Refinement of communication and states in models of embedded systems

Refinement of Communication and States in Models of Embedded Systems

Harsh Beohar

December 2012

IPA Dissertation Series 2013-01
Typeset using LaTeX2e
Printed by University Press Facilities, Eindhoven
Cover design by Paul Verspaget

A catalogue record is available
from the Eindhoven University of Technology Library
ISBN: 978-90-386-3327-5

The work in this thesis has been carried out under the auspices of the research school IPA (Institute for Programming research and Algorithmics).

The author was employed at the Eindhoven University of Technology, supported by the Seventh Research Framework Programme of the European Commission (MULTIFORM project, Grant number: INFSO-ICT-224249).

Refinement of Communication and States in Models of Embedded Systems

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de
Technische Universiteit Eindhoven, op gezag van de
rector magnificus, prof.dr.ir. C.J. van Duijn, voor een
commissie aangewezen door het College voor
Promoties in het openbaar te verdedigen
op dinsdag 22 januari 2013 om 16.00 uur

door

Harsh Beohar

geboren te Jabalpur, India

Dit proefschrift is goedgekeurd door de promotoren:

prof.dr. J.C.M. Baeten
en
prof.dr.ir. J.E. Rooda

Copromotor:
dr.ir. P.J.L. Cuijpers

"The whole of science is nothing more than a refinement of everyday thinking."

Albert Einstein

Summary

Refinement of Communication and States in models of Embedded Systems

This thesis addresses two particular issues related to the design of embedded systems; namely, refinement of communication and refinement of states.

The refinement of communication deals with the issue of implementing a synchronous system in an asynchronous way such that two systems are behaviourally equivalent. As a result, correctness of an asynchronous system can be achieved by establishing correctness on its synchronous version, which is computationally cheaper than analysing the latter. The research objective was to find conditions that ensure the addition of buffers does not modify the behaviour of a given synchronous system. We show that it is possible to obtain better desynchronisability conditions (even for finer equivalence like branching bisimulation) by changing the properties of the communication protocol. This is in contrast with the previous works where the focus was only on restricting the communicating components.

The refinement of states deals with the stepwise development of hybrid systems. Such a concept was absent in the Compositional Interchange Format (CIF), a modelling language for embedded systems based on hybrid automata and some process algebraic operators. The research objective was to develop a compositional operational semantics of CIF with hierarchy (HCIF). We show that by referring only to the transition system of the substructures (not to their syntactic representation), the semantics of HCIF operators is almost unchanged with respect to their CIF versions. Furthermore, a definition to eliminate hierarchy in a HCIF model is presented. As a result, the existing simulation tools and the transformation tools to other timed or hybrid languages can be reused upon the elimination of hierarchy from a HCIF model.

Acknowledgements

I would like to thank prof.dr. Jos Baeten, prof.dr.ir. Koos Rooda, and dr.ir. Pieter Cuijpers for trusting me as a PhD candidate in the MULTIFORM project (Integrated Multi-formalism Tool Support for the Design of Networked Embedded Control Systems). Professor Baeten's various proposals and reviews on my research work has always given me motivation and confidence in continuing my doctoral study. I very much appreciate professor Rooda's efforts for keeping me free from the overheads of MULTIFORM project and for his help in formulating research questions.

If this thesis is written in a readable way, then it is due to painstaking effort taken by my daily supervisor dr.ir. Pieter Cuijpers. Pieter with his friendly spirit has played a key role in the development of my mathematical, writing, and presentation skills over these last four years. I also want to thank Pieter for the Dutch translation of the summary of my thesis. It was a great privilege to work with you and I wish to express my profoundest gratitude to Pieter for sharing his knowledge with me.

I would like to thank the reading committee: prof.dr. Kim Larsen, prof.dr. Ursula Goltz, and prof.dr.ir. Jan Friso Groote. The valuable comments and feedback provided by them has significantly improved the material of this thesis. My sincere gratitude to prof.dr. Manohara Pai M. M. for participating in my defense committee at the very last moment and also for his support during my master's study without whom this PhD campaign would have been impossible.

I would also like to thank dr. Erik de Vink and dr. Ruurd Kuipers for reading the important chapters of this thesis. Apart from reviewing this thesis, Erik was always open to discuss variety of things with me. In particular, his advice on solving problems in a concrete setting led to the introduction of half-duplex buffers, which I realized in the context of Multimover case-study that resulted in better conditions for desynchronisation. I am also grateful to Ruurd for his help in preparing the viva-voce required to defend my thesis and results.

I would also like to thank the members of Formal Methods Group and Formal System Analysis group for their support during this period: Ammar Osaiweran, Bas Luttik, Erik de Vink, Francien Dechesne, Frank Stappers, Hans Zantema, Helle Hansen, Jan Friso Groote, Jeroen Keiren, Jos Baeten, Kees Huizing, Meivan Cheng, Maciej Gazda, Matthias Rafffelsieper, MohammadReza Mousavi, Muhammad Atif, Neda Noroozi, Pieter Cuijpers, Rob Hoogerwoord, Ruurd Kuiper, Simona Orzan, Sjoerd Cranen, Sonja Georgievska, Suzana Andova, Tineke van den Bosch, Wan Fokkink, and Wieger Wesselink.

An important aspect of this thesis was developed with the cooperation of System Engineering Group at the Department of Mechanical Engineering, TU/e. I would like to thank Allan van Hulst, Asia van de Mortel-Fronczak, Bert van Beek, Dennis Hendriks, Damian Nadales, Evgeniy Ivanov, Henk van Rooy, Jasen Markovski, Koos Rooda, Konstantin Starkov, Michel Reniers, Mihaly Petreczky, Ramon Schiffelers, and Rong Su for the pleasant working atmosphere. Especially, I want to thank Allan, Damian, Evgeniy, and Konstantin for making the "Bunker" a friendly place to work in.

I would like to thank all my friends for supporting me during my PhD study. Special thanks goes to Ashwin, Debjyoti “Bax” Bera, Ernest “Dj Takla”, Kiran, Manj(n)u, Mehaal “Chand” Rai, McEnroe “Before” Dsilva, Pallmall, Poojith, Ram “the Hulk”, (S)Uttam, Ujwal “Captain Cook”, and Waqar for the great time in Eindhoven and Amersfoort.

Finally, I would like to thank my parents, my brother Yash, fufaji and bua for their everlasting support, care, and love.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The field of embedded systems is an engineering discipline in which the idea is to create a product by embedding a processing device. Some characteristics of an embedded system are the following (see [58] for a more elaborate list).

- Embedded systems are *hybrid* by nature, i.e., they contain a mix of continuous and discrete dynamics.
- Embedded systems are *reactive*, i.e., they continuously interact with their environment.
- Embedded systems are *application specific*, i.e., only the intended software will be running on the processors.

Due to the increasing influence of safety-critical embedded systems (like a control system in an aeroplane) in our society, establishing correctness of software running on these systems become vital. To this end, various researchers (see [2, 23], and the references therein) have proposed the *model-based engineering* approach for the design of embedded systems. One objective of this approach is to detect and correct errors at an early stage of the system development. The following observation by Boehm and Basili [21] estimates the cost paid in missing the above goal: *finding and fixing a software problem after delivery is often 100 times more expensive than finding and fixing it during the requirement and design phase.*

In the model-based engineering approach, first, the system under design is decomposed into a number of components. Second, for every component an abstract model is constructed in a formal language and rigorous analyses are performed with respect to requirements. Third, this abstract model of a component is refined and analysed until a detailed model of the component is

achieved that sufficiently captures its implementation; this process is known as *refinement*.

Roughly, refinement corresponds to the addition of some unspecified aspect of behaviour in an abstract model. For instance,

- Refinement of *communication*: how to implement the interaction among components of a system?
- Refinement of *states*: how to specify the behaviour of a system (or component) in a hierarchical way?
- Refinement of *time* and *resources* [49]: how to implement timing and resource constraints in a concrete model such that its behaviour can be measured in some metric with respect to the abstract model?

In this thesis, we shall focus on only two kinds of refinement; namely, refinement of communication and refinement of states.

1.1 Refinement of communication

The components of a system are not just stand-alone entities, rather, they interact with each other to perform a certain global task. Message passing [50] is a programming paradigm in which software components send and receive messages either synchronously or asynchronously. In synchronous communication components must be physically coupled, making it possible to execute corresponding send and receive messages simultaneously. Asynchronous communication is used when components are placed physically apart. The corresponding send and receive messages are then decoupled and the messages travel via a buffer from a sender to its recipient.

As stated in [36], an issue associated with synchronous and asynchronous communication is *though synchronous systems are easier to develop and understand than asynchronous systems, the implementation architectures one has to aim at are usually of an asynchronous nature*. One reason for this is that the designer has a freedom (in synchronous communication) of not specifying behaviour for the orphan messages in the model of a receiver. Informally, a message is *orphan* [55] if it cannot be read by a receiver from its input buffer. This raises an obvious research question (Figure 1.1): *how to implement a synchronous system in an asynchronous way?*

We tackle[1] this problem by developing different *desynchronisation* techniques. Desynchronisation, a term coined by Fischer and Janssen [36] in

[1] For the refinement of communication, we restrict ourselves to only a discrete model of an embedded system.

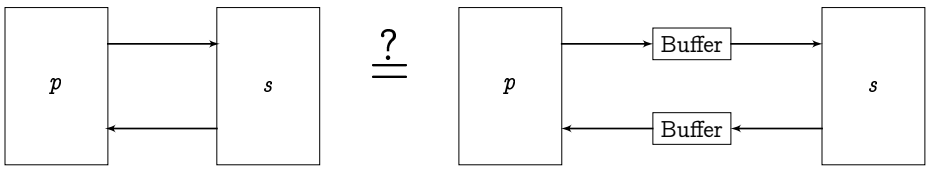


FIGURE 1.1: Refinement of communication.

concurrency theory, is a technique that constructs a correct asynchronous system from a given synchronous system. The motive is to find conditions under which the behaviour of a synchronous system is insensitive to the addition of buffers using an abstraction scheme. An abstraction scheme hides certain details in the asynchronous system that are irrelevant with respect to the synchronous system and ensures that the two systems performs identical events. Formally, the goal is to find conditions that render the synchronous and asynchronous system behaviourally equivalent under a suitable abstraction scheme.

As a result, a *correct* asynchronous system is behaviourally equivalent to the synchronous system and is orphan free. The related works [17, 24, 36, 55, 67] in literature focussed on only one of these correctness criteria, not both. However, in this thesis we will develop desynchronisation techniques that guarantees orphan freedom, whenever the synchronous system and the asynchronous system are behaviourally equivalent.

The methods we use for studying desynchronizability in this thesis stem from process algebra and concurrency theory [6]. We do not fix a set of desirable properties a priori between a synchronous system and its asynchronous version, but rather aim for desynchronizability modulo a behavioral equivalence that preserves a large set of possibly desirable properties. To be as general as possible, we take branching bisimulation as our behavioral equivalence of choice, which is the strongest equivalence used in interleaving concurrency theory in the presence of silent actions (see [76]).

The work on desynchronisation in this thesis spans over Chapters 3-6. A more detailed account of related work is given in the respective chapters.

- In Chapter 3, we develop a desynchronisation technique for the synchronous systems that are synthesised by the supervisory control theory of Ramadge and Wonham [69]. This theory provides an automatic synthesis of a supervisor that *synchronously* controls a plant (a model of hardware) such that the given requirements are satisfied. The inability to synthesise a controller for a plant in the presence of asynchronous communication is one of the drawbacks in applying the supervisory control theory in practice [33].

The main result of this chapter is the partial characterisation of desynchronisation modulo branching bisimulation [77]. In other words, the synthesised supervisor controls the same plant in the presence of buffers such that the synchronous system and its asynchronous version satisfy the same requirements. Furthermore, the result obtained is in contrast with the previous works [13, 80], where synthesis algorithms are designed to compute a new supervisor in the presence of buffers.

- Chapter 4 considers the desynchronisation of a concrete synchronous system. Informally, a synchronous system is *concrete* if it executes only observable events. The objective is to obtain a weaker set of conditions for desynchronisability than the conditions of Chapter 3. To this end, we study the desynchronisation problem in the presence of a half-duplex mechanism, where a component of a system can only send a message, whenever its input buffers are empty. An intuitive example of a half-duplex mechanism is the situation where two individuals communicate over a walkie-talkie.

  The main results of this chapter are the characterisation of desynchronisation modulo branching bisimulation in the presence of a half-duplex mechanism. I.e., we not only show that our conditions guarantee the desynchronisability of a concrete synchronous system; but, also that any desynchronisable concrete synchronous system satisfies our conditions.

  Furthermore, we also give the characterisation of desynchronisation modulo contra-simulation (a weaker equivalence than branching bisimulation, see [78]) because certain non-deterministic choices in a synchronous system can become delayed in the corresponding asynchronous system. Note that the delaying of non-deterministic choices was essentially the point which led to the development of contra-simulation in concurrency theory (see [78]).

- In Chapter 5, we apply the developed desynchronisation techniques to desynchronise a case-study called the pusher-lift system [75].

- Lastly, in Chapter 6 we first address the issue of obtaining an efficient asynchronous implementation of a synchronous system. Second, the desynchronisation of a network of synchronously communicating processes is addressed. Third, the issue of desynchronising a non-concrete synchronous system whose alphabet contains invisible actions is treated.

1.2 Refinement of states

The components of an embedded system are not monolithic entities, rather they are composed from different smaller components. In computer science, the development of hierarchical automata / statecharts [47, 54, 61] has led

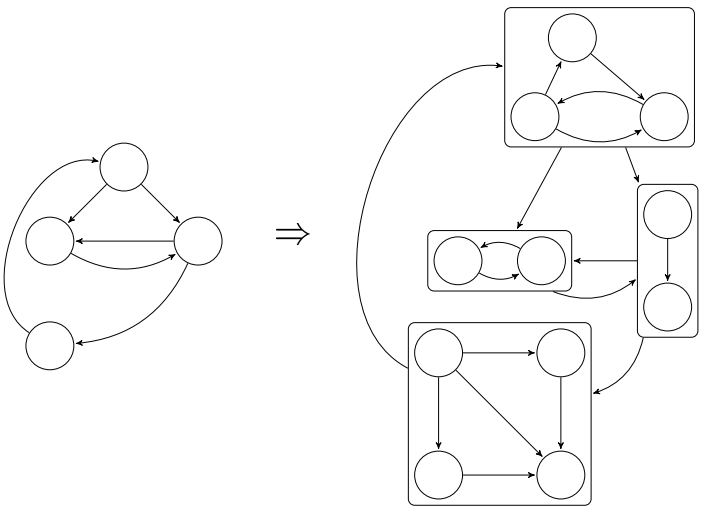


FIGURE 1.2: An illustration of a hierarchical automaton.

to the stepwise development of complex discrete systems (see Figure 1.2 for an illustration of a hierarchical automaton). This is because hierarchical automata allow a gradual, level by level description of system behaviour [47]. Such a concept is absent in the Compositional Interchange Format (CIF) [7].

CIF is a modelling language for embedded systems based on hybrid automata [48] and is designed for two purposes; namely, as a specification language for hybrid systems, and as an interchange format for allowing model transformations between other languages for hybrid systems.

In Chapter 7, we extend the CIF language with the concept of hierarchy, resulting in a new specification language called the Hierarchical Compositional Interchange format (HCIF). A benefit of using hierarchical hybrid automata in the design phase is that it helps the designers to communicate among themselves that are involved in the development of an embedded system. For instance, a control engineer might be interested in only the specification of the continuous behaviour of a component, whereas a software engineer might only be interested in the specification of the discrete behaviour.

The semantics of a hierarchical automaton is defined in a compositional manner, by referring only to the transition system of the substructures and not to their syntactic representation. This compositional introduction of hierarchy allows us to keep the semantics of the HCIF operators almost unchanged with respect to their CIF versions. For this purpose we introduce the automaton postfix operator to define the overall behaviour of a hierarchical automaton, whereas the previous works [27, 54, 61] on the semantics of hierarchical automata requires tree-structure on the set of locations. Consequently, additional concepts from tree-structures, like least common ancestors, children

of a location, etc., complicate the semantics and thus, bringing considerable differences between the semantics of CIF and HCIF [20].

1.3 Origin of the chapters

The contents of Chapter 3 are taken from [17, 19], although the sufficient conditions for desynchronisability when bags are used as buffers are improved. Essentially, the diamond property [17, Definition 11.] is relaxed by allowing diamonds only on certain actions (see Definition 3.9). However, a sub-class of synchronous systems is studied in Chapter 3 in comparison to the synchronous systems in [17]. To be precise, the sets of external actions of both the plant and its supervisor are assumed to be empty because the sufficient conditions obtained are restrictive even without the set of external actions (see Chapter 5 that supports this fact). Furthermore, sufficient conditions for desynchronisability when queues are used as buffers are new.

The characterisation of desynchronisation modulo branching bisimulation in Chapter 4 was published in [18]; although, the characterisation of desynchronisation modulo contra-simulation is not yet published elsewhere. Furthermore, the above assumption on the set of external actions is discarded while developing the desynchronisation techniques in this chapter. The case-study presented in Chapter 5 is adopted from the lecture notes on supervisory control theory [75]. The material in Chapter 6 is also new and is unpublished elsewhere.

Finally, the contents of Chapter 7 originate from [20, 64]. This is a result of joint work with Damian Nadales from the System Engineering group, TU/e. As a result, a version of this chapter can also be found in [63]. Note that certain mathematical notations are changed with respect to [63] to prevent any conflicts with the refinement of communications part of this thesis.

This thesis can be read in the order depicted in Figure 1.3, due to the independent nature of the refinement of communications and refinement of states.

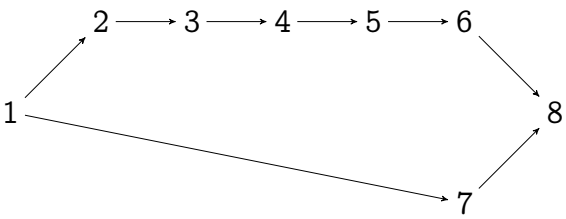


FIGURE 1.3: Outline of the thesis, where the numbers [1-8] represent the numbers of the chapters.

Chapter 2

Preliminaries

This chapter is devoted to establish the mathematical notations, and definitions required for refinement of communication, i.e., for Chapters 3-6.

2.1 Basic definitions

In the sequel, we model the world as a labelled transition system space [6] in which all behaviors of interest are represented. Components of a system as well as their compositions are called *processes* and are represented by pointing out an initial state $p \in \mathbb{P}$ in the labelled transition system space. A process p is then formed by all reachable states from the initial state $p \in \mathbb{P}$.

2.1.1 Labelled transition system

Definition 2.1. A *labelled transition system space* (simply, labelled transition system) is a tuple $(\mathbb{P}, A_\tau, \rightarrow, \sqcup)$, where

- $\mathbb{P}$ is a set of states.
- A is the set of actions. We set $A_\tau = A \uplus \{\tau\}$, where τ is the so called *invisible action*, and let the symbols $\alpha, \alpha', \alpha_1, \cdots$ range over the set A_τ.
- $\rightarrow \subseteq \mathbb{P} \times A_\tau \times \mathbb{P}$ is the transition relation.
- $\sqcup \subseteq \mathbb{P}$ is the *emptiness* predicate over the set of states and its purpose is to observe the states of an asynchronous system that consists of empty buffer contents (see Subsection 2.2.3).

The notation $q \xrightarrow{\alpha} q'$ denotes an element $(q, \alpha, q') \in \rightarrow$ and $q\sqcup$ denotes a state q satisfying the predicate $\sqcup$. Furthermore, we write $q\blacksquare$ in lieu of $\neg(q\sqcup)$. We write $q \xnrightarrow{\alpha}$ as a shorthand for $\nexists q'. \left[q \xrightarrow{\alpha} q'\right]$. A state $q \in \mathbb{P}$ is called *deadlocked*, denoted $q \not\rightarrow$, if $\forall \alpha. \left[q \xnrightarrow{\alpha}\right]$. For a given initial state $q \in \mathbb{P}$, the set of reachable states $\mathfrak{R}(q)$ is defined as the smallest set such that:

$$q \in \mathfrak{R}(q), \text{ and } \forall q_1, q_2, \alpha. \left[q_1 \in \mathfrak{R}(q) \wedge q_1 \xrightarrow{\alpha} q_2 \Rightarrow q_2 \in \mathfrak{R}(q)\right].$$

A process q is *finite* if the sets $\mathfrak{R}(q)$ and $\rightarrow_q$ are finite, where $\rightarrow_q$ is the restriction of $\rightarrow$ to $\mathfrak{R}(q)$. Sometimes, we refer to finite processes as automata.

Definition 2.2 (Concrete and deterministic processes)**.** Let $\text{Alph}(q)$ denote the alphabet of the process q: $\text{Alph}(q) = \left\{\alpha \mid \exists q_1, q_2. \left[q_1 \in \mathfrak{R}(q) \wedge q_1 \xrightarrow{\alpha} q_2\right]\right\}$. A process q is *concrete* iff $\tau \notin \text{Alph}(q)$. Furthermore, a process q is *deterministic* iff $\forall q', q_1, q_2. \left[q' \in \mathfrak{R}(q) \wedge q' \xrightarrow{\alpha} q_1 \wedge q' \xrightarrow{\alpha} q_2 \Rightarrow q_1 = q_2\right]$.

2.1.2 Sequences and multisets

The data structures used in queues and bags are sequences and multisets, respectively. Thus, in this subsection we define sequences and multisets with certain operations over them.

Let A^* denote the set of all *finite sequences* generated from the set of actions with the *empty sequence* denoted by the symbol ϵ. The *concatenation* of any two sequences $w_1, w_2 \in A^*$ is denoted by $w_1.w_2$. Let $\pi : A^* \rightarrow A^*$ be a *permutation* function that returns a permutation of a sequence. We write $w_1 =_\pi w_2$ to denote that $w_2 \in A^*$ is a *permutation* of the sequence $w_1 \in A^*$ such that $w_2 = \pi(w_1)$. A projection of a sequence $w \in A^*$ onto a set $B \subseteq A$, denoted $w \uparrow B$, is a sequence inductively defined as:

$$\epsilon \uparrow B = \epsilon, \text{ and } (\alpha.w) \uparrow B = \begin{cases} \alpha.(w \uparrow B), & \text{if } \alpha \in B \\ w \uparrow B, & \text{if } \alpha \notin B \end{cases}.$$

A *multiset* ξ over the set of actions A is a function $\xi : A \rightarrow \mathbb{N}$ that returns the corresponding multiplicity of the elements. We write the *empty multiset* as $\varepsilon : A \rightarrow 0$, where $\forall \alpha \in A.[\varepsilon(\alpha) = 0]$. We write $A^\star$ to denote the set of all multisets generated from the set A.

