f} \wedge \textsc{Det}(\alpha)$. By Lemma 7, we also know $H' \neq \emptyset$. This means that $\exists n \geq 1$ such that

$$H' = \bigcup_{i=1}^{n} H'_i \text{ such that } p \overset{\alpha}{\Longrightarrow}_T q_i \text{ and } H'_i \subseteq T_{q_i} \text{ for each } i \in \{1,\ldots,n\} \tag{D.3}$$

  There are two subcases to consider:

– If $\mathsf{f}' = \text{false}$, we know by Lemma 12 that $\exists t \in H'$ such that $(\{t\},\mathsf{f}') \models_{\text{DET}} [\alpha]\varphi$. From (D.3), we also know $t \in H'_k$ for some $k \in \{1,\ldots,n\}$ and that $p \overset{\alpha}{\Longrightarrow}_T q_k$ and $H'_k \subseteq T_{q_k}$. Using Lemma 13, we deduce that $q_k \notin [\![\varphi]\!]$, and by Proposition D.6, we obtain $p \overset{\alpha}{\Longrightarrow} q_k$. Thus, we can conclude that $p \notin \{q \mid p \overset{\alpha}{\Longrightarrow} q \text{ implies } q \notin [\![\varphi]\!]\} = [\![[\alpha]\varphi]\!]$, as required.

– If $\mathsf{f}' = \text{true}$, then $\text{DET}(\alpha)$. From Lemma 5, we know $T_{q_i} = T_{q_j}$ for all $i,j \in \{1,\ldots,n\}$, which implies $H' \subseteq T_{q_k}$ for all $k \in \{1,\ldots,n\}$. We can thus use the IH and obtain $q_k \notin [\![\varphi]\!]$. By Proposition D.6 and the fact that $p \overset{\alpha}{\Longrightarrow}_T q_k$, we also know $p \overset{\alpha}{\Longrightarrow} q_k$. Thus, we can conclude that $p \notin \{q \mid p \overset{\alpha}{\Longrightarrow} q \text{ implies } q \notin [\![\varphi]\!]\} = [\![[\alpha]\varphi]\!]$, as required.

- Case vUmPre, *i.e.,* $(H,\mathsf{f}) \models_{\text{DET}} [\alpha]\varphi$ because $(H',\mathsf{f}') \models_{\text{DET}} [\alpha]\varphi$ where $H' = \mathit{suffix}(H,\gamma)$ and $\mathsf{f}' = \mathsf{f} \wedge \text{DET}(\gamma)$. Due to our assumption that all internal actions iAct are deterministic, *i.e.,* $\text{DET}(\gamma)$, then $\mathsf{f}' = \text{true}$. By Lemma 7, we also know $H' \neq \emptyset$. This means that $\exists n \geq 1$ such that

$$H' = \bigcup_{i=1}^{n} H'_i \text{ such that } p \overset{\gamma}{\Longrightarrow}_T q_i \text{ and } H'_i \subseteq T_{q_i} \quad \text{for each } i \in \{1,\ldots,n\}$$

  From Lemma 5, we know $T_{q_i} = T_{q_j}$ for all $i,j \in \{1,\ldots,n\}$, which implies $H' \subseteq T_{q_k}$ for all $k \in \{1,\ldots,n\}$. We can thus use the IH and obtain $q_k \notin [\![[\alpha]\varphi]\!]$. By Proposition D.6 and $p \overset{\gamma}{\Longrightarrow}_T q_k$, we also know $p \Longrightarrow q_k$. Our result, $p \notin [\![[\alpha]\varphi]\!]$, follows by definition of $[\![-]\!]$.

- Case vAndL. We know $(H,\mathsf{f}) \models_{\text{DET}} \varphi \wedge \psi$ because $(H,\mathsf{f}) \models_v \varphi$. By the IH, we obtain $p \notin [\![\varphi]\!]$, which implies that $p \notin [\![\varphi]\!] \cap [\![\psi]\!] = [\![\varphi \wedge \psi]\!]$.

- Case vAndR. Proof is analogous to that for vAndL.

- Case vOr. We know $(H,\text{true}) \models_{\text{DET}} \varphi \vee \psi$ because $(H,\text{true}) \models_v \varphi$ and $(H,\text{true}) \models_v \psi$. By the IH, we obtain $p \notin [\![\varphi]\!]$ and $p \notin [\![\psi]\!]$, which implies that $p \notin [\![\varphi]\!] \cup [\![\psi]\!] = [\![\varphi \vee \psi]\!]$.

- Case vMax. We know $(p,\mathsf{f}) \models_v \max X.\varphi$ because $(H,\mathsf{f}) \models_v \varphi[^{\max X.\varphi}/_X]$. By the IH, we obtain $p \notin [\![\varphi[^{\max X.\varphi}/_X]]\!] = [\![\max X.\varphi]\!]$. □

We now prove Theorem 9.10, restated below.

Theorem D.16. Suppose $\text{DET}(\gamma) = \text{true}$ for all internal actions $\gamma \in \text{iAct}$. For all (closed) formulae $\varphi \in \text{sHML}^{\vee}_{\text{DET}}$, if $p \notin [\![\varphi]\!]$ then $\big(\exists H \subseteq T_p \text{ such that } H \models_{\text{DET}} \varphi\big)$.

Proof. Follows from Lemma 17 below, by letting $\mathsf{f} = \text{true}$. □

Lemma 17. If $\text{DET}(\gamma) = \text{true}$ for all $\gamma \in \text{iAct}$, then for all $p \in \text{Prc}$, $\varphi \in \text{sHML}^{\vee}_{\text{DET}}$ and $\mathsf{f} \in \text{Bool}$, if $\mathsf{f} \vdash_{\text{DET}} \varphi$ and $p \notin [\![\varphi]\!]$ then $\big(\exists H \subseteq T_p \text{ such that } (H,\mathsf{f}) \models_{\text{DET}} \varphi\big)$.

Proof. Suppose $p \in [\![\varphi]\!]$. Since $H \subseteq T_p$, it suffices to show $(T_p,\text{true}) \models_{\text{DET}} \varphi$. This follows from Lemma 18 below. □

Lemma 18. If $\text{DET}(\gamma) = \text{true}$ for all $\gamma \in \text{iAct}$, then for all $p \in \text{Prc}$, $\varphi \in \text{sHML}^{\vee}_{\text{DET}}$ and $\mathsf{f} \in \text{Bool}$, if $\mathsf{f} \vdash_{\text{DET}} \varphi$ and $p \notin [\![\varphi]\!]$ then $(T_p,\mathsf{f}) \models_{\text{DET}} \varphi$.

Proof. The proof proceeds by rule induction on $\mathsf{f} \vdash_{\text{DET}} \varphi$.

- Case cA, *i.e.,* $\mathsf{f} \vdash_{\text{Det}} \varphi$ where $\varphi \in \{\mathsf{ff}, \mathsf{tt}, X\}$. Assume that $p \notin [\![\varphi]\!]$. When $\varphi = \mathsf{ff}$, we immediately obtain that $(T_p, \mathsf{f}) \models_{\text{Det}} \mathsf{ff}$ by rule vF. When $\varphi = \mathsf{tt}$, the statement is vacuously true since $[\![\mathsf{tt}]\!] = \textsc{Prc}$ and thus $p \in [\![\varphi]\!]$. Also, we cannot have that $\varphi = X$; we are assuming φ is closed.
- Case cUm, *i.e.,* $\mathsf{f} \vdash_{\text{Det}} [\alpha]\varphi$ because $\mathsf{f} \wedge \textsc{Det}(\alpha) \vdash_{\text{Det}} \varphi$. Assume $p \notin [\![[\alpha]\varphi]\!]$. This means that there exists q such that $p \overset{\alpha}{\Longrightarrow} q$ and $q \notin [\![\varphi]\!]$. By Proposition D.6, we know $p \overset{t}{\Longrightarrow}_T p' \overset{\alpha}{\Longrightarrow}_T p'' \overset{t'}{\Longrightarrow}_T q$ for some p', p'' and $t, t' \in \textsc{Act}^*$, which implies $p'' \notin [\![\varphi]\!]$. There are three subcases to consider:
  - When $\mathsf{f} = \mathsf{true} = \textsc{Det}(\alpha)$, we have $\mathsf{true} \vdash_{\text{Det}} \varphi$. By the IH, we deduce $(T_{p''}, \mathsf{true}) \models_{\text{Det}} \varphi$, *i.e.,* $(T_{p''}, \mathsf{true} \wedge \textsc{Det}(\alpha)) \models_{\text{Det}} \varphi$. Applying rule vUm, we obtain $(T_{p'}, \mathsf{true}) \models_{\text{Det}} [\alpha]\varphi$. Applying rule vUmPre $|t|$ times, we conclude $(T_p, \mathsf{true}) \models_{\text{Det}} [\alpha]\varphi$.
  - When $\mathsf{f} = \mathsf{false}$, the proof is analogous to that for the previous case.
  - When $\mathsf{f} = \mathsf{true}$ and $\textsc{Det}(\alpha) = \mathsf{false}$, we have $\mathsf{false} \vdash_{\text{Det}} \varphi$. By the IH, we get $(T_{p''}, \mathsf{false}) \models_{\text{Det}} \varphi$ *i.e.,* $(T_{p''}, \mathsf{true} \wedge \textsc{Det}(\alpha)) \models_{\text{Det}} \varphi$. Applying rule vUm, we obtain $(T_{p'}, \mathsf{true}) \models_{\text{Det}} [\alpha]\varphi$. By the assumption that $\textsc{Det}(\gamma) = \mathsf{true}$, then we also know that for $t = \gamma_1 \cdots \gamma_n$, we have $\textsc{Det}(\gamma_i) = \mathsf{true}$. We can thus apply rule vUmPre n times and conclude $(T_p, \mathsf{true}) \models_{\text{Det}} [\alpha]\varphi$.
- Case cAnd, *i.e.,* $\mathsf{f} \vdash_{\text{Det}} \varphi \wedge \psi$ because $\mathsf{f} \vdash_{\text{Det}} \varphi$ and $\mathsf{f} \vdash_{\text{Det}} \psi$. Assume $p \notin [\![\varphi \wedge \psi]\!]$. This implies that either $p \notin [\![\varphi]\!]$ or $p \notin [\![\psi]\!]$. Without loss of generality, suppose the former. By the IH, we obtain $(T_p, \mathsf{f}) \models_{\text{Det}} \varphi$. Our result, $(T_p, \mathsf{f}) \models_{\text{Det}} \varphi \wedge \psi$, follows by rule vAnd.
- Case cOr, *i.e.,* $\mathsf{true} \vdash_{\text{Det}} \varphi \vee \psi$ because $\mathsf{true} \vdash_{\text{Det}} \varphi$ and $\mathsf{true} \vdash_{\text{Det}} \psi$. Assume $p \notin [\![\varphi \vee \psi]\!]$. This implies that $p \notin [\![\varphi]\!]$ and $p \notin [\![\psi]\!]$. By the IH, we obtain $(T_p, \mathsf{true}) \models_{\text{Det}} \varphi$ and $(T_p, \mathsf{true}) \models_{\text{Det}} \psi$. Our result, $(T_p, \mathsf{true}) \models_{\text{Det}} \varphi \vee \psi$, follows by rule vOr.
- Case cMax, *i.e.,* $\mathsf{f} \vdash_{\text{Det}} \max X.\varphi$ because $\mathsf{f} \vdash_{\text{Det}} \varphi[^{\max X.\varphi}/_X]$. Assume $p \notin [\![\max X.\varphi]\!]$. Since $[\![\max X.\varphi]\!] = [\![\varphi[^{\max X.\varphi}/_X]]\!]$, we also know $p \in [\![\varphi[^{\max X.\varphi}/_X]]\!]$. By the IH, we obtain $(T_p, \mathsf{f}) \models_{\text{Det}} \varphi[^{\max X.\varphi}/_X]$. Our result, $(T_p, \mathsf{f}) \models_{\text{Det}} \max X.\varphi$, follows by rule vMax. □

E Proving Monitor Correctness

In this section, we prove the instrumentation and monitor properties of Chapter 8. Before delving into the technical results, we present the full operational semantics for monitors and instrumentation in Part II of this thesis. Concretely, Figure E.1 consolidates the rules in Figure 3.1 and Figure 7.1.

Monitor Syntax

$$m,n \in \textsc{Mon} ::= \mathsf{no} \mid \mathsf{end} \mid \alpha.m \mid \mathrm{rec}X.m \mid X \mid m \oplus n \mid m \otimes n \qquad (\odot \in \{\oplus, \otimes\})$$

Monitor Semantics

$$\textsc{mEnd}\ \frac{}{(t,\mathsf{end}) \xrightarrow{\alpha}_H (t\alpha,\mathsf{end})} \qquad \textsc{mVrP1L}\ \frac{t \in H}{(t,\mathsf{no} \odot n) \xrightarrow{\tau}_H (t,n)} \qquad \textsc{mVrP2L}\ \frac{t \notin H}{(t,\mathsf{no} \odot n) \xrightarrow{\tau}_H (t,\mathsf{no})} \qquad \textsc{mAct}\ \frac{}{(t,\alpha.m) \xrightarrow{\alpha}_H (t\alpha,m)}$$

$$\textsc{mRec}\ \frac{}{(t,\mathrm{rec}X.m) \xrightarrow{\tau}_H (t,m[\mathrm{rec}X.m/X])} \qquad \textsc{mTauL}\ \frac{(t,m) \xrightarrow{\tau}_H (t,m')}{(t,m \odot n) \xrightarrow{\tau}_H (t,m' \odot n)}$$

$$\textsc{mPar1}\ \frac{(t,m) \xrightarrow{\alpha}_H (t',m') \quad (t,n) \xrightarrow{\alpha}_H (t',n')}{(t,m \odot n) \xrightarrow{\alpha}_H (t',m' \odot n')} \qquad \textsc{mPar2L}\ \frac{n \neq \mathsf{no} \quad (t,m) \xrightarrow{\alpha}_H (t',m') \quad (t,n) \overset{\alpha}{\not\to}_H \quad (t,n) \overset{\tau}{\not\to}_H}{(t,m \odot n) \xrightarrow{\alpha}_H (t',m')}$$

Instrumentation Semantics

$$\textsc{iNo}\ \frac{}{H \triangleright (t,\mathsf{no}) \triangleleft p \xrightarrow{\tau} H, t \triangleright (t,\mathsf{end}) \triangleleft p} \qquad \textsc{iTer}\ \frac{m \neq \mathsf{no} \quad p \xrightarrow{\alpha} p' \quad (t,m) \overset{\alpha}{\not\to}_H \quad (t,m) \overset{\tau}{\not\to}_H}{H \triangleright (t,m) \triangleleft p \xrightarrow{\alpha} H \triangleright (t\alpha,\mathsf{end}) \triangleleft p'}$$

$$\textsc{iAsS}\ \frac{m \neq \mathsf{no} \quad p \xrightarrow{\tau} p'}{H \triangleright (t,m) \triangleleft p \xrightarrow{\tau} H \triangleright (t,m) \triangleleft p'} \qquad \textsc{iAsI}\ \frac{m \neq \mathsf{no} \quad p \xrightarrow{\gamma} p'}{H \triangleright (t,m) \triangleleft p \xrightarrow{\tau} H \triangleright (t\gamma,m) \triangleleft p'} \qquad \textsc{iAsM}\ \frac{(t,m) \xrightarrow{\tau}_H (t',m')}{H \triangleright (t,m) \triangleleft p \xrightarrow{\tau} H \triangleright (t',m') \triangleleft p}$$

$$\textsc{iMon}\ \frac{p \xrightarrow{\alpha} p' \quad (t,m) \xrightarrow{\alpha}_H (t',m')}{H \triangleright (t,m) \triangleleft p \xrightarrow{\alpha} H \triangleright (t',m') \triangleleft p'}$$

Figure E.1. *Monitors and Instrumentation for non-deterministic SUSs*

E.1 Instrumentation Properties

The first result we prove is Proposition 8.1, restated below. We note that this proof is very similar to that for Proposition 4.1 in Appendix B.

Proposition E.0 (Veracity). For any history H, monitor m, system p, and sequence of actions $\eta_1,\ldots,\eta_n$,

$$\text{if } H \triangleright (\epsilon,m) \triangleleft p \xrightarrow{\eta_1} \cdots \xrightarrow{\eta_n} H' \triangleright (t,m') \triangleleft p' \text{ then } p \overset{t}{\Longrightarrow}_T p'$$

Proof. The proof proceeds by induction on n.

For the base case, when $n = 0$, the result is immediate.

For the inductive case, when $n = k+1$, the transitions are as follows:

$$H \triangleright (\epsilon,m) \triangleleft p \xrightarrow{\eta_1} \cdots \xrightarrow{\eta_k} H' \triangleright (t,m') \triangleleft p' \xrightarrow{\eta_{k+1}} H'' \triangleright (t',m'') \triangleleft p''$$

We show that $p \overset{t'}{\Longrightarrow}_T p''$.
By the IH and $H \triangleright (\epsilon,m) \triangleleft p \xrightarrow{\eta_1} \cdots \xrightarrow{\eta_k} H' \triangleright (t,m') \triangleleft p'$, we obtain that

$$p \overset{t}{\Longrightarrow}_T p' \tag{E.1}$$

By case analysis, $H' \triangleright (t,m') \triangleleft p' \xrightarrow{\eta_{k+1}} H'' \triangleright (t',m'') \triangleleft p''$ could have been derived via several rules:

- Using rule iNo, then $p'' = p'$ and $t = t'$, which implies that $p' \overset{\epsilon}{\Longrightarrow}_T p''$. By eq. (E.1), we conclude that $p \overset{t}{\Longrightarrow}_T p' \overset{\epsilon}{\Longrightarrow}_T p''$, *i.e.,* $p \overset{t}{\Longrightarrow}_T p''$.
- Using rule iTer, then $t' = t\alpha$ and $p' \xrightarrow{\alpha} p''$ for some $\alpha \in$ eAct, which implies that $p' \overset{\alpha}{\Longrightarrow}_T p''$. By eq. (E.1), we conclude that $p \overset{t}{\Longrightarrow}_T p' \overset{\alpha}{\Longrightarrow}_T p''$, *i.e.,* $p \overset{t\alpha}{\Longrightarrow}_T p''$.
- Using rule iAsS, then $t = t'$ and $p' \xrightarrow{\tau} p''$, which implies that $p' \overset{\epsilon}{\Longrightarrow}_T p''$. By eq. (E.1), we conclude that $p \overset{t}{\Longrightarrow}_T p' \overset{\epsilon}{\Longrightarrow}_T p''$, *i.e.,* $p \overset{t}{\Longrightarrow}_T p''$.
- Using rule iAsI, then $t' = t\gamma$ and $p' \xrightarrow{\gamma} p''$ for some $\gamma \in$ iAct, which implies $p' \overset{\gamma}{\Longrightarrow}_T p''$. By eq. (E.1), we conclude that $p \overset{t}{\Longrightarrow}_T p' \overset{\gamma}{\Longrightarrow}_T p''$, *i.e.,* $p \overset{t\gamma}{\Longrightarrow}_T p''$.
- Using rule iAsM, then $p' = p''$ and $(t,m') \xrightarrow{\tau}_H (t',m'')$. By Lemma 2, we obtain $t = t'$, and since $p' = p''$, we obtain $p' \overset{\epsilon}{\Longrightarrow}_T p''$. Using eq. (E.1), we conclude $p \overset{t}{\Longrightarrow}_T p' \overset{\epsilon}{\Longrightarrow}_T p''$, *i.e.,* $p \overset{t}{\Longrightarrow}_T p''$.
- Using rule iMon, then $p' \xrightarrow{\alpha} p''$ and $(t,m') \xrightarrow{\alpha}_H (t',m'')$ for some $\alpha \in$ eAct. By Lemma 1, we obtain that $t' = t\alpha$, and since $p' \xrightarrow{\alpha} p''$ we know that $p' \overset{\alpha}{\Longrightarrow}_T p''$. Using eq. (E.1), we conclude $p \overset{t}{\Longrightarrow}_T p' \overset{\alpha}{\Longrightarrow}_T p''$, *i.e.,* $p \overset{t\alpha}{\Longrightarrow}_T p''$. □

E.2 Monitor Properties

In this section, we give the complete proof for Proposition 8.3 in Chapter 8. We note that this proof is very similar to that for Proposition 4.8 in Appendix B.

Lemma 1. For any history H, traces t,u, flag f and monitor m, if $\mathbf{rej}((H,t),\mathsf{f},m)$ then $\mathbf{rej}((H,tu),\mathsf{f},m)$

Proof. The proof proceeds by rule induction on $\mathbf{rej}((H,t),\mathsf{f},m)$.

- Case NO. Follows immediately because $\mathbf{rej}(H', \mathsf{f}, \mathsf{no})$ for all $H' \neq \emptyset$.
- Case ACT. We know $\mathbf{rej}((H,t), \mathsf{f}, \alpha.m)$ because $\mathbf{rej}(H', \mathsf{f}', m)$ where $H' = \mathit{suffix}((H,t),\alpha)$ and $\mathsf{f}' = \mathsf{f} \wedge \textsc{Det}(\alpha)$. There are two subcases to consider:
  - When $t = \alpha t'$, then $H' = (H'', t') = \mathit{suffix}((H,t),\alpha)$ for some H''. By the IH, we deduce $\mathbf{rej}((H'', t'u), \mathsf{f}', m)$. But by definition, we also know $(H'', t'u) = \mathit{suffix}((H,tu),\alpha)$, meaning that $\mathbf{rej}(\mathit{suffix}((H,tu),\alpha), \mathsf{f}', m)$. Our result, $\mathbf{rej}((H,tu), \mathsf{f}, \alpha.m)$, follows by rule ACT.
  - When $t = \beta t'$, we know by definition that $\mathit{suffix}((H,t),\alpha) = \mathit{suffix}(H,\alpha) = \mathit{suffix}((H,tu),\alpha)$. Our result, $\mathbf{rej}((H,tu), \mathsf{f}, \alpha.m)$, follows immediately by applying rule ACT.
- Case ACTPRE. Proof is similar to that for ACT.
- Case PARAL. We know that $\mathbf{rej}((H,t), \mathsf{f}, m' \otimes m'')$ because of $\mathbf{rej}((H,t), \mathsf{f}, m')$. By the IH, we obtain $\mathbf{rej}((H,tu), \mathsf{f}, m')$. Using rule PARAL, we can conclude $\mathbf{rej}((H,tu), \mathsf{f}, m' \otimes m'')$.
- Case PARAR. Proof is analogous to that for PARAR.
- Case PARO. We know $\mathbf{rej}((H,t), \mathsf{true}, m' \otimes m'')$ because $\mathbf{rej}((H,t), \mathsf{true}, m')$ and $\mathbf{rej}((H,t), \mathsf{true}, m'')$. By the IH, we obtain that $\mathbf{rej}((H,tu), \mathsf{true}, m')$ and $\mathbf{rej}((H,tu), \mathsf{true}, m'')$. Applying rule PARO, we conclude $\mathbf{rej}((H,tu), \mathsf{true}, m' \otimes m'')$.
- Case REC. We know $\mathbf{rej}((H,t), \mathsf{f}, \mathsf{rec}X.m)$ because $\mathbf{rej}((H,t), \mathsf{f}, m[^{\mathsf{rec}X.m}/_X])$. By the IH, we obtain $\mathbf{rej}((H,tu), \mathsf{f}, m[^{\mathsf{rec}X.m}/_X])$. Our result, $\mathbf{rej}((H,tu), \mathsf{f}, \mathsf{rec}X.m)$, follows by rule REC. □

F Proving Maximal Expressiveness

The proof for showing that $\text{sHML}^{\vee}_{\text{DET}}$ is maximally expressive w.r.t. our multi-run monitoring setup in Section 9.3 relies on the normalisation function $N_{\text{DET}}(-)$ in Definition 9.19. Concretely, in this chapter, we give the proof for Proposition 9.21, which relies on several additional results.

Lemma 1. Suppose $X \notin fv(m)$. Then for all $m,n \in \text{MON}$, $\mathsf{f},\mathsf{f}' \in \text{BOOL}$ and $\sigma \in \text{SUB}$, we have

$$N_{\text{DET}}(m,\mathsf{f},\sigma) = N_{\text{DET}}(m,\mathsf{f},\sigma[X \mapsto \langle n,\mathsf{f}'\rangle])$$

Proof. Suppose $X \notin fv(m)$. The proof proceeds by induction on the structure of m. The only interesting case is when $m = \text{rec}X.m'$. We have

$$\begin{aligned}
N_{\text{DET}}(\text{rec}X.m',\mathsf{f},\sigma) &= \text{rec}X.N_{\text{DET}}(m',\mathsf{f},\sigma[X \mapsto \langle \text{rec}X.m,\mathsf{f}\rangle]) \\
&= \text{rec}X.N_{\text{DET}}(m',\mathsf{f},\sigma[X \mapsto \langle n,\mathsf{f}'\rangle][X \mapsto \langle \text{rec}X.m,\mathsf{f}\rangle]) \text{ for some } n \text{ and } \mathsf{f}' \\
&\qquad \text{since } \sigma[\langle n,\mathsf{f}'\rangle][X \mapsto \langle \text{rec}X.m,\mathsf{f}\rangle] = \sigma[X \mapsto \langle \text{rec}X.m,\mathsf{f}\rangle] \\
&= N_{\text{DET}}(\text{rec}X.m',\mathsf{f},\sigma[X \mapsto \langle n,\mathsf{f}'\rangle])
\end{aligned}$$

The other cases are straightforward. □

Lemma 2. For any m, f and $\sigma \in \text{SUB}$, if $X \notin fv(m)$ and $X \notin fv(\mathbf{cod}(\sigma))$ then $X \notin fv(N_{\text{DET}}(m,\mathsf{f},\sigma))$.

Proof. Follows from the contrapositive of Lemma 3. □

Lemma 3. For all $m \in \text{MON}$, $\mathsf{f} \in \text{BOOL}$ and $\sigma \in \text{SUB}$, if $X \in fv(N_{\text{DET}}(m,\mathsf{f},\sigma))$ then either $X \in fv(m)$ or $X \in fv(\mathbf{cod}(\sigma))$.

Proof. Suppose $X \in fv(N_{\text{DET}}(m,\mathsf{f},\sigma))$. The proof proceeds by induction on the derivation of $N_{\text{DET}}(m,\mathsf{f},\sigma)$. We only outline the main cases:

- Case TVAR3, *i.e.,* $N_{\text{DET}}(Y,\mathsf{f},\sigma) = n$ because $\sigma(Y) = \langle m,\mathsf{f}'\rangle$ where $\mathsf{f}' \neq \mathsf{f}$, and $N_{\text{DET}}(m,\mathsf{f},\sigma) = n$. Since $fv(N_{\text{DET}}(Y,\mathsf{f},\sigma)) = fv(n) = fv(N_{\text{DET}}(m,\mathsf{f},\sigma))$, we can use the IH and obtain that either $X \in fv(m)$ or $X \in fv(\mathbf{cod}(\sigma))$. Since $\sigma(Y) = \langle m,\mathsf{f}'\rangle$, then it must be that $X \in fv(\mathbf{cod}(\sigma))$.
- Case TACT, *i.e.,* $N_{\text{DET}}(\alpha.m,\mathsf{f},\sigma) = \alpha.n$ because $N_{\text{DET}}(m,\mathsf{f} \wedge \text{DET}(\alpha),\sigma) = n$. Since $fv(N_{\text{DET}}(\alpha.m,\mathsf{f},\sigma)) = fv(\alpha.n) = fv(n) = fv(N_{\text{DET}}(m,\mathsf{f} \wedge \text{DET}(\alpha),\sigma))$, we can use the IH and obtain that either $X \in fv(m)$ or $X \in fv(\mathbf{cod}(\sigma))$. In turn, this implies that either $X \in fv(\alpha.m)$ or $X \in fv(\mathbf{cod}(\sigma))$.

- Case tRec, *i.e.,* $N_{\text{DET}}(\text{rec}\,Y.m,\mathsf{f},\sigma) = \text{rec}\,Y.n$ because $N_{\text{DET}}(m,\mathsf{f},\sigma') = n$ where $\sigma' = \sigma[Y \mapsto \langle \text{rec}\,Y.m,\mathsf{f}\rangle]$. Working up to α-equivalence, we can assume that $X \neq Y$. Since $X \in fv(\text{rec}\,Y.n) = fv(n) \setminus \{Y\}$, then $X \in fv(n)$. By the IH, we obtain that either $X \in fv(m)$ or $X \in fv(\mathbf{cod}(\sigma'))$. In case of the former, since $X \neq Y$, we deduce that $X \in fv(\text{rec}\,Y.m)$. In case of the latter, there are two subcases. If $X \in fv(\mathbf{cod}(\sigma))$, then we are done. Otherwise, if $X \in fv(\mathbf{cod}(\sigma'))$ but $X \notin fv(\mathbf{cod}(\sigma))$, then it must be that $X \in fv(\text{rec}\,Y.m)$.

The other cases follow with similar reasoning. □

Lemma 4. Given monitor $n \in \text{Mon}$ and $\sigma \in \text{Sub}$, suppose that $X \notin fv(\mathbf{cod}(\sigma))$ and $fv(n) \subseteq \{X\}$. Then for any monitor $m \in \text{Mon}$, we have that

$$N_{\text{DET}}(m,\text{false},\sigma[X \mapsto \langle \text{rec}X.n,\text{true}\rangle]) = N_{\text{DET}}(m,\text{false},\sigma[X \mapsto \langle \text{rec}X.n,\text{false}\rangle])[^{N_{\text{DET}}(\text{rec}X.n,\text{false},\sigma)}/_X]$$

Proof. The proof proceeds by induction on the structure of m.

- Case $m = X$. We have

$$\begin{aligned}
&N_{\text{DET}}(X,\text{false},\sigma[X \mapsto \langle \text{rec}X.n,\text{true}\rangle]) \\
&= N_{\text{DET}}(\text{rec}X.n,\text{false},\sigma[X \mapsto \langle \text{rec}X.n,\text{true}\rangle]) \\
&= \text{rec}X.N_{\text{DET}}(n,\text{false},\sigma[X \mapsto \langle \text{rec}X.n,\text{false}\rangle]) \\
&= \big(\text{rec}X.N_{\text{DET}}(n,\text{false},\sigma[X \mapsto \langle \text{rec}X.n,\text{false}\rangle])\big)[^{N_{\text{DET}}(\text{rec}X.n,\text{false},\sigma)}/_X] \text{ since } X \text{ is not free} \\
&= N_{\text{DET}}(\text{rec}X.n,\text{false},\sigma[X \mapsto \langle \text{rec}X.n,\text{false}\rangle])[^{N_{\text{DET}}(\text{rec}X.n,\text{false},\sigma)}/_X]
\end{aligned}$$

- Case $m = Y$. There are two subcases to consider. When $Y \notin \mathbf{dom}(\sigma)$, the proof is straightforward. When $\sigma(Y) = \langle m',\mathsf{f}\rangle$ for some m' and f, we have

$$\begin{aligned}
&N_{\text{DET}}(Y,\text{false},\sigma[X \mapsto \langle \text{rec}X.n,\text{true}\rangle]) \\
&= N_{\text{DET}}(m',\text{false},\sigma[X \mapsto \langle \text{rec}X.n,\text{true}\rangle]) \\
&= N_{\text{DET}}(m',\text{false},\sigma[X \mapsto \langle \text{rec}X.n,\text{false}\rangle]) \text{ by Lemma 1 because } X \notin fv(\mathbf{cod}(\sigma)) \text{ implies } X \notin fv(m') \\
&= N_{\text{DET}}(m',\text{false},\sigma[X \mapsto \langle \text{rec}X.n,\text{false}\rangle])[^{N_{\text{DET}}(\text{rec}X.n,\text{false},\sigma)}/_X] \\
&\quad \text{since } X \notin N_{\text{DET}}(m',\text{false},\sigma[X \mapsto \langle \text{rec}X.n,\text{false}\rangle]) \text{ by Lemma 2}
\end{aligned}$$

- Case $m = \text{rec}X.m'$. We have

$$\begin{aligned}
&N_{\text{DET}}(\text{rec}X.m',\text{false},\sigma[X \mapsto \langle \text{rec}X.n,\text{true}\rangle]) \\
&= N_{\text{DET}}(\text{rec}X.m',\text{false},\sigma[X \mapsto \langle \text{rec}X.n,\text{false}\rangle]) \text{ by Lemma 1 since } X \notin fv(\text{rec}X.m') \\
&= \text{rec}X.N_{\text{DET}}(m',\text{false},\sigma[X \mapsto \langle \text{rec}X.n,\text{false}\rangle]) \\
&= \big(\text{rec}X.N_{\text{DET}}(m',\text{false},\sigma[X \mapsto \langle \text{rec}X.n,\text{false}\rangle])\big)[^{N_{\text{DET}}(\text{rec}X.n,\text{false},\sigma)}/_X] \text{ since } X \text{ is not free} \\
&= N_{\text{DET}}(\text{rec}X.m',\text{false},\sigma[X \mapsto \langle \text{rec}X.n,\text{false}\rangle])[^{N_{\text{DET}}(\text{rec}X.n,\text{false},\sigma)}/_X]
\end{aligned}$$

- Case $m = \text{rec}\,Y.m'$. We have

$$\begin{aligned}
&N_{\text{DET}}(\text{rec}\,Y.m',\text{false},\sigma[X \mapsto \langle \text{rec}X.n,\text{true}\rangle]) \\
&= \text{rec}\,Y.N_{\text{DET}}(m',\text{false},\sigma[X \mapsto \langle \text{rec}X.n,\text{true}\rangle]) \\
&= \text{rec}\,Y.\big(N_{\text{DET}}(m',\text{false},\sigma[X \mapsto \langle \text{rec}X.n,\text{false}\rangle])[^{N_{\text{DET}}(\text{rec}X.n,\text{false},\sigma)}/_X]\big) \text{ by the IH} \\
&= \big(\text{rec}\,Y.N_{\text{DET}}(m',\text{false},\sigma[X \mapsto \langle \text{rec}X.n,\text{false}\rangle])\big)[^{N_{\text{DET}}(\text{rec}X.n,\text{false},\sigma)}/_X] \\
&= N_{\text{DET}}(\text{rec}\,Y.m',\text{false},\sigma[X \mapsto \langle \text{rec}X.n,\text{false}\rangle])[^{N_{\text{DET}}(\text{rec}X.n,\text{false},\sigma)}/_X]
\end{aligned}$$

The remaining cases are more straightforward. □

Lemma 5. Given monitor $n \in \textsc{Mon}$ and $\sigma \in \textsc{Sub}$, suppose $X \notin fv(\mathbf{cod}(\sigma))$ and $fv(n) \subseteq \{X\}$. For any monitor $m \in \textsc{Mon}$ and flag $\mathsf{f} \in \textsc{Bool}$, we have that

$$N_{\textsc{Det}}(m[^{\mathrm{rec}X.n}/_X], \mathsf{f}, \sigma) = N_{\textsc{Det}}(m, \mathsf{f}, \sigma')[^{N_{\textsc{Det}}(\mathrm{rec}X.n, \mathsf{f}, \sigma)}/_X]$$

where $\sigma' = \sigma[X \mapsto \langle \mathrm{rec}X.n, \mathsf{f} \rangle]$.

Proof. Let $\sigma' = \sigma[X \mapsto \langle \mathrm{rec}X.n, \mathsf{f} \rangle]$. Suppose $X \notin fv(\mathbf{cod}(\sigma))$ and $fv(n) \subseteq \{X\}$. Then $X \notin fv(\mathrm{rec}X.n)$ either. The proof proceeds by induction on the structure of m. We outline the main cases.

- Case $m = X$. There are three subcases to consider.
  - When $\sigma(X) = \langle m', \mathsf{f} \rangle$ for some m', we have

    $$\begin{aligned}
    &N_{\textsc{Det}}(X[^{\mathrm{rec}X.m}/_X], \mathsf{f}, \sigma) \\
    &= N_{\textsc{Det}}(\mathrm{rec}X.n, \mathsf{f}, \sigma) \\
    &= X[^{N_{\textsc{Det}}(\mathrm{rec}X.n, \mathsf{f}, \sigma)}/_X] \\
    &= N_{\textsc{Det}}(X, \mathsf{f}, \sigma')[^{N_{\textsc{Det}}(\mathrm{rec}X.n, \mathsf{f}, \sigma)}/_X] \text{ where } \sigma' = \sigma[X \mapsto \langle \mathrm{rec}X.n, \mathsf{f} \rangle] \text{ by Definition 9.19}
    \end{aligned}$$

  - When $\sigma(X) = \langle m', \mathsf{f}' \rangle$ for some m' and $\mathsf{f}' \neq \mathsf{f}$, the proof is similar.
  - When $X \notin \mathbf{dom}(\sigma)$, the proof is similar.
- Case $m = Y$. There are three subcases to consider.
  - When $\sigma(Y) = \langle m', \mathsf{f} \rangle$ for some m', we have

    $$\begin{aligned}
    N_{\textsc{Det}}(Y[^{\mathrm{rec}X.m}/_X], \mathsf{f}, \sigma) = N_{\textsc{Det}}(Y, \mathsf{f}, \sigma) = Y \text{ by Definition 9.19} \\
    = Y[^{N_{\textsc{Det}}(\mathrm{rec}X.n, \mathsf{f}, \sigma)}/_X] \\
    = N_{\textsc{Det}}(Y, \mathsf{f}, \sigma')[^{N_{\textsc{Det}}(\mathrm{rec}X.n, \mathsf{f}, \sigma)}/_X] \text{ by Definition 9.19}
    \end{aligned}$$

  - When $\sigma(Y) = \langle m', \mathsf{f}' \rangle$ for some m' and $\mathsf{f}' \neq \mathsf{f}$, we have

    $$\begin{aligned}
    &N_{\textsc{Det}}(Y[^{\mathrm{rec}X.m}/_X], \mathsf{f}, \sigma) \\
    &= N_{\textsc{Det}}(Y, \mathsf{f}, \sigma) = N_{\textsc{Det}}(m', \mathsf{f}, \sigma) \text{ by Definition 9.19} \\
    &= N_{\textsc{Det}}(m', \mathsf{f}, \sigma') \text{ where } \sigma' = \sigma[X \mapsto \langle \mathrm{rec}X.n, \mathsf{f} \rangle] \text{ by Lemma 1} \\
    &\qquad \text{because } X \notin fv(\mathbf{cod}(\sigma)) \text{ implies } X \notin fv(m') \\
    &= N_{\textsc{Det}}(m', \mathsf{f}, \sigma')[^{N_{\textsc{Det}}(\mathrm{rec}X.n, \mathsf{f}, \sigma)}/_X] \text{ since } X \notin N_{\textsc{Det}}(m', \mathsf{f}, \sigma') \text{ by Lemma 2} \\
    &= N_{\textsc{Det}}(Y, \mathsf{f}, \sigma')[^{N_{\textsc{Det}}(\mathrm{rec}X.n, \mathsf{f}, \sigma)}/_X] \text{ by Definition 9.19}
    \end{aligned}$$

  - When $Y \notin \mathbf{dom}(\sigma)$, the proof is straightforward.
- Case $m = \alpha.n$. We have

  $$\begin{aligned}
  &N_{\textsc{Det}}((\alpha.m')[^{\mathrm{rec}X.m}/_X], \mathsf{f}, \sigma) \\
  &= N_{\textsc{Det}}(\alpha.(m'[^{\mathrm{rec}X.m}/_X]), \mathsf{f}, \sigma) \\
  &= \alpha.N_{\textsc{Det}}(m'[^{\mathrm{rec}X.m}/_X], \mathsf{f}', \sigma) \text{ where } \mathsf{f}' = \mathsf{f} \wedge \textsc{Det}(\alpha) \\
  &= \alpha.\big(N_{\textsc{Det}}(m', \mathsf{f}', \sigma')[^{N_{\textsc{Det}}(\mathrm{rec}X.n, \mathsf{f}', \sigma)}/_X]\big) \text{ using the IH where } \sigma' = \sigma[X \mapsto \langle \mathrm{rec}X.n, \mathsf{f}' \rangle]
  \end{aligned}$$

There are two subcases to consider. If $\mathsf{f} = \mathsf{f}'$, we have

$$\begin{aligned}
&\alpha.\big(N_{\text{DET}}(m',\mathsf{f}',\sigma')[^{N_{\text{DET}}(\text{rec}X.n,\mathsf{f}',\sigma)}/_X]\big)\\
&= \alpha.\big(N_{\text{DET}}(m',\mathsf{f},\sigma')[^{N_{\text{DET}}(\text{rec}X.n,\mathsf{f},\sigma)}/_X]\big) \text{ where } \sigma' = \sigma[X \mapsto \langle \text{rec}X.n,\mathsf{f}\rangle]\\
&= \big(\alpha.N_{\text{DET}}(m',\mathsf{f},\sigma')\big)[^{N_{\text{DET}}(\text{rec}X.n,\mathsf{f},\sigma)}/_X]\\
&= N_{\text{DET}}(\alpha.m',\mathsf{f},\sigma')[^{N_{\text{DET}}(\text{rec}X.n,\mathsf{f},\sigma)}/_X] \text{ as required}
\end{aligned}$$

Otherwise, if $\mathsf{f} = \text{true}$ and $\mathsf{f}' = \text{false}$, we have

$$\begin{aligned}
&\alpha.\big(N_{\text{DET}}(m',\mathsf{f}',\sigma')[^{N_{\text{DET}}(\text{rec}X.n,\mathsf{f}',\sigma)}/_X]\big)\\
&= \alpha.\big(N_{\text{DET}}(m',\text{false},\sigma')[^{N_{\text{DET}}(\text{rec}X.n,\text{false},\sigma)}/_X]\big) \text{ where } \sigma' = \sigma[X \mapsto \langle \text{rec}X.n,\text{false}\rangle]\\
&= \alpha.\big(N_{\text{DET}}(m',\text{false},\sigma')[^{N_{\text{DET}}(\text{rec}X.n,\text{false},\sigma)}/_X]\big)[^{N_{\text{DET}}(\text{rec}X.n,\text{true},\sigma)}/_X] \text{ since } X \text{ is not free}\\
&= \alpha.N_{\text{DET}}(m',\text{true},\sigma'')[^{N_{\text{DET}}(\text{rec}X.n,\text{true},\sigma)}/_X] \text{ by Lemma 4 where } \sigma'' = \sigma[X \mapsto \langle \text{rec}X.n,\text{true}\rangle]\\
&= N_{\text{DET}}(\alpha.m',\text{true},\sigma'')[^{N_{\text{DET}}(\text{rec}X.n,\text{true},\sigma)}/_X] \text{ as required}
\end{aligned}$$

- Case $m = \text{rec}X.m'$. We have

$$\begin{aligned}
&N_{\text{DET}}((\text{rec}X.m')[^{\text{rec}X.n}/_X],\mathsf{f},\sigma)\\
&= N_{\text{DET}}(\text{rec}X.m',\mathsf{f},\sigma) \text{ since } X \notin fv(\text{rec}X.m')\\
&= N_{\text{DET}}(\text{rec}X.m',\mathsf{f},\sigma') \text{ where } \sigma' = \sigma[X \mapsto \langle \text{rec}X.n,\mathsf{f}\rangle] \text{ using Lemma 1}\\
&= \text{rec}X.N_{\text{DET}}(m',\mathsf{f},\sigma'[X \mapsto \langle \text{rec}X.m',\mathsf{f}\rangle])\\
&= \big(\text{rec}X.N_{\text{DET}}(m',\mathsf{f},\sigma'[X \mapsto \langle \text{rec}X.m',\mathsf{f}\rangle])\big)[^{N_{\text{DET}}(\text{rec}X.n,\mathsf{f},\sigma)}/_X] \text{ since } X \text{ is not free}\\
&= N_{\text{DET}}(\text{rec}X.m',\mathsf{f},\sigma')[^{N_{\text{DET}}(\text{rec}X.n,\mathsf{f},\sigma)}/_X]
\end{aligned}$$

- Case $m = \text{rec}Y.m'$. We have

$$\begin{aligned}
&N_{\text{DET}}((\text{rec}Y.m')[^{\text{rec}X.n}/_X],\mathsf{f},\sigma)\\
&= N_{\text{DET}}(\text{rec}Y.\big(m'[^{\text{rec}X.n}/_X]\big),\mathsf{f},\sigma)\\
&= \text{rec}Y.N_{\text{DET}}(m'[^{\text{rec}X.n}/_X],\mathsf{f},\sigma') \text{ where } \sigma' = \sigma[Y \mapsto \langle \text{rec}Y.m'[^{\text{rec}X.n}/_X],\mathsf{f}\rangle]\\
&= \text{rec}Y.\big(N_{\text{DET}}(m',\mathsf{f},\sigma'')[^{N_{\text{DET}}(\text{rec}X.n,\sigma',\mathsf{f})}/_X]\big) \text{ by the IH where } \sigma'' = \sigma'[X \mapsto \langle \text{rec}X.n,\mathsf{f}\rangle]\\
&= \big(\text{rec}Y.N_{\text{DET}}(m',\mathsf{f},\sigma'')\big)[^{N_{\text{DET}}(\text{rec}X.n,\sigma',\mathsf{f})}/_X]\\
&= N_{\text{DET}}(\text{rec}Y.m',\mathsf{f},\sigma'')[^{N_{\text{DET}}(\text{rec}X.n,\sigma',\mathsf{f})}/_X]\\
&= N_{\text{DET}}(\text{rec}Y.m',\mathsf{f},\sigma'')[^{N_{\text{DET}}(\text{rec}X.n,\sigma'',\mathsf{f})}/_X] \text{ by Lemma 1}\\
&\qquad \text{since } Y \notin fv(\mathbf{cod}(\sigma')) \text{ and the assumption } fv(n) \subseteq \{X\} \text{ implies } Y \notin fv(\text{rec}X.n)
\end{aligned}$$

The remaining cases are more straightforward. □

Lemma 7 shows that the transformation function $N_{\text{DET}}(-)$ preserves history rejections. However, its proof relies on Lemma 6 below.

Lemma 6. Suppose that for any monitor $m \in \text{Mon}$ and $\sigma \in \text{Sub}$, $X \notin fv(\mathbf{cod}(\sigma))$ and $fv(m) \subseteq \{X\}$. If $\mathbf{rej}(H,\text{false},N_{\text{DET}}(m,\text{false},\sigma))$ then $\mathbf{rej}(H,\text{false},N_{\text{DET}}(m,\text{true},\sigma))$.

Proof. The proof proceeds by rule induction on the judgement $\mathbf{rej}(H,\text{false},N_{\text{DET}}(m,\text{false},\sigma))$. We outline the main cases.

- Case ACT, *i.e.,* $\mathbf{rej}(H,\mathsf{false},N_{\mathrm{DET}}(\alpha.m,\mathsf{false},\sigma))$ where $N_{\mathrm{DET}}(\alpha.m,\mathsf{false},\sigma)=\alpha.N_{\mathrm{DET}}(m,\mathsf{false},\sigma)$ because $\mathbf{rej}(H',\mathsf{false},N_{\mathrm{DET}}(m,\mathsf{false},\sigma))$ where $H'=\mathit{suffix}(H,\alpha)$. There are two subcases to consider:

  In the case that $\mathrm{DET}(\alpha)=\mathsf{false}$, we have $N_{\mathrm{DET}}(\alpha.m,\mathsf{false},\sigma)=N_{\mathrm{DET}}(\alpha.m,\mathsf{true},\sigma)$, which implies $\mathbf{rej}(H,\mathsf{false},N_{\mathrm{DET}}(\alpha.m,\mathsf{true},\sigma))$.

  In the case that $\mathrm{DET}(\alpha)=\mathsf{true}$, by the IH, we obtain $\mathbf{rej}(H',\mathsf{false},N_{\mathrm{DET}}(m,\mathsf{true},\sigma))$. Applying rule ACT, we get $\mathbf{rej}(H,\mathsf{false},\alpha.N_{\mathrm{DET}}(m,\mathsf{true},\sigma))$. Our result follows by the fact that $\alpha.N_{\mathrm{DET}}(m,\mathsf{true},\sigma)=N_{\mathrm{DET}}(\alpha.m,\mathsf{true},\sigma)$.

- Case REC, *i.e.,* $\mathbf{rej}(H,\mathsf{false},N_{\mathrm{DET}}(\mathrm{rec}X.m,\mathsf{false},\sigma))$ where, by Definition 9.19, we have that $N_{\mathrm{DET}}(\mathrm{rec}X.m,\mathsf{false},\sigma)=\mathrm{rec}X.N_{\mathrm{DET}}(m,\mathsf{false},\sigma')$ and $\sigma'=\sigma[X\mapsto\langle\mathrm{rec}X.m,\mathsf{false}\rangle]$ because

$$\mathbf{rej}(H,\mathsf{false},N_{\mathrm{DET}}(m,\mathsf{false},\sigma')[^{\mathrm{rec}X.N_{\mathrm{DET}}(m,\mathsf{false},\sigma)}/_X]) \tag{F.1}$$

  By Lemma 5, we also know that

$$N_{\mathrm{DET}}(m,\mathsf{false},\sigma')[^{\mathrm{rec}X.N_{\mathrm{DET}}(m,\mathsf{false},\sigma)}/_X]=N_{\mathrm{DET}}(m[^{\mathrm{rec}X.m}/_X],\mathsf{false},\sigma) \tag{F.2}$$

  By eqs. (F.1) and (F.2) and the IH, we deduce $\mathbf{rej}(H,\mathsf{false},N_{\mathrm{DET}}(m[^{\mathrm{rec}X.m}/_X],\mathsf{true},\sigma))$, which implies that $\mathbf{rej}(H,\mathsf{false},N_{\mathrm{DET}}(m,\mathsf{true},\sigma'')[^{\mathrm{rec}X.N_{\mathrm{DET}}(m,\mathsf{true},\sigma'')}/_X])$ where $\sigma''=\sigma[X\mapsto\langle\mathrm{rec}X.m,\mathsf{true}\rangle]$ by Lemma 5. Applying rule REC, we obtain $\mathbf{rej}(H,\mathsf{false},\mathrm{rec}X.N_{\mathrm{DET}}(m,\mathsf{true},\sigma''))$.
  Our result, $\mathbf{rej}(H,\mathsf{false},N_{\mathrm{DET}}(\mathrm{rec}X.m,\mathsf{true},\sigma))$, follows by Definition 9.19 since $\mathrm{rec}X.N_{\mathrm{DET}}(m,\mathsf{true},\sigma'')=N_{\mathrm{DET}}(\mathrm{rec}X.m,\mathsf{true},\sigma)$. □

Lemma 7. For any monitor $m\in\mathrm{MON}$, $\mathbf{rej}(H,\mathsf{f},m)$ iff $\mathbf{rej}(H,\mathsf{f},N_{\mathrm{DET}}(m,\mathsf{f},\emptyset))$.

Proof. For the "only if" direction, the proof proceeds by rule induction on $\mathbf{rej}(H,\mathsf{f},m)$. We outline the main cases.

- Case ACT, *i.e.,* $\mathbf{rej}(H,\mathsf{f},\alpha.m)$ because $\mathbf{rej}(H',\mathsf{f}',m)$ where $H'=\mathit{suffix}(H,\alpha)$ and $\mathsf{f}'=\mathsf{f}\wedge\mathrm{DET}(\alpha)$. By the IH, we obtain that $\mathbf{rej}(H',\mathsf{f}',N_{\mathrm{DET}}(m,\mathsf{f}',\emptyset))$. Applying rule ACT, we get $\mathbf{rej}(H,\mathsf{f},\alpha.N_{\mathrm{DET}}(m,\mathsf{f}',\emptyset))$ which, by Definition 9.19, implies that $\mathbf{rej}(H,\mathsf{f},N_{\mathrm{DET}}(\alpha.m,\mathsf{f},\emptyset))$.

- Case ACTI, *i.e.,* $\mathbf{rej}(H,\mathsf{f},\alpha.m)$ because $\mathbf{rej}(H',\mathsf{f}',\alpha.m)$ where $H'=\mathit{suffix}(H,\gamma)$ and $\mathsf{f}'=\mathsf{f}\wedge\mathrm{DET}(\gamma)$ for some $\gamma\in\mathrm{IACT}$. By the IH, we obtain $\mathbf{rej}(H',\mathsf{f}',N_{\mathrm{DET}}(\alpha.m,\mathsf{f}',\emptyset))$. There are two subcases to consider. If $\mathsf{f}=\mathsf{f}'$, we can apply rule ACTI and conclude $\mathbf{rej}(H,\mathsf{f},N_{\mathrm{DET}}(\alpha.m,\mathsf{f},\emptyset))$. If $\mathsf{f}\neq\mathsf{f}'$, *i.e.,* $\mathsf{f}=\mathsf{true}$ and $\mathsf{f}'=\mathsf{false}$, then by Lemma 6, we obtain $\mathbf{rej}(H',\mathsf{f}',N_{\mathrm{DET}}(\alpha.m,\mathsf{f},\emptyset))$. Applying rule ACTI, we conclude that $\mathbf{rej}(H,\mathsf{f},N_{\mathrm{DET}}(\alpha.m,\mathsf{f},\emptyset))$.

- Case REC, *i.e.,* $\mathbf{rej}(H,\mathsf{f},\mathrm{rec}X.m)$ because $\mathbf{rej}(H,\mathsf{f},m[^{\mathrm{rec}X.m}/_X])$.
  By the IH, we obtain $\mathbf{rej}(H,\mathsf{f},N_{\mathrm{DET}}(m[^{\mathrm{rec}X.m}/_X],\mathsf{f},\emptyset))$. Using Lemma 5 and Definition 9.19, we know

$$\begin{aligned}&N_{\mathrm{DET}}(m[^{\mathrm{rec}X.m}/_X],\mathsf{f},\emptyset)\\&=N_{\mathrm{DET}}(m,\mathsf{f},\{X\mapsto\langle\mathrm{rec}X.m,\mathsf{f}\rangle\})[^{N_{\mathrm{DET}}(\mathrm{rec}X.m,\mathsf{f},\emptyset)}/_X]\\&=N_{\mathrm{DET}}(m,\mathsf{f},\{X\mapsto\langle\mathrm{rec}X.m,\mathsf{f}\rangle\})[^{\mathrm{rec}X.N_{\mathrm{DET}}(m,\mathsf{f},\{X\mapsto\langle\mathrm{rec}X.m,\mathsf{f}\rangle\})}/_X]\end{aligned}$$

  Let $n=N_{\mathrm{DET}}(m,\mathsf{f},\{X\mapsto\langle\mathrm{rec}X.m,\mathsf{f}\rangle\})$. We thus have $\mathbf{rej}(H,\mathsf{f},n[^{\mathrm{rec}X.n}/_X])$.
  By rule REC, we obtain $\mathbf{rej}(H,\mathsf{f},\mathrm{rec}X.n)$. Our result, $\mathbf{rej}(H,\mathsf{f},N_{\mathrm{DET}}(\mathrm{rec}X.m,\mathsf{f},\emptyset))$, follows by Definition 9.19.

The proof for the "if" direction follows similarly by rule induction on $\mathbf{rej}(H,\mathsf{f},N_{\text{DET}}(m,\mathsf{f},\emptyset))$. The only case that differs slightly is that for REC.

- Case REC. We know $\mathbf{rej}(H,\mathsf{f},N_{\text{DET}}(\mathsf{rec}X.m,\mathsf{f},\emptyset))$, *i.e.,* $\mathbf{rej}(H,\mathsf{f},\mathsf{rec}X.n)$ where

$$\mathsf{rec}X.n = N_{\text{DET}}(\mathsf{rec}X.m,\mathsf{f},\emptyset) = \mathsf{rec}X.N_{\text{DET}}(m,\mathsf{f},\{X \mapsto \langle \mathsf{rec}X.m,\mathsf{f}\rangle\})$$

  because $\mathbf{rej}(H,\mathsf{f},n[^{\mathsf{rec}X.n}/_X])$. Using Definition 9.19 and Lemma 5, we know

$$\begin{aligned} n[^{\mathsf{rec}X.n}/_X] &= N_{\text{DET}}(m,\mathsf{f},\{X \mapsto \langle \mathsf{rec}X.m,\mathsf{f}\rangle\})[^{\mathsf{rec}X.N_{\text{DET}}(m,\mathsf{f},\{X \mapsto \langle \mathsf{rec}X.m,\mathsf{f}\rangle\})}/_X] \\ &= N_{\text{DET}}(m,\mathsf{f},\{X \mapsto \langle \mathsf{rec}X.m,\mathsf{f}\rangle\})[^{N_{\text{DET}}(\mathsf{rec}X.m,\mathsf{f},\emptyset)}/_X] \\ &= N_{\text{DET}}(m[^{\mathsf{rec}X.m}/_X],\mathsf{f},\emptyset) \end{aligned}$$

  We can thus rewrite $\mathbf{rej}(H,\mathsf{f},n[^{\mathsf{rec}X.n}/_X])$ as $\mathbf{rej}(H,\mathsf{f},N_{\text{DET}}(m[^{\mathsf{rec}X.m}/_X],\mathsf{f},\emptyset))$. By the IH, we obtain $\mathbf{rej}(H,\mathsf{f},m[^{\mathsf{rec}X.m}/_X])$. Applying rule REC, we can conclude $\mathbf{rej}(H,\mathsf{f},\mathsf{rec}X.m)$ as required. □

We are now in a position to prove the main result of this chapter, namely Proposition 9.21 from Section 9.3 which is restated below.

Proposition F.7. For all $m \in \text{MON}$ and $H \in \text{HST}$, $\mathbf{rej}_{\text{DET}}(H,m)$ iff $\mathbf{rej}_{\text{DET}}(H,N_{\text{DET}}(m))$.

Proof. Follows from Lemma 7, letting $\mathsf{f} = \mathsf{true}$. □

Appendix III

Practical Aspects

G Lower Bounds

In this chapter, we give the missing proofs in Section 10.2. To prove Proposition 10.9 and Theorem 10.10 from Section 10.2, we first present a few technical results, starting with several properties for the function $lb(-)$ in Definition 10.6, where the meta-function $fv(\varphi)$ returns the free recursion variables in φ.

Lemma 1. For all $\varphi, \psi, \chi \in \text{sHML}^{\vee}_{\text{NF}}$:

(i) $lb(\varphi[\psi/X]) = lb(\varphi[\max Y.\psi/X])$

(ii) $lb(\psi) \leq lb(\chi)$ implies $lb(\varphi[\psi/X]) \leq lb(\varphi[\chi/X])$

(iii) $lb(\varphi) \leq lb(\psi)$ implies $lb(\varphi) \leq lb(\psi[\varphi/X])$

Proof. We only give those for the main cases of (iii); the others follow with a similar but more straightforward argument and can be proven independently.

The proof of (iii) proceeds by induction on ψ. Assume $lb(\varphi) \leq lb(\psi)$. We show $lb(\varphi) \leq lb(\psi[\varphi/X])$.