Definition 2.3. Let $\xi : A \rightarrow \mathbb{N}$ be a multiset over the set A.

- With abuse of notation, we use the predicate $\in$ to denote an element that *belongs* to a multiset. It is defined as $\alpha \in \xi = \xi(\alpha) > 0$.

- The operator $\oplus$ denotes an *addition* of an element to a multiset. It is defined as $(\xi \oplus \alpha)(\alpha) = \xi(\alpha) + 1$ and $(\xi \oplus \alpha)(\alpha') = \xi(\alpha')$, for every $\alpha' \neq \alpha$.
- The operator $\ominus$ denotes a *removal* of an element from a multiset. It is defined as $(\xi \ominus \alpha)(\alpha) = \xi(\alpha) - 1$, if $\xi(\alpha) > 0$, $(\xi \ominus \alpha)(\alpha) = \xi(\alpha)$, if $\xi(\alpha) = 0$, and $(\xi \ominus \alpha)(\alpha') = \xi(\alpha')$ for every $\alpha' \neq \alpha$.

We define $\mathbf{S}(\xi)$ to denote the set of all sequences generated from a multiset ξ. For example, the set of all sequences generated from the multiset $\xi = \varepsilon \oplus a \oplus b$ is $\mathbf{S}(\xi) = \{a.b, b.a\}$.

Proposition 2.4. *Let $\xi \in A^*$ and $w_1 \in \mathbf{S}(\xi)$. If for some permutation π and $w_1 =_\pi w_2$, then $w_2 \in \mathbf{S}(\xi)$.*

We now define the *reachability relation* $\longrightarrow\!\!\!\!\!\rightarrow \subseteq \mathbb{P} \times A^* \times \mathbb{P}$ in the following recursive way. The definitions of the behavioural equivalence relations used in this thesis are based on this reachability relation.

$$q_1 \longrightarrow\!\!\!\!\!\rightarrow q_1 \ , \ \frac{q_1 \overset{w}{\longrightarrow\!\!\!\!\!\rightarrow} q' ,\ q' \overset{\tau}{\rightarrow} q_2}{q_1 \overset{w}{\longrightarrow\!\!\!\!\!\rightarrow} q_2} \ , \ \frac{q_1 \overset{w}{\longrightarrow\!\!\!\!\!\rightarrow} q' ,\ q' \overset{\alpha}{\rightarrow} q_2 ,\ \alpha \neq \tau}{q_1 \overset{w.\alpha}{\longrightarrow\!\!\!\!\!\rightarrow} q_2} \ .$$

2.1.3 Behavioural equivalence relations

In principle, a synchronous interaction can be simulated asynchronously by hiding and renaming certain interactions between the local components and the buffers (see Subsection 2.2.2). As a result, weak equivalences, which treat τ transitions as unobservable [76] can be used to compare a synchronous system and its asynchronous version. In this thesis, we concentrate on two such equivalences; namely, branching bisimulation [77], and contra-simulation [78].

Branching bisimulation [77] is a weak equivalence which abstracts away from the invisible transitions while preserving the branching structure of the processes. An advantage of using branching bisimulation equivalence is that it enables us to preserve most of the modal requirements between a synchronous system and an asynchronous system [28].

Definition 2.5. A binary relation $\mathcal{B} \subseteq \mathbb{P} \times \mathbb{P}$ is a *branching bisimulation* relation [77] iff the following transfer conditions are satisfied.

1. $\forall q, q_1, q', \alpha \in A_\tau . \Big[(q, q') \in \mathcal{B} \wedge q \overset{\alpha}{\rightarrow} q_1 \Rightarrow (\alpha = \tau \wedge (q_1, q') \in \mathcal{B}) \vee$
$\exists q_1', q_2' . \Big[q' \longrightarrow\!\!\!\!\!\rightarrow q_1' \overset{\alpha}{\rightarrow} q_2' \wedge (q, q_1') \in \mathcal{B} \wedge (q_1, q_2') \in \mathcal{B}\Big]\Big]$,
2. $\forall q, q', q_1', \alpha \in A_\tau . \Big[(q, q') \in \mathcal{B} \wedge q' \overset{\alpha}{\rightarrow} q_1' \Rightarrow (\alpha = \tau \wedge (q, q_1') \in \mathcal{B}) \vee$

$$\exists q_1, q_2.\Big[q \longrightarrow\!\!\!\!\!\rightarrow\ q_1 \xrightarrow{\alpha} q_2 \wedge (q_1, q') \in \mathcal{B} \wedge (q_2, q_1') \in \mathcal{B}\Big]\Big].$$

Furthermore, a relation $\mathcal{B} \subseteq \mathbb{P} \times \mathbb{P}$ is an $\sqcup$-*sensitive* (pronounced: emptiness sensitive) *branching bisimulation* relation iff $\mathcal{B}$ is a branching bisimulation relation satisfying the following transfer conditions:

3. $\forall q, q'.\Big[(q, q') \in \mathcal{B} \wedge q\sqcup \Rightarrow \exists q_1'.\Big[q' \longrightarrow\!\!\!\!\!\rightarrow\ q_1' \wedge q_1' \sqcup \wedge (q, q_1') \in \mathcal{B}\Big]\Big]$,
4. $\forall q, q'.\Big[(q, q') \in \mathcal{B} \wedge q'\sqcup \Rightarrow \exists q_1.\Big[q \longrightarrow\!\!\!\!\!\rightarrow\ q_1 \wedge q_1 \sqcup \wedge (q_1, q') \in \mathcal{B}\Big]\Big]$.

Two processes q and q' are ($\sqcup$-*sensitive*) *branching bisimilar*, denoted ($q \overset{\sqcup}{\leftrightarrow}_{\mathbf{b}} q'$) $q \leftrightarrow_{\mathbf{b}} q'$, if there is a (an $\sqcup$-sensitive) branching bisimulation relation $\mathcal{B}$ such that $(q, q') \in \mathcal{B}$.

Proposition 2.6. *Let q_1, q_2 be any two arbitrary processes such that $q_1 \overset{\sqcup}{\leftrightarrow}_{\mathbf{b}} q_2$. If $q_1 \overset{w}{\longrightarrow\!\!\!\!\!\rightarrow}\ q_3$ for some $w \in A^*$, then $\exists q_4.\Big[q_2 \overset{w}{\longrightarrow\!\!\!\!\!\rightarrow}\ q_4 \wedge q_3 \overset{\sqcup}{\leftrightarrow}_{\mathbf{b}} q_4\Big]$.*

Proposition 2.7. *Let q_1 be an arbitrary process and let q_2 be a concrete process. If $q_1 \overset{\sqcup}{\leftrightarrow}_{\mathbf{b}} q_2$, $q_1 \xrightarrow{\alpha} q_1'$, and $\alpha \neq \tau$, then $\exists q_2'.\Big[q_2 \xrightarrow{\alpha} q_2' \wedge q_1' \overset{\sqcup}{\leftrightarrow}_{\mathbf{b}} q_2'\Big]$.*

Definition 2.8. A transition $q \xrightarrow{\alpha} q'$ is *inert* [44] modulo $\leftrightarrow_{\mathbf{b}}$ ($\overset{\sqcup}{\leftrightarrow}_{\mathbf{b}}$), if $q \leftrightarrow_{\mathbf{b}} q'$ ($q \overset{\sqcup}{\leftrightarrow}_{\mathbf{b}} q'$).

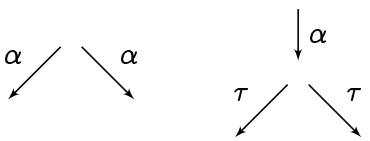


FIGURE 2.1: Delaying of choice, where $\alpha \neq \tau$.

In certain situations, branching bisimulation equivalence is hard to establish due to its demanding transfer conditions. In particular, an external non-deterministic choice (see left in Figure 2.1) in a synchronous system can be delayed (see right in Figure 2.1) while constructing an asynchronous system[1]. This phenomenon in concurrency theory is known as a *delaying of choice* [78]. For this purpose, contra-simulation equivalence [76, 78] was designed to be insensitive with the delaying of an external choice into an internal choice. This can be interpreted via the so called contra-simulation axiom $a.x + a.y = a.(\tau.x + \tau.y)$ (see [78] for more details).

Definition 2.9. A binary relation $\mathcal{C} \subseteq \mathbb{P} \times \mathbb{P}$ is a *contra-simulation* relation [78] iff the following transfer condition is satisfied.

$$\forall q_1, q_2, q_1', w \in A^*.\Big[(q_1, q_1') \in \mathcal{C} \wedge q_1 \overset{w}{\longrightarrow\!\!\!\!\!\rightarrow}\ q_2 \Rightarrow \exists q_2'.\Big[q_1' \overset{w}{\longrightarrow\!\!\!\!\!\rightarrow}\ q_2' \wedge (q_2', q_2) \in \mathcal{C}\Big]\Big].$$

[1] A concrete example is given in Chapter 4 (see Example 4.1).

Furthermore, a relation $\mathcal{C} \subseteq \mathbb{P} \times \mathbb{P}$ is a $\sqcup$-*sensitive contra-simulation* relation iff $\mathcal{C}$ is a contra-simulation relation satisfying the following transfer condition:

$$\forall q_1, q_1'.\ [(q_1, q_1') \in \mathcal{C} \wedge q_1 \sqcup \Rightarrow \exists q_2'.\ [q_1' \longrightarrow\!\!\!\!\!\rightarrow q_2' \wedge q_2' \sqcup \wedge (q_2', q_1) \in \mathcal{C}]]\ .$$

A process q_1 *contra-simulates* q_2, denoted $q_1 \preccurlyeq q_2$, if there exists a contra-simulation relation $\mathcal{C}$ such that $(q_1, q_2) \in \mathcal{C}$. Furthermore, two processes q_1, q_2 are *contra-similar*, denoted $q_1 \sim_{\mathbf{c}} q_2$, iff $q_1 \preccurlyeq q_2$ and $q_2 \preccurlyeq q_1$. It is assumed that the preorder $\preccurlyeq$ and the equivalence $\sim_{\mathbf{c}}$ are extended to $\preccurlyeq^{\sqcup}$ and $\sim_{\mathbf{c}}^{\sqcup}$, respectively, in the usual way.

Proposition 2.10. *Let q_1 be an arbitrary process and q_2 be a process such that $q_1 \preccurlyeq^{\sqcup} q_2$ and $q_2 \stackrel{\tau}{\not\rightarrow}$. Then, $q_2 \preccurlyeq^{\sqcup} q_1$.*

Proof. By instantiating Definition 2.9 we have

$$q_1 \preccurlyeq^{\sqcup} q_2 \wedge q_1 \longrightarrow\!\!\!\!\!\rightarrow q_1 \Rightarrow \exists q_2'.\ [q_2 \longrightarrow\!\!\!\!\!\rightarrow q_2' \wedge q_2' \preccurlyeq^{\sqcup} q_1]\ .$$

But, $q_2 \stackrel{\tau}{\not\rightarrow}$; thus $q_2' = q_2$. Similarly, instantiating the transfer condition of $\sqcup$ we have $q_1 \preccurlyeq^{\sqcup} q_2 \wedge q_1 \sqcup \Rightarrow \exists q_2'.\ [q_2 \longrightarrow\!\!\!\!\!\rightarrow q_2' \wedge q_2' \sqcup \wedge q_2' \preccurlyeq^{\sqcup} q_1]$. Since $q_2 \stackrel{\tau}{\not\rightarrow}$, we have $q_2' = q_2$. Hence, $q_2 \preccurlyeq^{\sqcup} q_1$. □

Henceforth, we let the symbols $\simeq$, $\simeq^{\sqcup}$ range over the sets $\{\leftrightarrows_{\mathbf{b}}, \sim_{\mathbf{c}}\}$, and $\{\leftrightarrows_{\mathbf{b}}^{\sqcup}, \sim_{\mathbf{c}}^{\sqcup}\}$, respectively.

2.2 From synchrony to asynchrony

The goal of this section is to give Structural Operational Semantics (SOS) [68] to both synchronous and asynchronous systems so that the equivalence problem can be studied on the induced labelled transition systems. Given two processes p, s we achieve this goal in three steps. First, we recall the semantics of the synchronous merge operator, the renaming operator, and the abstraction operator from the TCP process algebra [6]. The synchronous merge operator is used to model the given synchronous system, while the renaming and the abstraction operators are used in the construction of an asynchronous system. Second, we construct a buffered system by placing two buffers between the given processes p, s. Finally, we define four abstraction schemes on a buffered system using the renaming operator and the abstraction operator.

Consider a synchronous system as depicted in Figure 2.2. We identify two basic components p, s, which we assume to be processes in our labeled transition system. These processes are composed into a synchronous process $p \parallel s$ (commonly known as a synchronous system). The process $p \parallel s$ can perform four kinds of events; namely, the external actions of p and s that belong to

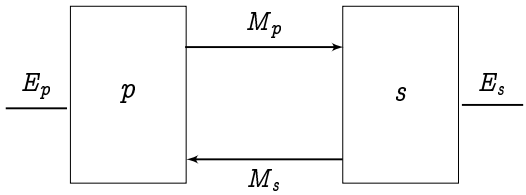


FIGURE 2.2: A synchronous system.

the sets E_p and E_s, respectively, and messages from p and s that belong to the sets M_p and M_s, respectively.

Furthermore, we make the distinction between the sending of a message (modeled for p by the set $!M_p = \{!m | m \in M_p\}$) and the receiving of that message (modeled for p by the set $?M_p = \{?m | m \in M_p\}$). Note that such a distinction is crucial when constructing an asynchronous system. We assume that the so obtained sets are all part of our alphabet and are all pairwise disjoint, i.e.,

$$E_p \uplus E_s \uplus M_p \uplus M_s \uplus !M_p \uplus !M_s \uplus ?M_p \uplus ?M_s \subseteq A.$$

Assuming that the processes p and s are already part of our labeled transition system, where p makes use of the actions $!M_p \uplus ?M_p \uplus E_p$ and s makes use of the actions $!M_s \uplus ?M_s \uplus E_s$, we can define the synchronous composition of p and s through SOS rules on the states of the transition system.

Formally, the *synchronous merge* is a binary operator $\|: \mathbb{P} \times \mathbb{P} \to \mathbb{P}$ that denotes the synchronous execution of two processes. The semantic rules for the synchronous merge operator are given in Table 2.1. The premise of each rule states the assumption on the states of the composed processes, and the conclusion gives the resulting transition for the composed state.

The *renaming* operator [6] is a unary operator $\rho_f : \mathbb{P} \to \mathbb{P}$ parameterised by the so-called *renaming function* $f : A \to A$ that maps the set of actions to itself. The *abstraction* operator [6] is also a unary operator $\tau_I : \mathbb{P} \to \mathbb{P}$ parameterised by a set of actions $I \subseteq A$. Intuitively, the abstraction operator renames all actions in the set I to the invisible action τ, whenever $\alpha \in I$. The SOS rules for the renaming operator and the abstraction operator are given in Table 2.1.

It should be noted that the branching bisimulation and the contra-simulation relations are congruence relations for the synchronous merge operator, the renaming operator, and the abstraction operator (see [6] for branching bisimulation, and [78] for contra-simulation). Formally, an equivalence relation $\simeq$ is a *congruence* relation with respect to a k-ary operator Θ, if for all processes q_i, q_i' (for $i \in [1, k]$) such that $q_i \simeq q_i'$, then

$$\Theta(q_1, \cdots, q_k) \simeq \Theta(q_1', \cdots, q_k').$$

$$\frac{p_1 \xrightarrow{!m} p_2,\ s_1 \xrightarrow{?m} s_2}{p_1 \parallel s_1 \xrightarrow{m} p_2 \parallel s_2}\ (1) \qquad \frac{p_1 \xrightarrow{?n} p_2,\ s_1 \xrightarrow{!n} s_2}{p_1 \parallel s_1 \xrightarrow{n} p_2 \parallel s_2}\ (2)$$

$$\frac{p_1 \xrightarrow{\alpha} p_2,\ \alpha \in E_p \cup \{\tau\}}{p_1 \parallel s_1 \xrightarrow{\alpha} p_2 \parallel s_1}\ (3) \qquad \frac{s_1 \xrightarrow{\alpha} s_2,\ \alpha \in E_s \cup \{\tau\}}{p_1 \parallel s_1 \xrightarrow{\alpha} p_1 \parallel s_2}\ (4)$$

$$\frac{q \xrightarrow{\alpha} q',\ \alpha \notin I}{\tau_I(q) \xrightarrow{\alpha} \tau_I(q')}\ (5) \qquad \frac{q \xrightarrow{\alpha} q',\ \alpha \in I}{\tau_I(q) \xrightarrow{\tau} \tau_I(q')}\ (6) \qquad \frac{q \xrightarrow{\tau} q'}{\tau_I(q) \xrightarrow{\tau} \tau_I(q')}\ (7)$$

$$\frac{q \xrightarrow{\alpha} q'}{\rho_f(q) \xrightarrow{f(\alpha)} \rho_f(q')}\ (8) \qquad \frac{q \xrightarrow{\tau} q'}{\rho_f(q) \xrightarrow{\tau} \rho_f(q')}\ (9)$$

TABLE 2.1: SOS rules for $\parallel$, $\rho_f()$, and $\tau_I()$ operators.

Let $\square = \square_p \uplus \square_s$, where $\square \in \{M, E, !M, ?M\}$. With abuse of notations, we define an *input* projection function $? : (M \cup E)^* \to ?M^*$, an *output* projection function $! : (M \cup E)^* \to (!M \cup E)^*$, and a *message* projection function $\bar{} : (M \cup E)^* \to M^*$ in the following way.

1. $?\epsilon = \epsilon$, $?(e.w) = w$, $?(m.w) = ?m.?w$, where $e \in E$ and $w \in (M \cup E)^*$.
2. $!\epsilon = \epsilon$, $!(e.w) = e.!w$, $!(m.w) = !m.!w$, where $e \in E$ and $w \in (M \cup E)^*$.
3. $\bar{\epsilon} = \epsilon$, $\overline{e.w} = \bar{w}$, and $\overline{m.w} = m.\bar{w}$, where $e \in E$ and $w \in (M \cup E)^*$.

Proposition 2.11. *Let $p \parallel s$ be a synchronous system.*

1. *If $u \in (M_s \cup E_s)^*$, $p_1 \parallel s_1 \xrightarrow{u} p_2 \parallel s_2$ then $p_1 \xrightarrow{?u} p_2 \wedge s_1 \xrightarrow{!u} s_2$.*
2. *If $v \in (M_p \cup E_p)^*$, $p_1 \parallel s_1 \xrightarrow{v} p_2 \parallel s_2$ then $s_1 \xrightarrow{?v} s_2 \wedge p_1 \xrightarrow{!v} p_2$.*

Proof. Straightforward by induction on u and v. □

2.2.1 Buffered systems

Next, we turn our attention to the construction of a *buffered system*, which is the result of placing two buffers between the processes p, s of a given synchronous system $p \parallel s$. Unlike, in the case of a synchronous system $p \parallel s$, the processes p, s in a buffered system interact with each other by reading and

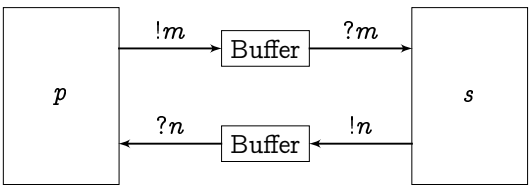


FIGURE 2.3: A buffered system.

writing messages in their input and output buffers, respectively. Intuitively, a buffered system performs the send $!m$ action, whenever a sender (either, p, or s) sends the message $!m$ to its output buffer. Similarly, a buffered system performs the receive $?m$ action, whenever a receiver reads the message $?m$ from its input buffer (see Figure 2.3).

The final ingredient required to give formal semantics of a buffered system is the kind of data-structure used to store the messages. In this thesis, we consider two kinds of lossless and unbounded buffers: queues and bags. To formally denote a buffered system with queues, we introduce a family of operators called *queue-asynchronous merge*, notation $\_\,|[\mu,\nu]|\,\_$, for every $\mu \in {M_s}^*, \nu \in {M_p}^*$. Similarly, a buffered system with bags is written as $p\,|\{\xi,\zeta\}|\,s$, where $\_\,|\{\xi,\zeta\}|\,\_$ is a family of operators called *bag-asynchronous merge*, for every $\xi \in M_s^\star, \zeta \in M_p^\star$. Henceforth, the contents of the input buffer attached to the processes p and s are denoted by $\mu \in {M_s}^*$ ($\xi \in M_s^\star$) and $\nu \in {M_p}^*$ ($\zeta \in M_p^\star$), respectively. The semantic rules of bag-asynchronous merge and queue-asynchronous merge are presented in Table 2.2.

2.2.2 Abstraction schemes: which ones to hide?

With the construction of a buffered system, we have already made a step towards asynchrony from synchrony. A problem with defining equivalence between a synchronous system and its buffered version is that *asynchronous composition needs two actions for the communication of a message while synchronous composition only needs one.* Furthermore, any two equivalent systems should at-least have the same alphabet. The usual process algebraic way to solve this issue is by defining an abstraction scheme, translating certain actions from the asynchronous system to actions from the synchronous system while hiding the remaining ones. In principle, there are four abstraction schemes as shown in Figure 2.4:

$\mathbf{A}_1$. hiding the interactions between the process p and the buffers, and renaming the interactions between the process s and the buffers to the messages from the process s. (see Figure 2.4(a)). The expression of the form $\alpha_1 \to \alpha_2$ (for $\alpha_1, \alpha_2 \in A_\tau$) in Figure 2.4(a) denotes the renaming of the action α_1 to α_2.

$$\frac{p \xrightarrow{!m} p'}{\begin{array}{l}(p\,|[\mu,\nu]|\,s) \xrightarrow{!m} (p'\,|[\mu,\nu.m]|\,s)\\ (p\,|\{\xi,\zeta\}|\,s) \xrightarrow{!m} (p'\,|\{\xi,\zeta \oplus m\}|\,s)\end{array}} \quad (10) \qquad \frac{s \xrightarrow{!n} s'}{\begin{array}{l}(p\,|[\mu,\nu]|\,s) \xrightarrow{!n} (p\,|[\mu.n,\nu]|\,s')\\ (p\,|\{\xi,\zeta\}|\,s) \xrightarrow{!n} (p\,|\{\xi \oplus n,\zeta\}|\,s')\end{array}} \quad (11)$$

$$\frac{p \xrightarrow{?n} p',\ \mu = n.\mu'}{(p\,|[\mu,\nu]|\,s) \xrightarrow{?n} (p'\,|[\mu',\nu]|\,s)} \quad (12) \qquad \frac{p \xrightarrow{?n} p',\ \xi(n) > 0}{(p\,|\{\xi,\zeta\}|\,s) \xrightarrow{?n} (p'\,|\{\xi \ominus n,\nu\}|\,s)} \quad (13)$$

$$\frac{s \xrightarrow{?m} s',\ \nu = m.\nu'}{(p\,|[\mu,\nu]|\,s) \xrightarrow{?m} (p\,|[\mu,\nu']|\,s')} \quad (14) \qquad \frac{s \xrightarrow{?m} s',\ \zeta(m) > 0}{(p\,|\{\xi,\zeta\}|\,s) \xrightarrow{?m} (p\,|\{\xi,\zeta \ominus m\}|\,s')} \quad (15)$$

$$\frac{p \xrightarrow{\alpha} p',\ \alpha \in E_p \cup \{\tau\}}{\begin{array}{l}(p\,|[\mu,\nu]|\,s) \xrightarrow{\alpha} (p'\,|[\mu,\nu]|\,s)\\ (p\,|\{\xi,\zeta\}|\,s) \xrightarrow{\alpha} (p'\,|\{\xi,\zeta\}|\,s)\end{array}} \quad (16) \qquad \frac{s \xrightarrow{\alpha} s',\ \alpha \in E_s \cup \{\tau\}}{\begin{array}{l}(p\,|[\mu,\nu]|\,s) \xrightarrow{\alpha} (p\,|[\mu,\nu]|\,s')\\ (p\,|\{\xi,\zeta\}|\,s) \xrightarrow{\alpha} (p\,|\{\xi,\zeta\}|\,s')\end{array}} \quad (17)$$

TABLE 2.2: SOS rules for buffered systems.

$\mathbf{A}_2$. hiding the interactions between the process s and the buffers, and renaming the interactions between the process p and the buffers to the messages from p. (see Figure 2.4(b)).

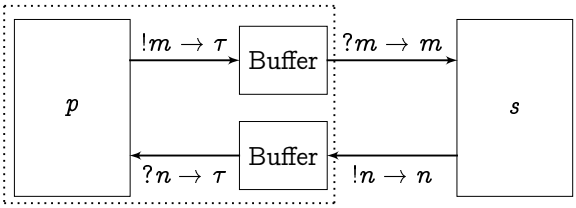
$\mathbf{A}_3$. hiding the interactions between the processes p, s and their input buffers, and renaming the interactions between the processes p, s and their output buffers to the messages from p, s. (see Figure 2.4(c)).

$\mathbf{A}_4$. hiding the interactions between the processes p, s and their output buffers, and renaming the interactions between the processes p, s and their input buffers to the messages from p, s. (see Figure 2.4(d)).

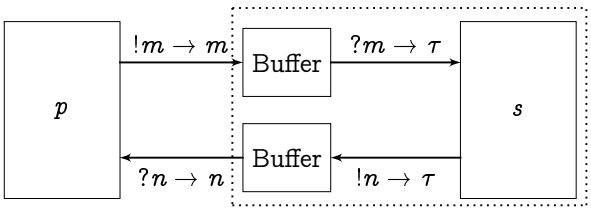
To formally define the aforementioned abstraction schemes, we first define the sets H_i for the abstraction schemes $\mathbf{A}_i$ in the following way, for every $i \in [1, 4]$. Intuitively, the set H_1 contains the send actions and the receive actions of the process p, which are the result of the interactions between the process p and the two buffers in a buffered system (recall the rules from Table 2.2). Similarly, explanations can be given for the sets H_2, H_3, and H_4.

$$H_1 = !M_p \uplus ?M_p \qquad H_2 = !M_s \uplus ?M_s$$
$$H_3 = ?M_p \uplus ?M_s \qquad H_4 = !M_p \uplus !M_s.$$

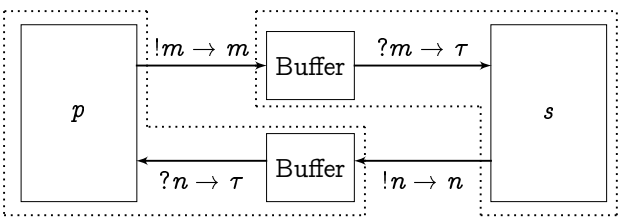
Next, define the renaming functions $\eta_i : A \to A$ for the abstraction schemes $\mathbf{A}_i$ in the following way, for every $i \in [1, 4]$. Intuitively, the function η_1



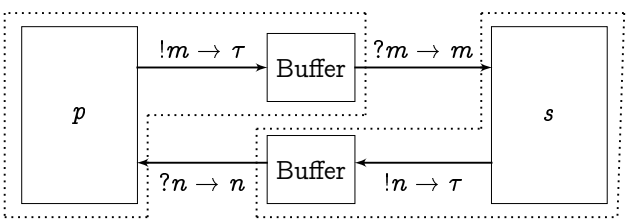
(a) Abstraction scheme $\mathbf{A}_1$.



(b) Abstraction scheme $\mathbf{A}_2$.



(c) Abstraction scheme $\mathbf{A}_3$.



(d) Abstraction scheme $\mathbf{A}_4$.

FIGURE 2.4: Abstraction schemes.

renames the send actions and the receive actions to the messages, which are the result of the interactions between the process s and the two buffers. Similarly, explanations can be given for the functions η_2, η_3, and η_4.

$$\eta_1(\alpha) = \begin{cases} m, & \alpha = ?m,\ ?m \in ?M_s, \\ n, & \alpha = !n,\ !n \in !M_s. \end{cases} \quad \eta_2(\alpha) = \begin{cases} n, & \alpha = ?n,\ ?n \in ?M_p, \\ m, & \alpha = !m,\ !m \in ?M_p. \end{cases}$$

$$\eta_3(\alpha) = \begin{cases} m, & \alpha = !m,\ !m \in !M_p, \\ n, & \alpha = !n,\ !n \in !M_s. \end{cases} \quad \eta_4(\alpha) = \begin{cases} n, & \alpha = ?n,\ ?n \in ?M_p, \\ m, & \alpha = ?m,\ ?m \in ?M_s. \end{cases}$$

For brevity, write $\tau_{H_i}(\rho_{\eta_i}(q))$ as $\nabla_i(q)$, for some $q \in \mathbb{P}$, and $i \in [1,4]$ is the identifier of an abstraction scheme. Thus, from a given synchronous system $p \parallel s$, the following variants of asynchronous systems are constructed: $\nabla_i(p \,|[\epsilon,\epsilon]|\, s)$, $\nabla_i(p \,|\{\varepsilon,\varepsilon\}|\, s)$, for $i \in [1,4]$. The equivalence problems studied

Chapter	Equivalence	Buffers
3.	$p \parallel s \leftrightarrows_{\mathbf{b}} \nabla_1(p \,\|[\epsilon,\epsilon]|\, s)$ $p \parallel s \leftrightarrows_{\mathbf{b}} \nabla_1(p \,\|\{\varepsilon,\varepsilon\}|\, s)$	Queues and bags.
4.	$p \parallel s \leftrightarrows^{\sqcup}_{\mathbf{b}} \nabla_3(p \,\|[\epsilon,\epsilon]|\, s)$ $p \parallel s \sim^{\sqcup}_{\mathbf{c}} \nabla_3(p \,\|[\epsilon,\epsilon]|\, s)$	Queues

TABLE 2.3: Equivalence problems studied in Chapters 3 and 4.

in Chapters 3, 4 are stated formally in Table 2.3. The motivation of choosing a particular abstraction scheme can be found in the respective chapters. In the next subsection, we clarify the role of $\sqcup$-sensitive versions of branching bisimulation and contra-simulation in this thesis.

2.2.3 The emptiness predicate $\sqcup$

Recall from Chapter 1 that orphan freedom is the other correctness criterion which we envisage on an asynchronous system, apart from being equivalent to the given synchronous system. The orphan freedom property is similar to the unspecified reception from the field of Communicating Finite State Machines (CFSM) (see [24, 42]), which is formulated only for queues because queues are the preferred choice of buffers in CFSM. Informally, a state $\nabla_i(p \,|[\mu,\nu]|\, s)$ (for $i \in [1,4]$) in an asynchronous system have *unspecified reception* if the states p, s has no input transitions of the form $p \xrightarrow{?n} p'$ and $s \xrightarrow{?m} s'$, whenever $\mu = n.\mu'$ and $\nu = m.\nu'$, respectively. In this thesis, we shall focus on a more general property than unspecified reception, called orphan freedom.

Definition 2.12. An asynchronous system $\nabla_i(p \,|[\epsilon,\epsilon]|\, s)$ with queues as buffers, is *orphan free* iff for every reachable states $\nabla_i(p_1 \,|[\mu,\nu]|\, s_1) \in \mathfrak{R}(\nabla_i(p \,|[\epsilon,\epsilon]|\, s))$ the following condition holds: $\exists p_2, s_2, w.\ \left[\nabla_i(p_1 \,|[\mu,\nu]|\, s_1) \overset{w}{\twoheadrightarrow} \nabla_i(p_2 \,|[\epsilon,\epsilon]|\, s_2)\right]$. Similarly, an asynchronous system $\nabla_i(p \,|\{\varepsilon,\varepsilon\}|\, s)$ with bags as buffers, is *orphan free* iff for every reachable states $\nabla_i(p_1 \,|\{\xi,\zeta\}|\, s_1) \in \mathfrak{R}(\nabla_i(p \,|\{\xi,\zeta\}|\, s))$ the following condition holds: $\exists p_2, s_2, w.\ \left[\nabla_i(p_1 \,|\{\xi,\zeta\}|\, s_1) \overset{w}{\twoheadrightarrow} \nabla_i(p_2 \,|\{\varepsilon,\varepsilon\}|\, s_2)\right]$.

An advantage when a synchronous system and its asynchronous version are equivalent modulo $\simeq^{\sqcup}$ is that it ensures orphan freedom in the asynchronous system. Essentially, this property is ensured by the transfer conditions in $\sqcup$-sensitive versions of the branching bisimulation and the contra-simulation relations (see Theorem 2.13). To establish this fact, we define the semantic rules for the emptiness predicate $\sqcup$ in Table 2.4.

$$\frac{}{p \parallel s\sqcup}\ (18) \qquad \frac{}{\begin{array}{l} p\,|[\epsilon,\epsilon]|\ s\sqcup \\ p\,|\{\epsilon,\epsilon\}|\ s\sqcup \end{array}}\ (19) \qquad \frac{q\sqcup}{\rho_f(q)\sqcup}\ (20) \qquad \frac{q\sqcup}{\tau_I(q)\sqcup}\ (21)$$

TABLE 2.4: SOS rules for the emptiness predicate.

Theorem 2.13. *Let* $i \in \{1,2,3,4\}$.

1. *If* $p \parallel s \simeq^{\sqcup} \nabla_i(p\,|[\epsilon,\epsilon]|\ s)$, *then the asynchronous system* $\nabla_i(p\,|[\epsilon,\epsilon]|\ s)$ *is orphan free.*
2. *If* $p \parallel s \simeq^{\sqcup} \nabla_i(p\,|\{\epsilon,\epsilon\}|\ s)$, *then the asynchronous system* $\nabla_i(p\,|\{\epsilon,\epsilon\}|\ s)$ *is orphan free.*

Proof. We give the proof of only Item 1; Item 2 can be proved in similar way. We show that if $p \parallel s \simeq^{\sqcup} \nabla_i(p\,|[\epsilon,\epsilon]|\ s)$, then

$$\forall p_1, s_1, \mu, \nu.\Big[\nabla_i(p_1\,|[\mu,\nu]|\ s_1) \in \mathfrak{R}(\nabla_i(p\,|[\epsilon,\epsilon]|\ s)) \Rightarrow \exists p_2, s_2.[\nabla_i(p_1\,|[\mu,\nu]|\ s_1) \longrightarrow\!\!\!\!\!\rightarrow \nabla_i(p_2\,|[\epsilon,\epsilon]|\ s_2)]\Big].$$

If $\mu, \nu = \epsilon$, then the result holds trivially. So suppose, otherwise $\mu \neq \epsilon \vee \nu \neq \epsilon$.

1. Abstraction scheme $\mathbf{A}_1$. It is given that $p \parallel s \simeq^{\sqcup} \nabla_1(p\,|[\epsilon,\epsilon]|\ s)$. So we get the following two cases:
   (a) When $\simeq^{\sqcup} = \underline{\leftrightarrow}^{\sqcup}_{\mathbf{b}}$. Since $\nabla_1(p_1\,|[\mu,\nu]|\ s_1) \in \mathfrak{R}(\nabla_1(p\,|[\epsilon,\epsilon]|\ s))$, there exists w such that $\nabla_1(p\,|[\epsilon,\epsilon]|\ s) \overset{w}{\longrightarrow\!\!\!\!\!\rightarrow} \nabla_1(p_1\,|[\mu,\nu]|\ s_1)$. From Proposition 2.6 we have $\exists q.\,[q \in \mathfrak{R}(p \parallel s \wedge q \underline{\leftrightarrow}^{\sqcup}_{\mathbf{b}} \nabla_1(p_1\,|[\mu,\nu]|\ s_1))]$. From the transfer conditions of branching bisimulation we get $q\sqcup \Rightarrow \exists p_2, s_2.[\nabla_1(p_1\,|[\mu,\nu]|\ s_1) \longrightarrow\!\!\!\!\!\rightarrow \nabla_1(p_2\,|[\epsilon,\epsilon]|\ s_2)]$. But, from the semantics of the abstraction scheme $\mathbf{A}_1$ we know that reading the messages by the process p_1 from its input queue is observable. Thus, there cannot be trace with a non-empty sequence of τ's from the state $\nabla_1(p_1\,|[\mu,\nu]|\ s_1)$ such that the content μ is completely read. Hence, this case is inapplicable.
   (b) When $\simeq^{\sqcup} = \sim^{\sqcup}_{\mathbf{c}}$. Similar to the previous case.
2. Abstraction scheme $\mathbf{A}_2$. Similar to the previous case.
3. Abstraction scheme $\mathbf{A}_3$. It is given that $p \parallel s \simeq^{\sqcup} \nabla_3(p\,|[\epsilon,\epsilon]|\ s)$. So we get the following two cases:

(a) When $\simeq^{\sqcup} = \leftrightarrows^{\sqcup}_{\mathbf{b}}$. Since $\nabla_3(p_1 \,|[\mu,\nu]|\, s_1) \in \mathfrak{R}(\nabla_3(p \,|[\epsilon,\epsilon]|\, s))$, there exists w such that $\nabla_3(p \,|[\epsilon,\epsilon]|\, s) \xrightarrow{w}\!\!\!\!\rightarrow \nabla_3(p_1 \,|[\mu,\nu]|\, s_1)$. From Proposition 2.6 we have $\exists q.\Big[q \in \mathfrak{R}(p \parallel s) \wedge q \leftrightarrows^{\sqcup}_{\mathbf{b}} \nabla_3(p_1 \,|[\mu,\nu]|\, s_1)\Big]$. From the transfer conditions of branching bisimulation, we get

$$q\sqcup \Rightarrow \exists p_2, s_2.\Big[\nabla_3(p_1 \,|[\mu,\nu]|\, s_1) \longrightarrow\!\!\!\!\rightarrow \nabla_3(p_2 \,|[\epsilon,\epsilon]|\, s_2)\Big].$$

(b) When $\simeq^{\sqcup} = \sim^{\sqcup}_{\mathbf{c}}$. Since $\nabla_3(p_1 \,|[\mu,\nu]|\, s_1) \in \mathfrak{R}(\nabla_3(p \,|[\epsilon,\epsilon]|\, s))$, there exists w such that $\nabla_3(p \,|[\epsilon,\epsilon]|\, s) \xrightarrow{w}\!\!\!\!\rightarrow \nabla_3(p_1 \,|[\mu,\nu]|\, s_1)$. But, $p \parallel s \sim^{\sqcup}_{\mathbf{c}} \nabla_3(p \,|[\epsilon,\epsilon]|\, s)$. Using the transfer condition of contra-simulation we get $\exists q.\Big[p \parallel s \xrightarrow{w}\!\!\!\!\rightarrow q \wedge q \preccurlyeq \nabla_3(p_1 \,|[\mu,\nu]|\, s_1)\Big]$. Again, using the transfer condition of contra-simulation we have

$$q\sqcup \Rightarrow \exists p_2, s_2.\Big[\nabla_3(p_1 \,|[\mu,\nu]|\, s_1) \longrightarrow\!\!\!\!\rightarrow \nabla_3(p_2 \,|[\epsilon,\epsilon]|\, s_2)\Big].$$

4. Abstraction scheme $\mathbf{A}_4$. Similar to the Case 1. □

Even though, Theorem 2.13 seems to solve the issue of preserving orphan freedom; however, this solution is an unsatisfactory one for the asynchronous systems that are constructed with the abstraction schemes $\mathbf{A}_1, \mathbf{A}_2$, and $\mathbf{A}_4$. The reason is that a state with non-empty buffer contents in an asynchronous system can never be related to a state in a synchronous system by the $\sqcup$-sensitive versions of the branching bisimulation or the contra-simulation relations, whenever the abstraction schemes $\mathbf{A}_1, \mathbf{A}_2$, and $\mathbf{A}_4$ are used. The following lemma states this fact formally.

Lemma 2.14. *Let* $i \in \{1, 2, 4\}$.

1. *If* $p_1 \parallel s_1 \simeq^{\sqcup} \nabla_i(p_2 \,|[\mu,\nu]|\, s_2)$, *then* $\mu = \epsilon$, $\nu = \epsilon$.
2. *If* $p_1 \parallel s_1 \simeq^{\sqcup} \nabla_i(p_2 \,|\{\xi,\zeta\}|\, s_2)$, *then* $\xi = \varepsilon$, $\zeta = \varepsilon$.

Proof. By following the reasoning in Case 1(a) of Theorem 2.13 we know that $\mu, \nu = \epsilon$ in the context of the abstraction scheme $\mathbf{A}_1$. Likewise, the other cases can be proved. □

In hindsight, the abstraction scheme $\mathbf{A}_3$ allows an elegant and a reasonable way to establish the orphan freedom for an asynchronous system in comparison with the other abstraction schemes. This is one of the advantages of using the abstraction scheme $\mathbf{A}_3$ over the other abstraction schemes. In Chapter 4, we will revisit this comparison of the abstraction schemes with the aim to find weaker conditions (than those in Chapter 3) for a synchronous system and an asynchronous system to be equivalent.

Chapter 3

Desynchronising a plant and its supervisor

Recall (from Chapter 1) that our objective is to desynchronise a given synchronous system, i.e., to establish that the synchronous system and its asynchronous version are equivalent under suitable conditions. Considering that the choice of buffers and the abstraction schemes contribute to varieties of asynchronous systems, this makes the above objective a challenging one. In principle, we would like to select a buffer and an abstraction scheme which result in weaker conditions (than any other combination of buffer and abstraction schemes) for the equivalence to hold between a synchronous system and its asynchronous version.

So we take a first step towards this goal by studying the equivalence problem for the following setup. The synchronous systems are synthesised using supervisory control theory, both queues and bags are used as buffers, and the abstraction scheme $\mathbf{A}_1$ is used for the construction of asynchronous systems.

The synchronous systems of interests are synthesised by supervisory control theory of Ramadge and Wonham [69]. In particular, these systems do not contain external actions in their alphabets and are deterministic by definition (cf. [69]). These restrictions essentially simplify the required conditions for desynchronisability. Later, in Chapter 4, we study the effects of having external actions and non-determinism in the context of desynchronisation.

In spite of the above theoretical motive, there is also a practical reason to study desynchronisation in supervisory control theory. In particular, the task of implementing a supervisory controller is non-trivial, even though, there are different theories [52, 65, 69, 79] that allow an automatic synthesis of these controllers in the form of automata. One of the reasons for this discord is the

asynchronous interaction between a plant and its supervisor in implementations, whereas the existing supervisory control theories assume synchronous interaction. In supervisory control theory literature, this issue is known as the inexact synchronisation problem and is considered to be a major obstacle in applying supervisory control theory in practice (see [33]).

Finally, throughout this chapter, the abstraction scheme $\mathbf{A}_1$ is used in the construction of asynchronous systems. The rationale behind the choice of $\mathbf{A}_1$ is that a supervisor s and the synchronous system $p \parallel s$ are isomorphic in the monolithic supervisory control theory [69], modulo the difference in the type of action labels. This is because in the synthesis of supervisors no transitions are introduced that a plant cannot execute. As a consequence, the supervisor model remains unaffected in the abstraction scheme $\mathbf{A}_1$; while, in the other abstraction schemes ($\mathbf{A}_2$, $\mathbf{A}_3$ and $\mathbf{A}_4$) this fact does not hold.

3.1 Introduction to supervisory control theory

Supervisory control theory of Ramadge and Wonham [69] provides the automatic synthesis of a supervisor that controls a plant in such a way that a legal behaviour is achieved. In supervisory control theory terminology,

- the model of an uncontrolled hardware is known as the *plant*,
- the model that specifies the legal behaviour is known as the *requirement*,
- the model that forces the plant to meet the requirement by interacting with it is known as the *supervisor*.
- the synchronous composition of a plant and its supervisor is known as *closed-loop system* or *supervised plant* [57]. However, in this thesis, we prefer the terminology synchronous system than closed-loop system.