- When $\psi = Y$, we have $lb(Y) = \infty$ and $lb(\varphi) \leq lb(Y)$. If $X = Y$, result follows immediately since $X[\varphi/X] = \varphi$. If $X \neq Y$, result also follows immediately since $Y[\varphi/X] = Y$.
- When $\psi = [\alpha]\psi'$, we have $lb(\varphi) \leq lb([\alpha]\psi') = lb(\psi')$. By the IH, we deduce $lb(\varphi) \leq lb(\psi'[\varphi/X])$, which implies $lb(\varphi) \leq lb(([\alpha]\psi')[\varphi/X])$.
- When $\psi = \psi_1 \wedge \psi_2$, we have $lb(\varphi) \leq min(lb(\psi_1), lb(\psi_2))$, which implies $lb(\varphi) \leq lb(\psi_1)$ and $lb(\varphi) \leq lb(\psi_2)$. By the IH, we obtain $lb(\varphi) \leq lb(\psi_1[\varphi/X])$ and $lb(\psi_2[\varphi/X])$. Our result, $lb(\varphi) \leq lb((\psi_1 \wedge \psi_2)[\varphi/X])$, follows since $lb((\psi_1 \wedge \psi_2)[\varphi/X]) = min(lb(\psi_1[\varphi/X]), lb(\psi_2[\varphi/X]))$.
- When $\psi = \psi_1 \vee \psi_2$, then $lb(\varphi) \leq lb(\psi_1 \vee \psi_2) = lb(\psi_1) + lb(\psi_2) + 1$. There are two subcases to consider:
  - When $lb(\varphi) \leq lb(\psi_1)$ or $lb(\varphi) \leq lb(\psi_2)$. W.l.o.g. suppose the former. By the IH, we obtain $lb(\varphi) \leq lb(\psi_1[\varphi/X])$, which implies that $lb(\varphi) \leq lb(\psi_1[\varphi/X]) + lb(\psi_2[\varphi/X]) + 1 = lb((\psi_1 \vee \psi_2)[\varphi/X])$.
  - When $lb(\varphi) > lb(\psi_1)$ and $lb(\varphi) > lb(\psi_2)$. By the IH, we obtain $lb(\psi_1) \leq lb(\psi_1[\psi_1/X])$ and $lb(\psi_2) \leq lb(\psi_2[\psi_2/X])$. But by Lemma 1((ii)), we also know $lb(\psi_1[\psi_1/X]) \leq lb(\psi_1[\varphi/X])$ and $lb(\psi_2[\psi_2/X]) \leq lb(\psi_2[\varphi/X])$. This implies that $lb(\varphi) \leq lb(\psi_1) + lb(\psi_2) + 1 \leq lb(\psi_1[\varphi/X]) + lb(\psi_2[\varphi/X]) = lb((\psi_1 \vee \psi_2)[\varphi/X])$, as required.
- When $\psi = \max Y.\psi'$, then $lb(\varphi) \leq lb(\max Y.\psi') = lb(\psi')$. There are two sub-cases to consider:
  If $X = Y$, our result is immediate since $(\max X.\psi')[\varphi/X] = \max X.\psi'$.
  If $X \neq Y$, then by the IH, we obtain $lb(\varphi) \leq lb(\psi'[\varphi/X]) = lb((\max Y.\psi')[\varphi/X])$.

This concludes our proof. □

Corollary G.2. For all $\varphi \in \text{sHML}^{\vee}_{\text{NF}}$, $lb(\varphi) \leq lb(\varphi[\varphi/X])$.

Proof. Follows from Lemma 1(iii) by letting $\varphi = \psi$. □

We give an alternative definition to the violation relation, $\models_{\text{DET}}$, in Definition 9.8 that is specific to $\text{sHML}^{\vee}_{\text{NF}}$ formulae (in contrast to $\models_{\text{DET}}$ which is defined over $\text{sHML}^{\vee}_{\text{DET}}$), namely Definition G.3. Theorem G.8 then shows that these two definitions correspond.

Definition G.3. The *separation violation relation*, denoted as $\models^{s}_{\text{DET}}$, is the least relation of the form $(\text{HST} \times \text{BOOL} \times \text{sHML}^{\vee}_{\text{NF}})$ satisfying the following rules:

$$\textsc{svF}\ \frac{H \neq \emptyset}{(H,\mathsf{f}) \models^{s}_{\text{DET}} \mathsf{ff}} \qquad \textsc{svMax}\ \frac{(H,\mathsf{f}) \models^{s}_{\text{DET}} \varphi[\max X.\varphi/X]}{(H,\mathsf{f}) \models^{s}_{\text{DET}} \max X.\varphi} \qquad \textsc{svUm}\ \frac{H' = \mathit{suffix}(H,\alpha) \quad \mathsf{f}' = \mathsf{f} \wedge \text{DET}(\alpha) \quad (H',\mathsf{f}') \models^{s}_{\text{DET}} \varphi}{(H,\mathsf{f}) \models^{s}_{\text{DET}} [\alpha]\varphi}$$

$$\textsc{svUmPre}\ \frac{H' = \mathit{suffix}(H,\gamma) \quad \mathsf{f}' = \mathsf{f} \wedge \text{DET}(\gamma) \quad (H',\mathsf{f}') \models^{s}_{\text{DET}} [\alpha]\varphi}{(H,\mathsf{f}) \models^{s}_{\text{DET}} [\alpha]\varphi} \qquad \textsc{svAndL}\ \frac{(H,\mathsf{f}) \models^{s}_{\text{DET}} \varphi}{(H,\mathsf{f}) \models^{s}_{\text{DET}} \varphi \wedge \psi} \qquad \textsc{svAndR}\ \frac{(H,\mathsf{f}) \models^{s}_{\text{DET}} \psi}{(H,\mathsf{f}) \models^{s}_{\text{DET}} \varphi \wedge \psi}$$

$$\textsc{svOr}\ \frac{H = H_1 \uplus H_2 \quad (H_1,\mathsf{true}) \models^{s}_{\text{DET}} \varphi \quad (H_2,\mathsf{true}) \models^{s}_{\text{DET}} \psi}{(H,\mathsf{true}) \models^{s}_{\text{DET}} \varphi \vee \psi}$$

We write $H \models^{s}_{\text{DET}} \varphi$ to mean $(H,\mathsf{true}) \models^{s}_{\text{DET}} \varphi$. ■

Width irrevocability also holds for the separation violation relation, Lemma 4.

Lemma 4. For all $\varphi \in \text{sHML}^{\vee}_{\text{NF}}$, if $(H,\mathsf{f}) \models^{s}_{\text{DET}} \varphi$ then $(H \cup H',\mathsf{f}) \models^{s}_{\text{DET}} \varphi$.

Proof. The proof proceeds by rule induction. We only give the proof for the case when $(H,\mathsf{f}) \models^{s}_{\text{DET}} \varphi$ is derived via rule svOr; the other cases are straightforward.

- We know $(H,\mathsf{f}) \models^{s}_{\text{DET}} \varphi_1 \vee \varphi_2$ because $H = H_1 \uplus H_2$ and $(H_1,\mathsf{f}) \models^{s}_{\text{DET}} \varphi_1$ and $(H_2,\mathsf{f}) \models^{s}_{\text{DET}} \varphi_2$. We show that $(H \cup H',\mathsf{f}) \models^{s}_{\text{DET}} \varphi_1 \vee \varphi_2$. Let $H'' = H \backslash H'$ where $H_1 \cap H' = \emptyset$ and $H_2 \cap H' = \emptyset$. By the IH, we know $(H_1 \cup H'',\mathsf{f}) \models^{s}_{\text{DET}} \varphi_1$. Since $(H_1 \cup H'') \cap H_2 = \emptyset$, we conclude $(H \cup H',\mathsf{f}) \models^{s}_{\text{DET}} \varphi_1 \vee \varphi_2$ via rule svOr. □

Lemma 6 shows that all violating systems for $\bigvee_{i \in I} [\alpha_i]\varphi_i$ from $\text{sHML}^{\vee}_{\text{NF}}$ can violate the sub-formulae $[\alpha_i]\varphi_i$ through disjoint histories. This result relies on the helper function $\mathit{start}(H,\alpha) = \{t \mid t = \alpha t' \in H\}$, returning the set of all traces in H that are prefixed with a sequence of internal actions $\gamma \in \text{IACT}^*$, followed by an α action. *E.g.* when $H = \{\delta_1 rsa, \delta_2 rsc, ars\}$, then $\mathit{start}(H,r) = \{\delta_1 rsa, \delta_2 rsc\}$.

Lemma 5. For all α,β such that $\alpha \neq \beta$, $\mathit{start}(H,\alpha) \cap \mathit{start}(H,\beta) = \emptyset$.

Proof. Straightforward by definition. □

Lemma 6. For any (closed) formula $\bigvee_{i \in I} [\alpha_i]\varphi_i \in \text{sHML}^{\vee}_{\text{NF}}$ and history $H \in \text{HST}$,

$$\text{if } (H,\mathsf{f}) \models^{s}_{\text{DET}} \bigvee_{i \in I} [\alpha_i]\varphi_i \text{ then } \left(\mathit{start}(H,\alpha_i),\mathsf{f}\right) \models^{s}_{\text{DET}} [\alpha_i]\varphi_i \text{ for each } i \in I$$

Proof. The proof proceeds by numerical induction on the size of I.

- For the base case, $I = \{1\}$, *i.e.,* $(H,\mathsf{f}) \models^{s}_{\text{DET}} [\alpha]\varphi$. Using Lemma 11, we know $\exists t \in \text{ACT}^{*}$ such that $H' = \mathit{suffix}(H,t\alpha)$ and $\mathsf{f}' = \mathsf{f} \wedge \text{DET}(t\alpha)$ and $(H',\mathsf{f}') \models_{\text{DET}} \varphi$. Let $H'' = \{\alpha u \mid u \in H'\}$, *i.e.,* all traces in H' prefixed with action α. Applying rule SVUM, we obtain $(H'', \mathsf{f} \wedge \text{DET}(t)) \models^{s}_{\text{DET}} [\alpha]\varphi$. Let $H''' = \{tt' \mid t' \in H'\}$, *i.e.,* all traces in H'' prefixed with trace t. Applying rule SVUMPRE n times where n is the length of trace t, we obtain $(H'',\mathsf{f}) \models_{\text{DET}} [\alpha]\varphi$. Since, by definition, $H''' \subseteq \mathit{start}(H,\alpha)$, we can use Lemma 4 to conclude that $(\mathit{start}(H,\alpha),\mathsf{f}) \models^{s}_{\text{DET}} [\alpha]\varphi$.
- For the inductive case, $I=\{1,\ldots,n{+}1\}$. The judgment $(H,\mathsf{f}) \models^{s}_{\text{DET}} \bigvee_{i\in I}[\alpha_i]\varphi_i$ can be expanded to

$$(H,\mathsf{f}) \models^{s}_{\text{DET}} \big([\alpha_1]\varphi_1\big) \vee \Big(\bigvee_{j\in J}[\alpha_j]\varphi_j\Big) \text{ where } J = \{2,\ldots,n{+}1\}$$

  By case analysis, this could have only been derived via rule SVOR, which means that there exist H',H'' such that $H{=}H' \uplus H''$ and $(H',\mathsf{f}) \models^{s}_{\text{DET}} [\alpha_1]\varphi_1$ and $(H'',\mathsf{f}) \models^{s}_{\text{DET}} \bigvee_{j\in J}[\alpha_j]\varphi_j$. Using Lemma 4, we deduce $(p,H) \models^{s}_{\text{DET}} [\alpha_1]\varphi_1$ and $(p,H) \models^{s}_{\text{DET}} \bigvee_{j\in J}[\alpha_j]\varphi_j$. By the IH, we obtain $(\mathit{start}(H,\alpha_1),\mathsf{f}) \models^{s}_{\text{DET}} [\alpha_1]\varphi_1$ and $(\mathit{start}(H,\alpha_j),\mathsf{f}) \models^{s}_{\text{DET}} [\alpha_j]\varphi_j$ for all $j \in J$. We can thus conclude $(\mathit{start}(H,\alpha_i),\mathsf{f}) \models^{s}_{\text{DET}} [\alpha_i]\varphi_i$ for all $i \in I$, as required. □

We show that a similar result holds for the violation relation $\models_{\text{DET}}$.

Lemma 7. For any (closed) formula $\bigvee_{i\in I}[\alpha_i]\varphi_i \in \text{sHML}^{\vee}_{\text{NF}}$ and history $H \subseteq T_p$,

$$\text{if } (H,\mathsf{f}) \models_{\text{DET}} \bigvee_{i\in I}[\alpha_i]\varphi_i \text{ then } \big(\mathit{start}(H,\alpha_i),\mathsf{f}\big) \models_{\text{DET}} [\alpha_i]\varphi_i \text{ for each } i \in I$$

Proof. Proof is similar, but more straightforward, to that for Lemma 7. □

Theorem G.8 (Correspondence). For all $\varphi \in \text{sHML}^{\vee}_{\text{NF}}$ and $H \in \text{HST}$, $(H,\mathsf{f}) \models_{\text{DET}} \varphi$ iff $(H,\mathsf{f}) \models^{s}_{\text{DET}} \varphi$.

Proof. For the *if* direction, we show $(H,\mathsf{f}) \models^{s}_{\text{DET}} \varphi$ implies $(H,\mathsf{f}) \models_{\text{DET}} \varphi$. The proof is by rule induction. We only give the case for when $(H,\mathsf{f}) \models_{\text{DET}} \varphi$ is derived via rule SVOR; all other cases are homogeneous.

- We know $(H,\mathsf{f}) \models^{s}_{\text{DET}} \varphi \vee \psi$ because $\exists H_1,H_2$ such that $H = H_1 \uplus H_2$, $(H_1,\mathsf{f}) \models^{s}_{\text{DET}} \varphi$ and $(H_2,\mathsf{f}) \models^{s}_{\text{DET}} \psi$. By the IH, we deduce $(H_1,\mathsf{f}) \models_{\text{DET}} \varphi$ and $(H_2,\mathsf{f}) \models_{\text{DET}} \psi$, which imply $(H,\mathsf{f}) \models_{\text{DET}} \varphi$ and $(H,\mathsf{f}) \models_{\text{DET}} \psi$ by Proposition D.8. Our result, $(H,\mathsf{f}) \models_{\text{DET}} \varphi \vee \psi$, follows via rule VOR.

For the *only if* direction, we show $(H,\mathsf{f}) \models_{\text{DET}} \varphi$ implies $(H,\mathsf{f}) \models^{s}_{\text{DET}} \varphi$. Again, the proof is by rule induction and we only give that for when $(p,H) \models_{\text{DET}} \varphi$ is derived via rule VOR.

- We know $(H,\mathsf{f}) \models_{\text{DET}} \varphi \vee \psi$ because $(H,\mathsf{f}) \models_{\text{DET}} \varphi$ and $(H,\mathsf{f}) \models_{\text{DET}} \psi$. By the IH, we deduce $(H,\mathsf{f}) \models^{s}_{\text{DET}} \varphi$ and $(H,\mathsf{f}) \models^{s}_{\text{DET}} \psi$. Since $\varphi \vee \psi \in \text{sHML}^{\vee}_{\text{NF}}$, then $\varphi = \bigvee_{i\in I}[\alpha_i]\varphi_i$ and $\psi = \bigvee_{j\in J}[\alpha_j]\psi_j$ where $I \cap J = \emptyset$. By Lemma 6, we obtain $\big(\mathit{start}(H,\alpha_i),\mathsf{f}\big) \models^{s}_{\text{DET}} \varphi_i$ and $\big(\mathit{start}(H,\alpha_j),\mathsf{f}\big) \models^{s}_{\text{DET}} \psi_j$ for each $i \in I$ and $j \in J$. By Lemma 5, we also know $\bigcap_{k\in I\cup J} \mathit{start}(H,\alpha_k) = \emptyset$. Repeatedly applying rule SVOR, we deduce $\big(\bigcup_{i\in I} \mathit{start}(H,\alpha_i),\mathsf{f}\big) \models^{s}_{\text{DET}} \bigvee_{i\in I}[\alpha_i]\varphi_i$ and $\big(\bigcup_{j\in J} \mathit{start}(H,\alpha_j),\mathsf{f}\big) \models^{s}_{\text{DET}} \bigvee_{j\in J}[\alpha_i]\psi_j$. By rule SVOR again, we get $\big(\bigcup_{k\in I\cup J} \mathit{start}(H,\alpha_k),\mathsf{f}\big) \models^{s}_{\text{DET}} \varphi \vee \psi$. Our result, $(H,\mathsf{f}) \models^{s}_{\text{DET}} \varphi \vee \psi$, follows via Lemma 4 since $\bigcup_{k\in I\cup J} \mathit{start}(H,\alpha_k) \subseteq H$.

□

Equipped with these technical results, we prove Proposition 10.9 and Theorem 10.10. Theorem 10.10 is a direct consequence of Lemma 9.

Proposition G.8 (Disjoint Violating Histories). For all (closed) formulae $\varphi \vee \psi \in \text{sHML}^{\vee}_{\text{NF}}$ and histories $H \in \text{HST}$, if $H \models_{\text{DET}} \varphi \vee \psi$ then $H = H' \uplus H''$ such that $H' \models_{\text{DET}} \varphi$ and $H'' \models_{\text{DET}} \psi$.

Proof. Assume $(H,\mathsf{f}) \models_{\text{DET}} \varphi \vee \psi$. Since $\varphi \vee \psi \in \text{sHML}^{\vee}_{\text{NF}}$, we know $\varphi = \bigvee_{i\in I}[\alpha_i]\varphi_i'$ and $\psi = \bigvee_{j\in J}[\alpha_j]\psi_j'$ where $I \cap J = \emptyset$. By Lemma 7, we deduce $(\mathit{start}(H,\alpha_i),\mathsf{f}) \models_{\text{DET}} [\alpha_i]\varphi_i'$ and $(\mathit{start}(H,\alpha_j),\mathsf{f}) \models_{\text{DET}} [\alpha_j]\psi_j'$ for each $i \in I$ and $j \in J$. Let $H_1 = \bigcup_{i\in I} \mathit{start}(H,\alpha_i)$ and $H_2 = \bigcup_{j\in J} \mathit{start}(H,\alpha_j)$. Repeatedly applying rule vOr, we obtain $(H_1,\mathsf{f}) \models_{\text{DET}} \varphi$ and $(H_2,\mathsf{f}) \models_{\text{DET}} \psi$. From Lemma 5, we know $H_1 \cap H_2 = \emptyset$. Let $H_3 = H \backslash H_1$. By Proposition D.8, we get $(H_1,\mathsf{f}) \models_{\text{DET}} \varphi$ and $(H_2 \cup H_3,\mathsf{f}) \models_{\text{DET}} \psi$ where $H = H_1 \uplus (H_2 \cup H_3)$ as required. □

Lemma 9. For any (closed) formula $\varphi \in \text{sHML}^{\vee}_{\text{NF}}$ and $H \in \text{HST}$, if $(H,\mathsf{f}) \models^{s}_{\text{DET}} \varphi$ then $|H| \geq lb(\varphi)+1$.

Proof. The proof is by rule induction.

- Case vF, *i.e.,* $(H,\mathsf{f}) \models^{s}_{\text{DET}} \mathsf{ff}$ where $H \neq \emptyset$. Thus $|H| \geq 1 = lb(\mathsf{ff})+1$.
- Case vUm, *i.e.,* $(H,\mathsf{f}) \models^{s}_{\text{DET}} [\alpha]\varphi$ because $(H',\mathsf{f}') \models^{s}_{\text{DET}} \varphi$ where $H' = \mathit{suffix}(H,\alpha)$ and $\mathsf{f}' = \mathsf{f} \wedge \text{DET}(\alpha)$. By the IH, we deduce $|H'| \geq lb(\varphi)+1$. Let $H'' = \{\alpha t \mid t \in H'\}$. Since $H'' \subseteq H$, then $|H| \geq |H''| = |H'| \geq lb(\varphi)+1 = lb([\alpha]\varphi)+1$.
- Case vUmPre, *i.e.,* $(H,\mathsf{f}) \models^{s}_{\text{DET}} [\alpha]\varphi$ because $(H',\mathsf{f}') \models^{s}_{\text{DET}} [\alpha]\varphi$ where $H' = \mathit{suffix}(H,\gamma)$ and $\mathsf{f}' = \mathsf{f} \wedge \text{DET}(\gamma)$. By the IH, we deduce $|H'| \geq lb([\alpha]\varphi)+1$. Let $H'' = \{\gamma t \mid t \in H'\}$. Since $H'' \subseteq H$, then $|H| \geq |H''| = |H'| \geq lb([\alpha]\varphi)+1$.
- Case vAndL, *i.e.,* $(H,\mathsf{f}) \models^{s}_{\text{DET}} \varphi \wedge \psi$ because $(H,\mathsf{f}) \models^{s}_{\text{DET}} \varphi$. By the IH, $|H| \geq lb(\varphi)+1 \geq \min(lb(\varphi),lb(\psi))+1 = lb(\varphi \wedge \psi)+1$.
- Case vAndR. Analogous to previous case.
- Case vOr, *i.e.,* $(H,\mathsf{true}) \models^{s}_{\text{DET}} \varphi \vee \psi$ because $H = H_1 \uplus H_2$ and $(H_1,\mathsf{true}) \models^{s}_{\text{DET}} \varphi$ and $(H_2,\mathsf{true}) \models^{s}_{\text{DET}} \varphi$. By the IH, we obtain $|H_1| \geq lb(\varphi)+1$ and $|H_2| \geq lb(\psi)+1$, which means that $|H| = |H_1|+|H_2| \geq lb(\varphi)+lb(\psi)+2 = lb(\varphi \vee \psi)+1$.
- Case vMax, *i.e.,* $(H,\mathsf{f}) \models^{s}_{\text{DET}} \max X.\varphi$ because $(H,\mathsf{f}) \models^{s}_{\text{DET}} \varphi[^{\max X.\varphi}/_X]$. By the IH, Lemma 1(i) and Corollary G.2, we conclude $|H| \geq lb(\varphi[^{\max X.\varphi}/_X])+1 = lb(\varphi[^{\varphi}/_X])+1 \geq lb(\varphi)+1 = lb(\max X.\varphi)+1$. □

We are now in a position to prove the main result of this chapter, namely Theorem 10.10, restated below.

Theorem G.10 (History Lower Bounds). For all (closed) formulae $\varphi \in \text{sHML}^{\vee}_{\text{NF}}$ and histories $H \in \text{HST}$, if $H \models_{\text{DET}} \varphi$ then $|H| \geq lb(\varphi)+1$.

Proof. Follows from Lemma 9 and Theorem G.8. □

H Properties of Actor Systems

In this section, we give the missing proofs from Chapter 11. Before embarking on this endeavour, we give some general properties of actor systems.

Lemma 1. If $K \mid O \triangleright A \xrightarrow{i?v} B$ then $i \in \mathit{fId}(A)$.

Proof. Straightforward by rule induction. □

Corollary H.2. If $i \notin \mathit{fId}(A)$ then $K \mid O \triangleright A \xrightarrow{i?v}\!\!\!\!\!\!/\;\; B$.

Proof. Straightforward by rule induction. □

Lemma 3. If $K \mid O \triangleright A \xrightarrow{\tau} B$ then $\mathit{fId}(B) \subseteq \mathit{fId}(A)$.

Proof. Straightforward by rule induction. □

Lemma 4 below states that if a system can perform an input action, then that transition is always possible, regardless of the external observer O w.r.t. which it is executing.

Lemma 4. If $K \mid O \triangleright A \xrightarrow{i?v} B$ then $K \mid O' \triangleright A \xrightarrow{i?v} B$ for every observer O'.

Proof. The proof proceeds by induction on $K \mid O \triangleright A \xrightarrow{i?v} B$.

- Case RCV, *i.e.,* $K \mid O \triangleright i[e \triangleleft q] \xrightarrow{i?v} i[e \triangleleft q{:}v]$. Our result, $K \mid O' \triangleright i[e \triangleleft q] \xrightarrow{i?v} i[e \triangleleft q{:}v]$, follows by rule RCV.
- Case SCP1, *i.e.,* $K \mid O \triangleright (\nu\, j)A \xrightarrow{i?v} (\nu\, j)B$ because $K, j \mid O \triangleright A \xrightarrow{i?v} B$ and $j \sharp \mathit{fn}(i?v)$. By the IH, we obtain $K, j \mid O' \triangleright A \xrightarrow{i?v} B$. Our result, $K \mid O' \triangleright (\nu\, j)A \xrightarrow{i?v} (\nu\, j)B$, follows by rule SCP1.
- Case PARL, *i.e.,* $K \mid O \triangleright A \parallel B \xrightarrow{i?v} A' \parallel B$ because $K \mid O \triangleright A \xrightarrow{i?v} A'$ and $i \sharp \mathit{fId}(B)$. By the IH, we obtain $K \mid O' \triangleright A \xrightarrow{i?v} A'$. Our result, $K \mid O' \triangleright A \parallel B \xrightarrow{i?v} A' \parallel B$, follows by PARL.
- Case PARR. The proof is analogous to that for PARL.

This completes our proof. □

Similarly, Lemma 5 states that if a system can τ-transition, then that transition is always possible, regardless of the knowledge K and external observer O w.r.t. which it is executing.

Lemma 5. If $K \mid O \triangleright A \xrightarrow{\tau} K \mid O \triangleright B$ then $K' \mid O' \triangleright A \xrightarrow{\tau} K' \mid O' \triangleright B$ for all $K, K', O, O' \subseteq \text{PID}$.

Proof. Straightforward by rule induction. □

We also prove several results that provide insights into the structure and behaviour of actor systems.

Lemma 6. If $A \equiv A_1 \parallel A_2$ then one of the following statements must hold:

- $A = A_1$ and $A_2 = \mathbf{0}$, or $A = A_2$ and $A_1 = \mathbf{0}$
- $A_1 \equiv (\nu \vec{h}_1) A_1' \parallel (\nu \vec{h}_2) A_1''$ and $A_2 \equiv (\nu \vec{h}_3) A_2' \parallel (\nu \vec{h}_4) A_2''$ and $A = (\nu \vec{h}_1, \vec{h}_2, \vec{h}_3, \vec{h}_4)(B_1 \parallel B_2)$ and $B_1 = A_1' \parallel A_2'$ and $B_2 = A_1'' \parallel A_2''$.

Proof. By rule induction on $A \equiv A_1 \parallel A_2$. □

Lemma 7. If $A \equiv A_1 \parallel A_2$ and $K \mid O \triangleright A \xrightarrow{i?v} B$ then

(i) either $K \mid O \triangleright A_1 \xrightarrow{i?v} A_1'$ and $B \equiv A_1' \parallel A_2$;
(ii) or $K \mid O \triangleright A_2 \xrightarrow{i?v} A_2'$ and $B \equiv A_1 \parallel A_2'$.

Proof. Suppose that $A \equiv A_1 \parallel A_2$ and $K \mid O \triangleright A \xrightarrow{i?v} B$. The proof is by rule induction on the latter.

- Case RCV, *i.e.,* $A = i[e \triangleleft q]$ and $A' = i[e \triangleleft q{:}v]$. Since $A = A_1 \parallel A_2$, then by Lemma 6, we must have either $A = A_1$ and $A_2 = \mathbf{0}$, or $A = A_2$ and $A_1 = \mathbf{0}$. In the first case, condition (1) is satisfied since $i[e \triangleleft q{:}v] \equiv i[e \triangleleft q{:}v] \parallel \mathbf{0}$. Otherwise, condition (2) is satisfied since $i[e \triangleleft q{:}v] \equiv \mathbf{0} \parallel i[e \triangleleft q{:}v]$.
- Case PARL, *i.e.,* $A = A_3 \parallel A_4$ and $B = B_3 \parallel A_4$ because $K \mid O \triangleright A_3 \xrightarrow{i?v} B_3$. By Lemma 6 and $A = A_3 \parallel A_4$, we must have $A_1 \equiv A_1' \parallel A_1''$ and $A_2 \equiv A_2' \parallel A_2''$ and $A_3 = A_1' \parallel A_2'$ and $A_4 = A_1'' \parallel A_2''$. Using the facts that $A_3 = A_1' \parallel A_2'$ and $K \mid O \triangleright A_3 \xrightarrow{i?v} B_3$ and the IH, we know that

$$\text{either } K \mid O \triangleright A_1' \xrightarrow{i?v} B_1' \text{ and } B_3 \equiv B_1' \parallel A_2'$$
$$\text{or } K \mid O \triangleright A_2' \xrightarrow{i?v} B_2' \text{ and } B_3 \equiv A_1' \parallel B_2'$$

  Applying rule PARL on the transitions and rule SCTXP on the equivalences, these resp. give us that

$$\text{either } K \mid O \triangleright A_1' \parallel A_1'' \xrightarrow{i?v} B_1' \parallel A_1'' \text{ and } B_3 \parallel A_4 \equiv (B_1' \parallel A_2') \parallel A_4$$
$$\text{or } K \mid O \triangleright A_2' \parallel A_2'' \xrightarrow{i?v} B_2' \parallel A_2'' \text{ and } B_3 \parallel A_4 \equiv (A_1' \parallel B_2') \parallel A_4$$

  Using $A_4 = A_1'' \parallel A_2''$, $A_1 = A_1' \parallel A_1''$, $A_2 = A_2' \parallel A_2''$, $B = B_3 \parallel A_4$ and rules for $\equiv$, we can rewrite this as

$$\text{either } K \mid O \triangleright A_1 \xrightarrow{i?v} B_1' \parallel A_1'' \text{ and } B \equiv (B_1' \parallel A_1'') \parallel A_2$$
$$\text{or } K \mid O \triangleright A_2 \xrightarrow{i?v} B_2' \parallel A_2'' \text{ and } B \equiv A_1 \parallel (B_2' \parallel A_2'')$$

  which correspond to conditions (1) and (2).
- Case PARR, similar to that for PARL.
- Case SCP1, *i.e.,* $A = (\nu j) A'$ and $B = (\nu j) A''$ because $K, j \mid O \triangleright A' \xrightarrow{i?v} A''$ and $j \sharp fn(i,v)$. By Lemma 6, there are two subcases to consider:
  - When $A = A_1$ and $A_2 = \mathbf{0}$, by this and $K \mid O \triangleright (\nu j) A' \xrightarrow{i?v} (\nu j) A''$, we know $K \mid O \triangleright A_1 \xrightarrow{i?v} (\nu j) A''$. By rule SNIL, we conclude $B = (\nu j) A'' \equiv B \parallel \mathbf{0} = B \parallel A_2$ as required.