The basic entities in Ramadge and Wonham's supervisory control theory (plant, requirement, and supervisor) are modelled as *deterministic automata*. The alphabets of these automata are partitioned into two disjoint subsets: *controllable* actions and *uncontrollable* actions. The idea behind this partition is that the supervisor can enable, or disable controllable actions, but it cannot disable the uncontrollable actions of a plant. In this thesis, we follow the input-output interpretation [8, 13] between a plant and its supervisor, wherein *the uncontrollable actions are the output messages from a plant to a supervisor and the controllable actions are the output messages from a supervisor to a plant*. In other words, the sets of uncontrollable actions and controllable actions are $!M_p$ and $?M_p$, respectively.

Definition 3.1. A *plant* is a concrete and deterministic process $p \in \mathbb{P}$ such that $E_p = \emptyset$. Similarly, a *supervisor* is a concrete and deterministic process $s \in \mathbb{P}$ such that $E_s = \emptyset$.

The conditions $E_p = \emptyset$ and $E_s = \emptyset$ ensure that a plant and its supervisor do not contain external actions. This is because in the theory of Ramadge and Wonham a plant model contains only either controllable, or uncontrollable actions. Furthermore, the synthesis procedure ensures that a supervisor restricts the behaviour of the plant by synchronising with some of the controllable actions and no extra transitions are introduced in a supervisor that a plant cannot execute.

A requirement is a process specifying the legal interaction that must occur while the plant and its supervisor are interacting such that a required task (for which the supervisor is synthesised) is completed.

Definition 3.2. A *requirement* is a concrete and deterministic process $r \in \mathbb{P}$ such that $\mathrm{Alph}(r) \subseteq M$.

Throughout this chapter, we use the letters p to denote a plant, s to denote a supervisor, and r to denote a requirement.

Now, we can state the *control problem* as follows: given a plant p and a requirement r, find a supervisor $s \in \mathbb{P}$ such that

$$p \parallel s \leftrightarrows_{\mathbf{b}} r.$$

In this chapter, we are not interested in how this supervisor is computed and rather assume that we are provided with a solution to the above equation. Note that in supervisory control literature the control problem is usually based on language equivalence, but branching bisimilarity coincides with language equivalence in the presence of determinism and in the absence of τ actions [31]. We use branching bisimulation as the equality in the above equation because desynchronisation may introduce non-determinism through the abstraction of certain interactions in the asynchronous system. Branching bisimulation then ensures that the moments of choice that are specified in the requirement r are also retained by the synchronous system $p \parallel s$.

3.2 Desynchronisation using queues

In this section, we study the equivalence problem between a synchronous system and its asynchronous version, where queues are used as buffers. As already mentioned, the abstraction scheme $\mathbf{A}_1$ is used in the construction of an asynchronous system from the given synthesised synchronous system.

Definition 3.3. A synchronous system $p \parallel s$ is *desynchronisable with respect to the abstraction scheme* $\mathbf{A}_1$ *and queues*, if $p \parallel s \leftrightarrows_{\mathbf{b}} \nabla_1(p \,|[\epsilon, \epsilon]|\, s)$.

Throughout this chapter, by a *synthesised synchronous system* we mean a synchronous system synthesised using the supervisory control theory of Ramadge and Wonham [69].

Proposition 3.4. *If $p \parallel s$ is a synthesised synchronous system, then $p \parallel s$ is a concrete and deterministic process.*

3.2.1 Conditions of desynchronisability

In this section, we first present three conditions on a synchronous system. Later, we show that these three conditions together form a sufficient condition for desynchronisation with respect to the abstraction scheme $\mathbf{A}_1$ and queues.

Definition 3.5. A binary relation $\mathcal{W} \subseteq \mathbb{P} \times \mathbb{P}$ is called a *well-posedness* relation iff the following conditions are satisfied.

1. $\forall p_1, s_1, p_2, m. \Big[p_1 \xrightarrow{!m} p_2 \wedge (p_1, s_1) \in \mathcal{W} \Rightarrow \exists s_2. \Big[s_1 \xrightarrow{?m} s_2\Big] \wedge \forall s_2. \Big[s_1 \xrightarrow{?m} s_2 \Rightarrow (p_2, s_2) \in \mathcal{W}\Big]\Big]$,
2. $\forall p_1, s_1, p_2, e \in E_p. \Big[p_1 \xrightarrow{e} p_2 \wedge (p_1, s_1) \in \mathcal{W} \Rightarrow (p_2, s_1) \in \mathcal{W}\Big]$,
3. $\forall p_1, s_1, s_2, m. \Big[s_1 \xrightarrow{!m} s_2 \wedge (p_1, s_1) \in \mathcal{W} \Rightarrow \exists p_2. \Big[p_1 \xrightarrow{?m} p_2\Big] \wedge \forall p_2. \Big[p_1 \xrightarrow{?m} p_2 \Rightarrow (p_2, s_2) \in \mathcal{W}\Big]\Big]$,
4. $\forall p_1, s_1, s_2, e \in E_s. \Big[s_1 \xrightarrow{e} s_2 \wedge (p_1, s_1) \in \mathcal{W} \Rightarrow (p_1, s_2) \in \mathcal{W}\Big]$.

Two processes p, s are said to be *well-posed* if there exists a well-posedness relation $\mathcal{W}$ such that $(p, s) \in \mathcal{W}$.

Condition 1 and 3 assert not only that there exists a matching receive message in the receiver, but every matching receive message in the receiver will lead to a well-posed state. Condition 2 and 4 asserts that the execution of an external action in p and s, respectively, will lead to a well-posed state. Note that Condition 1 can be simplified in the following way under the assumption of deterministic processes.

$$\forall p_1, s_1, p_2, m. \Big[p_1 \xrightarrow{!m} p_2 \wedge (p_1, s_1) \in \mathcal{W} \Rightarrow \exists s_2. \Big[s_1 \xrightarrow{?m} s_2 \wedge (p_2, s_2) \in \mathcal{W}\Big]\Big]. \tag{3.1}$$

Likewise, Condition 3 can be simplified under the assumption of deterministic processes. In Definition 3.5, we defined well-posedness in a general way to handle non-deterministic processes in Chapter 4.

Intuitively, if a sender and its receiver are not well-posed, then in a synchronous system the send messages will be blocked. However, in an asynchronous system, such send messages will lead to *orphans*, i.e., messages that remain forever in the buffer. In turn, orphans leads to deadlocking communication (except in some pathological cases).

Proposition 3.6. *Let $p \parallel s$ be a concrete and a well-posed synchronous system with $\mathcal{W}$ being the witnessing well-posedness relation, i.e., $(p, s) \in \mathcal{W}$. Then, $\forall p_1, s_1. [p_1 \parallel s_1 \in \mathfrak{R}(p \parallel s) \Rightarrow (p_1, s_1) \in \mathcal{W}]$.*

Lemma 3.7 (Generalised well-posedness)**.** *Let $p \parallel s$ be a well-posed and concrete synchronous system. Let $u \in (M_s \cup E_s)^*$ and $v \in (M_p \cup E_p)^*$.*

1. *If $p_1 \parallel s_1 \in \mathfrak{R}(p \parallel s)$ and $s_1 \xrightarrow{!u} \!\!\!\!\rightarrow\, s_2$, then $\exists p_2. \left[p_1 \parallel s_1 \xrightarrow{u} \!\!\!\!\rightarrow\, p_2 \parallel s_2\right]$.*
2. *If $p_1 \parallel s_1 \in \mathfrak{R}(p \parallel s)$ and $p_1 \xrightarrow{!v} \!\!\!\!\rightarrow\, p_2$, then $\exists s_2. \left[p_1 \parallel s_1 \xrightarrow{v} \!\!\!\!\rightarrow\, p_2 \parallel s_2\right]$.*

Proof. Straightforward from the induction on u (v) and application of well-posedness definition. □

The next condition which we would like to discuss is called *X-singular*, where $X \subseteq A$. Informally, a process q is X-singular if every reachable state executes a unique element from the set X. Note that the following definition does not exclude non-deterministic choices at a state.

Definition 3.8. Let $X \subseteq A$. A process $q \in \mathbb{P}$ is *X-singular* if every reachable state $q_1 \in \mathfrak{R}(q)$ satisfies the following condition:

$$\forall q_2, q_3, \alpha_1, \alpha_2 \in X. \left[q_1 \xrightarrow{\alpha_1} q_2 \wedge q_1 \xrightarrow{\alpha_2} q_3 \Rightarrow \alpha_1 = \alpha_2\right].$$

Example 3.1. *Consider the following transition systems of p_1 and s_1:*

$$(\{p_1, p_2, p_3\}, \{p_1 \xrightarrow{!m_1} p_2, p_1 \xrightarrow{!m_2} p_3\}), \text{ and}$$
$$(\{s_1, s_2, s_3\}, \{s_1 \xrightarrow{?m_1} s_2, s_1 \xrightarrow{?m_2} s_3\}).$$

The transition system of the synchronous system $p_1 \parallel s_1$ and the asynchronous system $\nabla_1(p_1 \,|[\epsilon, \epsilon]|\, s_1)$ are shown in Figure 3.1. An attempt to construct a branching bisimulation relation between the two systems will fail to relate the states $p_1 \parallel s_1$ and $\nabla_1(p_2 \,|[\epsilon, m_1]|\, s_1)$. This is because the state $\nabla_1(p_2 \,|[\epsilon, m_1]|\, s_1)$ cannot simulate the transition $p_1 \parallel s_1 \xrightarrow{m_2} p_3 \parallel s_3$. Thus, the deterministic choice of messages from the plant-side is

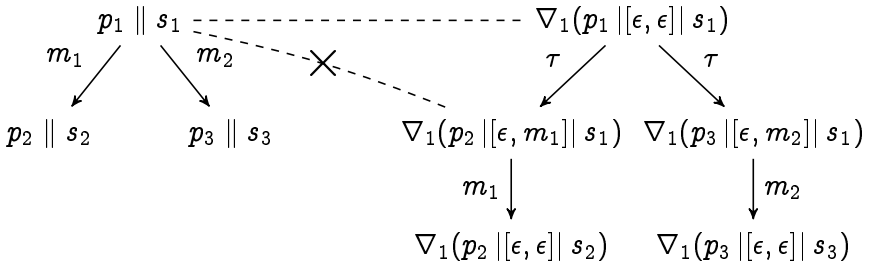


FIGURE 3.1: Role of the condition C_p-singular.

transformed into an internal non-deterministic choice by the abstraction scheme $\mathbf{A}_1$ *that cannot be resolved.*

In contrast, suppose $p_1 \parallel s_1 \leftrightarrows_{\mathbf{b}} \nabla_1(p_2 \,|[\epsilon, m_1]|\, s_1)$. *Then, from the transfer conditions of branching bisimulation we have there exists* q_1, q_2 *such that*

$$\nabla_1(p_2 \,|[\epsilon, m_1]|\, s_1) \longrightarrow\!\!\!\!\!\rightarrow\ q_1 \xrightarrow{m_2} q_2 \wedge q_1 \leftrightarrows_{\mathbf{b}} p_1 \parallel s_1 \wedge q_2 \leftrightarrows_{\mathbf{b}} p_3 \parallel s_3.$$

Since, concrete processes p, s *are used to construct the asynchronous system we have* $q_1 = \nabla_1(p_4 \,|[\epsilon, m_1.m_2]|\, s_2)$, *for some* $p_4 \in \mathbb{P}$ *such that*

$$\nabla_1(p_2 \,|[\epsilon, m_1]|\, s_1) \xrightarrow{\tau} \nabla_1(p_4 \,|[\epsilon, m_1.m_2]|\, s_2).$$

Notice that there cannot exists a process q_2 *with* $\nabla_1(p_4 \,|[\epsilon, m_1.m_2]|\, s_2) \xrightarrow{m_2} q_2$ *because queues are used as buffers and the message* m_2 *can only be read by* s_2 *after it has read the message* m_1. *Thus, this suggests that the condition* M_p*-singular is also a candidate for the necessary conditions of desynchronisation modulo* $\leftrightarrows_{\mathbf{b}}$ *when queues are used.*

The final condition which we would like to discuss is (X, Y)-*diamond* property (commonly known as diamond property), where $X, Y \subseteq A$. Informally, Definition 3.9 asserts that the execution of two distinct actions $\alpha \in X$ and $\beta \in Y$ from a state in a synchronous system cannot disable the execution of each other; furthermore, the traces $\alpha.\beta$ and $\beta.\alpha$ commute.

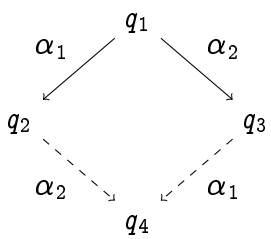


FIGURE 3.2: An illustration of Definition 3.9.

Definition 3.9. Let $X, Y \subseteq A$ be any two subsets of the set of actions. A process $q \in \mathbb{P}$ satisfies the (X, Y)-*diamond* property if every reachable state $q_1 \in \mathfrak{R}(q)$ satisfies the following condition (see Figure 3.2):

$$\forall q_2, q_3, \alpha_1 \in X, \alpha_2 \in Y. \Big[q_1 \xrightarrow{\alpha_1} q_2 \wedge q_1 \xrightarrow{\alpha_2} q_3 \wedge \alpha_1 \neq \alpha_2 \Rightarrow \exists q_4. \Big[q_2 \xrightarrow{\alpha_2} q_4 \wedge q_3 \xrightarrow{\alpha_1} q_4 \Big] \Big] .$$

Example 3.2. *Consider the following transition systems of the processes* p_1, s_1*:* $(\{p_1, p_2, p_3\}, \{p_1 \xrightarrow{!m} p_2, p_1 \xrightarrow{?n} p_3\})$ *and* $(\{s_1, s_2, s_3\}, \{s_1 \xrightarrow{?m} s_2, s_1 \xrightarrow{!n} s_3\})$*. The transition systems of the synchronous system* $p_1 \parallel s_1$ *and the asynchronous system* $\nabla_1(p_1 \,|[\epsilon, \epsilon]|\, s_1)$ *are shown as solid lines in Figure 3.3. Observe that if we do not consider the dashed transitions in Figure 3.3 the two transition systems are already branching bisimilar; however, the asynchronous system is not orphan free.*

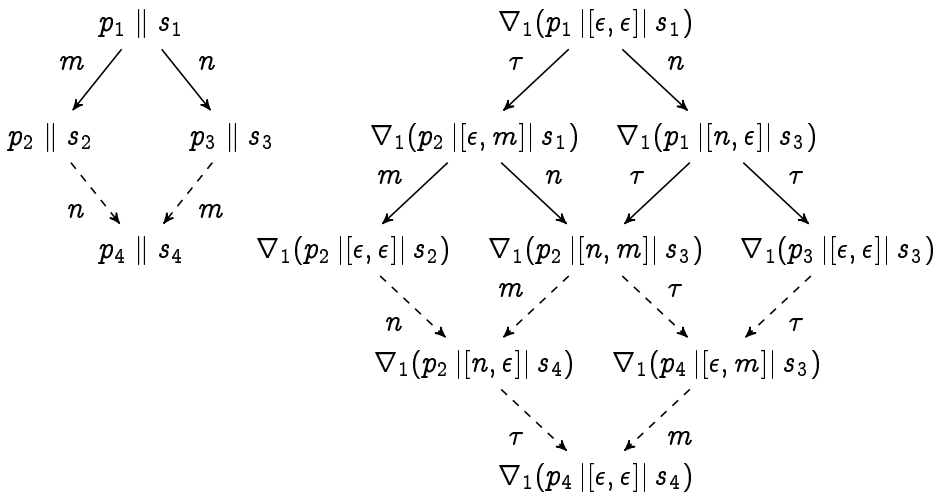


FIGURE 3.3: Illustration of diamond property.

In particular, the state $\nabla_1(p_2 \,|[n, m]|\, s_3)$ *contains non-empty queue contents and there is no way to empty these contents based on the given transition systems of* p_1 *and* s_1*. To this end, the* (M_p, M_s)*-diamond property ensures that there exists the transitions* $p_3 \parallel s_3 \xrightarrow{m} p_4 \parallel s_4$ *(for some* $p_4, s_4 \in \mathbb{P}$*) and* $p_2 \parallel s_2 \xrightarrow{n} p_4 \parallel s_4$ *in the synchronous system. As a result, the asynchronous system is orphan free and all the invisible transitions in it are inert modulo* $\leftrightarrows_{\mathbf{b}}$ *(see Figure 3.3).*

Corollary 3.10 (Generalised (X, Y) diamond property). *Let* $X, Y \subseteq A$ *and suppose* $w_1 \in X^*, w_2 \in Y^*$ *such that* $w_1 \cap w_2 = \emptyset$[1]*. Suppose a concrete*

[1]It is assumed that the sequences are flattened into sets before applying the operation $\cap$.

and deterministic process $q \in \mathbb{P}$ *satisfies the* (X, Y)*-diamond property. If* $q_1 \in \mathfrak{R}(q)$, $q_1 \xrightarrow{w_1} \!\!\!\!\rightarrow\, q_2$, *and* $q_1 \xrightarrow{w_2} \!\!\!\!\rightarrow\, q_3$, *then* $\exists q_4.\, \left[q_2 \xrightarrow{w_2} \!\!\!\!\rightarrow\, q_4 \wedge q_3 \xrightarrow{w_1} \!\!\!\!\rightarrow\, q_4\right]$.

Proof. By induction on w_1, w_2, and application of Definition 3.9. □

We are now ready to prove the main result of this section.

Theorem 3.11. *Let* $p \parallel s$ *be a synthesised synchronous system such that* p, s *are well-posed. If* $p \parallel s$ *is* M_p*-singular and satisfies the* (M_p, M_s)*-diamond property, then* $p \parallel s \leftrightarrows_{\mathbf{b}} \nabla_1(p \,|[\epsilon, \epsilon]|\, s)$.

Proof. Define a relation $\mathcal{B}$ in the following way:

$$\begin{aligned}\mathcal{B} = \Big\{ (p_1 \parallel s_1, \nabla_1(p_2 \,|[\mu, \nu]|\, s_1)) \;\Big|\; & p_1 \parallel s_1 \in \mathfrak{R}(p \parallel s) \wedge \\ & \nabla_1(p_2 \,|[\mu, \nu]|\, s_1) \in \mathfrak{R}(\nabla_1(p \,|[\epsilon, \epsilon]|\, s)) \wedge \\ & \exists p_2', s_2', p_2, s_2.\, \left[p_2' \parallel s_2' \xrightarrow{\mu} \!\!\!\!\rightarrow\, p_1 \parallel s_1 \wedge p_2' \parallel s_2' \xrightarrow{\nu} \!\!\!\!\rightarrow\, p_2 \parallel s_2\right] \Big\}.\end{aligned}$$

The construction of $\mathcal{B}$ is based on the following idea. A reachable state $p_1 \parallel s_1 \in \mathfrak{R}(p \parallel s)$ in the synchronous system is related to those states in the asynchronous system $\nabla_1(p \,|[\epsilon, \epsilon]|\, s)$ that contain the same supervisor state s. Next, we need to show that the relation $\mathcal{B}$ is a branching bisimulation relation, which can be found in Appendix A as Theorem A.1. □

It is interesting to note that the above definition of the relation $\mathcal{B}$ is independent of the size of the queues attached to the processes p, s. Thus, the preconditions in Theorem 3.11 are also sufficient for desynchronisability modulo branching bisimulation, even if queues of finite size with back-pressure are used to construct an asynchronous system with the abstraction scheme $\mathbf{A}_1$.

Theorem 3.12 (Orphan freedom). *Suppose a synthesised synchronous system* $p \parallel s$ *satisfies the preconditions of Theorem 3.11. Then, the asynchronous system* $\nabla_1(p \,|[\epsilon, \epsilon]|\, s)$ *is orphan free.*

Proof. We need to show that for every reachable state $\nabla_1(p_1 \,|[\mu, \nu]|\, s_1)$ in the asynchronous system $\nabla_1(p \,|[\epsilon, \epsilon]|\, s)$ the following holds:

$$\exists w, p_2, s_2.\, \left[\nabla_1(p_1 \,|[\mu, \nu]|\, s_1) \xrightarrow{w} \!\!\!\!\rightarrow\, \nabla_1(p_2 \,|[\epsilon, \epsilon]|\, s_2)\right].$$

So pick a reachable state $\nabla_1(p_1 \,|[\mu, \nu]|\, s_1) \in \mathfrak{R}(\nabla_1(p \,|[\epsilon, \epsilon]|\, s))$ in the asynchronous system. Recall the definition of the relation $\mathcal{B}$ from Theorem 3.11. We showed that if the conditions of this theorem hold then such a relation $\mathcal{B}$

exists. Thus, by definition of $\mathcal{B}$ we have $\exists p_1'.\,[(p_1' \parallel s_1, \nabla_1(p_1 \,|[\mu,\nu]|\, s_1)) \in \mathcal{B}]$. Furthermore, we have (construction of $\mathcal{B}$):

$$\exists p_2', s_2', s_2.\left[p_2' \parallel s_2' \xrightarrow{\mu} p_1' \parallel s_1 \wedge p_2' \parallel s_2' \xrightarrow{\nu} p_1 \parallel s_2\right].$$

Next, applying Corollary 3.10 we get

$$\exists p_3, s_3.\left[p_1' \parallel s_1 \xrightarrow{\nu} p_3 \parallel s_3 \wedge p_1 \parallel s_2 \xrightarrow{\mu} p_3 \parallel s_3\right].$$

Thus, we get $s_1 \xrightarrow{?\nu} s_3$ and $p_1 \xrightarrow{?\mu} p_3$ (see Proposition 2.11 for definitions of the sequences $?\nu, ?\mu$). Using these transitions we get the desired result:

$$\nabla_1(p_1 \,|[\mu,\nu]|\, s_1) \longrightarrow \nabla_1(p_3 \,|[\epsilon,\nu]|\, s_1) \xrightarrow{\nu} \nabla_1(p_3 \,|[\epsilon,\epsilon]|\, s_3). \qquad \square$$

3.3 Desynchronisation using bags

In this section, we find the conditions that ensure branching bisimulation equivalence between the given synchronous system $p \parallel s$ and the asynchronous system $\nabla_1(p \,|\{\varepsilon,\varepsilon\}|\, s)$, where bags are used as buffers.

Definition 3.13. A synchronous system $p \parallel s$ is *desynchronisable with respect to the abstraction scheme* $\mathbf{A}_1$ *and bags*, if $p \parallel s \leftrightarrows_{\mathbf{b}} \nabla_1(p \,|\{\varepsilon,\varepsilon\}|\, s)$.

3.3.1 Condition of desynchronisability

We begin by discarding the condition M_p-singular as a first change in the sufficient condition for desynchronisability when substituting queues by bags as buffers. Recall the synchronous system in Example 3.1, where the condition M_p-singular forbids the deterministic choices between the messages of the process p_1. This was required because the supervisor's state s_1 in $\nabla_1(p_2 \,|[\epsilon, m_1]|\, s_1)$ (see Figure 3.1) cannot read the message m_2 before reading the message m_1 due to the FIFO order maintained by the queues, whenever $p_2 \xrightarrow{!m_2} p_4$ and $s_1 \xrightarrow{?m_2} s_4$, for some $p_4, s_4 \in \mathbb{P}$. Thus, disallowing the state $\nabla_1(p_2 \,|[\epsilon, m_1]|\, s_1)$ to simulate the transition $p_1 \parallel s_1 \xrightarrow{m_2} p_2 \parallel s_2$ that violates one of the transfer conditions of the branching bisimulation relation. Note that the above issue disappears when bags are used to desynchronise a synchronous system because the messages are stored in an arbitrary order.

Example 3.3. *Consider the synchronous system $p_1 \parallel s_1$ of Example 3.1 and its asynchronous version $\nabla_1(p_1 \,|\{\varepsilon,\varepsilon\}|\, s_1)$ with bags as buffers. The transition system of the asynchronous system $\nabla_1(p_1 \,|\{\varepsilon,\varepsilon\}|\, s_1)$ is shown as solid lines in Figure 3.4. To relate the states $p_1 \parallel s_1$ and $\nabla_1(p_2 \,|\{\varepsilon, m_1\}|\, s_1)$*

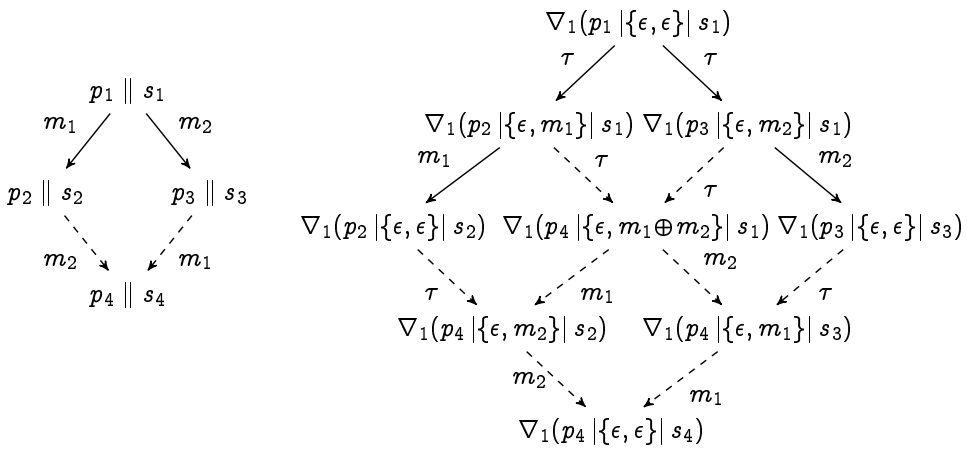


FIGURE 3.4: Example 3.4 in the new asynchronous setting.

by a branching bisimulation relation, the state $\nabla_1(p_2 |\{\epsilon, m_1\}| s_1)$ *must simulate the transition* $p_1 \parallel s_1 \xrightarrow{m_2} p_3 \parallel s_3$. *Similarly, to relate the states* $p_1 \parallel s_1$ *and* $\nabla_1(p_3 |\{\epsilon, m_2\}| s_1)$ *by a branching bisimulation relation, the state* $\nabla_1(p_3 |\{\epsilon, m_2\}| s_1)$ *must simulate the transition* $p_1 \parallel s_1 \xrightarrow{m_1} p_2 \parallel s_2$. *For this purpose, we require the transitions* $p_2 \parallel s_2 \xrightarrow{m_2} p_4 \parallel s_4$ *and* $p_3 \parallel s_3 \xrightarrow{m_1} p_4 \parallel s_4$*; thus, completing the diamond in the synchronous system* $p_1 \parallel s_1$. *Consequently, all the invisible transitions in the asynchronous system* $\nabla_1(p_1 |\{\epsilon, \epsilon\}| s_1)$ *are inert modulo* $\leftrightarrows_{\mathbf{b}}$ *(see Figure 3.4).*

Thus, the choice between the messages (say, m_1, m_2) of the plant process do not disable the execution of each other, and the traces $m_1.m_2$, and $m_2.m_1$ commute. Note that this can be encoded in the (X, Y)-diamond property (Definition 3.9) by substituting X as M_p, and Y as $M_p \cup M_s$.

Unfortunately, both well-posedness and $(M_p, M_p \cup M_s)$-diamond property are not sufficient to guarantee an asynchronous system is orphan free when bags are used as buffers. The following example illustrates this fact.

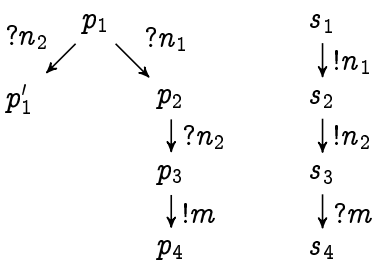


FIGURE 3.5: A plant and its supervisor in Example 3.4.

Example 3.4. *Consider the behaviour of a plant p_1 and its supervisor s_1 as shown in Figure 3.5.*

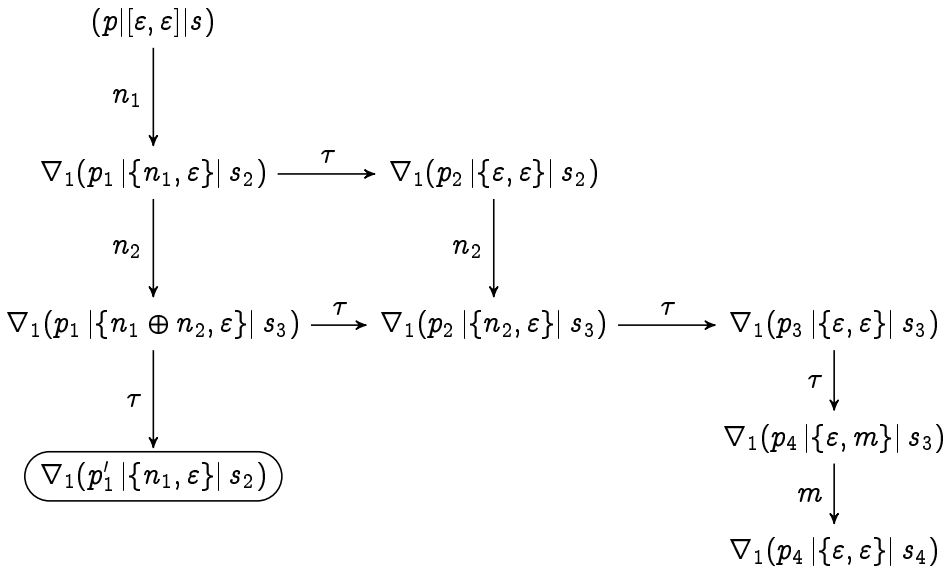


FIGURE 3.6: An example showing a deadlock in the asynchronous system that is absent in the synchronous system.

The transition system of the synchronous system $p_1 \parallel s_1$ is:

$$\left(\{p_i \parallel s_i \mid i \in [1,3]\}, \left\{ \begin{array}{l} p_1 \parallel s_1 \xrightarrow{n_1} p_2 \parallel s_2, p_2 \parallel s_2 \xrightarrow{n_2} p_3 \parallel s_3, \\ p_3 \parallel s_3 \xrightarrow{m} p_4 \parallel s_4 \end{array} \right\}\right).$$

The transition system generated by the asynchronous $\nabla_1(p\,|\{\varepsilon,\varepsilon\}|\,s)$ is depicted in Figure 3.6. Since the messages in the asynchronous system $\nabla_1(p\,|\{\varepsilon,\varepsilon\}|\,s)$ can delay in the buffers, so the supervisor can perform the output sequence $!n_1.!n_2$ without allowing any moves from the plant p; thus, resulting in the state $\nabla_1(p\,|\{n_1\oplus n_2,\varepsilon\}|\,s_2)$. Now, due to the deterministic choice between $?n_1$ and $?n_2$ at the state p_1, the plant can remove the message n_2 from its input bag before the message n_1 and leading the asynchronous system into a deadlock state (shown as rounded rectangle in Figure 3.6).

Thus, the following reordering property is designed to eliminate such scenarios.

Definition 3.14. A synchronous system $p_1 \parallel s_1$ satisfies the *reordering* property iff every reachable state $p_1 \parallel s_1 \in \mathfrak{R}(p \parallel s)$ satisfies the following conditions.

1. $\forall p_2, s_2, p_3, n, \mu.\Big[p_1 \parallel s_1 \xrightarrow{\mu.n} \!\!\!\!\rightarrow\; p_2 \parallel s_2 \wedge p_1 \xrightarrow{?n} p_3 \wedge n \in M_s \wedge \mu \in {M_s}^* \Rightarrow$

$$\exists s_3, \mu'. \left[p_1 \parallel s_1 \xrightarrow{n} p_3 \parallel s_3 \xrightarrow{\mu'} \!\!\!\!\rightarrow\; p_2 \parallel s_2 \wedge \mu =_\pi \mu' \right] \Big],$$

2. $\forall p_2, s_2, s_3, m, \nu. \Big[p_1 \parallel s_1 \xrightarrow{\nu.m} \!\!\!\!\rightarrow\; p_2 \parallel s_2 \wedge s_1 \xrightarrow{?m} s_3 \wedge m \in M_p \wedge \nu \in {M_p}^* \Rightarrow$

$$\exists p_3, \nu'. \left[p_1 \parallel s_1 \xrightarrow{m} p_3 \parallel s_3 \xrightarrow{\nu'} \!\!\!\!\rightarrow\; p_2 \parallel s_2 \wedge \nu' =_\pi \nu \right] \Big].$$

Informally, Condition 1 of Definition 3.14 states that if the plant is willing to receive an input $?n$ and the supervisor offers the message $n \in M_s$ after performing the sequence of messages $\mu \in {M_s}^*$, then the plant can read the message n before reading the sequence of messages μ' such that μ' is a permutation of the sequence μ. This is due to the nature of bags that are inserted between the plant and its supervisor.

Theorem 3.15. *Let $p \parallel s$ be a synthesised synchronous system such that p, s are well-posed. If $p \parallel s$ satisfies the $(M_p, M_p \cup M_s)$ diamond property and the reordering property then $p \parallel s \leftrightarrows_{\mathbf{b}} \nabla_1(p \,|\{\varepsilon, \varepsilon\}|\, s)$.*

Proof. Define a relation $\mathcal{B}$ in the following way:

$$\begin{aligned}
\mathcal{B} = \Big\{ & (p_1 \parallel s_1, \nabla_1(p_2 \,|\{\xi, \zeta\}|\, s_1)) \;\Big|\; p_1 \parallel s_1 \in \mathfrak{R}(p \parallel s) \wedge \\
& \nabla_1(p_2 \,|\{\xi, \zeta\}|\, s_1) \in \mathfrak{R}(\nabla_1(p \,|\{\varepsilon, \varepsilon\}|\, s)) \wedge \\
& \exists p_2', s_2', p_2, s_2, \mu, \nu. \Big[\mu \in \mathbf{S}(\xi) \wedge p_2' \parallel s_2' \xrightarrow{\mu} \!\!\!\!\rightarrow\; p_1 \parallel s_1 \wedge \\
& \qquad \nu \in \mathbf{S}(\zeta) \wedge p_2' \parallel s_2' \xrightarrow{\nu} \!\!\!\!\rightarrow\; p_2 \parallel s_2 \Big] \bigvee \qquad \text{(C1)} \\
& \exists p_2', s_2', p_2, s_2, \mu, \nu. \Big[\mu \in \mathbf{S}(\xi) \wedge p_2 \parallel s_2 \xrightarrow{\mu} \!\!\!\!\rightarrow\; p_2' \parallel s_2' \wedge \\
& \qquad \nu \in \mathbf{S}(\zeta) \wedge p_1 \parallel s_1 \xrightarrow{\nu} \!\!\!\!\rightarrow\; p_2' \parallel s_2' \Big] \Big\}. \qquad \text{(C2)}
\end{aligned}$$

The remainder of the proof which shows that $\mathcal{B}$ is a branching bisimulation relation can be found in Appendix A as Theorem A.2. □

Like in the case of queues, notice that the above definition of the relation $\mathcal{B}$ is also independent of the size of bags. Thus, the preconditions of Theorem 3.15 are also sufficient for desynchronisability when bags with bounded size and back-pressure are used.

Theorem 3.16 (Orphan freedom). *Suppose a synthesised synchronous system $p \parallel s$ satisfies the preconditions of Theorem 3.15. Then the asynchronous system $\nabla_1(p \,|\{\varepsilon, \varepsilon\}|\, s)$ is orphan free.*

Proof. Pick a reachable state $\nabla_1(p_1 \,|\{\xi,\zeta\}|\, s_1) \in \mathfrak{R}(\nabla_1(p\,|[\varepsilon,\varepsilon]|\, s))$ in the asynchronous system and recall the definition of the relation $\mathcal{B}$ from Theorem 3.15. We showed that if the conditions of this theorem then such a relation $\mathcal{B}$ exists. Thus, by definition of $\mathcal{B}$ we have $\exists p_1'.\,[(p_1' \parallel s_1, \nabla_1(p_1\,|\{\xi,\zeta\}|\, s_1)) \in \mathcal{B}]$.

1. Either, $(p_1' \parallel s_1, \nabla_1(p_1\,|\{\xi,\zeta\}|\, s_1)) \in \mathcal{B}$ due to the condition in Equation C1. By Corollary 3.10 we get

$$\exists p_3, s_3.\left[p_1' \parallel s_1 \xrightarrow{\ \nu\ } p_3 \parallel s_3 \wedge p_1 \parallel s_2 \xrightarrow{\ \mu\ } p_3 \parallel s_3\right].$$

   Applying Proposition 2.11 on the above transitions we get $s_1 \xrightarrow{\ ?\nu\ } s_3$ and $p_1 \xrightarrow{\ ?\mu\ } p_3$. Thus, we get

$$\nabla_1(p_1\,|\{\xi,\zeta\}|\, s_1) \longrightarrow \nabla_1(p_3\,|\{\varepsilon,\zeta\}|\, s_1) \xrightarrow{\ \nu\ } \nabla_1(p_3\,|\{\varepsilon,\varepsilon\}|\, s_3).$$

2. Or, $(p_1' \parallel s_1, \nabla_1(p_1\,|\{\xi,\zeta\}|\, s_1)) \in \mathcal{B}$ due to the condition in Equation C2. Similar to the previous case. □

3.4 Well-posedness for free

Recall that in the supervisory control theory of Ramadge and Wonham [69] a supervisor can enable, or disable controllable actions, but it cannot disable the uncontrollable actions of a plant. In other words, a supervisor, by construction, always has a matching receive transition for every send transition by a plant. Thus, Condition 1 of Definition 3.5 is automatically satisfied by the synthesis procedure of a supervisor because the supervisor is a deterministic process and under the assumption of determinism Condition 1 can be simplified as the condition in Equation (3.1) (Page 24).

This observation raises a question: Is it always possible to guarantee the well-posedness between a plant and its synthesised supervisor? In this section, we provide two ways to ensure the well-posedness between a given plant and its supervisor. These two solutions are different in nature: one is a synthesis technique that establishes the well-posedness for free; while, the other uses the Supremica tool-set [1] to ensure well-posedness.

3.4.1 Well-posedness via synthesis

The following steps are involved in our synthesis technique.

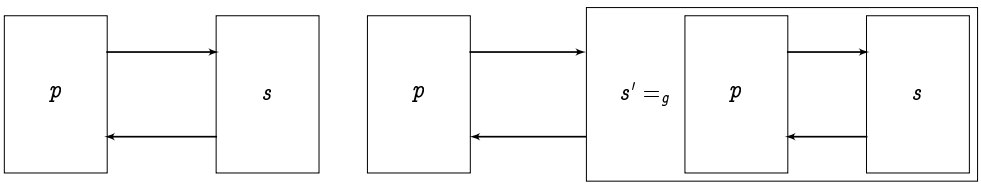


FIGURE 3.7: Well-posedness via synthesis.

- Given a plant p and a requirement r, synthesise a supervisor s using the synthesis procedure of [69].
- Next, construct a new supervisor s' as the synchronous parallel composition of the processes p and s with the renaming of messages into either, send or receive messages (see Figure 3.7). This renaming of messages ensures that the new supervisor s' and the old plant p can understand each other's requests. To this end, define a renaming function $g : M \to A$ in the following way:

$$g(m) = \begin{cases} !m, & \text{if } m \in M_s \\ ?m, & \text{if } m \in M_p \end{cases} . \tag{3.2}$$

Controllability is the central notion in supervisory control theory [10, 34, 79]. In essence, it gives sufficient and necessary conditions for the existence of a supervisor when a plant and a requirement model are given. Intuitively, it asserts that the supervisor must always synchronise with an uncontrollable action (or, a send message) generated by the plant.

Definition 3.17. A supervisor s is *controllable* with respect to a plant p if

$$\forall p_1, s_1, p_2, m. \left[p_1 \parallel s_1 \in \mathfrak{R}(p \parallel s) \wedge p_1 \xrightarrow{!m} p_2 \Rightarrow \exists s_2. \left[s_1 \xrightarrow{?m} s_2 \right] \right].$$

Note that the above definition does not ensure the existence of a supervisor if the given plant p is non-deterministic (cf. [10, 34]). Since a plant p and its supervisor s are deterministic in Ramadge and Wonham's framework, the above definition is sufficient to prove the well-posedness between the plant p and the new supervisor $\rho_g(p \parallel s)$.

Theorem 3.18. *Let p, s be any two concrete and deterministic processes. If s is controllable with respect to p, then the processes p and $\rho_g(p \parallel s)$ are well-posed.*

Proof. Define a binary relation $\mathcal{W} = \{(p_1, \rho_g(p_1 \parallel s_1)) \mid p_1 \parallel s_1 \in \mathfrak{R}(p \parallel s)\}$. Next, we show that the relation $\mathcal{W}$ is a well-posed relation.

1. When $p_1 \xrightarrow{!m} p_2$ and $(p_1, \rho_g(p_1 \parallel s_1)) \in \mathcal{W}$. By definition of $\mathcal{W}$ we have $p_1 \parallel s_1 \in \mathfrak{R}(p \parallel s)$. And from Definition 3.20 we get $\exists s_2. \left[s_1 \xrightarrow{?m} s_2\right]$. Thus, $p_1 \parallel s_1 \xrightarrow{m} p_2 \parallel s_2$. Hence, $\rho_g(p_1 \parallel s_1) \xrightarrow{?m} \rho_g(p_2 \parallel s_2)$ (because $m \in M_p$), and $p_2 \parallel s_2 \in \mathfrak{R}(p \parallel s)$. Clearly, by definition of $\mathcal{W}$ we have $(p_2, \rho_g(p_2 \parallel s_2)) \in \mathcal{W}$.

2. When $\rho_g(p_1 \parallel s_1) \xrightarrow{!n} \rho_g(p_2 \parallel s_2)$ and $(p_1, \rho_g(p_1 \parallel s_1)) \in \mathcal{W}$. By definition of $\mathcal{W}$ we have $p_1 \parallel s_1 \in \mathfrak{R}(p \parallel s)$. And by definition of the renaming function g we know that $p_1 \parallel s_1 \xrightarrow{n} p_2 \parallel s_2$, and $n \in M_s$. Since, the sets M_p, M_s are disjoint; thus, by semantics we have $s_1 \xrightarrow{!n} s_2$ and $p_1 \xrightarrow{?n} p_2$. Clearly, we have $p_2 \parallel s_2 \in \mathfrak{R}(p \parallel s)$. By definition of $\mathcal{W}$ we conclude that $(p_2, \rho_g(p_2 \parallel s_2)) \in \mathcal{W}$. □

Given a plant p and its synthesised supervisor s, we showed that it is possible to synthesise a new supervisor s' such that it is already well-posed with the plant p. The next theorem states that the synchronous systems $p \parallel s, p \parallel s'$ are equivalent modulo strong bisimulation (Definition B.1).

Theorem 3.19. *Let $p \parallel s$ be a synthesised synchronous system. Let $s' = \rho_g(p \parallel s)$. Then, $p \parallel s \leftrightarrows p \parallel s'$.*

Proof. Define a relation in the following way:

$$\mathcal{S} = \{(p_1 \parallel s_1, p_1 \parallel s_1') \mid p_1 \parallel s_1 \in \mathfrak{R}(p \parallel s) \wedge s_1' = \rho_g(p_1 \parallel s_1)\}.$$

Next, we show that the relation $\mathcal{S}$ is a strong bisimulation relation.

1. When $p_1 \parallel s_1 \xrightarrow{\alpha} p_2 \parallel s_2$ and $(p_1 \parallel s_1, p_1 \parallel s_1') \in \mathcal{S}$. Since the alphabet of a synthesised synchronous system contains elements from the set $M_p \cup M_s$. We get the following cases:

   (a) When $\alpha = m$, for some $m \in M_p$. From semantics we have $p_1 \xrightarrow{!m} p_2$, and $s_1 \xrightarrow{?m} s_2$. Also, by construction of $\mathcal{S}$ we have $p_1 \parallel s_1 \in \mathfrak{R}(p \parallel s) \wedge s_1' = \rho_g(p_1 \parallel s_1)$. From the definition of the function g we and the given transition we get $\rho_g(p_1 \parallel s_1) \xrightarrow{?m} \rho_g(p_2 \parallel s_2)$. Let $s_2' = \rho_g(p_2 \parallel s_2)$. Clearly, $p_1 \parallel s_1' \xrightarrow{m} p_2 \parallel s_2'$. Finally, from the construction of $\mathcal{S}$ we conclude that $(p_2 \parallel s_2, p_2 \parallel s_2') \in \mathcal{S}$.

   (b) When $\alpha = n$, for some $n \in M_s$. Similar to the previous case.