– When $A_1 \equiv (\nu \vec{h}_1)A_1' \parallel (\nu \vec{h}_2)A_1''$ and $A_2 \equiv (\nu \vec{h}_3)A_2' \parallel (\nu \vec{h}_4)A_2''$ such that

$$A = (\nu \vec{h}_1, \vec{h}_2, \vec{h}_3, \vec{h}_4)(A_3 \parallel A_4) \quad \text{where} \quad A_3 = A_1' \parallel A_2' \quad \text{and} \quad A_4 = A_1'' \parallel A_2''$$

Since $A = (\nu j)A'$, we must have that $j \in \vec{h}_i$ for some $i \in \{1,2,3,4\}$. Consider the case for when $i = 1$; other cases follow with similar reasoning. Let $\vec{h}_5 = \vec{h}_1 \setminus \{j\}$. Then we know

$$A' = (\nu \vec{h}_5, \vec{h}_2, \vec{h}_3, \vec{h}_4)(A_3 \parallel A_4) \tag{H.1}$$

Since we work up to α-conversion of bound entities, we can assume that $\vec{h}_1 \sharp fn(A_4)$ and $\vec{h}_3 \sharp fn(A_4)$ and $\vec{h}_2 \sharp fn(A_3)$ and $\vec{h}_4 \sharp fn(A_3)$. Using the fact that $\vec{h}_5 \subset \vec{h}_1$ and the rules defining $\equiv$ in Figure 11.2, we also know

$$A' \equiv (\nu \vec{h}_5, \vec{h}_3)A_3 \parallel (\nu \vec{h}_2, \vec{h}_4)A_4 \tag{H.2}$$

By eq. (H.1), the fact that $K, j \mid O \triangleright A' \xrightarrow{i?v} A''$ and the IH, we obtain that

$$\text{either} \quad K, j \mid O \triangleright (\nu \vec{h}_5, \vec{h}_3)A_3 \xrightarrow{i?v} B_1 \text{ and } A'' \equiv B_1 \parallel (\nu \vec{h}_2, \vec{h}_4)A_4 \tag{H.3}$$

$$\text{or} \quad K, j \mid O \triangleright (\nu \vec{h}_2, \vec{h}_4)A_4 \xrightarrow{i?v} B_2 \text{ and } A'' \equiv (\nu \vec{h}_5, \vec{h}_3)A_3 \parallel B_2 \tag{H.4}$$

If eq. (H.3) holds, applying rules SCP1,SCTXS,SEXT,SCOM gives us that

$$(\nu \vec{h}_1, \vec{h}_3)A_3 \xrightarrow{i?v} (\nu j)B_1 \quad \text{and} \quad (\nu j)A'' \equiv (\nu j)B_1 \parallel (\nu \vec{h}_2, \vec{h}_4)A_4$$

which corresponds to statement (1) as required.
If eq. (H.4) holds, applying rules SCP1,SCTXS,SEXT,SCOM give us

$$K \mid O \triangleright (\nu j, \vec{h}_2, \vec{h}_4)A_4 \xrightarrow{i?v} (\nu j)B_2 \quad \text{and} \quad (\nu j)A'' \equiv (\nu \vec{h}_5, \vec{h}_3)A_3 \parallel (\nu j)B_2$$

which corresponds to statement (2) as required.

• Case STRN, proof is straightforward.

This completes our proof. □

Corollary H.8. If $i[e \triangleleft q] \equiv A$ and $K \mid O \triangleright A \xrightarrow{i?v} B$ then $B \equiv i[e \triangleleft q{:}v]$.

Proof. Follows from Lemma 7 since $K \mid O \triangleright i[e \triangleleft q] \xrightarrow{i?v} i[e \triangleleft q{:}v]$ and $i[e \triangleleft q] \equiv A \parallel \mathbf{0}$. □

Lemma 9. If $A \equiv A_1 \parallel A_2$ and $K \mid O \triangleright A \xrightarrow{i!v} B$ then

(i) either $K \mid O \triangleright A_1 \xrightarrow{i!v} A_1'$ and $B \equiv A_1' \parallel A_2$;
(ii) or $K \mid O \triangleright A_2 \xrightarrow{i!v} A_2'$ and $B \equiv A_1 \parallel A_2'$.

Proof. The proof is similar to that for Lemma 7. □

Lemma 10. If $A \equiv (\nu i)A'$ and $K \mid O \triangleright A \xrightarrow{\eta} B$ and $i \sharp fn(\eta)$ then $B \equiv (\nu i)B'$ and $K, i \mid O \triangleright A' \xrightarrow{\eta} B'$.

Proof. Proof is straightforward. □

Lemma 11. If $A \equiv A_1 \parallel A_2$ and $K \mid O \triangleright A \xrightarrow{\eta} B$ then one of the following statements must hold:

(i) $K \mid O \triangleright A_1 \xrightarrow{\eta} A_1'$ and $B \equiv A_1' \parallel A_2$ and $sbj(\eta) \sharp fId(A_2)$
(ii) $K \mid O \triangleright A_2 \xrightarrow{\eta} A_2'$ and $B \equiv A_1 \parallel A_2'$ and $sbj(\eta) \sharp fId(A_1)$
(iii) $\eta = com(i,v)$ and $K \mid fId(A_2) \triangleright A_1 \xrightarrow{i!v} B_1$ and $K \mid fId(A_1) \triangleright A_2 \xrightarrow{i?v} B_2$ and $B \equiv B_1 \parallel B_2$
(iv) $\eta = com(i,v)$ and $K \mid fId(A_2) \triangleright A_1 \xrightarrow{i?v} B_1$ and $K \mid fId(A_1) \triangleright A_2 \xrightarrow{i!v} B_2$ and $B \equiv B_1 \parallel B_2$
(v) $\eta = ncom$ and $K \mid fId(A_2) \triangleright A_1 \xrightarrow{i \uparrow j} B_1$ and $K \mid fId(A_1) \triangleright A_2 \xrightarrow{i?j} B_2$ and $B \equiv (\nu\, j)(B_1 \parallel B_2)$
(vi) $\eta = ncom$ and $K \mid fId(A_2) \triangleright A_1 \xrightarrow{i?j} B_1$ and $K \mid fId(A_1) \triangleright A_2 \xrightarrow{i \uparrow j} B_2$ and $B \equiv (\nu\, j)(B_1 \parallel B_2)$

Proof. We omit the proof due to its length. However, it can be proven via rule induction on $K \mid O \triangleright A \xrightarrow{\eta} B$, using also Lemma 6. The method is similar to that for Lemma 7. □

H.1 Actor Structural Equivalence and Silent Actions

This section is dedicated to the proof of Proposition 11.4 from Section 11.2, stating that (non-traceable) silent transitions are confluent w.r.t. other actions.

Proposition H.11 (τ-Confluence). If $K \mid O \triangleright A \xrightarrow{\tau} A'$ and $K \mid O \triangleright A \xrightarrow{\eta} A''$, then either $\eta = \tau$ and $A' \equiv A''$ or there exists an actor system B and moves $K \mid O \triangleright A' \xrightarrow{\eta} B$ and $\mathbf{aft}(K \mid O, \eta) \triangleright A'' \xrightarrow{\tau} B$.

Proof. Intuitively, this is true because if $K \mid O \triangleright A$ does two *different* moves $K \mid O \triangleright A \xrightarrow{\tau} A'$ and $K \mid O \triangleright A \xrightarrow{\eta} A''$, then both moves must have occurred in different components of A. The proof proceeds by induction on the derivation of the first move, $K \mid O \triangleright A \xrightarrow{\tau} A'$.

All axioms are trivial. Rules commL, commR, ncommL, ncommR, scp2 and opn do not occur in any derivation of a τ move. For the inductive cases, the only non-straightforward rule is parL (the proof for parR is analogous). If $K \mid O \triangleright A \xrightarrow{\tau} K \mid O \triangleright A'$ was derived using this rule, then

$$\begin{aligned} &K \mid O \triangleright A_1 \parallel A_2 \xrightarrow{\tau} K \mid O \triangleright A_1' \parallel A_2 \\ &\text{because } K \mid O \triangleright A_1 \xrightarrow{\tau} K \mid O \triangleright A_1' \end{aligned} \tag{H.5}$$

We examine the proof for the second move, which must be of the form $K \mid O \triangleright A_1 \parallel A_2 \xrightarrow{\eta} \mathbf{aft}(K \mid O, \eta) \triangleright A''$. By Lemma 11, there are six possible ways how this could have occurred. We focus on the main cases:

- $A'' = A_1 \parallel A_2'$ and $K \mid O \triangleright A_2 \xrightarrow{\eta} K' \triangleright A_2'$ where $K' \mid O' = \mathbf{aft}(K \mid O, \eta)$ and $sbj(\eta) \sharp fId(A_1)$. Diagrammatically

$$\begin{array}{ccc} K \mid O \triangleright A_1 \parallel A_2 & \xrightarrow{\quad\tau\quad} & K \mid O \triangleright A_1' \parallel A_2 \\ \Big\downarrow \eta & & \\ K' \mid O' \triangleright A_1 \parallel A_2' & & \end{array}$$

  From Lemma 5 and eq. (H.5), we get $K' \mid O' \triangleright A_1 \xrightarrow{\tau} K' \mid O' \triangleright A_1'$. Since $sbj(\tau) = \emptyset$, thus $sbj(\tau) \sharp fId(A_2)$, we can apply rule parL to get the move $K' \mid O' \triangleright A_1 \parallel A_2' \xrightarrow{\tau} K' \mid O' \triangleright A_1' \parallel A_2'$. By Lemma 3, we also know $fId(A_1') \subseteq fId(A_1)$, which implies $sbj(\eta) \sharp fId(A_1')$. Rule parR can thus be applied to $K \mid O \triangleright A_2 \xrightarrow{\eta} K' \mid O' \triangleright A_2'$

to obtain the move $K \mid O \triangleright A_1' \parallel A_2 \xrightarrow{\eta} K' \mid O' \triangleright A_1' \parallel A_2'$. This gives us the required commuting diagram

$$\begin{array}{ccc} K \mid O \triangleright A_1 \parallel A_2 & \xrightarrow{\quad\tau\quad} & K \mid O \triangleright A_1' \parallel A_2 \\ \eta \downarrow & & \eta \downarrow \\ K' \mid O' \triangleright A_1 \parallel A_2' & \xrightarrow{\quad\tau\quad} & K' \mid O' \triangleright A_1' \parallel A_2' \end{array}$$

- $A'' = A_1' \parallel A_2$ and $K \mid O \triangleright A_1 \xrightarrow{\eta} K' \mid O' \triangleright A_1'$ where $K' \mid O' = \mathbf{aft}(K \mid O, \eta)$ and $sbj(\eta) \sharp fId(A_1)$. In other words, we have to complete the diagram

$$\begin{array}{ccc} K \mid O \triangleright A_1 \parallel A_2 & \xrightarrow{\quad\tau\quad} & K \mid O \triangleright A_1' \parallel A_2 \\ \eta \downarrow & & \eta \vdots \\ K' \mid O' \triangleright A_1'' \parallel A_2 & \dashrightarrow^{\tau} & K' \mid O' \triangleright A_1''' \parallel A_2 \end{array}$$

But note that we also have the diagram

$$\begin{array}{ccc} K \mid O \triangleright A_1 & \xrightarrow{\quad\tau\quad} & K \mid O \triangleright A_1' \\ \eta \downarrow & & \\ K' \mid O' \triangleright A_1'' & & \end{array}$$

that can be completed by induction as

$$\begin{array}{ccc} K \mid O \triangleright A_1 & \xrightarrow{\quad\tau\quad} & K \mid O \triangleright A_1' \\ \eta \downarrow & & \eta \downarrow \\ K' \mid O' \triangleright A_1'' & \xrightarrow{\quad\tau\quad} & K' \mid O' \triangleright A_1''' \end{array}$$

Applying PARL twice on $K' \mid O' \triangleright A_1'' \xrightarrow{\tau} K' \mid O' \triangleright A_1'''$ and $K \mid O \triangleright A_1' \xrightarrow{\eta} K' \mid O' \triangleright A_1'''$ give the two required moves.

- $\eta = ncom$ and $A'' = (\nu\, j)\left(A_1'' \parallel A_2''\right)$ and $K \mid fId(A_2) \triangleright A_1 \xrightarrow{i \uparrow j} K' \mid O' \triangleright A_1''$ and $K \mid fId(A_1) \triangleright A_2 \xrightarrow{i?j} K'' \mid O'' \triangleright A_2''$ for some name $i, j \in \text{PID}$ where $K' \mid O' = \mathbf{aft}(K \mid fId(A_2), i \uparrow j)$ and $K'' \mid O'' = \mathbf{aft}(K \mid fId(A_1), i?j)$. In other words, we have to complete the diagram

$$\begin{array}{ccc} K \mid O \triangleright A_1 \parallel A_2 & \xrightarrow{\quad\tau\quad} & K \mid O \triangleright A_1' \parallel A_2 \\ ncom \downarrow & & ncom \vdots \\ K \mid O \triangleright (\nu\, j)\left(A_1'' \parallel A_2''\right) & \dashrightarrow^{\tau} & K \mid O \triangleright (\nu\, j)\left(A_1''' \parallel A_2''\right) \end{array}$$

But note that we also have the diagram

$$\begin{array}{ccc} K \mid fId(A_2) \triangleright A_1 & \xrightarrow{\quad\tau\quad} & K \mid fId(A_2) \triangleright A_1' \\ i \uparrow j \downarrow & & \\ K' \mid O' \triangleright A_1'' & & \end{array}$$

that can be completed by induction as

$$\begin{array}{ccc} K \mid \mathit{fId}(A_2) \triangleright A_1 & \xrightarrow{\quad\tau\quad} & K \mid \mathit{fId}(A_2) \triangleright A_1' \\ {\scriptstyle i\uparrow j}\downarrow & & \downarrow{\scriptstyle i\uparrow j} \\ K' \mid O' \triangleright A_1'' & \xrightarrow{\quad\tau\quad} & K' \mid O' \triangleright A_1''' \end{array}$$

Applying NCOMML to $K \mid \mathit{fId}(A_2) \triangleright A_1' \xrightarrow{i\uparrow j} K' \mid O' \triangleright A_1'''$ and $K \mid \mathit{fId}(A_1) \triangleright A_2 \xrightarrow{i?j} K'' \mid O'' \triangleright A_2''$ gives the first required move

$$K \mid O \triangleright A_1' \parallel A_2 \xrightarrow{ncom} K \mid O \triangleright (\nu\, j)(A_1''' \parallel A_2'')$$

By Lemma 5 and $K; \mid O' \triangleright A_1'' \xrightarrow{\tau} K' \mid O' \triangleright A_1'''$, we also know $K \mid O \triangleright A_1'' \xrightarrow{\tau} K \mid O \triangleright A_1'''$. By rule PARL, we get $K \mid O \triangleright A_1'' \parallel A_2'' \xrightarrow{\tau} K \mid O \triangleright A_1''' \parallel A_2''$. Then applying SCP1, we obtain the second required move

$$K \mid O \triangleright (\nu\, j)(A_1'' \parallel A_2'') \xrightarrow{\tau} K \mid O \triangleright (\nu\, j)(A_1''' \parallel A_2'')$$

The remaining cases follow with similar reasoning. □

H.2 Actor Traceable Actions

In this section, we prove Propositions 11.6 to 11.8 from Section 11.3, respectively stating that input actions, output actions and communication actions are deterministic.

Proposition H.11 (Input Determinacy). For all systems $K \mid O \triangleright A$ and input actions $i?v$, if $K \mid O \triangleright A \xrightarrow{i?v} A'$ and $K \mid O \triangleright A \xrightarrow{i?v} A''$ then $A' \equiv A''$.

Proof. Follows from Lemma 12 below since $A \equiv A$. □

Lemma 12. For any $A \equiv A'$, if $K \mid O \triangleright A \xrightarrow{i?v} B$ and $K \mid O \triangleright A' \xrightarrow{i?v} B'$ then $B \equiv B'$.

Proof. Suppose $A \equiv A'$ and $K \mid O \triangleright A \xrightarrow{i?v} B$ and $K \mid O \triangleright A' \xrightarrow{i?v} B'$. We show $B \equiv B'$. The proof proceeds by induction on the first move.

- Case RCV, *i.e.,* $K \mid O \triangleright i[e \triangleleft q] \xrightarrow{i?v} i[e \triangleleft q{:}v]$. For the second move, we thus have $K \mid O \triangleright A' \xrightarrow{i?v} B'$ where $i[e \triangleleft q] \equiv A'$. By Corollary H.8, we obtain that $B' \equiv i[e \triangleleft q{:}v]$. Our result, $i[e \triangleleft q{:}v] \equiv B'$, follows by symmetry.
- Case SCP1, *i.e.,* $K \mid O \triangleright (\nu\, j)A \xrightarrow{i?v} (\nu\, j)B$ because $K, j \mid O \triangleright A \xrightarrow{i?v} B$ because $j \sharp \mathit{fn}(i!v)$. For the second move, we have $K \mid O \triangleright A' \xrightarrow{i?v} B'$ where $A' \equiv (\nu\, j)A$. By Lemma 10, we know $K, j \mid O \triangleright A \xrightarrow{i?v} B''$ and $B' \equiv (\nu\, j)B''$. By the IH and the fact that $A \equiv A$, we obtain $B \equiv B''$. Our result, $(\nu\, j)B \equiv (\nu\, j)B''$, follows by rule SCTXS.
- Case PARL, *i.e.,* $K \mid O \triangleright A_1 \parallel A_2 \xrightarrow{i?v} B_1 \parallel A_2$ because $K \mid O \triangleright A_1 \xrightarrow{i?v} B_1$ and $\mathit{sbj}(i?v) \sharp \mathit{fId}(A_2)$. For the second move, we have $K \mid O \triangleright A' \xrightarrow{i?v} B'$ where $A' \equiv A_1 \parallel A_2$. By Lemma 7, Corollary H.2 and $i \sharp \mathit{fId}(A_2)$, we know $K \mid O \triangleright A_1 \xrightarrow{i?v} B_1'$ and $B' \equiv B_1' \parallel A_2$ for some B_1'. By the IH, $B_1 \equiv B_1'$. Our result, $B_1 \parallel A_2 \equiv B_1' \parallel A_2$, follows by SCTXP.

- Case parR, proof is analogous to that for case parL.
- Case str, *i.e.,* $K \mid O \triangleright A \xrightarrow{i?v} B$ because $A \equiv A''$ and $K \mid O \triangleright A'' \xrightarrow{i?v} B''$ and $B'' \equiv B$. For the second move, we have $K \mid O \triangleright A' \xrightarrow{i?v} B'$ where $A' \equiv A''$. By transitivity, we know $A'' \equiv A'$ as well. Using the IH, we thus obtain that $B'' \equiv B'$. Our result, $B \equiv B'$, follows by symmetry and transitivity. □

The proof for Proposition 11.7 (output determinacy) relies on Lemma 13 below, which describes the structure of actors capable of performing an output action $i!v$.

Lemma 13. If $K \mid O \triangleright A \xrightarrow{i!v} B$ then $A \equiv A' \parallel i\langle v\rangle$ and $B \equiv A'$.

Proof. Suppose $K \mid O \triangleright A \xrightarrow{i!v} B$. The proof proceeds by induction on $K \mid O \triangleright A \xrightarrow{i!v} B$.

- Case snd2, *i.e.,* $K \mid O \triangleright i\langle v\rangle \xrightarrow{i!v} \mathbf{0}$ where $i \in O$. Result is immediate by rules sNil, sCom.
- Case scp1, *i.e.,* $K \mid O \triangleright (\nu\, j)A \xrightarrow{i!v} (\nu\, j)B$ because $K, j \mid O \triangleright A \xrightarrow{i!v} B$ and $j \sharp fn(i!v)$. By the IH, we obtain that $A \equiv A' \parallel i\langle v\rangle$ and $B \equiv A'$ for some A'. Applying rule sCtxt, we get $(\nu\, j)A \equiv (\nu\, j)(A' \parallel i\langle v\rangle)$ and $(\nu\, j)B \equiv (\nu\, j)A'$.
  There only remains to show that $(\nu\, j)A \equiv ((\nu\, j)A') \parallel i\langle v\rangle$. Since $j \sharp fn(i!v)$, we know $j \sharp fn(i\langle v\rangle)$. Thus, by rule sExt, $(\nu\, j)(A' \parallel i\langle v\rangle) \equiv ((\nu\, j)A') \parallel i\langle v\rangle$. Our result, $(\nu\, j)A \equiv ((\nu\, j)A') \parallel i\langle v\rangle$, follows by transitivity.
- Case parL, *i.e.,* $K \mid O \triangleright A_1 \parallel A_2 \xrightarrow{i!v} B_1 \parallel A_2$ because $K \mid O \triangleright A_1 \xrightarrow{i!v} B_1$ and $i \sharp fld(A_2)$. By the IH, we obtain $A_1 \equiv A_1' \parallel i\langle v\rangle$ and $B_1 \equiv A_1'$ for some A_1'. Applying rule sCtxP, we get $A_1 \parallel A_2 \equiv (A_1' \parallel i\langle v\rangle) \parallel A_2$ and $B_1 \parallel A_2 \equiv A_1' \parallel A_2$.
  There remains to show $A_1 \parallel A_2 \equiv (A_1' \parallel A_2) \parallel i\langle v\rangle$; this follows by rules sAss, sCom.
- Case parR, analogous to that of parL.
- Case sTrn, *i.e.,* $K \mid O \triangleright A \xrightarrow{i!v} B$ because $A \equiv A''$, $B \equiv B'$ and $K \mid O \triangleright A'' \xrightarrow{i\langle v\rangle} B'$. By the IH, $A'' \equiv A' \parallel i\langle v\rangle$ and $B' \equiv A'$. By transitivity and symmetry of $\equiv$, we can thus conclude $A \equiv A' \parallel i\langle v\rangle$ and $B \equiv A'$. □

We can now prove Proposition 11.7, restated below.

Proposition H.13 (Output Determinacy). For all systems $K \mid O \triangleright A$ and output actions $i!v$, if $K \mid O \triangleright A \xrightarrow{i!v} A'$ and $K \mid O \triangleright A \xrightarrow{i!v} A''$ then $A' \equiv A''$.

Proof. Follows from Lemma 14 below since $A \equiv A$. □

Lemma 14. For any $A \equiv A'$, if $K \mid O \triangleright A \xrightarrow{i!v} B$ and $K \mid O \triangleright A' \xrightarrow{i!v} B'$ then $B \equiv B'$.

Proof. Suppose $A \equiv A'$ and $K \mid O \triangleright A \xrightarrow{i!v} B$ and $K \mid O \triangleright A' \xrightarrow{i!v} B'$. We show $B \equiv B'$. The proof proceeds by induction on the first move.

- Case snd2, *i.e.,* $K \mid O \triangleright i\langle v\rangle \xrightarrow{i!v} \mathbf{0}$ where $i \in O$. For the second move, we have $K \mid O \triangleright A' \xrightarrow{i!v} B'$ where $A' \equiv i\langle v\rangle$. By Lemma 13, we know $A' \equiv A'' \parallel i\langle v\rangle$ and $B' \equiv A''$ for some A''. But since $A' \equiv i\langle v\rangle$, we can show $A'' \equiv \mathbf{0}$. Our result, $\mathbf{0} \equiv B'$, follows.
- Case scp1, *i.e.,* $K \mid O \triangleright (\nu\, j)A \xrightarrow{i!v} (\nu\, j)B$ because $K, j \mid O \triangleright A \xrightarrow{i!v} B$ and $j \sharp fn(v) \cup \{i\}$. Consider the second move, $K \mid O \triangleright A' \xrightarrow{i!v} B'$ where $(\nu\, j)A \equiv A'$. By Lemma 10, we know $B' \equiv (\nu\, j)B''$ and $K, j \mid O \triangleright A \xrightarrow{i!v} B''$. By the IH, we obtain that $B \equiv B''$. By rule sCtxS, $(\nu\, j)B \equiv (\nu\, j)B''$. Our result, $(\nu\, j)B \equiv B'$, follows by transitivity/symmetry.

- Case parL, *i.e.,* $K \mid O \triangleright A_1 \parallel A_2 \xrightarrow{i!v} B_1 \parallel A_2$ because $K \mid O \triangleright A_1 \xrightarrow{i!v} B_1$ and $i \sharp fId(A_2)$. Consider the second move, $K \mid O \triangleright A' \xrightarrow{i!v} B'$ where $A_1 \parallel A_2 \equiv A'$. By Lemma 9, we have two sub-cases:
  - For the first subcase, $K \mid O \triangleright A_1 \xrightarrow{i!v} B_1'$ and $B' \equiv B_1' \parallel A_2$. By the IH, we obtain $B_1 \equiv B_1'$. By rule sCxtP, $B_1 \parallel A_2 \equiv B_1' \parallel A_2$. Our result, $B_1 \parallel A_2 \equiv B'$, follows by transitivity/symmetry.
  - For the second subcase, $K \mid O \triangleright A_2 \xrightarrow{i!v} B_2$ and $B' \equiv A_1 \parallel B_2$. Lemma 13, we obtain $A_1 \equiv A_1' \parallel i\langle v \rangle$ and $A_2 \equiv A_2' \parallel i\langle v \rangle$ and $B_1 \equiv A_1'$ and $B_2 \equiv A_2'$. This implies that

$$\begin{aligned} B_1 \parallel A_2 &\equiv A_1' \parallel (A_2' \parallel i\langle v \rangle) \\ &\equiv (A_1' \parallel i\langle v \rangle) \parallel A_2' \text{ using rules sCom and sAss} \\ &\equiv A_1 \parallel B_2 \equiv B' \end{aligned}$$

    Our result, $B_1 \parallel A_2 \equiv B'$, follows by transitivity.
- Case parR, analogous to that of parL.
- Case str, *i.e.,* $K \mid O \triangleright A \xrightarrow{i!v} B$ because $A \equiv A''$ and $K \mid O \triangleright A'' \xrightarrow{i!v} B''$ and $B'' \equiv B$. For the second move, we have $K \mid O \triangleright A' \xrightarrow{i!v} B'$ where $A' \equiv A''$. By transitivity, we know $A'' \equiv A'$ as well. Using the IH, we thus obtain that $B'' \equiv B'$. Our result, $B \equiv B'$, follows by symmetry and transitivity. □

The proof for Proposition 11.8 (communication determinacy) relies on Lemma 15, describing the structure of an actor system capable of performing an internal communication action $com(i,v)$.

Lemma 15. If $K \mid O \triangleright A \xrightarrow{com(i,v)} B$ then

(i) $A \equiv A' \parallel i\langle v \rangle$;
(ii) $K \mid O \triangleright A' \xrightarrow{i?v} B'$ for some B';
(iii) $B \equiv B'$.

Proof. Assume $K \mid O \triangleright A \xrightarrow{com(i,v)} B$. We proceed by rule induction, outlining only the main cases; the remaining follow similarly.

- Case commL, *i.e.,* $A = A_1 \parallel A_2$ and $B = B_1 \parallel B_2$ because $K \mid fId(A_2) \triangleright A_1 \xrightarrow{i!v} B_1$ and $K \mid fId(A_1) \triangleright A_2 \xrightarrow{i?v} B_2$. By Lemma 13, we know $A_1 \equiv A_1' \parallel i\langle v \rangle$ and $B_1 \equiv A_1'$. By rules sCtxP, sAss, sCom and transitivity, this implies $A_1 \parallel A_2 \equiv (A_1' \parallel A_2) \parallel i\langle v \rangle$, giving us (i).
  Applying rule parR on $K \mid fId(A_1) \triangleright A_2 \xrightarrow{i?v} B_2$, we obtain $K \mid fId(A_1) \triangleright A_1' \parallel A_2 \xrightarrow{i?v} A_1' \parallel B_2$. Using Lemma 4, we get $K \mid O \triangleright A_1' \parallel A_2 \xrightarrow{i?v} A_1' \parallel B_2$, giving us (ii).
  The result in (iii), namely $B_1 \parallel B_2 \equiv A_1' \parallel B_2$, follows from $B_1 \equiv A_1'$ and rule sCtxP.
- Case parL, *i.e.,* $A = A_1 \parallel A_2$ and $B = B_1 \parallel A_2$ because $K \mid O \triangleright A_1 \xrightarrow{com(i,v)} B_1$. By the IH, we obtain that

$$A_1 \equiv A_1' \parallel i\langle v \rangle \tag{H.6}$$

$$K \mid O \triangleright A_1' \xrightarrow{i?v} B_1' \tag{H.7}$$

$$B_1 \equiv B_1' \tag{H.8}$$

  The result in (i), namely $A_1 \parallel A_2 \equiv (A_1' \parallel A_2) \parallel i\langle v \rangle$, follows by eq. (H.6), rules sCtxP, sAss, sCom and transitivity.

The result in (ii), namely $K \mid O \triangleright A_1' \parallel A_2 \xrightarrow{i?v} B_1' \parallel A_2$, follows by eq. (H.7) and rule parL.
The result in (iii), namely $B_1 \parallel A_2 \equiv B_1' \parallel A_2$, follows from eq. (H.8) and rule sCtxP.

- Case scp1, $A=(\nu j)A'$ and $B=(\nu i)B'$ because $K, j \triangleright A' \xrightarrow{com(i,v)} B'$ and $j \sharp fn(com(i,v))$ where $fn(com(i,v))=\{i,v\}$. By the IH, we obtain

$$A' \equiv B'' \parallel i\langle v\rangle \tag{H.9}$$

$$K, j \mid O \triangleright A'' \xrightarrow{i?v} B'' \tag{H.10}$$

$$B' \equiv B'' \tag{H.11}$$

Applying rule sCtxS on eq. (H.9), we get $(\nu j)A' \equiv (\nu j)(A'' \parallel i\langle v\rangle)$. Since $j \sharp \{i,v\}$, we can use rule sExt to obtain $(\nu j)A' \equiv ((\nu j)A'') \parallel i\langle v\rangle$, giving us the result in (i).
The result in (ii), namely $K \mid O \triangleright (\nu j)A'' \xrightarrow{i?v} (\nu j)B''$, follows by eq. (H.10) and rule scp1.
The result in (iii), namely $(\nu j)B' \equiv (\nu j)B''$, follows by eq. (H.11) and rule sCtxS. □

We can now prove that communication actions are deterministic, Proposition 11.8 restated below.

Proposition H.15 (Communication Determinacy). For all systems $K \mid O \triangleright A$ and internal communication actions $com(i,v)$, if $K \mid O \triangleright A \xrightarrow{com(i,v)} A'$ and $K \mid O \triangleright A \xrightarrow{com(i,v)} A''$ then $A' \equiv A''$.

Proof. Suppose $K \mid O \triangleright A \xrightarrow{com(i,v)} B$ and $K \mid O \triangleright A \xrightarrow{com(i,v)} B'$. We need to show $B \equiv B'$. By Lemma 15 and $K \mid O \triangleright A \xrightarrow{com(i,v)} B$, we know there exist some actor system C such that

$$A \equiv C \parallel i\langle v\rangle \quad \text{and} \quad K \mid O \triangleright C \xrightarrow{i?v} C' \quad \text{and} \quad B \equiv C'$$

Similarly, by Lemma 15 and $K \mid O \triangleright A \xrightarrow{com(i,v)} B'$, we know there exist some actor system D such that

$$A \equiv D \parallel i\langle v\rangle \quad \text{and} \quad K \mid O \triangleright D \xrightarrow{i?v} D' \quad \text{and} \quad B' \equiv D'$$

Since $A \equiv C \parallel i\langle v\rangle$ and $A \equiv D \parallel i\langle v\rangle$, we also know that $C \parallel i\langle v\rangle \equiv D \parallel i\langle v\rangle$. By case analysis, this could have only been derived using rule sCtxP, which gives us $C \equiv D$. Thus, by rule sTrn and $K \mid O \triangleright D \xrightarrow{i?v} D'$, we know $K \mid O \triangleright C \xrightarrow{i?v} D'$ as well. Using the facts that $K \mid O \triangleright C \xrightarrow{i?v} D'$ and $K \mid O \triangleright C \xrightarrow{i?v} C'$ and Proposition 11.6, we obtain $C' \equiv D'$. Since $B \equiv C'$ and $B' \equiv D'$, using transitivity/symmetry, we can conclude $B \equiv B'$. □

Appendix IV

Confluence and Determinism for Runtime Verification

I General Results

In this section, we state a few general results.