2. When $p_1 \parallel s_1' \xrightarrow{\alpha} p_2 \parallel s_2'$ and $(p_1 \parallel s_1, p_1 \parallel s_1') \in \mathcal{S}$. Since the alphabet of a synthesised synchronous system contains elements from the set $M_p \cup M_s$. We get the following cases:

(a) When $\alpha = m$, for some $m \in M_p$. Then, from semantics we have $p_1 \xrightarrow{!m} p_2$ and $s_1' \xrightarrow{?m} s_2'$. And from the construction of $\mathcal{S}$ we have $p_1 \parallel s_1 \in \mathfrak{R}(p \parallel s) \wedge s_1' = \rho_g(p_1 \parallel s_1)$. Furthermore, from the definition of g, the transition $s_1' \xrightarrow{?m} s_2'$, the semantics of $\parallel$, and the fact $s_1' = \rho_g(p_1 \parallel s_1)$ we get $p_1 \parallel s_1 \xrightarrow{m} p_2' \parallel s_2$ and $s_2' = \rho_g(p_2' \parallel s_2)$, for some $p_2', s_2 \in \mathbb{P}$. Thus, $p_1 \xrightarrow{!m} p_2' \wedge s_1 \xrightarrow{?m} s_2$.
But, the plant p in a synthesised synchronous system is deterministic (cf. Proposition 3.4). Thus, $p_1 \xrightarrow{!m} p_2' \wedge p_1 \xrightarrow{!m} p_2 \Rightarrow p_2 = p_2'$. Finally, by the construction of $\mathcal{S}$ we have $(p_2 \parallel s_2, p_2 \parallel s_2') \in \mathcal{S}$.

(b) When $\alpha = n$, for some $n \in M_s$. Similar to the previous case. □

3.4.2 Well-posedness via Supremica

Another alternative to establish a well-posed combination of the plant and its supervisor is by using the "inverse-controllability" feature of Supremica tool [1]. Intuitively, it asserts that a plant must always synchronise with the controllable actions (or, the output messages) of a supervisor.

Definition 3.20. A plant p is *inverse-controllable* with respect to a supervisor s if $\forall p_1, s_1, s_2, n. \left[p_1 \parallel s_1 \in \mathfrak{R}(p \parallel s) \wedge s_1 \xrightarrow{!n} s_2 \Rightarrow \exists p_2. \left[p_1 \xrightarrow{?n} p_2\right]\right]$.

Theorem 3.21. *Let p, s be concrete and deterministic processes. If s is controllable with respect to p and p is inverse-controllable with respect to s, then p and s are well-posed.*

Proof. Define a relation $\mathcal{W} = \{(p_1, s_1) \mid p_1 \parallel s_1 \in \mathfrak{R}(p \parallel s)\}$. Then, the result that $\mathcal{W}$ is a well-posed relation follows directly from the definition of controllability, inverse-controllability, and determinism. □

3.5 Related work

The main focus in the literature of supervisory control theory is to deal with the inexact synchronisation problem is by synthesising a supervisor in the presence of an asynchronous plant. Balemi was the first to consider this issue in the context of supervisory control theory [13]. An *input-output* interpretation was adopted between a plant and its supervisor. Furthermore, a special delay operator was introduced to model the delay in communication between the plant and its supervisor. Although the main result of [13] was the existence of a supervisor for an asynchronous plant, Balemi's construction was partially

asynchronous in nature as noticed in [80]. In [80], this requirement was relaxed; necessary and sufficient conditions were provided for the existence of a controller under the presence of bounded queues.

In addition, the synthesis algorithm of [80] ensures that the synchronous system is desynchronisable modulo language equivalence. The computational complexity of this algorithm exponentially depends on the delay bound (apart from being linear and quadratic on the size of plant p and requirement r, respectively). This is in contrast to the desynchronisation techniques developed in this thesis (not only this chapter), which are independent of the size of the buffers. Thus, our approach seems to be computationally cheaper than the one developed in [80], however this conjecture needs to be verified by analysing the complexities associated with the conditions presented here.

3.6 Conclusions

The goal of this chapter was to examine under what conditions a supervisor that is synthesised using supervisory control theory [69] can control an asynchronous plant without constructing the transition system of an asynchronous system. The main results obtained in this chapter are the following.

1. The conditions well-posedness, M_p-singular, and (M_p, M_s)-diamond are sufficient for desynchronisability of a synthesised synchronous system with respect to the abstraction scheme $\mathbf{A}_1$ and queues (Theorem 3.11).
2. The conditions well-posedness, $(M_p, M_s \cup M_p)$-diamond, and reordering property are sufficient for desynchronisability of a synthesised synchronous system with respect to the abstraction scheme $\mathbf{A}_1$ and bags (Theorem 3.15).
3. Finally, we also showed how to ensure that a given plant and its supervisor are always well-posed (Theorem 3.18 and Theorem 3.21).

In hindsight, we found sufficient conditions in this chapter under which a synthesised synchronous system is branching bisimilar to its corresponding asynchronous system (in the presence of either, queues, or bags). Thus, the synthesised supervisor can control the same plant in the presence of buffers such that the synchronous system and its asynchronous version satisfy the same requirements. The question whether these conditions are reasonable or not to apply in practice is the point of the next chapter.

Chapter 4

Desynchronisation of concrete synchronous systems

In the previous chapter, we found certain conditions on the synchronous systems using the abstraction scheme $\mathbf{A}_1$ for the desynchronisation modulo branching bisimulation. Subsequently, the following question arises: Are these conditions reasonable to apply in practice?

Unfortunately, these conditions are too strict to apply even for the synchronous systems that are synthesised from supervisory control theory. For instance, in the case of queues, the condition M_p-singular says that a plant must execute only a unique send-transition at every reachable state in a synchronous system. Furthermore, in the case of bags, the reordering property (an extra condition with respect to the case of queues) puts restriction on both plant and its supervisor so that they are insensitive to the ordering of messages.

Thus, the above observations lead us to search for a weaker collection of sufficient conditions for desynchronisability than the ones obtained in the previous chapter. This is essentially the objective of this chapter. As we know that the construction of an asynchronous system depends upon the choice of buffers and abstraction schemes, these two factors will play key role in achieving the aforementioned goal.

In particular, we show that by adopting the abstraction scheme $\mathbf{A}_3$ and half-duplex queues the condition M_p-singular and the diamond property disappears from the sufficient condition of desynchronisability. Furthermore, we also show that every synchronous system of supervisory control theory is desynchronisable by construction under this setup. Note that in this chapter the synchronous systems can contain nondeterministic choice and external actions.

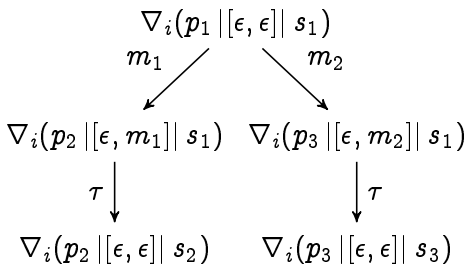


FIGURE 4.1: A way to prevent M_p-singularity, where $i \in \{2,3\}$.

4.1 A quest for weaker conditions

4.1.1 Why queues?

By now it is clear that we want to restrict the behaviour of an asynchronous system so that an equivalence between the synchronous system and its asynchronous version can be established. To this end, we choose queues as buffers rather than bags because an extra condition called the reordering property (Definition 3.14) arises due to the arbitrary order in which the messages are stored in the case of bags. Intuitively, an asynchronous system with bags has more behaviour than an asynchronous system with queues; hence, condition(s) to restrict the former will be stronger than the conditions to restrict the latter.

4.1.2 Why abstraction scheme $\mathbf{A}_3$?

In Chapter 3, we discovered that the deterministic choices between the messages of the process p were transformed into the internal choices in the asynchronous system due the abstraction scheme $\mathbf{A}_1$. In addition, the resulting invisible transitions were non-inert modulo $\leftrightarrows_{\mathbf{b}}$ (see Example 3.1). One way to prevent this – instead of imposing the condition M_p-singular – is by considering the abstraction schemes $\mathbf{A}_2, \mathbf{A}_3$, where the external choice between the messages of the process p remains intact.

Reconsider the synchronous system of Example 3.1. The transition systems of the asynchronous systems with queues and the abstraction schemes $\mathbf{A}_2, \mathbf{A}_3$ are shown in Figure 4.1. Observe that in both the abstraction schemes $\mathbf{A}_2, \mathbf{A}_3$, the output messages $!m_1, !m_2$ of the process p_1 are renamed into the messages m_1, m_2; thus, the choice between m_1, m_2 remains visible. On the other hand, the output messages of the process p_1 are made invisible in the abstraction schemes $\mathbf{A}_1, \mathbf{A}_4$; thus, the external choice between messages m_1, m_2 of the process p_1 is not preserved in $\mathbf{A}_1, \mathbf{A}_4$.

Therefore, by constructing asynchronous systems using the abstraction schemes $\mathbf{A}_2$, $\mathbf{A}_3$ it is possible to eliminate the condition M_p-singular from the sufficient conditions for desynchronisability. However, in case of the abstraction scheme $\mathbf{A}_2$, the issue of preserving the external deterministic choice is not completely resolved. It turns out that in this case a dual version of the condition M_p-singular, i.e., M_s-*singular* arises because (inherently) the abstraction scheme $\mathbf{A}_2$ is a dual version of the abstraction scheme $\mathbf{A}_1$.

Consider the following definition of a synchronous system $p_1 \parallel s_1$:

$$\left(\{p_i \parallel s_i \mid i \in \{1,2,3\}\}, \{p_1 \parallel s_1 \xrightarrow{n_1} p_2 \parallel s_2, p_1 \parallel s_1 \xrightarrow{n_2} p_3 \parallel s_3\}\right),$$

where $n_1, n_2 \in M_s$. Notice that in the transition system[1] of the asynchronous system $\nabla_2(p_1 \,|[\epsilon,\epsilon]|\, s_1)$ the choice between n_1, n_2 is again transformed into an internal choice. The reason is that the output messages of the process s_1 are renamed into invisible actions in the abstraction scheme $\mathbf{A}_2$ (compare this with the abstraction scheme $\mathbf{A}_1$ and observe the duality).

On the other hand, in the abstraction scheme $\mathbf{A}_3$ the output messages of both the processes p, s are renamed to the corresponding messages, i.e., elements of the sets M_p, M_s, respectively. Thus, the choice between the messages of processes p, s remains visible.

4.1.3 Half-duplex queues

Another property which we want to eliminate from the sufficient conditions for desynchronisability is (M_p, M_s)-diamond property because it is hard to establish in practice. Consider a simplified model of a drive motor (a device used in an Automated Guided Vehicle [38]), which can move in a forward direction '*fwmove*', or in a backward direction '*bwmove*'. The drive motor can only move in the forward direction in its initial state. The requirement is to design a controller such that the event change direction '*chdir*' always executes before altering the direction of the motor. The models of the drive motor, the controller, and the synchronous system are shown in Figure 4.2. The dotted lines shows the extra transitions that are required in the synchronous system to satisfy the (M_p, M_s)-diamond property. Observe that if the synchronous system satisfies the (M_p, M_s)-diamond property, then it violates the requirement. Thus, it is impossible to satisfy the (M_p, M_s)-diamond property in these situations unless we adapt the model of plant or supervisor, or we adapt the way in which the desynchronisation is performed.

To encompass the application of desynchronisation, it is necessary to eliminate the diamonds between the messages m, n when $m \in M_p$, $n \in M_s$. For this

[1] We ask the reader to verify this observation by drawing the transition system of the asynchronous system $\nabla_2(p_1 \,|[\epsilon,\epsilon]|\, s_1)$.

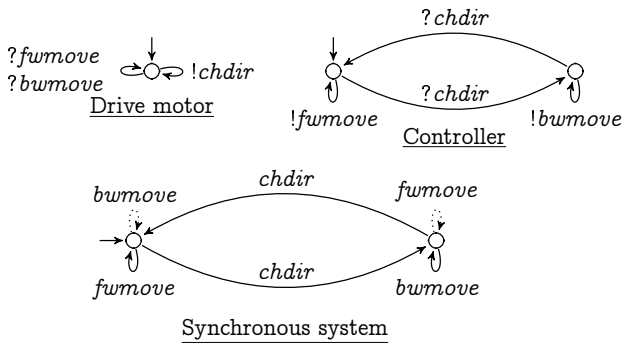


FIGURE 4.2: An illustration showing the impossibility of establishing the diamond property in a synchronous system.

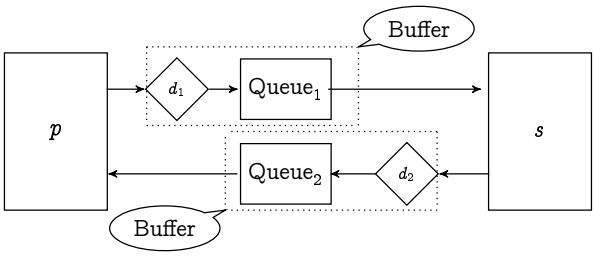


FIGURE 4.3: A buffered system with the half-duplex mechanism. The diamonds d_1, d_2 represent the half-duplex condition.

purpose, we introduce the *half-duplex* mechanism [24], which asserts that a sender is allowed to send a message, whenever its input queue is empty. The modified architecture of a buffered system is shown in Figure 4.3.

Formally, a buffered system with half-duplex mechanism will be composed by a new family of operators $\_ \,||[\mu, \nu]||_{\mathrm{h}}\, \_$, for every $\mu \in {M_s}^*, \nu \in {M_p}^*$. Semantically, we model the half-duplex mechanism in Table 4.1 by replacing Rules 10 and 11 from Table 2.2 with the following Rules $10'$ and $11'$, respectively. Rule $12'$ states that the remaining rules of Table 2.2 are just inherited. Observe that the rules are similar to those we used before (Table 2.2), except that either the left or the right queue remains empty at all times.

Next, we argue that in the context of half-duplex mechanism, the abstraction scheme $\mathbf{A}_3$ will yield a less restrictive condition for desynchronisation in comparison to the other abstraction schemes. This is essentially due to the abstraction scheme $\mathbf{A}_3$ that preserves the external choices between the messages of distinct local processes present at a state in a synchronous system upon adding buffers. The abstraction schemes $\mathbf{A}_1$, $\mathbf{A}_2$, and $\mathbf{A}_4$ fail to preserve these nondeterministic choices, thus disallowing the two systems to be related by a branching bisimulation relation. Notice that here we are highlighting the

$$\frac{p \xrightarrow{!m} p'}{p\,|[\epsilon,\nu]|_{\mathrm{h}}\, s \xrightarrow{!m} p'\,|[\epsilon,\nu.m]|_{\mathrm{h}}\, s}\ (10') \qquad \frac{s \xrightarrow{!n} s'}{p\,|[\mu,\epsilon]|_{\mathrm{h}}\, s \xrightarrow{!n} p\,|[\mu.n,\epsilon]|_{\mathrm{h}}\, s'}\ (11')$$

$$\frac{p_1\,|[\mu_1,\nu_1]|\, s_1 \xrightarrow{\alpha} p_2\,|[\mu_2,\nu_2]|\, s_2,\ \alpha \notin !M_p \cup !M_s}{p_1\,|[\mu_1,\nu_1]|_{\mathrm{h}}\, s_1 \xrightarrow{\alpha} p_2\,|[\mu_2,\nu_2]|_{\mathrm{h}}\, s_2}\ (12') \ .$$

TABLE 4.1: SOS rules for the buffered systems with half-duplex queues.

preservation of an external choice between the messages of *distinct local processes*; whereas, in the previous subsection, we focused on the preservation of an external choice between the messages of *a local process*.

Consider the following transition system of a synchronous system $p_1 \parallel s_1$: $(\{p_1 \parallel s_1, p_2 \parallel s_2, p_3 \parallel s_3\}, \{p_1 \parallel s_1 \xrightarrow{n} p_2 \parallel s_2, p_1 \parallel s_1 \xrightarrow{m} p_3 \parallel s_3\})$, where $n \in M_s$ and $m \in M_p$. In Figure 4.4, the transition systems induced by different abstraction schemes are drawn. The dotted lines show an attempt to construct a branching bisimulation relation between the synchronous system and the asynchronous systems with the respective abstraction schemes. Observe that a branching bisimulation relation only exists in the case of abstraction scheme $\mathbf{A}_3$. For instance, consider the case of $\mathbf{A}_1$, where the states $p_1 \parallel s_1$ and $\nabla_1(p_3\,|[\epsilon,m]|_{\mathrm{h}}\, s_1)$ are not branching bisimilar. This is due to the half-duplex mechanism which prevents the process s_1 to send the message $!n$ and thus, disallowing the state $\nabla_1(p_3\,|[\epsilon,m]|_{\mathrm{h}}\, s_1)$ to simulate the matching message n. Similar reasons can be given for the abstraction schemes $\mathbf{A}_2$ and $\mathbf{A}_4$.

So far, in Chapter 3, we focussed on the properties that the communicating processes should have in order to ensure desynchronisability. Therefore, our research hypothesis is that it may be possible *to find better desynchronisability conditions by changing the properties of the communication protocol*. In the remainder of this chapter, we take a first step in that direction. In particular, in the presence of half-duplex queues and the abstraction scheme $\mathbf{A}_3$ we show that a synchronous system is desynchronisable modulo branching bisimulation if and only if it is well-posed, strongly independent of external actions, and input-deterministic. Moreover, we also show that by dropping the adverb 'strongly' and the condition input-deterministic from the above, the resulting conditions characterises desynchronisability modulo contra-simulation.

4.2 Towards completeness of our characterisation

The following lemmas are useful in proving the completeness of our characterisation.

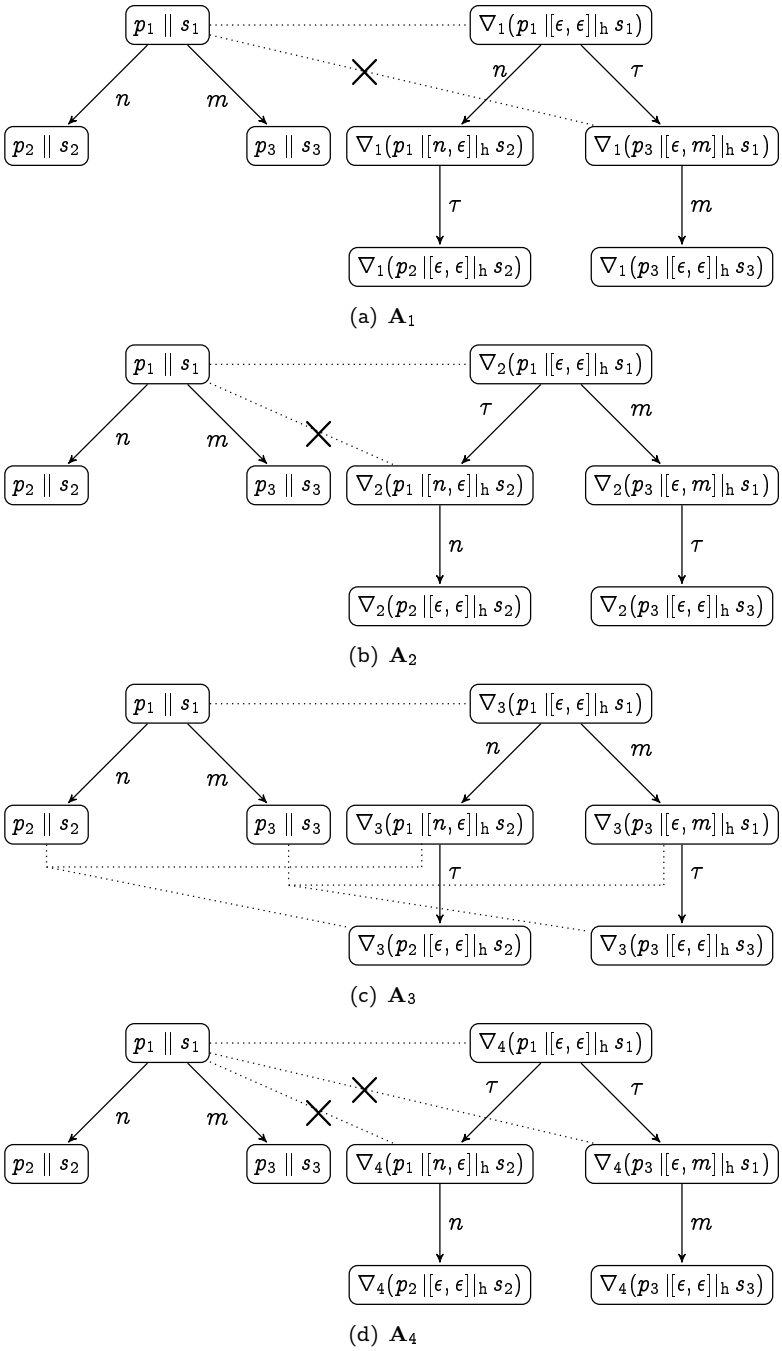


FIGURE 4.4: Different behaviour induced by the abstractions schemes $\mathbf{A}_1, \mathbf{A}_2, \mathbf{A}_3$, and $\mathbf{A}_4$ in the presence of half-duplex mechanism.

Lemma 4.1. *If* $p_1 \parallel s_1 \xrightarrow{w}\!\!\!\!\rightarrow p_2 \parallel s_2$ *then* $\nabla_3(p_1 \,|[\epsilon,\epsilon]|\, s_1) \xrightarrow{w}\!\!\!\!\rightarrow \nabla_3(p_2 \,|[\epsilon,\epsilon]|\, s_2)$.

Proof. Straightforward by induction on w. □

Lemma 4.2 states that all the invisible transitions induced by the abstraction scheme $\mathbf{A}_3$ are inert modulo branching bisimulation, whenever a concrete synchronous system is desynchronisable.

Lemma 4.2. *Suppose* $p \parallel s$ *is a concrete synchronous system,* $p \parallel s \leftrightarrows^{\sqcup}_{\mathbf{b}} \nabla_3(p \,|[\epsilon,\epsilon]|\, s)$, *and* $\nabla_3(p_1 \,|[\mu_1,\nu_1]|\, s_1) \in \mathfrak{R}(\nabla_3(p \,|[\epsilon,\epsilon]|\, s))$. *If*

$$\nabla_3(p_1 \,|[\mu_1,\nu_1]|\, s_1) \xrightarrow{\tau} \nabla_3(p_2 \,|[\mu_2,\nu_2]|\, s_2),$$

then $\nabla_3(p_1 \,|[\mu_1,\nu_1]|\, s_1) \leftrightarrows^{\sqcup}_{\mathbf{b}} \nabla_3(p_2 \,|[\mu_2,\nu_2]|\, s_2)$.