Lemma 1. If $K \triangleright P \xrightarrow{a?b} Q$ then $a \in \mathit{fIn}(P)$.

Proof. Straightforward by rule induction on $K \triangleright P \xrightarrow{a?b} Q$. □

Lemma 2. If $K \triangleright P \xrightarrow{com(a,b)} Q$ then $a,b \in K$.

Proof. Straightforward by rule induction. □

Lemma 3. If $K \triangleright P \xrightarrow{a\uparrow b} Q$ then $\mathit{fIn}(Q) \subseteq \mathit{fIn}(P) \cup \{b\}$.

Proof. Straightforward by rule induction. □

Lemma 4. If $K \triangleright P \xrightarrow{a\uparrow b} Q$ then $\exists P' \in \textsc{Prc}$ such that $P \equiv (\nu b)P'$.

Proof. The proof proceeds by induction on $K \triangleright P \xrightarrow{a\uparrow b} Q$.

- Case pScp1, *i.e.,* $K \triangleright (\nu c)P \xrightarrow{a\uparrow b} (\nu c)Q$ because $K,c \triangleright P \xrightarrow{a\uparrow b}$ and $c \sharp a\uparrow b$. By the IH, we know $\exists P'$ such that $P \equiv (\nu b)P'$. Using rules sCtxtS and sSwp, we obtain $(\nu c)P \equiv (\nu c)(\nu b)P' \equiv (\nu b)(\nu c)P'$ as required.
- Case pOpn, *i.e.,* $K \triangleright (\nu b)P \xrightarrow{a\uparrow b} Q$ because $K,b \triangleright P \xrightarrow{a!b} Q$. Our result, $(\nu b)P \equiv (\nu b)P$, follows immediately.
- Case pPar, *i.e.,* $K \triangleright P_1 \parallel P_2 \xrightarrow{a\uparrow b} Q_1 \parallel P_2$ because $K \triangleright P_1 \xrightarrow{a\uparrow b} Q_1$. By the IH, we know $\exists P_1'$ such that $P_1 \equiv (\nu b)P_1'$. Since we are working up to α-equivalence, we can assume that b is different from every free name, *i.e.,* $b \sharp \mathit{fn}(P_2)$. Using rule sExt, we conclude $P_1 \parallel P_2 \equiv ((\nu b)P_1') \parallel P_2 \equiv (\nu b)(P_1' \parallel P_2)$.
- Case pStr, *i.e.,* $K \triangleright P \xrightarrow{a\uparrow b} Q$ because $P \equiv P''$ and $Q \equiv Q'$ and $K \triangleright P'' \xrightarrow{a\uparrow b} Q'$. By the IH, we obtain $P'' \equiv (\nu b)P'$ for some P'. Our result, $P \equiv (\nu b)P'$, follows by transitivity and symmetry.

This concludes our proof. □

Lemma 5. If $K,c \triangleright P \xrightarrow{\eta} Q$ and $c \sharp \mathit{fn}(P)$ then $K \triangleright P \xrightarrow{\eta} Q$.

Proof. By rule induction on $K,c \triangleright P \xrightarrow{\eta} Q$. □

Lemma 6. Suppose $c \sharp \mathit{fn}(P)$. If $K \triangleright P \xrightarrow{a?b} Q$ where $b \neq c$ then $c \sharp \mathit{fn}(Q)$.

Proof. Straightforward by rule induction. □

Inversion Lemmas

We also prove several results that provide additional insights into the structure and behaviour of systems from Figure 12.1.

Lemma 7. If $P \equiv P_1 \parallel P_2$ then either of the following statements must hold:

(i) $P = P_1$ and $P_2 = \mathbf{0}$, or $P = P_2$ and $P_1 = \mathbf{0}$;

(ii) $P_1 \equiv (\nu \vec{d}_1) P_1' \parallel (\nu \vec{d}_2) P_1''$ and $P_2 \equiv (\nu \vec{d}_3) P_2' \parallel (\nu \vec{d}_4) P_2''$ and $P = (\nu \vec{d}_1, \vec{d}_2, \vec{d}_3, \vec{d}_4)(Q_1 \parallel Q_2)$ and $Q_1 = P_1' \parallel P_2'$ and $Q_2 = P_1'' \parallel P_2''$.

Proof. By rule induction on $P \equiv P_1 \parallel P_2$. We omit the proof as it is quite long and tedious. □

Lemma 8 (Input Inversion). If $P \equiv P_1 \parallel P_2$ and $K \triangleright P \xrightarrow{a?b} Q$, then one of the following holds:

(i) $K \triangleright P_1 \xrightarrow{a?b} P_1'$ and $Q \equiv P_1' \parallel P_2$
(ii) $K \triangleright P_2 \xrightarrow{a?b} P_2'$ and $Q \equiv P_1 \parallel P_2'$

Proof. Suppose that $P \equiv P_1 \parallel P_2$ and $K \triangleright P \xrightarrow{a?b} Q$. The proof proceeds by rule induction on the latter.

- Case pIn, *i.e.,* $P = a?x.P'$ and $Q = P'[b/x]$. Since $P = P_1 \parallel P_2$, then by Lemma 7, we must either have $P = P_1$ and $P_2 = \mathbf{0}$, or $P = P_2$ and $P_1 = \mathbf{0}$. In the first case, the statement in (i) is satisfied since $P'[b/x] \equiv P'[b/x] \parallel \mathbf{0}$. Otherwise, the statement in (ii) is satisfied since $P'[b/x] \equiv \mathbf{0} \parallel P'[b/x]$.
- Case pPar, *i.e.,* $P = P_3 \parallel P_4$ and $Q = Q_3 \parallel P_4$ because $K \triangleright P_3 \xrightarrow{a?b} Q_3$. By Lemma 7 and $P = P_3 \parallel P_4$, we must have $P_1 \equiv P_1' \parallel P_1''$ and $P_2 \equiv P_2' \parallel P_2''$ and $P_3 = P_1' \parallel P_2'$ and $P_4 = P_1'' \parallel P_2''$. Using the facts that $P_3 = P_1' \parallel P_2'$ and $K \triangleright P_3 \xrightarrow{a?b} Q_3$ and the IH, we know that

$$\text{either } K \triangleright P_1' \xrightarrow{a?b} Q_1' \text{ and } Q_3 \equiv Q_1' \parallel P_2'$$
$$\text{or } K \triangleright P_2' \xrightarrow{a?b} Q_2' \text{ and } Q_3 \equiv P_1' \parallel Q_2'$$

  Applying rule pPar on the transitions and rule sCtxP on the equivalences above, these respectively give us that

$$\text{either } K \triangleright P_1' \parallel P_1'' \xrightarrow{a?b} Q_1' \parallel P_1'' \text{ and } Q_3 \parallel P_4 \equiv (Q_1' \parallel P_2') \parallel P_4$$
$$\text{or } K \triangleright P_2' \parallel P_2'' \xrightarrow{a?b} Q_2' \parallel P_2'' \text{ and } Q_3 \parallel P_4 \equiv (P_1' \parallel Q_2') \parallel P_4$$

  Using $P_4 = P_1'' \parallel P_2''$, $P_1 = P_1' \parallel P_1''$, $P_2 = P_2' \parallel P_2''$, $Q = Q_3 \parallel P_4$ and rules for $\equiv$, we obtain that

$$\text{either } K \triangleright P_1 \xrightarrow{a?b} Q_1' \parallel P_1'' \text{ and } Q \equiv (Q_1' \parallel P_1'') \parallel P_2$$
$$\text{or } K \triangleright P_2 \xrightarrow{a?b} Q_2' \parallel P_2'' \text{ and } Q \equiv P_1 \parallel (Q_2' \parallel P_2'')$$

  which resp. correspond to the statements in (i) and (ii).
- Case pScp1, *i.e.,* $P = (\nu c)P'$ and $Q = (\nu c)Q'$ because $K, c \triangleright P' \xrightarrow{a?c} Q'$ and $c \sharp \{a, b\}$. By Lemma 7, there are two subcases to consider:

– When $P{=}P_1$ and $P_2{=}\mathbf{0}$: since $K \triangleright P \xrightarrow{a?b} Q$ where $P{=}(\nu c)P'$ and $Q{=}(\nu c)Q'$, we have that $K \triangleright P_1 \xrightarrow{a?b} Q$. By rule sNil, we also know $Q \equiv Q \parallel \mathbf{0} = Q \parallel P_2$, satisfying the statement in (i).

– When $P_1 \equiv (\nu \vec{d}_1)P_1' \parallel (\nu \vec{d}_2)P_1''$ and $P_2 \equiv (\nu \vec{d}_3)P_2' \parallel (\nu \vec{d}_4)P_2''$ such that

$$P = (\nu \vec{d}_1, \vec{d}_2, \vec{d}_3, \vec{d}_4)(P_3 \parallel P_4)$$
$$\text{where } P_3 = P_1' \parallel P_2' \text{ and } P_4 = P_1'' \parallel P_2'' \text{ and } \vec{d}_i \sharp \vec{d}_j \text{ for all } i \neq j$$

Since $P = (\nu c)P'$, then $c \in \vec{d}_i$ for some $i \in \{1,2,3,4\}$. Suppose $i = 1$; other cases follow with similar reasoning. Letting $\vec{d}_5 = \vec{d}_1 \setminus \{c\}$, we have

$$\begin{aligned} P_1 &\equiv (\nu c, \vec{d}_5)P_1' \parallel (\nu \vec{d}_2)P_1'' \\ &\equiv (\nu c)((\nu \vec{d}_5)P_1' \parallel (\nu \vec{d}_2)P_1'') \quad \text{since we can assume } c \sharp \vec{d}_2 \\ &= (\nu c)P_5 \quad \text{where} \quad P_5 = (\nu \vec{d}_5)P_1' \parallel (\nu \vec{d}_2)P_1'' \end{aligned} \tag{I.1}$$

From the assumption $P \equiv P_1 \parallel P_2$ and (I.1), we also know that

$$\begin{aligned} P &\equiv ((\nu c)P_5) \parallel P_2 \\ &\equiv (\nu c)(P_5 \parallel P_2) \quad \text{since we can assume } c \sharp fn(P_1'') \cup fn(P_2) \end{aligned}$$

Since $P = (\nu c)P'$, by case analysis, we know $P \equiv (\nu c)(P_5 \parallel P_2)$ could have only been derived using rule sCtxS, meaning $P' \equiv P_5 \parallel P_2$. By $P' \equiv P_5 \parallel P_2$ and $K, c \triangleright P' \xrightarrow{a?b} Q'$ and the IH, we obtain that

$$\text{either } K, c \triangleright P_5 \xrightarrow{a?b} P_5' \text{ and } Q' \equiv P_5' \parallel P_2 \tag{I.2}$$
$$\text{or } K, c \triangleright P_2 \xrightarrow{a?b} P_2' \text{ and } Q' \equiv P_5 \parallel P_2' \tag{I.3}$$

If (I.2) holds, applying rules sScp1 and sCtxS gives us that

$$K \triangleright (\nu c)P_5 \xrightarrow{a?b} (\nu c)P_5' \text{ and } (\nu c)Q' \equiv (\nu c)(P_5' \parallel P_2)$$

Since $c \sharp fn(P_2)$, then $(\nu c)(P_5' \parallel P_2) \equiv (\nu c)P_5' \parallel P_2$. Using (I.1) and rule sTrn, we conclude

$$K \triangleright P_1 \xrightarrow{a?b} (\nu c)P_5' \quad \text{and} \quad (\nu c)Q' \equiv ((\nu c)P_5') \parallel P_2$$

which corresponds to the required result in (i).
If (I.3) holds, using Lemma 5 and the fact that $c \sharp fn(P_2)$, we obtain $K \triangleright P_2 \xrightarrow{a?b} P_2'$. Also, by rule sCxtS and $(\nu c)Q' \equiv ((\nu c)P_5') \parallel P_2$, we get $(\nu c)Q' \equiv (\nu c)(P_5 \parallel P_2')$. By Lemma 6 and the fact that $c \sharp \{a,b\}$, then $a \sharp fn(P_2')$, which implies $(\nu c)(P_5 \parallel P_2') \equiv (\nu c)P_5 \parallel P_2'$ as well. By (I.1), we thus have that

$$K \triangleright P_2 \xrightarrow{a?b} P_2' \quad \text{and} \quad (\nu c)Q' \equiv P_1 \parallel P_2'$$

which corresponds to the required result in (ii).

• Case sTrn, proof is straightforward.

This completes our proof. □

Lemma 9 (Output Inversion). If $P \equiv P_1 \parallel P_2$ and $K \triangleright P \xrightarrow{a!b} Q$ then one of the following statements hold:

(i) $K \triangleright P_1 \xrightarrow{a!b} P_1'$ and $Q \equiv P_1' \parallel P_2$
(ii) $K \triangleright P_2 \xrightarrow{a!b} P_2'$ and $Q \equiv P_1 \parallel P_2'$

Proof. The proof is similar to that for Lemma 8. □

Corollary I.10 (Output Inversion). If $K \triangleright P \parallel a!b.\mathbf{0} \xrightarrow{\eta} Q \parallel a!b.\mathbf{0}$ then $K \triangleright P \xrightarrow{\eta} Q$.

Proof. Follows from Lemma 9. □

Lemma 11 (Scoping Inversion). If $P \equiv (\nu c)P'$ and $K \triangleright P \xrightarrow{\eta} Q$ and $c \sharp fn(\eta)$ then $Q \equiv (\nu c)Q'$ and $K, c \triangleright P' \xrightarrow{\eta} Q'$.

Proof. Proof is straightforward by rule induction on $K \triangleright P \xrightarrow{\eta} Q$. □

Lemma 12 assumes a function $\mathrm{s} : \textsc{Prc} \mapsto \mathbb{N}$ that returns the number of scoped names in P. *E.g.* $\mathrm{s}((\nu b)a!b.\mathbf{0}) = 1$ whereas $\mathrm{s}(a!b.\mathbf{0}) = 0$.

Lemma 12. If $P \equiv Q$ then $\mathrm{s}(P) = \mathrm{s}(Q)$.

Proof. Straightforward by induction on $P \equiv Q$. □

J Determinism and Confluence Results

J.1 Asynchrony Results

In this section, we give the missing proof from Sections 13.1 and 13.1.1. We start by proving a number of general results. Concretely, Lemma 1 describes the structure of systems capable of performing an output action of the form $a!b$.

Lemma 1. If $K \triangleright P \xrightarrow{a!b} Q$ then there exists $\vec{d} \subseteq \textsc{Chans}$ and $P_1, P_2 \in \textsc{Prc}$ such that $\vec{d} \sharp \{a,b\}$ and $P \equiv (\nu \vec{d})(P_1 \parallel a!b.P_2)$ and $Q \equiv (\nu \vec{d})(P_1 \parallel P_2)$.

Proof. The proof is by rule induction on $K \triangleright P \xrightarrow{a!b} Q$.

- Case pOut, *i.e.,* $K \triangleright a!b.P' \xrightarrow{a!b} P'$. Result is immediate.
- Case pScp1, *i.e.,* $K \triangleright (\nu c)P' \xrightarrow{a!b} (\nu c)Q'$ because $K, c \triangleright P' \xrightarrow{a!b} Q'$ and $c \sharp \{a,b\}$. By the IH, we obtain that $P' \equiv (\nu \vec{d})(P_1 \parallel a!b.P_2)$ and $Q' \equiv (\nu \vec{d})(P_1 \parallel P_2)$ for some $\vec{d} \subseteq \textsc{Chans}$ and $P_1, P_2 \in \textsc{Prc}$. By rule sCtxS,

  $$(\nu c)P' \equiv (\nu c)(\nu \vec{d})(P_1 \parallel a!b.P_2) \quad \text{and} \quad (\nu c)Q' \equiv (\nu c)(\nu \vec{d})(P_1 \parallel P_2)$$

  Letting $\vec{e} = \vec{d} \cup \{c\}$, our result follows, *i.e.,* $(\nu c)P' \equiv (\nu \vec{e})(P_1 \parallel a!b.P_2)$ and $(\nu c)Q' \equiv (\nu \vec{e})(P_1 \parallel P_2)$.
- Case pPar, *i.e.,* $K \triangleright P_1 \parallel P_2 \xrightarrow{a!b} Q_1 \parallel P_2$ because $K \triangleright P_1 \xrightarrow{a!b} Q_1$. By the IH, we obtain $P_1 \equiv (\nu \vec{d})(P_1' \parallel a\langle b \rangle P_1'')$ and $Q_1 \equiv (\nu \vec{d})(P_1' \parallel P_1'')$ for some $\vec{d} \subseteq \textsc{Chans}$ and $P_1', P_1'' \in \textsc{Prc}$. Applying rule sCtxP, we get

  $$P_1 \parallel P_2 \equiv \big((\nu \vec{d})(P_1' \parallel a!b.P_1'')\big) \parallel P_2 \quad \text{and} \quad Q_1 \parallel P_2 \equiv \big((\nu \vec{d})(P_1' \parallel P_1'')\big) \parallel P_2$$

  Since we are working up to the α-renaming of bound names, we can assume $\vec{d} \sharp fn(P_2)$. Thus, using rules sCom,sExt,sAss, we obtain

  $$P_1 \parallel P_2 \equiv (\nu \vec{d})((P_1' \parallel P_2) \parallel a!b.P_1'') \quad \text{and} \quad Q_1 \parallel P_2 \equiv (\nu \vec{d})((P_1' \parallel P_2) \parallel P_1'')$$

  Our result follows by letting $P_3 = P_1' \parallel P_2$.
- Case sTrn, *i.e.,* $K \triangleright P \xrightarrow{a!b} Q$ because $P \equiv P''$, $Q \equiv Q'$ and $K \triangleright P'' \xrightarrow{a!b} Q'$. By the IH, we obtain $P'' \equiv (\nu \vec{d})(P_1 \parallel a!b.P_2)$ and $Q' \equiv (\nu \vec{d})(P_1 \parallel P_2)$. By transitivity/symmetry of $\equiv$, we conclude $P \equiv (\nu \vec{d})(P_1 \parallel a!b.P_2)$ and $Q \equiv (\nu \vec{d})(P_1 \parallel P_2)$. □

Similarly, Lemma 2 describes the structure of systems capable of scope extruding channel name b via the action $a\uparrow b$.

Lemma 2. If $K \triangleright P \xrightarrow{a\uparrow b} Q$ then there exists $\vec{d} \subseteq \textsc{Chans}$ and $P_1, P_2 \in \textsc{Prc}$ such that $\vec{d} \sharp a$, $b \in \vec{d}$ and $P \equiv (\nu \vec{d})(P_1 \parallel a!b.P_2)$ and $Q \equiv (\nu \vec{d})(P_1 \parallel P_2)$.

Proof. Similar to that for Lemma 1. □

Using these results, we are now in a position to prove the main result of this section, namely Theorem 13.5 from Section 13.1, restated below.

Theorem J.3 (Output Determinacy). A fully asynchronous system $K \triangleright P$ is deterministic w.r.t. any output action $a!b$.

Proof. Suppose $K \triangleright P$ has asynchronous outputs on channel name a and $K \triangleright P \xrightarrow{a!b} Q$ and $K \triangleright P \xrightarrow{a!b} Q'$. Let $K' = \mathbf{aft}(K, a!b)$. We have to show $K' \models Q \approx Q'$.
By Lemma 1, we know there exist channel lists $\vec{d},\vec{e} \subseteq \text{CHANS}$ and processes $P_1,P_2,P_3,P_4 \in \text{PRC}$ such that

$$P \equiv (\nu \vec{d})(P_1 \parallel a!b.P_2) \qquad \text{and} \qquad Q \equiv (\nu \vec{d})(P_1 \parallel P_2) \tag{J.1}$$

$$P \equiv (\nu \vec{e})(P_3 \parallel a!b.P_4) \qquad \text{and} \qquad Q' \equiv (\nu \vec{e})(P_3 \parallel P_4) \tag{J.2}$$

where $\vec{d} \sharp \{a,b\}$ and $\vec{e} \sharp \{a,b\}$. By transitivity/symmetry of $\equiv$, we obtain that

$$(\nu \vec{d})(P_1 \parallel a!b.P_2) \equiv (\nu \vec{e})(P_3 \parallel a!b.P_4)$$

Let $\vec{d'} = \vec{e} \setminus \vec{d}$ and $\vec{e'} = \vec{d} \setminus \vec{e}$. By Lemma 12, we must have that $P_1 \equiv (\nu \vec{d'})P_1'$ and $P_2 \equiv (\nu \vec{e'})P_2'$, which implies

$$(\nu \vec{d})(((\nu \vec{d'})P_1') \parallel a!b.P_2) \equiv (\nu \vec{e})(((\nu \vec{e'})P_3') \parallel a!b.P_4)$$

Since we are working up to the α-renaming of bound names, we can assume that $\vec{d'} \sharp fn(a!b.P_2)$ and $\vec{e'} \sharp fn(a!b.P_4)$. By rules sEXT,sCTXS,sCOM,sCTXP, and transitivity of $\equiv$, we obtain that

$$(\nu \vec{f})(P_1' \parallel a!b.P_2) \equiv (\nu \vec{f})\big(P_3' \parallel a!b.P_4\big) \tag{J.3}$$

where $\vec{f} = \vec{d} \cup \vec{d'} = \vec{e} \cup \vec{e'}$. Only rule sCTXS could have been used, which means

$$P_1' \parallel a!b.P_2 \equiv P_3' \parallel a!b.P_4 \tag{J.4}$$

Using similar reasoning, we also know that

$$Q \equiv (\nu \vec{f})(P_1' \parallel P_2) \qquad \text{and} \qquad Q' \equiv (\nu \vec{f})(P_3' \parallel P_4) \tag{J.5}$$

The statement in eq. (J.4) gives us a number of sub-cases, but their corresponding proofs are long and tedious. We focus on the main ones:

- When $a!b.P_2 = a!b.P_4$ and $P_1' \equiv P_3'$, by the statement in eq. (J.5) and the structural equivalence rules in Definition 12.5, we obtain that $Q \equiv Q'$. Our result, $\mathbf{aft}(K, a!b) \models Q \approx Q'$, follows by Theorem 12.11.
- When $P_1' \equiv R \parallel R'$ and $P_3' \equiv R \parallel a!b.P_2$ and $a!b.P_4 \equiv R'$, we know that

  $$P_1' \parallel P_2 \equiv (R \parallel R') \parallel P_2 \equiv (R \parallel a!b.P_4) \parallel P_2 \quad \text{and} \quad P_3' \parallel P_4 \equiv (R \parallel a!b.P_2) \parallel P_4$$

  Using rule sCTXS, we obtain that

  $$(\nu \vec{f})(P_1' \parallel P_2) \equiv (\nu \vec{f})\big((R \parallel a!b.P_4) \parallel P_2\big) \quad \text{and} \quad (\nu \vec{f})(P_3' \parallel P_4) \equiv (\nu \vec{f})\big((R \parallel a!b.P_2) \parallel P_4\big)$$

By the structural equivalence rules pScp1,pParL,pParR,pOut, we also know that

$$K' \triangleright (\nu \vec{f})\big((R \parallel a!b.P_4) \parallel P_2\big) \xrightarrow{a!b} (\nu \vec{f})\big((R \parallel P_4) \parallel P_2\big) \quad \text{and}$$
$$K' \triangleright (\nu \vec{f})\big((R \parallel a!b.P_2) \parallel P_4\big) \xrightarrow{a!b} (\nu \vec{f})\big((R \parallel P_2) \parallel P_4\big)$$

By the assumption that $K \triangleright P$ persistently outputs asynchronously and Definition 13.4, we obtain that

$$K' \models (\nu \vec{f})\big((R \parallel a!b.P_4) \parallel P_2\big) \approx (\nu \vec{f})\big((R \parallel P_4) \parallel P_2\big) \parallel a!b. \quad \text{and}$$
$$K' \models (\nu \vec{f})\big((R \parallel a!b.P_2) \parallel P_4\big) \approx (\nu \vec{f})\big((R \parallel P_2) \parallel P_4\big) \parallel a!b.$$

By the transitivity and symmetry of $\approx$, we obtain

$$K' \models (\nu \vec{f})\big((R \parallel a!b.P_4) \parallel P_2\big) \approx (\nu \vec{f})\big((R \parallel a!b.P_2) \parallel P_4\big)$$

Since $Q \equiv (\nu \vec{f})\big((R \parallel a!b.P_4) \parallel P_2\big)$ and $Q' \equiv (\nu \vec{f})\big((R \parallel a!b.P_2) \parallel P_4\big)$, our result $K' \models Q \approx Q'$ follows using Theorem 12.11 and transitivity/symmetry. □

J.2 Single Receiver Results

In this section, we give the missing proofs in Sections 13.2 and 13.2.1. Before embarking on this endeavour, we establish a few key technical results. Concretely, since from here onward we assume that processes do *not* contain any output prefixing, *i.e.,* $asy(P)$, Lemma 4 provides further insight into the structure of systems capable of performing output actions.

Lemma 4. If $asy(P)$ and $K \triangleright P \xrightarrow{a!b} Q$ then there exists $P' \in \text{Prc}$ such that $P \equiv P' \parallel a!b.\mathbf{0}$ and $Q \equiv P'$.

Proof. Suppose $K \triangleright P \xrightarrow{a!b} Q$. By Lemma 1, we know there exist $\vec{d} \subseteq \text{Chans}$ and $P_1, P_2 \in \text{Prc}$ such that $\vec{d} \sharp \{a,b\}$ and $P \equiv (\nu \vec{d})(P_1 \parallel a!b.P_2)$ and $Q \equiv (\nu \vec{d})(P_1 \parallel P_2)$. But by the assumption that $asy(P)$ and Proposition 13.10, we also know $asy((\nu \vec{d})(P_1 \parallel a!b.P_2))$, which implies $P_2 = \mathbf{0}$. Using the fact that $\vec{d} \sharp \{a,b\}$ and rules sCtxS,sNil,sExt,sCom, we obtain $P \equiv \big((\nu \vec{d})P_1\big) \parallel a!b.\mathbf{0}$ and $Q \equiv (\nu \vec{d})P_1$. Our result follows by letting $P' = (\nu \vec{d})P_1$. □

Corollary J.5. If $asy(P)$ and $K \triangleright P \xRightarrow{a!b} Q$ then there exists $P' \in \text{Prc}$ such that $P \equiv P' \parallel a!b.\mathbf{0}$ and $Q \equiv P'$. ■

Lemma 6. If $K \triangleright P$ has single receivers on a and $P \equiv Q$ then $K \triangleright Q$ has single receivers on a.

Proof. Suppose $K \triangleright P$ has single receivers on a and $P \equiv Q$. Suppose also $Q \equiv (\nu \vec{d})(Q_1 \parallel Q_2)$ where $\vec{d} \sharp a$ and $K \triangleright Q_1 \xrightarrow{a?\_} Q_1'$. We must show that $K \triangleright Q_2 \xrightarrow{a?} \not\rightarrow$. By transitivity, we know $P \equiv (\nu \vec{d})(Q_1 \parallel Q_2)$, and by the assumption that $K \triangleright P$ has single receivers on a, we immediately obtain that $K \triangleright Q_2 \xrightarrow{a?} \not\rightarrow$ as required. □

Lemma 7. If $K \triangleright (\nu b)P$ has single receivers on channel a then $K \triangleright P$ has single receivers on channel a.

Proof. Assume $K \triangleright (\nu b)P$ has single receivers on a. Assume also that $P \equiv (\nu \vec{d})(P_1 \parallel P_2)$ and $K \triangleright P_1 \xrightarrow{a?\_} Q_1$. We need to show $K \triangleright P_2 \xrightarrow{a?} \not\rightarrow$. Applying rule sCtxS from Definition 12.5 on $P \equiv (\nu \vec{d})(P_1 \parallel P_2)$, we obtain

that $(\nu b)P \equiv (\nu b)(\nu \vec{d})(P_1 \parallel P_2)$, *i.e.,* $(\nu b)P \equiv (\nu \vec{e})(P_1 \parallel P_2)$ where $\vec{e} = b,\vec{d}$. By the assumption that $K \triangleright (\nu b)P$ has single receivers on a and $K \triangleright P_1 \xrightarrow{a?\_} Q_1$, we immediately conclude $K \triangleright P_2 \xrightarrow{a?}\!\!\!\!\!\!/\;\;$ as required. □

Lemma 8. If $K \triangleright P \parallel Q$ has single receivers on a then $K \triangleright P$ and $K \triangleright Q$ have single receivers on a.

Proof. Assume $K \triangleright P \parallel Q$ has single receivers on a. Since $P \parallel Q \equiv P$, it suffices to show that $K \triangleright P$ has single receivers on a. Assume that $P \equiv (\nu \vec{d})(P_1 \parallel P_2)$ and $K \triangleright P_1 \xrightarrow{a?\_} Q_1$. We show $K \triangleright P_2 \xrightarrow{a?}\!\!\!\!\!\!/\;\;$.
By $P \equiv (\nu \vec{d})(P_1 \parallel P_2)$ and rule sCtxP, we know $P \parallel Q \equiv ((\nu \vec{d})(P_1 \parallel P_2)) \parallel Q$. Since we are working up to α-equivalence, we can assume $\vec{d} \sharp fn(Q)$. Repeatedly applying rule sExt, followed by rule sAss, we get

$$((\nu \vec{d})(P_1 \parallel P_2)) \parallel Q \;\equiv\; (\nu \vec{d})((P_1 \parallel P_2) \parallel Q) \;\equiv\; (\nu \vec{d})(P_1 \parallel (P_2 \parallel Q))$$

By transitivity, this implies $P \parallel Q \equiv (\nu \vec{d})(P_1 \parallel (P_2 \parallel Q))$. By the assumption that $K \triangleright P \parallel Q$ has single receivers on a and $K \triangleright P_1 \xrightarrow{a?\_}$, we deduce $K \triangleright P_2 \parallel Q \xrightarrow{a?}\!\!\!\!\!\!/\;\;$. We can thus conclude $K \triangleright P_2 \xrightarrow{a?}\!\!\!\!\!\!/\;\;$, as required. Otherwise, if $K \triangleright P_2 \xrightarrow{a?\_} P_2'$ for some P_2', then by rule pParL, we obtain $K \triangleright P_2 \parallel Q \xrightarrow{a?\_} P_2' \parallel Q$, a contradiction. □

Proving Input Determinacy

We are now in a position to prove Proposition 13.19 from Section 13.2. Its proof relies on Lemma 9 below.

Lemma 9 (Input Determinacy). Suppose system $K \triangleright P$ has single receivers on channel name a. If $P \equiv P'$ and $K \triangleright P \xrightarrow{a?b} Q$ and $K \triangleright P' \xrightarrow{a?b} Q'$ then $Q \equiv Q'$.

Proof. Suppose $K \triangleright P$ has single receivers on channel a. Suppose also that $K \triangleright P \xrightarrow{a?b} Q$ and $K \triangleright P' \xrightarrow{a?b} Q'$ where $P \equiv P'$. The proof proceeds by induction on the first move.

- Case pIn, *i.e.,* $P = a?x.P''$ and $Q = P''[b/x]$. Since $P \equiv P'$ where $P = a?x.P''$, we must have that $P' = P$. Thus, by Corollary H.8 and $K \triangleright P' \xrightarrow{a?b} Q'$, we obtain that $Q' \equiv P''[b/x]$.
- Case pScp1, *i.e.,* $P = (\nu c)P''$ and $Q = (\nu c)Q''$ because $K,c \triangleright P'' \xrightarrow{a?b} Q'$ and $c \sharp \{a,b\}$. By the assumptions that $P \equiv P'$ and $K \triangleright P' \xrightarrow{a?b} Q'$ and Lemma 11, we obtain that $K,c \triangleright P'' \xrightarrow{a?b} Q'''$ and $Q' \equiv (\nu c)Q'''$ for some Q'''. By the assumption that $K \triangleright P = K \triangleright (\nu c)P'$ has single receivers on a and Lemma 7, we also know $K \triangleright P'$ has single receivers on a, which implies $K,c \triangleright P'$ has single receivers on a. We can thus use the IH and obtain that $Q \equiv Q''$. Our result, $(\nu c)Q \equiv (\nu c)Q''$, follows by rule sCtxS.
- Case pPar, *i.e.,* $P = P_1 \parallel P_2$ and $Q = Q_1 \parallel P_2$ because $K \triangleright P_1 \xrightarrow{a?b} Q_1$. From the assumption that $K \triangleright P_1 \parallel P_2$ has single receivers on a, by Definition 13.14, we also know $K \triangleright P_2 \xrightarrow{a?b}\!\!\!\!\!\!/\;\;$. By Lemma 8 and the second move $K \triangleright P' \xrightarrow{a?b} Q'$ where $P \equiv P'$ and $P = P_1 \parallel P_2$, we obtain that

  $$\text{either } K \triangleright P_1 \xrightarrow{a?b} Q_1' \text{ and } Q' \equiv Q_1' \parallel P_2 \tag{J.6}$$

  $$\text{or } K \triangleright P_2 \xrightarrow{a?b} Q_2 \text{ and } Q' \equiv P_1 \parallel Q_2 \tag{J.7}$$

  However, the move in (J.7) contradicts the fact that $K \triangleright P_2 \xrightarrow{a?b}\!\!\!\!\!\!/\;\;$, which implies that (J.6) must hold. Thus by the IH, we obtain $Q_1 \equiv Q_1'$. Our result, $Q \equiv Q'$, follows by rule sCtxP.
- Case pStr, *i.e.,* $K \triangleright P \xrightarrow{a?b} Q$ because $P \equiv P''$, $Q \equiv Q''$ and $K \triangleright P'' \xrightarrow{a?b} Q''$. From the assumption that $P \equiv P'$ and transitivity/symmetry, we also know $P'' \equiv P'$. By this, moves $K \triangleright P'' \xrightarrow{a?b} Q''$ and $K \triangleright P' \xrightarrow{a?b} Q'$, and the IH, we obtain $Q'' \equiv Q'$. Our result, $Q \equiv Q'$, follows from the fact that $Q \equiv Q''$ and transitivity.

□

Theorem J.10 (Input Determinacy). If a system $K \triangleright P$ has single receivers on channel name a, then $K \triangleright P$ is deterministic w.r.t. all input actions $a?b$.

Proof. Follows by Lemma 9 since $P \equiv P$. □

Proving Communication Determinacy

The proof for Theorem 13.22 relies on Lemma 11, which describes the structure of a system capable of performing some internal communication, described by the action $com(a,b)$.

Lemma 11. If $K \triangleright P \xrightarrow{com(a,b)} Q$ then

(i) $P \equiv P' \parallel a!b.\mathbf{0}$;
(ii) $K \triangleright P' \xrightarrow{a?b} Q'$ for some Q';
(iii) $Q \equiv Q'$.

Proof. Assume $K \triangleright P \xrightarrow{com(a,b)} Q$. We proceed by rule induction, outlining only the main cases.

- Case pComm, *i.e.,* $P = P_1 \parallel P_2$ and $Q = Q_1 \parallel Q_2$ because $K \triangleright P_1 \xrightarrow{a!b} Q_1$ and $K \triangleright P_2 \xrightarrow{a?b} Q_2$.
  By Lemma 4, we know $P_1 \equiv P_1' \parallel a!b.\mathbf{0}$ and $Q_1 \equiv P_1'$. By rules sCtxP,sAss,sCom and transitivity, this implies $P_1 \parallel P_2 \equiv (P_1' \parallel P_2) \parallel a!b.\mathbf{0}$, giving us (i).
  Applying rule pPar on $K \triangleright P_2 \xrightarrow{a?b} Q_2$ gives us that $K \triangleright P_2 \parallel P_1' \xrightarrow{a?b} Q_2 \parallel P_1'$. Since $P_1' \parallel P_2 \equiv P_2 \parallel P_1'$ and $P_1' \parallel Q_2 \equiv Q_2 \parallel P_1'$ by rule sCom, the required result in (ii) follows by rule pStr.
  The result in (iii), namely $Q_1 \parallel Q_2 \equiv P_1' \parallel Q_2$, follows from $Q_1 \equiv P_1'$ and rule sCtxP.
- Case pPar, *i.e.,* $P = P_1 \parallel P_2$ and $Q = Q_1 \parallel P_2$ because $K \triangleright P_1 \xrightarrow{com(a,b)} Q_1$. By the IH, we obtain that

$$P_1 \equiv P_1' \parallel a!b.\mathbf{0} \tag{J.8}$$
$$K \triangleright P_1' \xrightarrow{a?b} Q_1' \tag{J.9}$$
$$Q_1 \equiv Q_1' \tag{J.10}$$

  The result in (i), namely $P_1 \parallel P_2 \equiv (P_1' \parallel P_2) \parallel a!b.\mathbf{0}$, follows by (J.8), the structural equivalence rules sCtxP, sAss, sCom and transitivity.
  The result in (ii), namely $K \triangleright P_1' \parallel P_2 \xrightarrow{a?b} Q_1' \parallel P_2$, follows by (J.9) and rule pPar.
  The result in (iii), namely $Q_1 \parallel P_2 \equiv Q_1' \parallel P_2$, follows from (J.10) and rule sCtxP.
- Case pScp1, $P = (\nu c)P'$ and $Q = (\nu c)Q'$ because $K, c \triangleright P' \xrightarrow{com(a,b)} Q'$ and $c \sharp fn(com(a,b))$ where $fn(com(a,b)) = \{a,b\}$. By the IH, we obtain

$$P' \equiv P'' \parallel a!b.\mathbf{0} \tag{J.11}$$
$$K, c \triangleright P'' \xrightarrow{a?b} Q'' \tag{J.12}$$
$$Q' \equiv Q'' \tag{J.13}$$

Applying rule sCtxS on (J.11), we get $(\nu c)P' \equiv (\nu c)(P'' \parallel a!b.\mathbf{0})$. Since $c\sharp\{a,b\}$, we can use rule sExt to obtain $(\nu c)P' \equiv ((\nu c)P'') \parallel a!b.\mathbf{0}$, giving us the result in (i).
The result in (ii), namely $K \triangleright (\nu c)P'' \xrightarrow{a?b} (\nu c)Q''$, follows by (J.12) and rule pScp1.
The result in (iii), namely $(\nu c)Q' \equiv (\nu c)Q''$, follows by (J.13) and rule sCtxS.

- Case pStr, the proof is straightforward.

This completes our proof. □

We are now in a position to prove that communication actions lead to structurally equivalent actor systems, as stated in Theorem 13.22.

Theorem J.12 (Internal Communication Determinacy). If $K \triangleright P$ has single receivers on channel name a, then $K \triangleright P$ is deterministic w.r.t. all internal communication actions $com(a,b)$.

Proof. Suppose $K \triangleright P \xrightarrow{com(a,b)} Q$ and $K \triangleright P \xrightarrow{com(a,b)} Q'$. We need to show $Q \equiv Q'$. By Lemma 11 and $K \triangleright P \xrightarrow{com(a,b)} Q$, we know that there exist some process R such that

$$P \equiv R \parallel a!b.\mathbf{0} \quad \text{and} \quad K \triangleright R \xrightarrow{a?b} R' \quad \text{and} \quad Q \equiv R'$$

Similarly, by Lemma 11 and $K \triangleright P \xrightarrow{com(a,b)} Q'$, we know that there exist some process S such that

$$P \equiv S \parallel a!b.\mathbf{0} \quad \text{and} \quad K \triangleright S \xrightarrow{a?b} S' \quad \text{and} \quad Q' \equiv S'$$

From $P \equiv R \parallel a!b.\mathbf{0}$ and $P \equiv S \parallel a!b.\mathbf{0}$, we know $R \parallel a!b.\mathbf{0} \equiv S \parallel a!b.\mathbf{0}$. From here, we can show that $R \equiv S$. Thus, by rule pStr and $K \triangleright S \xrightarrow{a?b} S'$, we know $K \triangleright R \xrightarrow{a?b} S'$ as well. Since $K \triangleright P$ has single receivers on a and $P \equiv S \parallel a!b.\mathbf{0}$, by Lemmas 6 and 8, we know $K \triangleright S$ also has single receivers on a. Thus, by $K \triangleright R \xrightarrow{a?b} S'$ and $K \triangleright R \xrightarrow{a?b} R'$ and Theorem 13.20, we obtain $R' \equiv S'$. Since $Q \equiv R'$ and $Q' \equiv S'$, we can conclude $Q \equiv Q'$, as required. □

Syntactic Conditions

In this section, we give the proofs for Propositions 13.28, 13.29 and 13.35. We start with Proposition 13.28 whose proof relies on Lemma 13 below.

Lemma 13. If $X \notin fV(R)$ and $Y \notin fV(Q)$ then $P[^{Q}/_{X}][^{R}/_{Y}]$.

Proof. Straightforward by induction on the structure of P. □

Proposition J.13. If $dr(P[^{\mathrm{rec}X.P}/_{X}], a, V \uplus \{X\})$ then $dr(P[^{\mathrm{rec}X.P}/_{X}], a, V)$.

Proof. Assume $dr(P[^{\mathrm{rec}X.P}/_{X}], a, V \cup \{X\})$. The proof is by rule induction on $dr(P[^{\mathrm{rec}X.P}/_{X}], a, V \cup \{X\})$. We only give the proof for the main cases:

- Case sRec1, *i.e.,* $dr(\mathrm{rec}\,Y.Q, a, V \cup \{X\})$ because $Y \in V \cup \{X\}$ and $dr(Q, a, V \cup \{X\})$. Since we know $\mathrm{rec}\,Y.Q = P[^{\mathrm{rec}X.P}/_{X}]$, we also know that $P = \mathrm{rec}\,Y.P'$ for some P' and $Q = P'[^{\mathrm{rec}X.P}/_{X}]$. Thus, we deduce $dr(P'[^{\mathrm{rec}X.P}/_{X}], a, V \cup \{X\})$. By the IH, we obtain $dr(P'[^{\mathrm{rec}X.P}/_{X}], a, V)$. By the assumption that all binders are different, we have that $X \neq Y$, which implies $X \in V$. We can thus apply rule sRec1 and obtain $dr(\mathrm{rec}\,Y.P'[^{\mathrm{rec}X.P}/_{X}], a, V)$, *i.e.,* $dr(\mathrm{rec}\,Y.Q, a, V)$ as required.

- Case sRec2, *i.e.,* $dr(\mathrm{rec}\,Y.Q, a, V \cup \{X\})$ because $Y \notin V \cup \{X\}$ and $dr(Q[^{\mathrm{rec}\,Y.Q}/_Y], a, V \cup \{X, Y\})$. Thus $X \neq Y$. Also, since $\mathrm{rec}\,Y.Q = P[^{\mathrm{rec}\,X.P}/_X]$, we know $P = \mathrm{rec}\,Y.P'$ for some P' and $Q = P'[^{\mathrm{rec}\,X.P}/_X]$, where $X \notin fV(\mathrm{rec}\,Y.Q)$ and $Y \notin fV(\mathrm{rec}\,X.P)$. By Lemma 13, this implies

$$Q[^{\mathrm{rec}\,Y.Q}/_Y] = P'[^{\mathrm{rec}\,X.P}/_X][^{\mathrm{rec}\,Y.Q}/_Y] = P'[^{\mathrm{rec}\,Y.Q}/_Y][^{\mathrm{rec}\,X.P}/_X]$$

  Thus, we have $dr(P'[^{\mathrm{rec}\,Y.Q}/_Y][^{\mathrm{rec}\,X.P}/_X], a, V \cup \{X, Y\})$. By the IH, we deduce

$$dr(P'[^{\mathrm{rec}\,Y.Q}/_Y][^{\mathrm{rec}\,X.P}/_X], a, V \cup \{Y\})$$

  which implies $dr(P'[^{\mathrm{rec}\,X.P}/_X][^{\mathrm{rec}\,Y.Q}/_Y], a, V \cup \{Y\})$ Applying rule sRec2, we can conclude that $dr(\mathrm{rec}\,Y.P'[^{\mathrm{rec}\,X.P}/_X], a, V)$, *i.e.,* $dr(\mathrm{rec}\,Y.Q, a, V)$ as required.

The proof for the remaining cases is straightforward. □

The proof for Proposition 13.29 relies on Lemma 14 below, stating that if a process has disjoint receivers on a given channel name, then this property holds for all structurally equivalent processes.

Lemma 14. If $P \equiv Q$ and $dr(P, a)$ then $dr(Q, a)$.

Proof. Straightforward by rule induction on $P \equiv Q$. □

Proposition J.14. For any valid interface $K \subseteq \textsc{Chans}$, if $dr(P, a)$ then $K \triangleright P$ has single receivers on a.

Proof. Suppose $dr(P, a)$ and $P \equiv (\nu \vec{d})(Q_1 \parallel Q_2)$ and $\vec{d} \sharp a$ and $K \triangleright Q_1 \xrightarrow{a?} Q_1'$. By Lemma 1, we deduce $a \in fIn(Q_1)$. By Lemma 14, we also know $dr((\nu \vec{d})(Q_1 \parallel Q_2), a)$. If $\vec{d} = \emptyset$ then $(\nu \vec{d})(Q_1 \parallel Q_2) = Q_1 \parallel Q_2$ which implies $dr(Q_1 \parallel Q_2, a)$. Otherwise, if $\vec{d} \neq \emptyset$ then by case analysis, we know $dr((\nu \vec{d})(Q_1 \parallel Q_2), a)$ could have only been derived using rule sScp a number of times, giving us $dr(Q_1 \parallel Q_2, a)$.

In both cases, we have that $dr(Q_1 \parallel Q_2, a)$. By case analysis, this could have only been derived using rule sPar, giving us $a \notin fIn(Q_1) \cap fIn(Q_2)$. Since we already know $a \in fIn(Q_1)$, then we must have $a \notin fIn(Q_2)$. By the contrapositive of Lemma 1, we can thus conclude that $K \triangleright Q_2 \xrightarrow{a?} \!\!\!\!/$. □

The proof for Proposition 13.35 relies on a number of additional results, namely Lemmas 15 to 19.

Lemma 15. If $loc(P)$ and $K \triangleright P \xrightarrow{a?b} Q$ then $fIn(Q) \subseteq fIn(P)$.

Proof. Straightforward by rule induction on $K \triangleright P \xrightarrow{a?b} Q$. □

Lemma 16. If $loc(P, V)$ then $loc(P[^b/_x], V)$.

Proof. Straightforward by induction on the structure of P. We outline the main case:

- Case $P = u?v.P'$. Since $loc(P, V)$ then $loc(P', V \cup \{v\})$ and $u \notin V$. By the assumption that all processes are closed, then u must be a channel, *i.e.,* $u \in \textsc{Chans}$. As a result, we know $P[^b/_x] = u?v.(P'[^b/_x])$. By the IH, we obtain $loc(P'[^b/_x], V \cup \{v\})$. By definition of $loc(-)$, this implies $loc(u?v.P'[^b/_x], V)$, *i.e.,* $loc(P[^b/_x], V)$ as required.

The proof for the other cases follows with similar, but more straightforward, reasoning. □

Lemma 17. If $loc(P, V \cup V')$ then $loc(P, V)$.

Proof. Straightforward by induction on the structure of P. □

Lemma 18. If $dr(P, a)$ and $loc(P, V)$ then $dr(P[^b/_x], a)$.

Proof. Straightforward by structural induction on P. □

We prove Proposition 13.35 in two parts: we first show that the locality constraint is preserved by all system transitions, Lemma 19, and then we show the disjoint receivers constraint is preserved by all system transitions, given that the initial process is local, Lemma 20.

Lemma 19 (Locality Preservation). If $loc(P)$ and $K \triangleright P \xrightarrow{\eta} Q$ then $loc(Q)$.

Proof. Straightforward by rule induction on $K \triangleright P \xrightarrow{\eta} Q$. We outline the main case:

- Case pIn, *i.e.,* $K \triangleright a?x.P' \xrightarrow{a?b} P'[^b/_x]$. Since $loc(a?x.P', V)$, we also know $loc(P', V \cup \{x\})$ and $a \notin V$. By Lemma 16, we obtain $loc(P'[^b/_x], V \cup \{x\})$. Our result, $loc(P'[^b/_x], V)$, follows by Lemma 17.

The proof for the remaining cases follow with similar but more straightforward reasoning. □

Lemma 20 (Disjoint Receivers Preservation). Suppose $loc(P)$ and $dr(P, a)$. If $K \triangleright P \xrightarrow{\eta} Q$ then $dr(Q, a)$.

Proof. By rule induction on $K \triangleright P \xrightarrow{\eta} Q$. Recall that $dr(P, a)$ denotes $dr(P, a, \emptyset)$; in what follows, we use the two interchangeably. Assume that $loc(P)$. By Lemma 19, we also know $loc(Q)$. We outline the main cases of our proof:

- Case pIn, *i.e.,* $K \triangleright b?x.P \xrightarrow{\eta} P[^b/_x]$ where $Q = P[^b/_x]$. Since $dr(b?x.P, a)$ then $dr(P, a)$. Our result, $dr(P[^b/_x], a)$, follows by Lemma 18 and the fact that $loc(Q)$.
- Case pRec, *i.e.,* $K \triangleright \mathrm{rec}X.P \xrightarrow{\tau} P[^{\mathrm{rec}X.P}/_X]$. Assume $dr(\mathrm{rec}X.P, a)$, *i.e.,* $dr(\mathrm{rec}X.P, a, \emptyset)$. Since $X \notin \emptyset$, then this could have only been derived using rule sRec2. Thus we know $dr(P[^{\mathrm{rec}X.P}/_X], a, \{X\})$. Our result, $dr(P[^{\mathrm{rec}X.P}/_X], a, \emptyset)$, follows by Proposition 13.28.
- Case pScp1, *i.e.,* $K \triangleright (\nu b)P \xrightarrow{\eta} (\nu b)Q$ where $K, b \triangleright P \xrightarrow{\eta} Q$ and $b \sharp \eta$. Since $dr((\nu b)P, a)$ and $loc((\nu b)P)$, then $dr(P, a)$ and $a \neq b$ and $loc(P)$. By the IH, $dr(P, a)$. Our result, $dr((\nu b)P)$, follows by rule sScp.
- Case pNComm, *i.e.,* $K \triangleright P \parallel Q \xrightarrow{ncom} (\nu c)(P' \parallel Q')$ because $K \triangleright P \xrightarrow{b\uparrow c} P'$ and $K \triangleright Q \xrightarrow{b?c} Q'$. We show $dr((\nu c)(P' \parallel Q'), a)$.

  From $loc(P \parallel Q)$, we know $loc(P)$ and $loc(Q)$. Also, since $dr(P \parallel Q, a)$ could have only been derived using rule sPar, we know $a \notin fIn(P) \cap fIn(Q)$ and $dr(P, a)$ and $dr(Q, a)$. By the IH, we obtain $dr(P', a)$ and $dr(Q', a)$.

  From $K \triangleright P \xrightarrow{b\uparrow c} P'$ and Lemma 4, we know $P \equiv (\nu c)P''$ for some P'', which by Lemma 14 and $dr(P, a)$, implies $dr((\nu c)P'', a)$. This could have only been derived using rule sScp, giving us $a \neq c$.

  From $K \triangleright P \xrightarrow{b\uparrow c} P'$ and Lemma 15, we know $fIn(P') \subseteq fIn(P) \cup \{c\}$. Similarly, from $K \triangleright Q \xrightarrow{b?c} Q'$

and Lemma 3, we know $fIn(Q') \subseteq fIn(Q)$. Thus, since $a \neq c$ and $a \notin fIn(P) \cap fIn(Q)$, we deduce that $a \notin (fIn(P) \cup \{c\}) \cap fIn(Q) \supseteq fIn(P') \cap fIn(Q')$, which implies $a \notin fIn(P') \cap fIn(Q')$. Applying rule sPar, we obtain $dr(P' \parallel Q', a)$. Since $a \neq c$, we can then apply rule sScp to conclude $dr((\nu c)(P' \parallel Q'), a)$.

We omit the proofs for the remaining cases as they follow with similar reasoning. □

Proposition J.20 (Preservation). Suppose $loc(P)$ and $dr(P,a)$. If $K \triangleright P \xrightarrow{\eta} Q$ then $loc(Q)$ and $dr(Q,a)$.

Proof. Follows from Lemmas 19 and 20. □

J.3 Receptiveness Results

In this section, we give the missing proofs in Sections 13.3 and 13.3.1. Before embarking on this endeavour, we establish a few key technical results. We start by defining the function $\mathbf{ext}(-)$ which, given an action, returns the set of scope extruded names.

Definition J.21 (Scope Extruded Names). The function $\mathrm{ext} : \textsc{Act} \mapsto \textsc{Chans}$ is defined as follows:

$$\mathbf{ext}(K,\eta) \stackrel{\text{def}}{=} \begin{cases} \{b\} & \text{if } \eta = a\uparrow b \\ \emptyset & \text{otherwise} \end{cases}$$ ■

Lemma 22 below states that whenever $K \triangleright P$ does not scope extrude channel name c, adding that channel name to K does not affecting the transitions that the system can perform.

Lemma 22. If $K \triangleright P \xrightarrow{\eta} Q$ where $c \sharp \mathbf{ext}(\eta)$ then $K \cup \{c\} \triangleright P \xrightarrow{\eta} Q$.

Proof. The proof proceeds by rule induction on $K \triangleright P \xrightarrow{\eta} Q$. We only outline the cases for rules pScp1 and pNComm:

- Case pNComm, *i.e.*, $K \triangleright P_1 \parallel P_2 \xrightarrow{ncom} (\nu b)(Q_1 \parallel Q_2)$ because $K \triangleright P_1 \xrightarrow{a\uparrow b} Q_1$ and $K \triangleright P_2 \xrightarrow{a?b} Q_2$. By $K \triangleright P_1 \xrightarrow{a\uparrow b} Q_1$ and Lemma 4, we know $P_1 \equiv (\nu b)P_1'$ for some P_1'. But since we are working up to α-equivalence, we can assume $b \neq c$, which implies $c \sharp a!b$. Thus, by $K \triangleright P_1 \xrightarrow{a\uparrow b} Q_1$ and the IH, we obtain $K \cup \{c\} \triangleright P_1 \xrightarrow{a\uparrow b} Q_1$. Similarly, by $K \triangleright P_2 \xrightarrow{a?b} Q_2$ and $c \sharp a?b$ (from definition) and the IH, we get $K \cup \{c\} \triangleright P_2 \xrightarrow{a?b} Q_2$. Our result, $K \cup \{c\} \triangleright P_1 \parallel P_2 \xrightarrow{ncom} (\nu b)(Q_1 \parallel Q_2)$, follows by rule pNComm.
- Case pScp1, *i.e.*, $K \triangleright (\nu b)P' \xrightarrow{\eta} (\nu b)Q'$ because $K, b \triangleright P' \xrightarrow{\eta} Q'$ and $b \sharp \eta$. By the assumption that $c \sharp \eta$ and the IH, we obtain $(K,b) \cup \{c\} \triangleright P' \xrightarrow{\eta} Q'$. Since we are working up to α-equivalence, we can also assume $b \neq c$, which implies $(K,b) \cup \{c\} = (K \cup \{c\}), b$. Applying rule pScp1 on $(K \cup \{c\}), b \triangleright P' \xrightarrow{\eta} Q'$, we conclude $K \cup \{c\} \triangleright (\nu b)P' \xrightarrow{\eta} (\nu b)Q'$ as required.

The proof for the remaining cases follows with similar reasoning. □

We also give a number of bisimilarity results. We start with Lemma 23, stating τ-transitions preserve bisimulation equivalence.

Lemma 23 (τ-Inertness). If $K \triangleright P \Rightarrow Q$ then $K \models P \approx Q$.

Proof. The proof is analogous to that for [83, Proposition 6.8]. □

Lemmas 24 and 25 respectively state that parallel composition and scoping preserve bisimilarity.

Lemma 24. If $K \models P \approx P'$ then $K \models P \parallel Q \approx P' \parallel Q$. ■

Lemma 25. If $K,c \models P \approx Q$ then $K \models (\nu c)P \approx (\nu c)Q$. ■

Lemma 26 states that extending the interface does not effect the bisimilarity of two systems.

Lemma 26. For any $c \in \textsc{Chans}$, if $K \models P \approx Q$ then $K,c \models P \approx Q$.

Proof. The proof is straightforward by definition. Intuitively, it holds because since we are working up to α-equivalence, we could simply rename the bound names if channel name c in already bound in either process P or process Q. □

Lemma 27 states that removing parallel output messages from bisimilar systems also preserves their bisimilarity.

Lemma 27. If $K \models P \parallel a!b.\mathbf{0} \approx Q \parallel a!b.\mathbf{0}$ then $K \models P \approx Q$.

Proof. Let $\mathcal{R}$ be the relation defined as follows:

$$\mathcal{R} = \left\{ \langle K \triangleright P, K \triangleright Q \rangle \quad | \quad K \models P \parallel a!b.\mathbf{0} \approx Q \parallel a!b.\mathbf{0} \right\}$$

To show $\mathcal{R}$ is a bisimilarity, pick a pair $\langle K \triangleright P, K \triangleright Q \rangle \in \mathcal{R}$ and assume $K \triangleright P \xrightarrow{\eta} K' \triangleright P'$ for some $\eta \in \textsc{Act}$. We show there exists a process Q' such that $K \triangleright Q \overset{\eta}{\Longrightarrow} K' \triangleright Q'$ and $\langle K' \triangleright P', K' \triangleright Q' \rangle \in \mathcal{R}$.

Applying rule pPar on $K \triangleright P \xrightarrow{\eta} K' \triangleright P'$, we obtain $K \triangleright P \parallel a!b.\mathbf{0} \xrightarrow{\eta} K' \triangleright P' \parallel a!b.\mathbf{0}$. By definition of $\mathcal{R}$, we also know that $K \models P \parallel a!b.\mathbf{0} \approx Q \parallel a!b.\mathbf{0}$. By the definition of $\approx$, there exists a process R such that

$$K \triangleright Q \parallel a!b.\mathbf{0} \overset{\eta}{\Longrightarrow} K' \triangleright R \quad \text{and} \quad K' \models P' \parallel a!b.\mathbf{0} \approx R \tag{J.14}$$

By rules pOut, pPar and pStr, we also know $K' \triangleright P' \parallel a!b.\mathbf{0} \xrightarrow{a!b} P'$. But since $K' \models P' \parallel a!b.\mathbf{0} \approx R$, there must exist a process S such that $K' \triangleright R \overset{a!b}{\Longrightarrow} S$ where $K' \models P' \approx S$. By Corollary J.5, we also know $R \equiv Q' \parallel a!b.\mathbf{0}$ for some Q', which by rule pStr implies that

$$K \triangleright Q \parallel a!b.\mathbf{0} \overset{\eta}{\Longrightarrow} K' \triangleright Q' \parallel a!b.\mathbf{0}$$

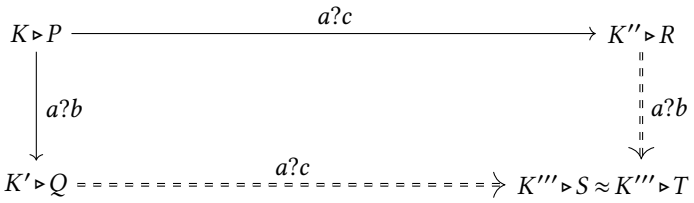
Using Corollary I.10, we obtain the required move, namely $K \triangleright Q \overset{\eta}{\Longrightarrow} K' \triangleright Q'$.
By $R \equiv Q' \parallel a!b.\mathbf{0}$ and Theorem 12.11, we also know $K' \models R \approx Q' \parallel a!b.\mathbf{0}$. Together with the statement in eq. (J.14) and symmetry/transitivity, this gives us that $K' \models P' \parallel a!b.\mathbf{0} \approx Q' \parallel a!b.\mathbf{0}$. By definition of $\mathcal{R}$, we can thus conclude that $\langle K' \triangleright P', K' \triangleright Q' \rangle \in \mathcal{R}$ as required. □

Proving Confluence w.r.t. Input Actions

Equipped with these results, we can now prove Theorem 13.46 from Section 13.3, restated below.