Proof. Since $p \parallel s$ is a concrete process, none of the τ-transitions in the asynchronous system can be matched by any related state in the synchronous system. Thus, all τ-transitions in the asynchronous system have to be inert modulo branching bisimulation. □

Lemma 4.3 states that two synchronous systems $p_1 \parallel s_1, p_2 \parallel s_2$ are indistinguishable modulo $\simeq^{\sqcup}$, whenever $p_1 \parallel s_1 \simeq^{\sqcup} \nabla_3(p_2 \,|[\epsilon,\epsilon]|\, s_2)$ and p_i, s_i are concrete processes (for $i \in \{1,2\}$).

Lemma 4.3. *Let* p_1, s_1, p_2, s_2 *be any four concrete processes. Then,*

$$p_1 \parallel s_1 \simeq^{\sqcup} \nabla_3(p_2 \,|[\epsilon,\epsilon]|\, s_2) \Rightarrow p_1 \parallel s_1 \simeq^{\sqcup} p_2 \parallel s_2 \quad .$$

Proof. It is easier to show the above results for strong bisimulation[2] [66], i.e., $p_1 \parallel s_1 \simeq^{\sqcup} \nabla_3(p_2 \,|[\epsilon,\epsilon]|\, s_2) \Rightarrow p_1 \parallel s_1 \leftrightarrows^{\sqcup} p_2 \parallel s_2$. Note that the desired result follows directly from Proposition B.2, which states that for the class of concrete processes the branching bisimulation and the contra-simulation equivalences coincides with the strong bisimulation equivalence. To see the former, define the following relations $\mathcal{S}_{\simeq^{\sqcup}}$ (for $\simeq^{\sqcup} \in \{\leftrightarrows^{\sqcup}_{\mathbf{b}}, \sim^{\sqcup}_{\mathbf{c}}\}$):

$$\mathcal{S}_{\simeq^{\sqcup}} = \Big\{ (p_3 \parallel s_3, p_4 \parallel s_4), (p_4 \parallel s_4, p_3 \parallel s_3) \Big|\; p_3 \parallel s_3 \in \mathfrak{R}(p_1 \parallel s_1) \wedge$$
$$\nabla_3(p_4 \,|[\epsilon,\epsilon]|\, s_4) \in \mathfrak{R}(\nabla_3(p_2 \,|[\epsilon,\epsilon]|\, s_2)) \wedge p_3 \parallel s_3 \simeq^{\sqcup} \nabla_3(p_4 \,|[\epsilon,\epsilon]|\, s_4) \Big\}.$$

Now, it is easy (although tedious) to show that the above relations are strong bisimulation relations. The complete proof can be found in Appendix B under the label Lemma B.3. □

[2] For the sake of completeness, the definition of strong bisimulation is given in the Appendix B as Definition B.1.

Lemma 4.4 states that if a concrete synchronous system is desynchronisable modulo $\simeq^{\sqcup}$, then every reachable state in the synchronous system is desynchronisable modulo $\simeq^{\sqcup}$. It is this property which is key in understanding the necessary conditions for desynchronisability modulo $\simeq^{\sqcup}$ (see the next section).

Lemma 4.4. *Let p, s be any two concrete processes. If $p \parallel s \simeq^{\sqcup} \nabla_3(p \,|[\epsilon,\epsilon]|\, s)$ then, $\forall p_1, s_1.\, [p_1 \parallel s_1 \in \mathfrak{R}(p \parallel s) \Rightarrow p_1 \parallel s_1 \simeq^{\sqcup} \nabla_3(p_1 \,|[\epsilon,\epsilon]|\, s_1)]$.*

Proof. We show the result for $\simeq^{\sqcup} = \sim^{\sqcup}_{\mathbf{c}}$. The proof for the case $\simeq^{\sqcup} = \underline{\leftrightarrow}^{\sqcup}_{\mathbf{b}}$ can be constructed along similar lines. Without loss of generality, let $p_1 \parallel s_1 \sim^{\sqcup}_{\mathbf{c}} \nabla_3(p_1 \,|[\epsilon,\epsilon]|\, s_1)$ for some $p_1 \parallel s_1 \in \mathfrak{R}(p \parallel s)$.

1. Suppose $p_1 \parallel s_1 \xrightarrow{n} p_2 \parallel s_2$, for some $n \in M_s$. Then we need to show that $p_2 \parallel s_2 \sim^{\sqcup}_{\mathbf{c}} \nabla_3(p_2 \,|[\epsilon,\epsilon]|\, s_2)$. Note that the above transition is due to Rule 2. Thus, we have $p_1 \xrightarrow{?n} p_2 \wedge s_1 \xrightarrow{!n} s_2$. Using this transition we get
$$\nabla_3(p_1 \,|[\epsilon,\epsilon]|\, s_1) \xrightarrow{n} \nabla_3(p_1 \,|[n,\epsilon]|\, s_2) \xrightarrow{\tau} \nabla_3(p_2 \,|[\epsilon,\epsilon]|\, s_2).$$
Thus, $\nabla_3(p_1 \,|[\epsilon,\epsilon]|\, s_1) \xrightarrow{n}\!\!\!\!\rightarrow \nabla_3(p_2 \,|[\epsilon,\epsilon]|\, s_2)$. But, from the induction hypothesis we have $p_1 \parallel s_1 \sim^{\sqcup}_{\mathbf{c}} \nabla_3(p_1 \,|[\epsilon,\epsilon]|\, s_1)$. Using the transfer condition of contra-simulation and the fact that the synchronous system is concrete we get $\exists p_2', s_2'.\, \left[p_1 \parallel s_1 \xrightarrow{n} p_2' \parallel s_2' \wedge p_2' \parallel s_2' \preccurlyeq^{\sqcup} \nabla_3(p_2 \,|[\epsilon,\epsilon]|\, s_2)\right]$. Since p, s are concrete processes so is p_2, s_2. Thus, $\nabla_3(p_2 \,|[\epsilon,\epsilon]|\, s_2) \stackrel{\tau}{\not\rightarrow}$. Applying Proposition 2.10 we get $\nabla_3(p_2 \,|[\epsilon,\epsilon]|\, s_2) \preccurlyeq^{\sqcup} p_2' \parallel s_2'$. Hence, $p_2' \parallel s_2' \sim^{\sqcup}_{\mathbf{c}} \nabla_3(p_2 \,|[\epsilon,\epsilon]|\, s_2)$. Now, from Lemma 4.3 we have $p_2 \parallel s_2 \sim^{\sqcup}_{\mathbf{c}} p_2' \parallel s_2'$. And from the transitivity of $\sim^{\sqcup}_{\mathbf{c}}$ we conclude that $p_2 \parallel s_2 \sim^{\sqcup}_{\mathbf{c}} \nabla_3(p_2 \,|[\epsilon,\epsilon]|\, s_2)$.
2. Suppose $p_1 \parallel s_1 \xrightarrow{\alpha} p_2 \parallel s_2$, for some $\alpha \in M_p \cup E_p \cup E_s$. Similar to the previous case. □

4.3 Characterisation

In this section, we first present the necessary conditions for desynchronisation of a concrete synchronous system modulo $\simeq^{\sqcup}$. Note that we show our conditions are necessary for desynchronisation modulo $\simeq^{\sqcup}$ even for the full-duplex queues. Subsequently, we prove that these conditions are also sufficient for the desynchronisation modulo $\simeq^{\sqcup}$ in the presence of a half-duplex mechanism.

4.3.1 Well-posedness

The first implication of desynchronisability modulo $\simeq^{\sqcup}$ is well-posedness as defined in Chapter 3.

Theorem 4.5. *Suppose p, s are concrete processes and $p \parallel s \simeq^{\sqcup} \nabla_3(p \,|[\epsilon,\epsilon]|\, s)$. Then p, s are well-posed.*

Proof. We show the result for $\simeq^{\sqcup} = \sim_{\mathbf{c}}^{\sqcup}$. The proof for the case $\simeq^{\sqcup} = \leftrightarrow_{\mathbf{b}}^{\sqcup}$ can be constructed along similar lines. Define a binary relation $\mathcal{W}$:

$$\mathcal{W} = \{(p_1, s_1) \mid \nabla_3(p_1 \,|[\epsilon,\epsilon]|\, s_1) \in \mathfrak{R}(\nabla_3(p \,|[\epsilon,\epsilon]|\, s))\}.$$

Next, we need to show that $\mathcal{W}$ is a well-posed relation.

1. Suppose $(p_1, s_1) \in \mathcal{W}$ and $p_1 \xrightarrow{!m} p_2$, for some $!m \in M_p$. By definition of $\mathcal{W}$ we have $\nabla_3(p_1 \,|[\epsilon,\epsilon]|\, s_1) \in \mathfrak{R}(\nabla_3(p \,|[\epsilon,\epsilon]|\, s))$. Using the above transition we have $\nabla_3(p_1 \,|[\epsilon,\epsilon]|\, s_1) \xrightarrow{m} \nabla_3(p_2 \,|[\epsilon,m]|\, s_1)$. Thus, $\nabla_3(p_2 \,|[\epsilon,m]|\, s_1) \in \mathfrak{R}(\nabla_3(p \,|[\epsilon,\epsilon]|\, s))$. Since $p \parallel s \sim_{\mathbf{c}}^{\sqcup} \nabla_3(p \,|[\epsilon,\epsilon]|\, s)$ so there exists some $q \in \mathfrak{R}(p \parallel s)$ such that $q \preccurlyeq^{\sqcup} \nabla_3(p_2 \,|[\epsilon,m]|\, s_1)$.

   But, $\nabla_3(p_2 \,|[\epsilon,m]|\, s_1)\!\downarrow$. Using the transfer condition of contra-simulation for the predicate $\sqcup$ we get

   $$\exists s_2.\, \left[\nabla_3(p_2 \,|[\epsilon,m]|\, s_1) \xrightarrow{\tau} \nabla_3(p_2 \,|[\epsilon,\epsilon]|\, s_2)\right].$$

   From the semantics and the construction of $\mathcal{W}$ we get $s_1 \xrightarrow{?m} s_2$ and $(p_2, s_2) \in \mathcal{W}$, respectively. Thus, we showed that there exists s_2 such that $s_1 \xrightarrow{?m} s_2$ and $(p_2, s_2) \in \mathcal{W}$. But, well-posedness also asserts that for every s_2 such that $s_1 \xrightarrow{?m} s_2$ and $(p_2, s_2) \in \mathcal{W}$. So pick a $s_2' \in \mathbb{P}$ such that $s_1 \xrightarrow{?m} s_2'$. Then we have $\nabla_3(p_2 \,|[\epsilon,m]|\, s_1) \xrightarrow{\tau} \nabla_3(p_2 \,|[\epsilon,\epsilon]|\, s_2')$. Finally, by the construction of $\mathcal{W}$ we conclude that $(p_2, s_2') \in \mathcal{W}$.

2. Suppose $(p_1, s_1) \in \mathcal{W}$ and $p_1 \xrightarrow{e} p_2$, for some $e \in E_p$. Then, by definition of $\mathcal{W}$ we have $\nabla_3(p_1 \,|[\epsilon,\epsilon]|\, s_1) \in \mathfrak{R}(\nabla_3(p \,|[\epsilon,\epsilon]|\, s))$. Using the above transition we have $\nabla_3(p_1 \,|[\epsilon,\epsilon]|\, s_1) \xrightarrow{e} \nabla_3(p_2 \,|[\epsilon,\epsilon]|\, s_1)$. Thus, $\nabla_3(p_2 \,|[\epsilon,\epsilon]|\, s_1) \in \mathfrak{R}(\nabla_3(p \,|[\epsilon,\epsilon]|\, s))$ and from the construction of $\mathcal{W}$ we get $(p_2, s_1) \in \mathcal{W}$.

3. Suppose $(p_1, s_1) \in \mathcal{W}$ and $s_1 \xrightarrow{\alpha} s_2$, for some $\alpha \in E_s \cup M_s$. Similar to the previous cases. □

4.3.2 Input-determinism

The second implication of desynchronisability modulo $\leftrightarrow_{\mathbf{b}}^{\sqcup}$ is that desynchronisable synchronous systems are input-deterministic.

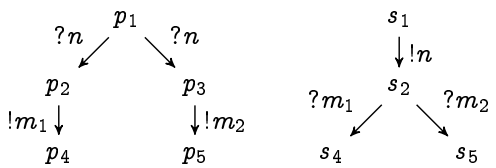
Definition 4.6. A process $p \parallel s$ is *input-deterministic modulo* $\leftrightarrow_{\mathbf{b}}^{\sqcup}$ iff every reachable state $p_1 \parallel s_1 \in \mathfrak{R}(p \parallel s)$ satisfies the following conditions.

1. for all p_2, s_2, p_3, whenever $p_1 \parallel s_1 \xrightarrow{u}\!\!\!\!\rightarrow p_2 \parallel s_2$ and $p_1 \parallel s_1 \xrightarrow{u}\!\!\!\!\rightarrow p_3 \parallel s_2$ for some $u \in (M_s \cup E_s)^*$, then $p_2 \parallel s_2 \leftrightarrows_{\mathbf{b}} p_3 \parallel s_2$.

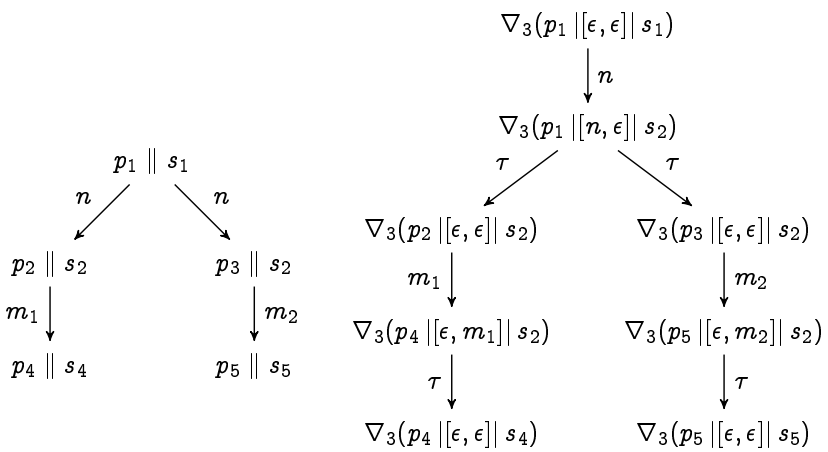
2. for all p_2, s_2, s_3, whenever $p_1 \parallel s_1 \xrightarrow{v}\!\!\!\!\rightarrow p_2 \parallel s_2$ and $p_1 \parallel s_1 \xrightarrow{v}\!\!\!\!\rightarrow p_2 \parallel s_3$ for some $v \in (M_p \cup E_p)^*$, then $p_2 \parallel s_2 \leftrightarrows_{\mathbf{b}} p_2 \parallel s_3$.

Intuitively, an input-deterministic synchronous system $p \parallel s$ make deterministic choices upon the reception of messages. It may perform nondeterministic external behavior, and it may also be nondeterministic when sending messages. This is because desynchronisation only delays nondeterministic choice on the input messages as shown in the next example.

Example 4.1. *Consider the two transition systems of the processes p, s as shown in Figure 4.5(a). The transition systems of the synchronous system and asynchronous system are shown in Figure 4.5(b). Clearly, the*



(a) Transitions systems of processes p, s in Example 4.1.



(b) Transition systems of $p \parallel s$ and $\nabla_3(p \,|[\epsilon,\epsilon]|\, s)$.

FIGURE 4.5: The role of input-determinism in desynchronisability.

following pairs of states: $(p_2 \parallel s_2, \nabla_3(p_1 \,|[n,\epsilon]|\, s_2))$, $(p_3 \parallel s_2, \nabla_3(p_1 \,|[n,\epsilon]|\, s_2))$ cannot be related by a branching bisimulation relation, i.e., $p_2 \parallel s_2 \not\leftrightarrows^{\sqcup}_{\mathbf{b}} \nabla_3(p_1 \,|[n,\epsilon]|\, s_2)$, $p_3 \parallel s_2 \not\leftrightarrows^{\sqcup}_{\mathbf{b}} \nabla_3(p_1 \,|[n,\epsilon]|\, s_2)$, respectively. This is because the state $p_2 \parallel s_2$ is unable to simulate the transition labelled with m_2; similarly, the state $p_3 \parallel s_2$ is unable to simulate the transition labelled with m_1. Note that in abstraction scheme $\mathbf{A}_3$, reading the messages from

the buffers by the local processes p, s are made invisible. Thus, the choice made by the local process p_1 will result in an internal nondeterministic choice at the state $\nabla_3(p_1 \,|[n, \epsilon]|\, s_1)$ in the asynchronous system, whereas in the synchronous system it will result in an external nondeterministic choice at the state $p_1 \parallel s_1$. This is an example of delaying of choice in the context of desynchronisation. Note that, in contrast, the synchronous system $p_1 \parallel s_1$ and the asynchronous system $\nabla_3(p_1 \,|[\epsilon, \epsilon]|\, s_1)$ depicted in Figure 4.5(b) are contra-similar.

At first sight, it seems obvious that Definition 4.6 can be equivalently defined in the following '*local*' way by restricting u, v to be a message from the process s and p (i.e., $u \in M_s$ and $v \in M_p$) in Conditions 1 and 2, respectively. Note that this formulation only leads to an equivalent formulation of input-determinism modulo syntactical equivalence (not branching bisimulation equivalence).

Definition 4.7. A synchronous system $p \parallel s$ is *locally* input deterministic modulo $=$ if every reachable state $p_1 \parallel s_1 \in \mathfrak{R}(p \parallel s)$ satisfies the following conditions.

1. for all p_2, s_2, p_3, whenever $p_1 \parallel s_1 \xrightarrow{n} p_2 \parallel s_2$ and $p_1 \parallel s_1 \xrightarrow{n} p_3 \parallel s_2$ for some $n \in M_s$, then $p_2 \parallel s_2 = p_3 \parallel s_2$.
2. for all p_2, s_2, s_3, whenever $p_1 \parallel s_1 \xrightarrow{m} p_2 \parallel s_2$ and $p_1 \parallel s_1 \xrightarrow{m} p_2 \parallel s_3$ for some $m \in M_p$, then $p_2 \parallel s_2 = p_2 \parallel s_3$.

Lemma 4.8. *A concrete synchronous system $p \parallel s$ is locally input deterministic modulo $=$ iff it is input deterministic modulo $=$.*

Proof. The *only-if* part follows directly from Definition 4.6. The *if* part follows by performing induction on sequences $u \in (M_s \cup E_s)^*$ ($v \in (M_p \cup E_p)^*$) and instantiating Definition 4.7. □

The next theorem states input-determinism modulo $\underline{\leftrightarrow}^{\sqcup}_{\mathbf{b}}$ is a necessary condition for desynchronisation modulo $\underline{\leftrightarrow}^{\sqcup}_{\mathbf{b}}$.

Theorem 4.9. *If p, s are concrete processes and $p \parallel s \,\underline{\leftrightarrow}^{\sqcup}_{\mathbf{b}}\, \nabla_3(p \,|[\epsilon, \epsilon]|\, s)$, then the synchronous system $p \parallel s$ is input-deterministic modulo $\underline{\leftrightarrow}^{\sqcup}_{\mathbf{b}}$.*

Proof. We need to show that if $p_1 \parallel s_1 \in \mathfrak{R}(p \parallel s)$, $p_1 \parallel s_1 \xrightarrow{u}\!\!\!\!\rightarrow\, p_2 \parallel s_2$, and $p_1 \parallel s_1 \xrightarrow{u}\!\!\!\!\rightarrow\, p_3 \parallel s_2$ (for $u \in (M_s \cup E_s)^*$), then $p_2 \parallel s_2 \,\underline{\leftrightarrow}^{\sqcup}_{\mathbf{b}}\, p_3 \parallel s_2$. From Proposition 2.11, we have $s_1 \xrightarrow{!u}\!\!\!\!\rightarrow\, s_2$, $p_1 \xrightarrow{?u}\!\!\!\!\rightarrow\, p_2$, and $p_1 \xrightarrow{?u}\!\!\!\!\rightarrow\, p_3$. From Lemma 4.4 we have $p_1 \parallel s_1 \,\underline{\leftrightarrow}^{\sqcup}_{\mathbf{b}}\, \nabla_3(p_1 \,|[\epsilon, \epsilon]|\, s_1)$. Using the sequence of transitions $s_1 \xrightarrow{!u}\!\!\!\!\rightarrow\, s_2$ we get $\nabla_3(p_1 \,|[\epsilon, \epsilon]|\, s_1) \xrightarrow{u}\!\!\!\!\rightarrow\, \nabla_3(p_1 \,|[\mu, \epsilon]|\, s_2)$, where

$\mu = \bar{u}$. Using the sequence of transitions $p_1 \xrightarrow{?u} p_2$, $p_1 \xrightarrow{?u} p_3$ we get $\nabla_3(p_1 \,|[\mu,\epsilon]|\, s_2) \longrightarrow \nabla_3(p_2 \,|[\epsilon,\epsilon]|\, s_2)$ and $\nabla_3(p_1 \,|[\mu,\epsilon]|\, s_2) \longrightarrow \nabla_3(p_3 \,|[\epsilon,\epsilon]|\, s_2)$, respectively.

Furthermore, from Lemma 4.2 we know that these τ-transitions are inert modulo $\leftrightarrows^{\sqcup}_{\mathbf{b}}$. Thus, we have $\nabla_3(p_2 \,|[\epsilon,\epsilon]|\, s_2) \leftrightarrows^{\sqcup}_{\mathbf{b}} \nabla_3(p_3 \,|[\epsilon,\epsilon]|\, s_2)$. Also, from Lemma 4.4 we have $p_2 \parallel s_2 \leftrightarrows^{\sqcup}_{\mathbf{b}} \nabla_3(p_2 \,|[\epsilon,\epsilon]|\, s_2)$ and $p_3 \parallel s_2 \leftrightarrows^{\sqcup}_{\mathbf{b}} \nabla_3(p_3 \,|[\epsilon,\epsilon]|\, s_2)$. Hence, by transitivity we get $p_2 \parallel s_2 \leftrightarrows^{\sqcup}_{\mathbf{b}} p_3 \parallel s_2$.