Theorem J.28 (Input Confluence). If a system $K \triangleright P$ is *strongly receptive* on channel a, then that system is confluent w.r.t. input actions $a?b$ and $a?c$ where $b \neq c$.

Proof. Suppose $K \triangleright P$ is strongly receptive on channel name a. Suppose also that $K \triangleright P \xrightarrow{a?b} K' \triangleright Q$ and $K \triangleright P \xrightarrow{a?c} K'' \triangleright R$ where $b \neq c$. We have to complete the following diagram, where $K' = K \cup \{b\}$, $K'' = K \cup \{b\}$ and $K''' = K \cup \{b,c\}$, the solid lines indicate the move that are given and the dotted lines are those that are required:



Assume that $P \equiv (\nu \vec{d})P'$ and $P' \not\equiv (\nu \vec{d})P''$ and $\vec{d} \sharp a$. Since P is strongly receptive on a, then we know that both $K' \triangleright Q$ and $K'' \triangleright R$ contain sub-processes that are bisimilar to the initial process P', that is:

$$Q \equiv (\nu \vec{d})(Q_1 \parallel Q_2) \quad \text{and} \quad K', \vec{d} \models Q_1 \approx P' \tag{J.15}$$

$$R \equiv (\nu \vec{d})(R_1 \parallel R_2) \quad \text{and} \quad K'', \vec{d} \models R_1 \approx P' \tag{J.16}$$

Applying rule pStr on the transition $K \triangleright P \xrightarrow{a?b} K' \triangleright Q$ and the equivalences $P \equiv (\nu \vec{d})P'$ and $Q \equiv (\nu \vec{d})(Q_1 \parallel Q_2)$, we obtain the transition $K \triangleright (\nu \vec{d})P' \xrightarrow{a?b} K' \triangleright (\nu \vec{d})(Q_1 \parallel Q_2)$. Since we are working up to α-equivalence, we can assume $b \sharp \vec{d}$. We can therefore repeatedly apply Lemma 11 and get $K, \vec{d} \triangleright P' \xrightarrow{a?b} K', \vec{d} \triangleright Q_1 \parallel Q_2$. Using Lemma 22, we can extend the interface of this transition by channel name c, which gives us $K'', \vec{d} \triangleright P' \xrightarrow{a?b} K''', \vec{d} \triangleright Q_1 \parallel Q_2$. By the equivalence in eq. (J.16) and the definition of $\approx$, we obtain

$$K'', \vec{d} \triangleright R_1 \xRightarrow{a?b} K''', \vec{d} \triangleright R_1' \qquad \text{where} \qquad K''', \vec{d} \models R_1' \approx Q_1 \parallel Q_2 \tag{J.17}$$

Repeatedly applying rules pPar and pScp1, we obtain $K'' \triangleright (\nu \vec{d})(R_1 \parallel R_2) \xRightarrow{a?b} K''' \triangleright (\nu \vec{d})(R_1' \parallel R_2)$. The first required move, namely $K'' \triangleright R \xRightarrow{a?b} K''' \triangleright (\nu \vec{d})(R_1' \parallel R_2)$, follows by the structural equivalence statement in eq. (J.16) and rule pStr.

Using similar reasoning, we can show that there exist a weak transition

$$K' \triangleright Q \xRightarrow{a?b} K''' \triangleright (\nu \vec{d})(Q_1' \parallel Q_2) \qquad \text{where} \qquad K''', \vec{d} \models Q_1' \approx R_1 \parallel R_2 \tag{J.18}$$

which gives us the second required move.

There only remains to show that $K''' \models (\nu \vec{d})(Q_1' \parallel Q_2) \approx (\nu \vec{d})(R_1' \parallel R_2)$.
From the bisimilarity statements in eqs. (J.15) to (J.18) and Lemma 24, we know that

$$K''', \vec{d} \;\models\; Q_1' \parallel Q_2 \;\approx\; (R_1 \parallel R_2) \parallel Q_2 \;\approx\; (P' \parallel R_2) \parallel Q_2$$

$$K''', \vec{d} \;\models\; R_1' \parallel R_2 \;\approx\; (Q_1 \parallel Q_2) \parallel R_2 \;\approx\; (P' \parallel Q_2) \parallel R_2$$

By symmetry/transitivity, we get $K''', \vec{d} \models Q_1' \parallel Q_2 \approx R_1' \parallel R_2$. Our result, $K''' \models (\nu \vec{d})(Q_1' \parallel Q_2) \approx (\nu \vec{d})(R_1' \parallel R_2)$, then follows by repeatedly applying Lemma 25. □

Proving Confluence w.r.t. Input Actions

The proof for Theorem 13.47 from Section 13.3 relies on a number of additional results.

Lemma 29. If $K \triangleright P \stackrel{a?b}{\Longrightarrow} Q$ and $b \in K$ then $K \triangleright P \parallel a!b.\mathbf{0} \stackrel{com(a,b)}{\Longrightarrow} Q$.

Proof. Assume $K \triangleright P \stackrel{a?b}{\Longrightarrow} Q$, which can be expanded as $K \triangleright P \Rightarrow K \triangleright P' \xrightarrow{a?b} K \triangleright P'' \Rightarrow K \triangleright Q$. Note that, since $b \in K$, then $\mathbf{aft}(K, a?b) = K$. Repeatedly applying rules pPar on $K \triangleright P \Rightarrow P'$ gives us $K \triangleright P \parallel a!b.\mathbf{0} \Rightarrow P' \parallel a!b.\mathbf{0}$. By rule pOut, we also know $K \triangleright a!b.\mathbf{0} \xrightarrow{a!b} \mathbf{0}$. Using rules pStr and pComm and $K \triangleright P' \xrightarrow{a?b} P''$ and $K \triangleright a!b.\mathbf{0} \xrightarrow{a!b} \mathbf{0}$ we thus obtain the move $K \triangleright P' \parallel a!b.\mathbf{0} \xrightarrow{com(a,b)} P'' \parallel \mathbf{0}$. Since $P'' \parallel \mathbf{0} \equiv P''$, by rule pStr, this implies $K \triangleright P' \parallel a!b.\mathbf{0} \xrightarrow{com(a,b)} P''$. Therefore, we have that

$$K \triangleright P \parallel a!b.\mathbf{0} \Rightarrow K \triangleright P' \parallel a!b.\mathbf{0} \xrightarrow{com(a,b)} K \triangleright P'' \Rightarrow Q$$

which implies that $K \triangleright P \parallel a!b.\mathbf{0} \stackrel{com(a,b)}{\Longrightarrow} Q$ as required. □

Lemma 30 describes the structure of a system capable of performing two different internal communication actions, described by the actions $com(a,b)$ and $com(a,c)$ with $b \neq c$.

Lemma 30. If $K \triangleright P \xrightarrow{com(a,b)} P'$ and $K \triangleright P \xrightarrow{com(a,c)} P''$ where $b \neq c$, then $\exists Q, Q', Q'' \in \text{Prc}$ such that

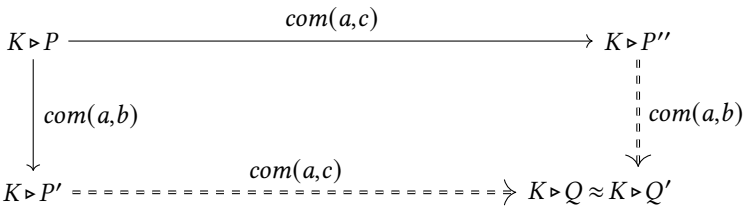
(i) $P \equiv Q \parallel a!b.\mathbf{0} \parallel a!c.\mathbf{0}$;
(ii) $K \triangleright Q \xrightarrow{a?b} Q'$ and $K \triangleright Q \xrightarrow{a?c} Q''$ for some processes Q' and Q'';
(iii) $P' \equiv Q' \parallel a!c.\mathbf{0}$ and $P'' \equiv Q'' \parallel a!b.\mathbf{0}$.

Proof. Follows as a corollary from Lemma 11 from Appendix J.2. □

We are now in a position to prove Theorem 13.47, restated below.

Theorem J.31 (Communication Confluence). If a system $K \triangleright P$ is *strongly receptive* on channel a, then that system is confluent w.r.t. communication actions $com(a,b)$ and $com(a,c)$ where $b \neq c$.

Proof. Suppose $K \triangleright P$ is strongly receptive on channel name a. Suppose also that $K \triangleright P \xrightarrow{com(a,b)} K \triangleright P'$ and $K \triangleright P \xrightarrow{com(a,c)} K \triangleright P''$ where $b \neq c$. We have to complete the following diagram, where the solid lines indicate the moves that are given and the dotted lines are those that are required.



By Lemma 30, we know there exists processes R, R', R'' such that

$$P \equiv R \parallel a!b.\mathbf{0} \parallel a!c.\mathbf{0} \tag{J.19}$$

$$K \triangleright Q \xrightarrow{a?b} K' \triangleright R' \text{ and } K \triangleright Q \xrightarrow{a?c} K'' \triangleright R'' \tag{J.20}$$

$$P' \equiv R' \parallel a!c.\mathbf{0} \text{ and } P'' \equiv R'' \parallel a!b.\mathbf{0} \tag{J.21}$$

Applying rule pPar on the transitions in eq. (J.20) gives us $K \triangleright R \parallel a!b.\mathbf{0} \parallel a!c.\mathbf{0} \xrightarrow{a?b} R' \parallel a!b.\mathbf{0} \parallel a!c.\mathbf{0}$ and $K \triangleright R \parallel a!b.\mathbf{0} \parallel a!c.\mathbf{0} \xrightarrow{a?c} R'' \parallel a!b.\mathbf{0} \parallel a!c.\mathbf{0}$. By rule pStr and the structural equivalence in eq. (J.19), we thus obtain the following moves:

$$K \triangleright P \xrightarrow{a?b} K' \triangleright R' \parallel a!b.\mathbf{0} \parallel a!c.\mathbf{0}$$

$$K \triangleright P \xrightarrow{a?c} K'' \triangleright R'' \parallel a!b.\mathbf{0} \parallel a!c.\mathbf{0}$$

But by Lemma 2, we also know $a,b,c \in K$, which means that $K = K' = K''$. Therefore, by the assumption that $K \triangleright P$ is strongly confluent on a and Theorem 13.46, we know there exist processes S,S' such that

$$\begin{array}{ccc} K \triangleright P & \xrightarrow{\quad a?c \quad} & K \triangleright Q_2 \parallel a!b.\mathbf{0} \parallel a!c.\mathbf{0} \\ \Big\downarrow{\scriptstyle a?b} & & \Big\Downarrow{\scriptstyle a?b} \\ K \triangleright Q_1 \parallel a!b.\mathbf{0} \parallel a!c.\mathbf{0} & \xRightarrow{\quad a?c \quad} & K \triangleright S \approx K \triangleright S' \end{array}$$

By Lemma 8 and the transition $K \triangleright R' \parallel a!b.\mathbf{0} \parallel a!c. \xRightarrow{a?b} K \triangleright S$ from the diagram above, we know there exists some process Q such that $K \triangleright R' \xRightarrow{a?c} K \triangleright Q$ and $S = Q \parallel a!b.\mathbf{0} \parallel a!c.\mathbf{0}$. Using Lemma 29, this also gives us $K \triangleright R' \parallel a!c.\mathbf{0} \xRightarrow{com(a,c)} K \triangleright Q$. By the structural equivalence in J.21 and rule sTrn, we obtain the first required move, namely $K \triangleright P' \xRightarrow{com(a,c)} K \triangleright Q$.

Using similar reasoning for the transition $K \triangleright R'' \parallel a!b.\mathbf{0} \parallel a!c.\mathbf{0} \xRightarrow{a?c} K \triangleright S'$ in the diagram above, we can obtain the second required move, $K \triangleright P'' \xRightarrow{com(a,b)} K \triangleright Q'$.

There only remains to show that $K \triangleright Q \approx K \triangleright Q'$. This follows by Lemma 27 and $K \triangleright S \approx K \triangleright S'$ from the diagram above where $S = Q \parallel a!b.\mathbf{0} \parallel a!c.\mathbf{0}$ and $S' = Q' \parallel a!b.\mathbf{0} \parallel a!c.\mathbf{0}$. □

Syntactic Conditions

In this section, we give the proofs for Propositions 13.52, 13.56 and 13.59. We start with Proposition 13.52, whose proof relies on Lemma 32 below.

Lemma 32. If $K \triangleright P \xrightarrow{\eta} P'$ then $K \triangleright P[Q/X] \xrightarrow{\eta} P'[Q/X]$.

Proof. The proof is straightforward by induction on the derivation of $K \triangleright P \xrightarrow{\eta} P'$. □

Proposition J.32. For any valid interface $K \subseteq$ Chans, if $a \in iac(P)$ then $K \triangleright P$ is receptive on a.

Proof. Assume that a is an input active channel of P, *i.e.,* $a \in iac(P)$. The proof is by induction on the structure of P. We outline the main cases:

- When $P = b?x.Q$, by the assumption that $a \in iac(P)$, we must have $b = a$. Our result follows by rule pIn since $b?x.Q \xrightarrow{b?c} Q[c/x]$ for any $c \in$ Chans.
- When $P = b!c.Q$, then by our assumption that $asy(P)$, we have $Q = \mathbf{0}$. Since $a \notin iac(\mathbf{0})$, then $a \notin iac(a!c.\mathbf{0})$, giving us a contradiction. Thus this case cannot arise.

- When $P = P_1 \parallel P_2$, then either $a \in P_1$ or $a \in P_2$. *W.l.o.g.*, suppose the former. By the IH, we obtain that $K \triangleright P_1$ is receptive on a, *i.e.,* $K \triangleright P_1 \Rightarrow \cdot \xrightarrow{a?\_} Q_1$ for some Q_1. Repeatedly applying rule pParL, we obtain $K \triangleright P_1 \parallel P_2 \Rightarrow \cdot \xrightarrow{a?\_} Q_1 \parallel P_2$, meaning that $K \triangleright P_1 \parallel P_2$ is receptive on a.
- When $P = \mathsf{rec}X.Q$, then $a \in iac(Q)$. By the IH, we get that $K \triangleright Q$ is receptive on a, *i.e.,* $K \triangleright Q \Rightarrow \cdot \xrightarrow{a?\_} Q'$ for some Q'. But by rule pRec, we also know $K \triangleright \mathsf{rec}X.Q \xrightarrow{\tau} Q[\mathsf{rec}X.Q/X]$. Using Lemma 32 and $K \triangleright Q \Rightarrow \cdot \xrightarrow{a?\_} Q'$, we obtain $Q[\mathsf{rec}X.Q/X] \Rightarrow \cdot \xrightarrow{a?\_} Q'[\mathsf{rec}X.Q/X]$. Our result follows.
- When $P = (\mathsf{if}\ a = a\ \mathsf{then}\ P'\ \mathsf{else}\ P'')$, then $a \in iac(P')$. By the IH, we obtain that $K \triangleright P'$ is receptive on channel name a.

The remaining cases follow with similar reasoning. □

The proof for Proposition 13.56 relies on a number of additional results. We start with Lemma 33 below, stating that if a process is input-replicated on a given channel name, then this property holds for all structurally equivalent processes.

Lemma 33. If $P \equiv Q$ and $r(P,a)$ then $r(Q,a)$.

Proof. Straightforward by rule induction on $P \equiv Q$. □

Lemma 34. If $r(P,a)$ and $r(Q,a)$ then $r(P[Q/X])$.

Proof. Straightforward by rule induction on $r(P,a)$. □

Lemma 35 asserts that the judgement $r(-)$ is closed under all τ-transitions.

Lemma 35. If $r(P,a)$ and $K \triangleright P \xrightarrow{\tau} Q$ then $r(Q,a)$.

Proof. Assume $r(P,a)$ and $K \triangleright P \xrightarrow{\tau} Q$. The proof proceeds by rule induction on $r(P,a)$.

- Rules rOut, rIn1, rIn2 could not have been used since $K \triangleright P \xrightarrow{\tau} Q$.
- Case rScp, *i.e.,* $r((\nu c)P', a)$ because $a \neq c$ and $r(P', a)$. By Lemma 11 and the move $(\nu c)P' \xrightarrow{\tau} Q$, we know that there exists a process P'' such that $Q \equiv (\nu c)P''$ and $K, c \triangleright P' \xrightarrow{\tau} P''$. Using $r(P', a)$ and $K, c \triangleright P' \xrightarrow{\tau} P''$ and the IH, we obtain $r(P'', a)$. Since $a \neq c$, applying rule rScp gives us $r((\nu c)P'', a)$. Our result, namely $r(Q, a)$, follows by the fact that $Q \equiv (\nu c)P''$ and Lemma 33.
- Case rRec1, *i.e.,* $r(\mathsf{rec}X.P', a)$ because $P' = a?x.(P'' \parallel X)$ where $X \notin fV(P'')$ and $r(P'', a)$. Since the move $\mathsf{rec}X.P' \xrightarrow{\tau} Q$ could have only been deriving using pRec, we know $Q = P'[\mathsf{rec}X.P'/X]$. Thus we have

$$\begin{aligned} P'[\mathsf{rec}X.P'/X] &= \big(a?x.(P'' \parallel X)\big)[\mathsf{rec}X.P'/X] \\ &= a?x.(P'' \parallel X([\mathsf{rec}X.P'/X])) \text{ since } X \notin fV(P'') \\ &= a?x.(P'' \parallel \mathsf{rec}X.P') \\ &= a?x.(P'' \parallel a?x.P'') \text{ since } P' = a?x.(P'' \parallel X) \end{aligned}$$

  Applying rule rIn1, we conclude $r(a?x.(P'' \parallel a?x.P''), a)$, *i.e.,* $r(Q, a)$.

- Case rRec2, *i.e.,* $r(\text{rec}X.P',a)$ because $P' \neq a?x.P''$ and $r(P',a)$. Since the move $\text{rec}X.P' \xrightarrow{\tau} Q$ could have only been deriving using pRec, we know $Q = P'[^{\text{rec}X.P'}/_X]$. Our result, $r(P'[^{\text{rec}X.P'}/_X],a)$, follows by Lemma 34 and the facts that $r(P',a)$ and $r(\text{rec}X.P',a)$.

The remaining cases follow with similar but more straightforward reasoning. □

Lemma 36 states that after inputting on channel name a, a process P satisfying the judgement $r(P,a)$ evolves into a process that contains a sub-process bisimilar to P.

Lemma 36. Suppose that $P \not\equiv (\nu c)P'$ for all $P' \in \text{Prc}$. If $r(P,a)$ and $K \triangleright P \xrightarrow{a?b} Q$, then there exist processes Q_1, Q_2 such that $Q \equiv Q_1 \parallel Q_2$ and $\mathbf{aft}(K,a?b) \models Q_1 \approx P$.

Proof. Suppose that $r(P,a)$ and $K \triangleright P \xrightarrow{a?b} Q$ where $K' = \mathbf{aft}(K,a?b)$. The proof proceeds by rule induction on the latter.

- Case pIn, *i.e.,* $K \triangleright a?x.P' \xrightarrow{a?b} Q$ where $Q = P'[^b/_x]$. By case analysis, $r(a?x.P',a)$ could have only been derived by rule rIn1, which implies that $P' = P'' \parallel *a?x.P''$ and $r(P'',a)$. We thus have

$$Q = P'[^b/_x] = \left(P'' \parallel *a?x.P''\right)[^b/_x] = P''[^b/_x] \parallel *a?x.P'' \text{ since } x \notin fv(*a?x.P'') \tag{J.22}$$

  In what follows, we show that $K' \models *a?x.P'' \approx a?x.P$, which gives us the required result.
  Since $*a?x.P'' = \text{rec}X.a?x.(P'' \parallel X)$, by rule pRec, we know that $K' \triangleright *a?x.P'' \xrightarrow{\tau} a?x.(P'' \parallel *a?x.P'')$. But by Lemma 23, we also know $K' \models *a?x.P'' \approx a?x.(P'' \parallel *a?x.P'')$. Since $P' = P'' \parallel *a?x.P''$, this means that $K' \models *a?x.P'' \approx a?x.P'$ as required.

- Case pPar, *i.e.,* $K \triangleright P_1 \parallel P_2 \xrightarrow{a?b} P_1' \parallel P_2$ because $K \triangleright P_1 \xrightarrow{a?b} P_1'$. Since $r(P_1 \parallel P_2,a)$ could have only been derived by rule rPar, we know $r(P_1,a)$. By the IH, we obtain that there exist Q_1, Q_2 such that $P_1' \equiv Q_1 \parallel Q_2$ and $K' \models Q_1 \approx P_1$ where $K' = \mathbf{aft}(K,a?b)$. Applying rule sCtxP to $P_1' \equiv Q_1 \parallel Q_2$, we obtain $P_1' \parallel P_2 \equiv (Q_1 \parallel Q_2) \parallel P_2$, which by rules sAss, sCom give us that $P_1' \parallel P_2 \equiv (Q_1 \parallel P_2) \parallel Q_2$. Also, using the bisimilarity $K' \models Q_1 \approx P_1$ and Lemma 24, we obtain $K' \models Q_1 \parallel P_2 \approx P_1 \parallel P_2$.
  We have thus show that for $P = P_1 \parallel P_2$ and $Q = P_1' \parallel P_2$, we have that $Q \equiv (Q_1 \parallel P_2) \parallel Q_2$ such that $K' \models Q_1 \parallel P_2 \approx P$ as required.

- Case pStr, the proof is straightforward.

We note that rule pScp1 could not have been used because of the assumption $P \not\equiv (\nu c)P'$ for all P'. □

Based on these results, we are now in a position to prove Proposition 13.56, restated below.

Proposition J.36. For any valid interface $K \subseteq \text{Chans}$, if $a \in iac(P)$ and $r(P,a)$ then system $K \triangleright P$ is strongly receptive on channel name a.

Proof. Suppose that $a \in iac(P)$ and $r(P,a)$. According to Definition 13.42, we need to show:

(i) System $K \triangleright P$ is receptive on a.
(ii) Suppose $P \equiv (\nu \vec{d})P'$ such that $P' \not\equiv (\nu c)P''$ for all $P'' \in \text{Prc}$. If $K' \triangleright P' \Longrightarrow K' \triangleright Q \xrightarrow{a?b} Q'$ where $K' = K,\vec{d}$ then $\exists Q_1, Q_2 \in \text{Prc}$ such that $Q' \equiv Q_1 \parallel Q_2$ and $\mathbf{aft}(K',a?b) \models Q_1 \approx P'$.

The result in (i) follows immediately by $a \in iac(P)$ and Proposition 13.52.

To show (ii), suppose that $P \equiv (\nu \vec{d})P'$ such that $P' \not\equiv (\nu c)P''$ for all processes $P'' \in \text{Prc}$. Suppose also that

$$K' \triangleright P' \Longrightarrow K' \triangleright Q \xrightarrow{a?b} K'' \triangleright Q'$$

where $K' = K, \vec{d}$ and $K'' = \mathbf{aft}(K', a?b)$. From Lemma 33 and the assumptions that $r(P,a)$ and $P \equiv (\nu \vec{d})P'$, we also know $r((\nu \vec{d}),a)$. By case analysis, this could have only been derived using rule rScp $|\vec{d}|$ times, which gives us that $r(P',a)$ as well. Together with Lemma 35 and the weak transition $K' \triangleright P' \Longrightarrow K' \triangleright Q$, this implies that $r(Q,a)$. Using Lemma 23, we also know $K' \models P' \approx Q$ which by Lemma 26 implies that

$$K'' \models P' \approx Q \tag{J.23}$$

By Lemma 36 and $r(Q,a)$ and $K' \triangleright Q \xrightarrow{a?b} K'' \triangleright Q'$, we obtain that there exists processes Q_1, Q_2 such that

$$Q' \equiv Q_1 \parallel Q_2 \quad \text{and} \quad K'' \models Q_1 \approx Q \tag{J.24}$$

Using symmetry/transitivity and the bisimilarities in eqs. (J.23) and (J.24), we conclude $K'' \models Q_1 \approx P'$. □

We prove Proposition 13.59 in three parts. Concretely, we first show that the locality constraint is preserved by all transitions; we already proved this in Lemma 19. Second, Lemma 38 shows that given a local process, the input-replication constraint is preserved by all transitions. Finally, Lemma 39 shows that given a local, input-replicated process on a given channel name, all reachable systems are also input-active on that channel. We start by proving Lemma 38, which relies on Lemma 37 below.

Lemma 37. If $loc(P,V)$ and $r(P,a)$ then $r(P[^b/_x],a)$.

Proof. The proof is straightforward by rule induction on $r(P,a)$. □

Lemma 38 (Input-Replication Preservation). If $loc(P)$ and $r(P,a)$ and $K \triangleright P \xrightarrow{\eta} Q$ then $r(P,a)$.

Proof. The proof is by rule induction on $r(P,a)$. We only outline the main cases:

- Rule rIn1, *i.e.,* $r(a?x.P',a)$ because $P' = P'' \parallel *a?x.P''$ and $r(P'',a)$. Assume $K \triangleright a?x.P' \xrightarrow{\eta} Q$. From this, we can show that we must have $\eta = a?b$ for some b and $Q \equiv P'[^b/_x]$; we omit the proof as it is quite straightforward. By the assumption that $loc(P)$, we also know $loc(P',\{x\})$. Using Lemma 37 and $r(P',a)$, we obtain $r(P'[^b/_x],a)$. Applying rule rRec1 on $r(P',a)$, we also know $r(*a?x.P')$. Applying rule rPar, we obtain $r(P''[^b/_x] \parallel *a?x.P'',a)$. Our result, $r(Q,a)$, follows by Lemma 33 and the fact that $Q \equiv P'[^b/_x] = P''[^b/_x] \parallel *a?x.P''$.

The proofs for cases rRec1, rRec2 and rScp are similar to those for Lemma 35. The remaining cases are straightforward. □

Lemma 39 (Input-Active Preservation). If $loc(P)$ and $r(P,a)$ and $a \in iac(P)$ and $K \triangleright P \xrightarrow{\eta} Q$ then $a \in iac(Q)$.

Proof. The proof is straightforward by rule induction on $r(P,a)$. □

Proposition J.39 (Preservation). Suppose that $a \in iac(P)$ and $r(P,a)$ and $loc(P)$. If $K \triangleright P \xrightarrow{\eta} Q$ then $a \in iac(P)$ and $r(Q,a)$ and $loc(Q)$.

Proof. Follows from Lemmas 19, 38 and 39. □

Bibliography

[1] Samson Abramsky. Observation equivalence as a testing equivalence. *TCS*, 53(2):225–241, 1987.

[2] Luca Aceto and Anna Ingólfsdóttir. Testing Hennessy-Milner Logic with Recursion. In *FoSSaCS*, volume 1578 of *LNCS*, pages 41–55. Springer, 1999.

[3] Luca Aceto, Anna Ingólfsdóttir, Kim Guldstrand Larsen, and Jiri Srba. *Reactive Systems: Modelling, Specification and Verification.* Cambridge U.P., 2007. ISBN 9780521875462.

[4] Luca Aceto, Arnar Birgisson, Anna Ingolfsdottir, MohammadReza Mousavi, and Michel A. Reniers. Rule formats for determinism and idempotence. *SCP*, 77(7–8):889–907, 2012.

[5] Luca Aceto, Antonis Achilleos, Adrian Francalanza, and Anna Ingólfsdóttir. Monitoring for silent actions. In *FSTTCS*, volume 93 of *LIPIcs*, pages 7:1–7:14. Schloss Dagstuhl, 2017.

[6] Luca Aceto, Antonis Achilleos, Adrian Francalanza, and Anna Ingólfsdóttir. Monitoring for Silent Actions. In *FSTTC*, volume 93 of *LIPIcs*, pages 7:1–7:14. Schloss Dagstuhl, 2017.

[7] Luca Aceto, Antonis Achilleos, Adrian Francalanza, and Anna Ingólfsdóttir. A Framework for Parameterized Monitorability. In *FOSSACS*, volume 10803 of *LNCS*, pages 203–220. Springer, 2018.

[8] Luca Aceto, Antonis Achilleos, Adrian Francalanza, Anna Ingólfsdóttir, and Karoliina Lehtinen. The Cost of Monitoring Alone. In *From Reactive Systems to Cyber-Physical Systems*, volume 11500 of *LNCS*, pages 259–275, 2019.

[9] Luca Aceto, Antonis Achilleos, Adrian Francalanza, Anna Ingólfsdóttir, and Karoliina Lehtinen. An Operational Guide to Monitorability. In *SEFM*, volume 11724 of *LNCS*, pages 433–453, 2019.

[10] Luca Aceto, Antonis Achilleos, Adrian Francalanza, Anna Ingólfsdóttir, and Karoliina Lehtinen. Adventures in Monitorability: From Branching to Linear Time and Back Again. *PACMPL*, 3(POPL): 52:1–52:29, 2019.

[11] Luca Aceto, Antonis Achilleos, Adrian Francalanza, Anna Ingólfsdóttir, and Sævar Örn Kjartansson. Determinizing monitors for HML with recursion. *JLAMP*, 111:100515, 2020.

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

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

[14] Luca Aceto, Duncan Paul Attard, Adrian Francalanza, and Anna Ingólfsdóttir. On Benchmarking for Concurrent Runtime Verification. In *FASE*, volume 12649 of *LNCS*, pages 3–23. Springer, 2021.

[15] Luca Aceto, Antonis Achilleos, Duncan Paul Attard, Léo Exibard, Adrian Francalanza, and Anna Ingólfsdóttir. A Monitoring Tool for Linear-Time μHML. In *COORDINATION*, volume 13271 of *LNCS*, pages 200–219. Springer, 2022.

[16] Luca Aceto, Ian Cassar, Adrian Francalanza, and Anna Ingólfsdóttir. Bidirectional runtime enforcement of first-order branching-time properties. *Log. Methods Comput. Sci.*, 19(1), 2023.

[17] Luca Aceto, Antonis Achilleos, Elli Anastasiadi, Adrian Francalanza, Daniele Gorla, and Jana Wagemaker. Centralized vs Decentralized Monitors for Hyperproperties. In *CONCUR*, volume 311 of *LIPIcs*, pages 4:1–4:19, 2024.

[18] Luca Aceto, Duncan Paul Attard, Adrian Francalanza, and Anna Ingólfsdóttir. Runtime instrumentation for reactive components. In *ECOOP*, volume 313 of *LIPIcs*, pages 2:1–2:33, 2024.

[19] Antonis Achilleos, Léo Exibard, Adrian Francalanza, Karoliina Lehtinen, and Jasmine Xuereb. A Synthesis Tool for Optimal Monitors in a Branching-Time Setting. In *COORDINATION*, volume 13271 of *LNCS*, pages 181–199, 2022.

[20] Gul Agha and Prasanna Thati. An Algebraic Theory of Actors and Its Application to a Simple Object-Based Language. In *From Object-Orientation to Formal Methods, Essays in Memory of Ole-Johan Dahl*, volume 2635 of *LNCS*, pages 26–57, 2004.

[21] Gul A. Agha. *ACTORS - A Model of Concurrent Computation in Distributed Systems.* MIT Press, 1990. ISBN 978-0-262-01092-4.

[22] Gul A. Agha. *ACTORS - A Model of Concurrent Computation in Distributed Systems.* MIT Press, 1990. ISBN 978-0-262-01092-4.

[23] Shreya Agrawal and Borzoo Bonakdarpour. Runtime Verification of k-Safety Hyperproperties in HyperLTL. In *IEEE*, pages 239–252, 2016.

[24] Wolfgang Ahrendt, Jesús Mauricio Chimento, Gordon J. Pace, and Gerardo Schneider. A specification language for static and runtime verification of data and control properties. In *FM*, volume 9109 of *LNCS*, pages 108–125. Springer, 2015.

[25] Bowen Alpern and Fred B. Schneider. Recognizing Safety and Liveness. *Distributed Comput.*, 2(3): 117–126, 1987.

[26] Roberto M. Amadio, Ilaria Castellani, and Davide Sangiorgi. On bisimulations for the asynchronous π-calculus. *TCS*, 195(2):291–324, 1998.

[27] Roberto M. Amadio, Gérard Boudol, and Cédric Lhoussaine. The receptive distributed pi-calculus. *ACM*, 25(5):549–577, 2003.

[28] Mouloud Amara, Giovanni Bernardi, Mohammad Foughali, and Adrian Francalanza. A Theory of (Linear-Time) Timed Monitors. In *39th European Conference on Object-Oriented Programming*

(ECOOP 2025), Leibniz International Proceedings in Informatics (LIPIcs), Dagstuhl, Germany, 2025. (to appear).

[29] Apple Inc. and the Swift project authors. *The Swift Programming Language (6.0 beta).* 2024.

[30] Duncan Paul Attard and Adrian Francalanza. A Monitoring Tool for a Branching-Time Logic. In *RV*, volume 10012 of *LNCS*, pages 473–481. Springer, 2016.

[31] Duncan Paul Attard and Adrian Francalanza. Trace Partitioning and Local Monitoring for Asynchronous Components. In *SEFM*, volume 10469 of *LNCS*, pages 219–235. Springer, 2017.

[32] Duncan Paul Attard, Luca Aceto, Antonis Achilleos, Adrian Francalanza, Anna Ingólfsdóttir, and Karoliina Lehtinen. Better Late Than Never or: Verifying Asynchronous Components at Runtime. In *IFIP*, volume 12719 of *LNCS*, pages 207–225. Springer, 2021.

[33] Giorgio Audrito, Ferruccio Damiani, Volker Stolz, Gianluca Torta, and Mirko Viroli. Distributed runtime verification by past-ctl and the field calculus. *Journal of Systems and Software*, 187:111251, 2022.

[34] Christel Baier, Joost-Pieter Katoen, and Kim Guldstrand Larsen. *Principles of model checking*. MIT press, 2008.

[35] Howard Barringer, Yliès Falcone, Klaus Havelund, Giles Reger, and David E. Rydeheard. Quantified Event Automata: Towards Expressive and Efficient Runtime Monitors. In *FM*, volume 7436 of *LNCS*, pages 68–84. Springer, 2012.

[36] Ezio Bartocci, Yliès Falcone, Adrian Francalanza, and Giles Reger. Introduction to Runtime Verification. In *Lectures on Runtime Verification - Introductory and Advanced Topics*, volume 10457 of *LNCS*, pages 1–33. Springer, 2018.

[37] Christian Bartolo Burlò, Adrian Francalanza, Alceste Scalas, Catia Trubiani, and Emilio Tuosto. PSTMonitor: Monitor synthesis from probabilistic session types. *SCP*, 2022.

[38] Andreas Bauer, Martin Leucker, and Christian Schallhart. Runtime verification for LTL and TLTL. *ACM*, 20(4):14:1–14:64, 2011.

[39] Gerd Behrmann, Alexandre David, and Kim G. Larsen. *A Tutorial on Uppaal*, pages 200–236. Springer, 2004.

[40] Jesper Bengtson and Joachim Parrow. Formalising the pi-calculus using nominal logic. *Log. Methods Comput. Sci.*, 5(2), 2009.

[41] Nataliia Bielova and Fabio Massacci. Do you really mean what you actually enforced? edited automata revisited. *IJIS*, 10(4):239–254, 2011.

[42] Borzoo Bonakdarpour, Pierre Fraigniaud, Sergio Rajsbaum, David A. Rosenblueth, and Corentin Travers. Decentralized Asynchronous Crash-resilient Runtime Verification. *JACM*, 69(5):34:1–34:31, 2022.

[43] Gérard Boudol. Asynchrony and the π-calculus. Research Report RR-1702, INRIA, 1992.

[44] Gérard Boudol. Typing the Use of Resources in a Concurrent Calculus (Extended Abstract). In *ASIAN*, volume 1345 of *LNCS*, pages 239–253, 1997.

[45] Janusz A. Brzozowski. Derivatives of Regular Expressions. *JACM*, 11(4):481–494, 1964.

[46] Ian Cassar, Adrian Francalanza, Luca Aceto, and Anna Ingólfsdóttir. A Survey of Runtime Monitoring Instrumentation Techniques. In *PrePost@iFM*, volume 254 of *EPTCS*, pages 15–28, 2017.

[47] Armando Castañeda and Gilde Valeria Rodríguez. Asynchronous wait-free runtime verification and enforcement of linearizability. In *PODC*, pages 90–101. ACM, 2023.

[48] Francesca Cesarini and Simon Thompson. *Erlang Programming - A Concurrent Approach to Software Development*. O'Reilly, 2009.

[49] Feng Chen and Grigore Rosu. Parametric Trace Slicing and Monitoring. In *TACAS*, volume 5505 of *LNCS*, pages 246–261. Springer, 2009.

[50] Alessandro Cimatti, Chun Tian, and Stefano Tonetta. Assumption-Based Runtime Verification with Partial Observability and Resets. In *Runtime Verification*, volume 11757 of *LNCS*, pages 165–184. Springer, 2019.

[51] Edmund M Clarke, Orna Grumberg, and Doron Peled. *Model Checking*. MIT press, 1999.

[52] Michael R. Clarkson and Fred B. Schneider. Hyperproperties. *JCS*, 18(6):1157–1210, 2010.

[53] Michael R. Clarkson, Bernd Finkbeiner, Masoud Koleini, Kristopher K. Micinski, Markus N. Rabe, and César Sánchez. Temporal logics for hyperproperties. In *POST*, volume 8414 of *LNCS*, pages 265–284. Springer, 2014.

[54] Sjoerd Cranen, Jan Friso Groote, Jeroen J. A. Keiren, Frank P. M. Stappers, Erik P. de Vink, Wieger Wesselink, and Tim A. C. Willemse. An Overview of the mCRL2 Toolset and Its Recent Advances. In *TACAS*, volume 7795 of *LNCS*, pages 199–213. Springer, 2013.

[55] Paulo Salem da Silva and Ana C.V. de Melo. Model checking merged program traces. *Electronic Notes in Theoretical Computer Science*, 240:97–112, 2009. ISSN 1571-0661. SBMF.

[56] Normann Decker, Martin Leucker, Daniel Thoma, Guillaume Brat, Arnaud Venet, and Neha Rungta. jUnit^rv–adding runtime verification to junit. In *NASA Formal Methods*, LNCS, pages 459–464. 2013.

[57] Ugo de'Liguoro and Luca Padovani. Mailbox Types for Unordered Interactions. In *ECOOP*, volume 109 of *LIPIcs*, pages 15:1–15:28, 2018.

[58] Ankush Desai, Tommaso Dreossi, and Sanjit A. Seshia. Combining model checking and runtime verification for safe robotics. In *RV*, volume 10548 of *LNCS*, pages 172–189. Springer, 2017.

[59] Edsger W. Dijkstra. Correctness concerns and, among other things, why they are resented. In *Proceedings of the International Conference on Reliable Software*, page 546. ACM, 1975.

[60] Antoine El-Hokayem and Yliès Falcone. Monitoring decentralized specifications. In Tevfik Bultan and Koushik Sen, editors, *ACM*, pages 125–135, 2017.

[61] E. Allen Emerson and Charanjit S. Jutla. The Complexity of Tree Automata and Logics of Programs. *SIAM J. Comput.*, 29(1):132–158, 1999.

[62] Yliès Falcone, Jean-Claude Fernandez, and Laurent Mounier. What can you verify and enforce at runtime? *Int. J. Softw. Tools Technol. Transf.*, 14(3):349–382, 2012.

[63] Yliès Falcone, Srdan Krstic, Giles Reger, and Dmitriy Traytel. A taxonomy for classifying runtime verification tools. *IJSTTT*, 23(2):255–284, 2021.

[64] Angelo Ferrando and Rafael C. Cardoso. Towards partial monitoring: Never too early to give in. *Science of Computer Programming*, 240:103220, 2025.

[65] Angelo Ferrando and Vadim Malvone. Towards the combination of model checking and runtime verification on multi-agent systems. In *PAAMS*, volume 13616 of *LNCS*, pages 140–152. Springer, 2022.

[66] Bernd Finkbeiner, Christopher Hahn, Marvin Stenger, and Leander Tentrup. Rvhyper: A runtime verification tool for temporal hyperproperties. In *TACAS (2)*, volume 10806 of *LNCS*, pages 194–200. Springer, 2018.

[67] Bernd Finkbeiner, Christopher Hahn, Marvin Stenger, and Leander Tentrup. Monitoring hyperproperties. *FMSD*, 54(3):336–363, 2019.

[68] Cédric Fournet and Georges Gonthier. The reflexive CHAM and the join-calculus. In *POPL*, pages 372–385, 1996.

[69] Simon Fowler, Sam Lindley, and Philip Wadler. Mixing metaphors: Actors as channels and channels as actors. In *ECOOP*, volume 74 of *LIPIcs*, pages 11:1–11:28, 2017.

[70] Pierre Fraigniaud, Sergio Rajsbaum, and Corentin Travers. On the Number of Opinions Needed for Fault-Tolerant Run-Time Monitoring in Distributed Systems. In *RV*, volume 8734 of *LNCS*, pages 92–107. Springer, 2014.

[71] Adrian Francalanza. A Theory of Monitors (Extended Abstract). In *FoSSaCS*, volume 9634 of *LNCS*, 2016. doi: 10.1007/978-3-662-49630-5_9.

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

[73] Adrian Francalanza. A Theory of Monitors. *Inf. Comput.*, 281:104704, 2021.

[74] Adrian Francalanza, Luca Aceto, and Anna Ingólfsdóttir. On Verifying Hennessy-Milner Logic with Recursion at Runtime. In *RV*, volume 9333 of *LNCS*, pages 71–86. Springer, 2015.

[75] Adrian Francalanza, Luca Aceto, Antonis Achilleos, Duncan Paul Attard, Ian Cassar, Dario Della Monica, and Anna Ingólfsdóttir. A foundation for runtime monitoring. In *RV*, volume 10548 of *LNCS*, pages 8–29. Springer, 2017.

[76] Adrian Francalanza, Luca Aceto, and Anna Ingólfsdóttir. Monitorability for the Hennessy-Milner logic with recursion. *FMSD*, 51(1):87–116, 2017.

[77] Mauro Gaspari and Gianluigi Zavattaro. An algebra of actors. In *FMOODS*, volume 139 of *IFIP*, 1999.

[78] Patrice Godefroid. Using partial orders to improve automatic verification methods. In *CAV*, volume 531 of *LNCS*, pages 176–185. Springer, 1990.

[79] J. Goodwin. *Learning Akka: Build Fault-tolerant, Concurrent, and Distributed Applications with Akka.* Community experience distilled. Packt Publishing, 2015. ISBN 9781784393007.

[80] Jan Friso Groote and M. P. A. Sellink. Confluence for Process Verification. *TCS*, 1996. doi: 10.1016/S0304-3975(96)80702-X.

[81] Jan Friso Groote and Jaco van de Pol. State space reduction using partial τ-confluence. In *MFCS*, volume 1893 of *LNCS*, pages 383–393. Springer, 2000.

[82] Joseph Y Halpern and Yoram Moses. A guide to completeness and complexity for modal logics of knowledge and belief. *Artificial intelligence*, 54(3):319–379, 1992.

[83] Matthew Hennessy. *A distributed Pi-calculus.* Cambridge University Press, 2007.

[84] Matthew Hennessy and James Riely. Resource Access Control in Systems of Mobile Agents. *Inf. Comput.*, 173(1):82–120, 2002.

[85] Thomas A. Henzinger and N. Ege Saraç. Monitorability Under Assumptions. In *RV*, volume 12399 of *LNCS*, pages 3–18. Springer, 2020.

[86] Thomas A. Henzinger and N. Ege Saraç. Quantitative and approximate monitoring. In *LICS*, pages 1–14. IEEE, 2021.

[87] Carl Hewitt, Peter Boehler Bishop, and Richard Steiger. A universal modular ACTOR formalism for artificial intelligence. In *IJCAI*, pages 235–245, 1973.

[88] Timothy L. Hinrichs, A. Prasad Sistla, and Lenore D. Zuck. Model check what you can, runtime verify the rest. In *HOWARD-60*, volume 42 of *EPiC Series in Computing*, pages 234–244. EasyChair, 2014.

[89] Kohei Honda and Mario Tokoro. An object calculus for asynchronous communication. In *ECOOP*, volume 512 of *LNCS*, pages 133–147, 1991.

[90] S. Juric. *Elixir in Action, Third Edition.* Manning, 2024. ISBN 9781633438514.

[91] Katarína Kejstová, Petr Rockai, and Jiri Barnat. From model checking to runtime verification and back. In *RV*, volume 10548 of *LNCS*, pages 225–240. Springer, 2017.

[92] Robert M. Keller. Formal Verification of Parallel Programs. *Commun. ACM*, 19(7):371–384, 1976.

[93] Yonit Kesten and Amir Pnueli. A compositional approach to ctl* verification. *TCS*, 331(2-3):397–428, 2005.

[94] Dexter Kozen. Results on the Propositional mu-Calculus. *TCS*, 27:333–354, 1983.

[95] Andreas Krebs, Arne Meier, Jonni Virtema, and Martin Zimmermann. Team Semantics for the Specification and Verification of Hyperproperties. In *MFCS*, volume 117 of *LIPIcs*, pages 10:1–10:16. Schloss Dagstuhl, 2018.

[96] Orna Kupferman, Moshe Y Vardi, and Pierre Wolper. An automata-theoretic approach to branching-time model checking. *JACM*, 47(2):312–360, 2000.

[97] Kim G. Larsen. Proof systems for satisfiability in hennessy-milner logic with recursion. *TCS*, 72(2): 265 – 288, 1990.

[98] I. Lee, H. Ben-Abdallah, S. Kannan, M. Kim, O. Sokolsky, and M. Viswanathan. A Monitoring and Checking Framework for Run-time Correctness Assurance. 1998.

[99] Martin Leucker. Sliding between Model Checking and Runtime Verification. In *RV*, volume 7687 of *LNCS*, pages 82–87, 2012.

[100] Martin Leucker and Christian Schallhart. A brief account of runtime verification. *JLAMP*, 78(5): 293–303, 2009.

[101] Jay Ligatti, Lujo Bauer, and David Walker. Edit automata: enforcement mechanisms for run-time security policies. *IJIS*, 4(1-2), 2005.

[102] Qingzhou Luo and Grigore Roşu. EnforceMOP: A Runtime Property Enforcement System for Multithreaded Programs. In *ISSTA*. ACM, 2013. doi: 10.1145/2483760.2483766.

[103] Radu Mateescu and Anton Wijs. Sequential and distributed on-the-fly computation of weak τ-confluence. *Sci. Comput. Program.*, 77(10-11):1075–1094, 2012.

[104] Massimo Merro and Davide Sangiorgi. On asynchrony in name-passing calculi. *Math. Struct. Comput. Sci.*, 14(5):715–767, 2004.

[105] Robin Milner. *A Calculus of Communicating Systems*, volume 92 of *LNCS*. Springer, 1980.

[106] Robin Milner. *Communication and Concurrency*. PHI Series in computer science. Prentice Hall, 1989.

[107] Dario Della Monica and Adrian Francalanza. Pushing Runtime Verification to the Limit: May Process Semantics Be With Us. In *OVERLAY@AI*IA*, volume 2509 of *CEUR Workshop Proceedings*, pages 47–52, 2019.

[108] Rumyana Neykova, Laura Bocchi, and Nobuko Yoshida. Timed runtime monitoring for multiparty conversations. *Formal aspects of computing*, 29(5):877–910, 2017.

[109] Rocco De Nicola and Matthew Hennessy. CCS without tau's. In *TAPSOFT*, LNCS, pages 138–152, 1987.

[110] Scott Owens, John H. Reppy, and Aaron Turon. Regular-expression derivatives re-examined. *JFP*, 19(2):173–190, 2009.

[111] Doron A. Peled. Partial order reduction: Linear and branching temporal logics and process algebras. In *Partial Order Methods in Verification, Proceedings of a DIMACS Workshop*, volume 29 of *DIMACS*, pages 233–257, 1996.

[112] James L. Peterson. *Petri Net Theory and the Modeling of Systems*. Prentice-Hall, 1981. ISBN 0136619835.

[113] Anna Philippou and David Walker. On confluence in the π-calculus. In *ICALP*, volume 1256 of *LNCS*, pages 314–324, 1997.

[114] Benjamin C. Pierce and David N. Turner. Pict: a programming language based on the Pi-Calculus. In *Proof, Language, and Interaction, Essays in Honour of Robin Milner*, pages 455–494, 2000.

[115] A. Pnueli and A. Zaks. Psl model checking and run-time verification via testers. In Jayadev Misra, Tobias Nipkow, and Emil Sekerinski, editors, *FM 2006: Formal Methods*, pages 573–586, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg. ISBN 978-3-540-37216-5.

[116] Y. S. Ramakrishna and Scott A. Smolka. Partial-order reduction in the weak modal mu-calculus. In Antoni W. Mazurkiewicz and Józef Winkowski, editors, *CONCUR*, volume 1243 of *LNCS*, pages 5–24, 1997.

[117] António Ravara and Vasco Thudichum Vasconcelos. Typing Non-uniform Concurrent Objects. In *CONCUR*, volume 1877 of *LNCS*, pages 474–488, 2000.

[118] Albert Rizaldi, Jonas Keinholz, Monika Huber, Jochen Feldle, Fabian Immler, Matthias Althoff, Eric Hilgendorf, and Tobias Nipkow. Formalising and Monitoring Traffic Rules for Autonomous Vehicles in Isabelle/HOL. In *IFM*, volume 10510 of *LNCS*, 2017. doi: 10.1007/978-3-319-66845-1_4.

[119] Michael Thomas Sanderson. *Proof techniques for CCS*. PhD thesis, The University of Edinburgh, 1983.

[120] Davide Sangiorgi. The name discipline of uniform receptiveness. *Theor. Comput. Sci.*, 221(1-2): 457–493, 1999.

[121] Davide Sangiorgi and David Walker. *The Pi-Calculus - a theory of mobile processes*. Cambridge University Press, 2001.

[122] Fred B. Schneider. Enforceable Security Policies. *ACM Trans. Inf. Syst. Secur.*, 3(1), 2000.

[123] Peter Selinger. First-order axioms for asynchrony. In *CONCUR*, volume 1243 of *LNCS*, pages 376–390. 1997. ISBN 3540631410.

[124] Koushik Sen, Grigore Rosu, and Gul Agha. Generating Optimal Linear Temporal Logic Monitors by Coinduction. In *ASIAN*, volume 2896 of *LNCS*, pages 260–275. Springer, 2003.

[125] Sandro Stucki, César Sánchez, Gerardo Schneider, and Borzoo Bonakdarpour. Gray-box monitoring of hyperproperties with an application to privacy. *FMSD*, pages 1–34, 2021.

[126] Antti Valmari. Stubborn set methods for process algebras. In *Partial Order Methods in Verification, Proceedings of a DIMACS Workshop*, volume 29 of *DIMACS*, pages 213–231, 1996.

[127] Jonni Virtema, Jana Hofmann, Bernd Finkbeiner, Juha Kontinen, and Fan Yang. Linear-Time Temporal Logic with Team Semantics: Expressivity and Complexity. In *IARCS*, volume 213 of *LIPIcs*, pages 52:1–52:17. Schloss Dagstuhl, 2021.

[128] Igor Walukiewicz. Completeness of Kozen's Axiomatisation of the Propositional mu-Calculus. In *IEEE*, pages 14–24, 1995.

[129] Shaohui Wang, Anaheed Ayoub, Oleg Sokolsky, and Insup Lee. Runtime Verification of Traces under Recording Uncertainty. In *RV*, volume 7186 of *LNCS*, pages 442–456. Springer, 2011.

[130] Mingsheng Ying. Weak confluence and τ-inertness. *TCS*, 238(1-2):465–475, 2000.

[131] Nobuko Yoshida, Kohei Honda, and Martin Berger. Linearity and bisimulation. *JLAMP*, 72(2): 207–238, 2007.

[132] Xian Zhang, Martin Leucker, and Wei Dong. Runtime verification with predictive semantics. In *NASA Formal Methods*, volume 7226 of *LNCS*, pages 418–432, 2012.

Acronyms

recHML Hennessy-Milner Logic with Recursion.

sHML Safety fragment of recHML.

sHML$^{\vee}$ sHML extended with disjunctions.

sHML$^{\vee}_{\text{Det}}$ sHML extended with disjunctions, as long as they are prefixed with deterministic actions.

CCS Calculus of Communicating Systems.

CTL Computation Tree Logic.

EVM Erlang Virtual Machine.

ILTS Instrumentable Labelled Transition System.

LTL Linear Temporal Logic.

LTS Labelled Transition System.

RV Runtime Verification.

SUS System Under Scrutiny.

Notation Index

Chapter 2

$\varphi,\psi,\ldots \in$ RECHML	RECHML formula
$\eta,\xi,\ldots \in$ ACT	External action or silent τ action
$\alpha,\beta,\ldots \in$ EACT	External action
$p,q,\ldots \in$ PRC	Systems
$t,u,\ldots \in$ TRC	Trace consisting of external actions
$[\![-]\!]$	Denotational semantic formula
$\equiv$	Equivalence relation

Part I

$m,n\ldots \in$ MON	Monitor
$H,H',\ldots \in$ HST	History
$\mathbf{rej}(H,m)$	Monitor m rejects history H
$\mathbf{rej}(p,m)$	Monitor m rejects system p
$\mathit{suffix}(H,\eta)$	Function returning the continuation of all traces in H prefixed with η
$\approx_H$	Monitor τ-equivalence relation
$(\!\vert-\vert\!)$	Function mapping formulae to monitors
$\models_v$	Violation Relation
$\sqsubseteq$	Language Inclusion
$\langle\langle-\rangle\rangle$	Reverse synthesises mapping monitors to formulae

Part II

$\eta,\xi,\ldots \in \textsc{Act}$	External action, internal action, or silent τ action
$\mu,\lambda,\ldots \in \textsc{tAct}$	Traceable action, *i.e.,* external or internal action
$\gamma,\delta,\ldots \in \textsc{eAct}$	Internal action
$\textsc{Det}$	Determinacy predicate
$t,u,\ldots \in \textsc{Trc}$	Trace consisting of external and internal actions
$\mathbf{rej}_{\textsc{Det}}(H,\mathrm{f},m)$	Monitor m rejects history H with flag f
$\mathbf{rej}_{\textsc{Det}}(H,m)$	Monitor m rejects history H with flag true
$\mathbf{rej}_{\textsc{Det}}(p,m)$	Monitor m rejects system p
$N_{\textsc{Det}}(-)$	Monitor normalisation function

Part III

$A,B,\ldots \in \textsc{Actr}$	Actor system
$e,d\ldots \in \textsc{Exp}$	Actor expression
$i,j,h\ldots \in \textsc{Pid}$	Actor names
$a,b,\ldots \in \textsc{Chans}$	Channel names
$a,b,\ldots \in \textsc{Atom}$	Atoms
$u,v,\ldots \in \textsc{Val}$	Values
$\mathbf{aft}(-)$	Interface and Observer Evolution
$\mathit{fIn}(-)$	Function returning free input channel names
$\mathit{fn}(-)$	Function returning free channel names
$\mathit{fv}(-)$	Function returning free variables
$\mathit{fV}(-)$	Function returning free recursion variables

Part IV

$P,P,\ldots \in \textsc{Prc}$	Processes in the π-calculus
$\approx$	Weak bisimilarity
$K \models P \approx Q$	Shorthand for $K \triangleright P \approx K \triangleright Q$

Formulae, Monitors and Systems Index

Throughout Parts I to III, we often refer to the following formulae, monitors and systems.

Formulae

$\varphi_0 \stackrel{\text{def}}{=} [s]\mathsf{ff} \wedge [a]\mathsf{ff} \wedge [c]\mathsf{ff}$

$\varphi_1 \stackrel{\text{def}}{=} [r]\mathsf{ff} \vee [c]\mathsf{ff}$

$\varphi_2 \stackrel{\text{def}}{=} [r]([s]\mathsf{ff} \vee [a]\mathsf{ff})$

$\varphi_3 \stackrel{\text{def}}{=} \big([r]([s]\mathsf{ff} \vee [a]\mathsf{ff})\big) \wedge \big([c]([r]\mathsf{ff} \wedge [s]\mathsf{ff} \wedge [a]\mathsf{ff} \wedge [c]\mathsf{ff})\big)$

$\varphi_4 \stackrel{\text{def}}{=} \max X.\big([r][s]X \wedge ([c]\mathsf{ff} \vee [a]\mathsf{ff})\big)$

$\varphi_5 \stackrel{\text{def}}{=} \max X.[r][s]\big(\max X.[r][s]X \wedge [a]X \wedge \mathsf{tt}\big) \wedge [a]X \wedge ([a]\mathsf{ff} \vee [c]\mathsf{ff})$

$\varphi_6 \stackrel{\text{def}}{=} [r]\big([s]\mathsf{ff} \vee [a]\mathsf{ff}\big) \vee [a]\mathsf{ff}$

$\varphi_7 \stackrel{\text{def}}{=} [r]\big([s]\mathsf{ff} \vee [a]\mathsf{ff}\big) \wedge [s]\mathsf{ff}$

$\varphi_8 \stackrel{\text{def}}{=} \max X.\big([a]\mathsf{ff} \vee ([c]\mathsf{ff} \wedge [r][s]X)\big)$

$\varphi_9 \stackrel{\text{def}}{=} [r]\mathsf{ff} \vee [r][s]\mathsf{ff}$

$\varphi_{10} \stackrel{\text{def}}{=} \max X.\big([r][s]X \wedge [a]X \wedge ([a]\mathsf{ff} \vee [c]\mathsf{ff})\big)$

Monitors

$m_1 \stackrel{\text{def}}{=} \mathrm{rec}X.\big(r.s.X \otimes (a.\mathsf{no} \oplus c.\mathsf{no})\big)$

$m_2 \stackrel{\text{def}}{=} \mathrm{rec}X.\big(r.s.X \otimes a.X \otimes (a.\mathsf{no} \oplus c.\mathsf{no})\big)$

$m_3 \stackrel{\text{def}}{=} r.s.a.\mathsf{no}$

$m_4 \stackrel{\text{def}}{=} r.(s.\mathsf{no} \oplus a.\mathsf{no})$

$m_5 \stackrel{\text{def}}{=} s.\mathsf{no} \otimes a.\mathsf{no} \otimes c.\mathsf{no}$

$m_6 \stackrel{\text{def}}{=} r.s.\big(\mathrm{rec}X.r.s.X \otimes a.X \otimes \mathsf{end}\big)$

$m_7 \stackrel{\text{def}}{=} \mathrm{rec}X.\big(a.\mathsf{no} \oplus (r.s.X \otimes c.\mathsf{no})\big)$

Systems

$p_1 \stackrel{\text{def}}{=} \mathrm{rec}X.\big(r.s.X + (a.X + c.\mathbf{0})\big)$

$p_2 \stackrel{\text{def}}{=} \mathrm{rec}X.\big(r.s.X + (\delta_1.a.X + \delta_2.c.\mathbf{0})\big)$

$p_3 \stackrel{\text{def}}{=} r.(s.\mathbf{0} + a.\mathbf{0}) + r.s.\mathbf{0}$

$p_4 \stackrel{\text{def}}{=} r.s.\mathbf{0} + r.a.\mathbf{0}$

$p_5 \stackrel{\text{def}}{=} r.(s.\mathbf{0} + a.\mathbf{0}) + r.(s.\mathbf{0} + a.\mathbf{0} + a.\mathbf{0})$

$p_6 \stackrel{\text{def}}{=} \delta_1.r.s.\mathbf{0} + \delta_2.r.a.\mathbf{0}$

$p_7 \stackrel{\text{def}}{=} \gamma.r.s.\mathbf{0} + \gamma.r.a.\mathbf{0}$

$p_8 \stackrel{\text{def}}{=} r.(\delta_1.s.\mathbf{0} + \delta_2.a.\mathbf{0})$

$p_9 \stackrel{\text{def}}{=} r.(\gamma.s.\mathbf{0} + \gamma.a.\mathbf{0})$

$p_{10} \stackrel{\text{def}}{=} \gamma.p_8 + \gamma.\mathbf{0}$