Likewise, the following statement can be proven: if $p_1 \parallel s_1 \in \mathfrak{R}(p \parallel s)$, $v \in (M_p \cup E_p)^*$, $p_1 \parallel s_1 \xrightarrow{v} p_2 \parallel s_2$, and $p_1 \parallel s_1 \xrightarrow{v} p_2 \parallel s_3$ then $p_2 \parallel s_2 \leftrightarrows^{\sqcup}_{\mathbf{b}} p_2 \parallel s_3$. □

4.3.3 Independence of external actions

The last implication of desynchronisability that we discuss is *independence of external actions*. Intuitively, it means that a receiver can always delay the execution of its own external action e in favor of receiving a sequence of messages u from the other process, without any consequence on its future behavior, i.e., the traces $e.u$ and $u.e$ commute up to $\simeq^{\sqcup}$. The reception of messages becomes independent of the external behavior in this way.

Definition 4.10. A synchronous system $p \parallel s$ is *E-independent modulo $\simeq^{\sqcup}$*, if every reachable state $p_1 \parallel s_1 \in \mathfrak{R}(p \parallel s)$ satisfies the following conditions (see Figure 4.6).

1. $\forall p_3, p_4, s_2, u, e. \Big[e \in E_p \wedge u \in (M_s \cup E_s)^* \wedge p_1 \parallel s_1 \xrightarrow{e} p_3 \parallel s_1 \xrightarrow{u} p_4 \parallel s_2 \Rightarrow$
   $\exists p_2, p_5, s_2'. \Big[p_1 \parallel s_1 \xrightarrow{u} p_2 \parallel s_2' \xrightarrow{e} p_5 \parallel s_2' \wedge p_4 \parallel s_2 \simeq^{\sqcup} p_5 \parallel s_2' \Big] \Big]$.
2. $\forall p_2, s_3, s_4, v, e. \Big[e \in E_s \wedge v \in (M_p \cup E_p)^* \wedge p_1 \parallel s_1 \xrightarrow{e} p_1 \parallel s_3 \xrightarrow{v} p_2 \parallel s_4 \Rightarrow$
   $\exists p_2', s_2, s_5. \Big[p_1 \parallel s_1 \xrightarrow{v} p_2' \parallel s_2 \xrightarrow{e} p_2' \parallel s_5 \wedge p_2 \parallel s_4 \simeq^{\sqcup} p_2' \parallel s_5 \Big] \Big]$.

Furthermore, a synchronous system $p \parallel s$ is *strong* E-independent modulo $\simeq^{\sqcup}$, if $s_2' = s_2$ in Condition 1 and $p_2' = p_2$ in Condition 2, respectively.

Just like in the case of input-determinism, it seems obvious that the above definition can be equivalently defined in the following '*local*' way by restricting u, v to be a message from the process s and p (i.e., $u \in M_s$ and $v \in M_p$) in Conditions 1 and 2, respectively. Note that this formulation only leads to an equivalent formulation of E-independence, not strong E-independence.

Proposition 4.11. *A concrete synchronous system $p \parallel s$ is E-independent modulo $\simeq^{\sqcup}$ iff $p \parallel s$ is locally E-independent modulo $\simeq^{\sqcup}$.*

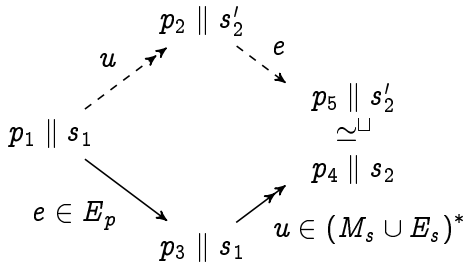


FIGURE 4.6: Illustration of Condition 1 in Definition 4.10.

Proof. The if-part follows directly from Definition 4.10 by letting $u \in M_s$ and $v \in M_p$ in Conditions 1 and 2, respectively. To prove the only-if part, assume $u = u'.\alpha$ such that $p_1 \parallel s_1 \xrightarrow{e} p_3 \parallel s_1 \xrightarrow{u'} p_4 \parallel s_2 \xrightarrow{\alpha} p_6 \parallel s_6$, for some $u' \in (M_s \cup E_s)^*$, $\alpha \in M_s \cup E_s$, $e \in E_p$, and $p_4, s_2 \in \mathbb{P}$. By induction hypothesis we have

$$\exists p_2, p_5, s_2'. \left[p_1 \parallel s_1 \xrightarrow{u'} p_2 \parallel s_2' \xrightarrow{e} p_5 \parallel s_2' \wedge p_4 \parallel s_2 \simeq^{\sqcup} p_5 \parallel s_2' \right].$$

Now, performing case-distinction on the type of α we get the following cases:

- Let $\alpha = e'$, for some $e' \in E_s$. Then, $p_4 = p_6$ and $s_2 \xrightarrow{e'} s_6$. Recall $p_4 \parallel s_2 \simeq^{\sqcup} p_5 \parallel s_2'$ from the induction hypothesis. Since the synchronous system $p \parallel s$ is concrete, then $p_4 \parallel s_2 \leftrightarrows^{\sqcup} p_5 \parallel s_2'$ in light of Proposition B.2. From the transfer conditions of strong bisimulation we get

  $$\exists p_7, s_6'. \left[p_5 \parallel s_2' \xrightarrow{e'} p_7 \parallel s_6' \wedge p_7 \parallel s_6' \leftrightarrows^{\sqcup} p_6 \parallel s_6 \right].$$

  Since, the sets E_p, E_s are disjoint, thus by the semantics we get $p_5 = p_7$.

- Let $\alpha = n$, for some $n \in M_s$. Then, $p_4 \xrightarrow{?n} p_6$ and $s_2 \xrightarrow{!n} s_6$. From the previous case we know that $p_4 \parallel s_2 \simeq^{\sqcup} p_5 \parallel s_2' \Rightarrow p_4 \parallel s_2 \leftrightarrows^{\sqcup} p_5 \parallel s_2'$. And from the transfer conditions of strong bisimulation we get

  $$\exists p_7, s_6'. \left[p_5 \parallel s_2' \xrightarrow{n} p_7 \parallel s_6' \wedge p_7 \parallel s_6' \leftrightarrows^{\sqcup} p_6 \parallel s_6 \right].$$

Likewise, Condition 2 of Definition 4.10 can be derived from its corresponding local formulation. □

In contrast to the above proposition, we have the following lemma for strong E-independence modulo $=$.

Lemma 4.12. *A concrete synchronous system $p \parallel s$ is locally strong E-independent of external actions iff it is strong E-independent of external actions modulo $=$.*

Proof. The *only-if* part follows directly from Definition 4.10. To prove the *if* part, assume the transitions $p_1 \parallel s_1 \xrightarrow{e} p_2 \parallel s_1 \xrightarrow{u}\!\!\!\!\rightarrow p_2' \parallel s_2$, where $p_1 \parallel s_1 \in \mathfrak{R}(p \parallel s)$, $e \in E_p$, and $u \in (M_s \cup E_s)^*$. We show by induction on u that there exists p_3 such that $p_1 \parallel s_1 \xrightarrow{u}\!\!\!\!\rightarrow p_3 \parallel s_2 \xrightarrow{e} p_2' \parallel s_2$. Without loss of generality, assume $u = u'.\alpha$ such that $p_2 \parallel s_1 \xrightarrow{u'}\!\!\!\!\rightarrow p_3' \parallel s_2' \xrightarrow{\alpha} p_2' \parallel s_2$, for some $p_3', s_2' \in \mathbb{P}$ and $u' \in (M_s \cup E_s)^*$ and $\alpha \in M_s \cup E_s$. Then, by induction hypothesis we have $p_1 \parallel s_1 \xrightarrow{u'}\!\!\!\!\rightarrow p_3 \parallel s_2' \xrightarrow{e} p_3' \parallel s_2'$, for some $p_3 \in \mathbb{P}$. Now performing case distinction on α we get the following cases.

1. Let $\alpha = e'$ for some $e' \in E_s$. Then, by the semantics we have $p_3' = p_2'$ and $s_2' \xrightarrow{e'} s_2$. Using this transition at the state $p_3 \parallel s_2'$ we get $p_3 \parallel s_2' \xrightarrow{e'} p_3 \parallel s_2$. But, from inductive hypothesis we have $p_3 \xrightarrow{e} p_2'$. Thus, $p_3 \parallel s_2 \xrightarrow{e} p_2' \parallel s_2$; hence, $p_1 \parallel s_1 \xrightarrow{u}\!\!\!\!\rightarrow p_3 \parallel s_2 \xrightarrow{e} p_2' \parallel s_2$ as required.
2. Let $\alpha = n$ for some $n \in M_s$. Then, $p_3 \parallel s_2' \xrightarrow{n} p_4 \parallel s_2 \xrightarrow{e} p_2' \parallel s_2$, for some $p_4 \in \mathbb{P}$. Thus, $p_1 \parallel s_1 \xrightarrow{u}\!\!\!\!\rightarrow p_4 \parallel s_2 \xrightarrow{e} p_2' \parallel s_2$ as required. □

The next theorem states that strong E-independent modulo $\underline{\leftrightarrow}^{\sqcup}_{\mathbf{b}}$ is a necessary condition for desynchronisation modulo $\underline{\leftrightarrow}^{\sqcup}_{\mathbf{b}}$.

Theorem 4.13. *If p, s are concrete processes and $p \parallel s \mathrel{\underline{\leftrightarrow}^{\sqcup}_{\mathbf{b}}} \nabla_3(p \,|[\epsilon,\epsilon]|\, s)$, then $p \parallel s$ is strong E-independent modulo $\underline{\leftrightarrow}^{\sqcup}_{\mathbf{b}}$.*

Proof. We need to show that if $p_1 \parallel s_1 \in \mathfrak{R}(p \parallel s)$, $e \in E_p, u \in (M_s \cup E_s)^*$, $p_1 \parallel s_1 \xrightarrow{e} p_3 \parallel s_1$, and $p_3 \parallel s_1 \xrightarrow{u}\!\!\!\!\rightarrow p_4 \parallel s_2$, then

$$\exists p_2, p_5.\left[p_1 \parallel s_1 \xrightarrow{n}\!\!\!\!\rightarrow p_2 \parallel s_2 \wedge p_2 \parallel s_2 \xrightarrow{e} p_5 \parallel s_2 \wedge p_4 \parallel s_2 \mathrel{\underline{\leftrightarrow}^{\sqcup}_{\mathbf{b}}} p_5 \parallel s_2\right].$$

Note that the above condition is a stronger version of Condition 1 in Definition 4.10. Applying Proposition 2.11 in the transitions $p_1 \parallel s_1 \xrightarrow{e} p_3 \parallel s_1 \xrightarrow{u}\!\!\!\!\rightarrow p_4 \parallel s_2$ we get

$$p_1 \xrightarrow{e} p_3 \wedge p_3 \xrightarrow{?u}\!\!\!\!\rightarrow p_4 \wedge s_1 \xrightarrow{!u}\!\!\!\!\rightarrow s_2. \tag{4.1}$$

And, from Lemma 4.1 we get $p_1 \parallel s_1 \in \mathfrak{R}(p \parallel s) \Rightarrow \nabla_3(p_1 \,|[\epsilon,\epsilon]|\, s_1) \in \mathfrak{R}(\nabla_3(p \,|[\epsilon,\epsilon]|\, s))$. Observe that using the transitions in Equation 4.1 we can derive the transitions in the asynchronous system depicted as solid lines in Figure 4.7, where $\mu = u$. Furthermore, from Theorem 4.5 we know that if $p \parallel s$ is a concrete process and $p \parallel s \mathrel{\underline{\leftrightarrow}^{\sqcup}_{\mathbf{b}}} \nabla_3(p \,|[\epsilon,\epsilon]|\, s)$ then p, s are well-posed.

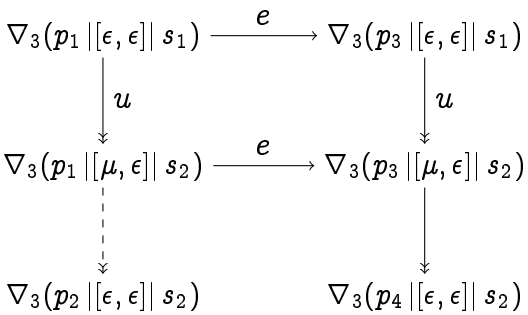


FIGURE 4.7: Partial transition system used in Theorem 4.13.

So let $\mathcal{W}$ be a well-posedness relation between p, s. Then, from Proposition 3.6 we get $(p_1, s_1) \in \mathcal{W}$. So, from Lemma 3.7, if $s_1 \xrightarrow{!u} s_2$ implies that $\exists p_2. \left[p_1 \xrightarrow{?u} p_2\right]$. Thus, we get the transition $\nabla_3(p_1 \,|[\mu,\epsilon]|\, s_1) \longrightarrow \nabla_3(p_2 \,|[\epsilon,\epsilon]|\, s_2)$ depicted as dashed line in Figure 4.7.

Now, from Lemma 4.2 we know that all invisible transitions in the asynchronous system $\nabla_3(p \,|[\epsilon,\epsilon]|\, s)$ are inert, so we have $\nabla_3(p_1 \,|[\mu,\epsilon]|\, s_2) \leftrightarrows^{\sqcup}_{\mathbf{b}} \nabla_3(p_2 \,|[\epsilon,\epsilon]|\, s_2)$. Using the transfer conditions of Definition 2.5 we get

$$\exists q', q''. \Big[\nabla_3(p_2 \,|[\epsilon,\epsilon]|\, s_2) \longrightarrow q' \xrightarrow{e} q'' \wedge q' \leftrightarrows^{\sqcup}_{\mathbf{b}} \nabla_3(p_1 \,|[\mu,\epsilon]|\, s_2) \wedge q'' \leftrightarrows^{\sqcup}_{\mathbf{b}} \nabla_3(p_3 \,|[\mu,\epsilon]|\, s_2)\Big].$$

But, p, s are concrete processes, so we have $q' = \nabla_3(p_2 \,|[\epsilon,\epsilon]|\, s_2)$. Since the sets E_p, E_s are disjoint, so from the semantics we know that the transition $\nabla_3(p_2 \,|[\epsilon,\epsilon]|\, s_2) \xrightarrow{e} q''$ is due to Rule 16. Thus,

$$\exists p_5. \left[q'' = \nabla_3(p_5 \,|[\epsilon,\epsilon]|\, s_2) \wedge p_2 \xrightarrow{e} p_5\right].$$

Using the above transition we get $p_2 \parallel s_2 \xrightarrow{e} p_5 \parallel s_2$. Moreover, from above we have $\nabla_3(p_5 \,|[\epsilon,\epsilon]|\, s_2) \leftrightarrows^{\sqcup}_{\mathbf{b}} \nabla_3(p_3 \,|[\mu,\epsilon]|\, s_2)$ ($\because q'' = \nabla_3(p_5 \,|[\epsilon,\epsilon]|\, s_2) \wedge q'' \leftrightarrows^{\sqcup}_{\mathbf{b}} \nabla_3(p_3 \,|[\mu,\epsilon]|\, s_2)$). Also, from Lemma 4.2 we have

$$\nabla_3(p_3 \,|[\mu,\epsilon]|\, s_2) \leftrightarrows^{\sqcup}_{\mathbf{b}} \nabla_3(p_4 \,|[\epsilon,\epsilon]|\, s_2).$$

Thus, from transitivity we have $\nabla_3(p_5 \,|[\epsilon,\epsilon]|\, s_2) \leftrightarrows^{\sqcup}_{\mathbf{b}} \nabla_3(p_4 \,|[\epsilon,\epsilon]|\, s_2)$. Furthermore, from Lemma 4.4 we have $x \parallel s_2 \leftrightarrows^{\sqcup}_{\mathbf{b}} \nabla_3(x \,|[\epsilon,\epsilon]|\, s_2)$ for $x \in \{p_4, p_5\}$. Hence, we can conclude that $p_4 \parallel s_2 \leftrightarrows^{\sqcup}_{\mathbf{b}} p_5 \parallel s_2$.

Likewise, the following statement can be proven: if $p_1 \parallel s_1 \in \mathfrak{R}(p \parallel s)$, $e \in E_s, v \in (M_p \cup E_p)^*$, $p_1 \parallel s_1 \xrightarrow{e} p_1 \parallel s_3$, and $p_1 \parallel s_3 \xrightarrow{v}\!\!\!\!\rightarrow\, p_2 \parallel s_4$, then

$$\exists s_2, s_5.\ \Big[p_1 \parallel s_1 \xrightarrow{v}\!\!\!\!\rightarrow\, p_2 \parallel s_2 \xrightarrow{e} p_2 \parallel s_5 \wedge p_2 \parallel s_4 \underline{\leftrightarrow}^{\sqcup}_{\mathbf{b}}\, p_2 \parallel s_5\Big]. \qquad \square$$

Theorem 4.14 demonstrates that E-independent modulo $\sim^{\sqcup}_{\mathbf{c}}$ is a necessary condition for desynchronisation modulo $\sim^{\sqcup}_{\mathbf{c}}$. Note that we use the weaker form of Definition 4.10 for desynchronisation modulo $\sim^{\sqcup}_{\mathbf{c}}$.

Theorem 4.14. *If p, s are concrete processes and $p \parallel s \sim^{\sqcup}_{\mathbf{c}} \nabla_3(p\,|[\epsilon,\epsilon]|\,s)$, then $p \parallel s$ is E-independent modulo $\sim^{\sqcup}_{\mathbf{c}}$.*

Proof. We first show that if $p_1 \parallel s_1 \in \mathfrak{R}(p \parallel s)$, $e \in E_p, u \in (M_s \cup E_s)^*$, $p_1 \parallel s_1 \xrightarrow{e} p_3 \parallel s_1$, and $p_3 \parallel s_1 \xrightarrow{u}\!\!\!\!\rightarrow\, p_4 \parallel s_2$, then

$$\exists p_2, p_5, s_2'.\ \Big[p_1 \parallel s_1 \xrightarrow{u}\!\!\!\!\rightarrow\, p_2 \parallel s_2' \xrightarrow{e} p_5 \parallel s_2' \wedge p_4 \parallel s_2 \simeq^{\sqcup} p_5 \parallel s_2'\Big].$$

Following the arguments of the previous proof we can derive the transitions in asynchronous system as depicted in Figure 4.7. Furthermore, from Lemma 4.4 we have $p_1 \parallel s_1 \sim^{\sqcup}_{\mathbf{c}} \nabla_3(p_1\,|[\epsilon,\epsilon]|\,s_1)$. Using transfer conditions of Definition 2.9 we know that $\nabla_3(p_1\,|[\epsilon,\epsilon]|\,s_1) \xrightarrow{u.e}\!\!\!\!\rightarrow\, \nabla_3(p_4\,|[\epsilon,\epsilon]|\,s_2)$ and $\nabla_3(p_1\,|[\epsilon,\epsilon]|\,s_1) \preccurlyeq^{\sqcup} p_1 \parallel s_1$ implies $\exists p_5, s_2'.\ \Big[p_1 \parallel s_1 \xrightarrow{u.e}\!\!\!\!\rightarrow\, p_5 \parallel s_2' \wedge p_5 \parallel s_2' \preccurlyeq^{\sqcup} \nabla_3(p_4\,|[\epsilon,\epsilon]|\,s_2)\Big]$. Since, p_4, s_2 are concrete processes so we have $\nabla_3(p_4\,|[\epsilon,\epsilon]|\,s_2) \xrightarrow{\tau}\!\!\!\!\!\!\rightarrow$. Now, from Proposition 2.10 we get $\nabla_3(p_4\,|[\epsilon,\epsilon]|\,s_2) \preccurlyeq^{\sqcup} p_5 \parallel s_2'$.

Thus, $p_5 \parallel s_2' \sim^{\sqcup}_{\mathbf{c}} \nabla_3(p_4\,|[\epsilon,\epsilon]|\,s_2)$. Now consider the sequence of transitions $p_1 \parallel s_1 \xrightarrow{n.e}\!\!\!\!\rightarrow\, p_5 \parallel s_2'$. and by decomposing it in the following way we get the desired result $\exists p_2.\ \Big[p_1 \parallel s_1 \xrightarrow{n}\!\!\!\!\rightarrow\, p_2 \parallel s_2' \xrightarrow{e} p_5 \parallel s_2'\Big]$. $\square$

4.3.4 Proof of sufficiency for desynchronisability

In this subsection, we prove that well-posedness, input-determinism modulo $\underline{\leftrightarrow}^{\sqcup}_{\mathbf{b}}$, and strong E-independence modulo $\underline{\leftrightarrow}^{\sqcup}_{\mathbf{b}}$ are also sufficient for desynchronisation of the concrete synchronous systems modulo $\underline{\leftrightarrow}^{\sqcup}_{\mathbf{b}}$, whenever buffers used are half-duplex queues (queues with half-duplex mechanism). Furthermore, we also show that dropping input-determinism modulo $\underline{\leftrightarrow}^{\sqcup}_{\mathbf{b}}$ and substituting strong E-independence modulo $\underline{\leftrightarrow}^{\sqcup}_{\mathbf{b}}$ by E-independence modulo $\sim^{\sqcup}_{\mathbf{c}}$ results in sufficient conditions for desynchronisation modulo $\sim^{\sqcup}_{\mathbf{c}}$.

Theorem 4.15. *Let p, s be concrete processes. If the synchronous system $p \parallel s$ is well-posed, input-deterministic modulo $\underline{\leftrightarrow}^{\sqcup}_{\mathbf{b}}$, and strong E-independent modulo $\underline{\leftrightarrow}^{\sqcup}_{\mathbf{b}}$, then $p \parallel s \underline{\leftrightarrow}^{\sqcup}_{\mathbf{b}} \nabla_3(p\,|[\epsilon,\epsilon]|_{\mathrm{h}}\,s)$.*

Proof. Define a relation $\mathcal{B}$ in the following way:

$$\begin{aligned}\mathcal{B} = \Big\{ & (p_1 \parallel s_1, \nabla_3(p_2 \,|[\mu,\nu]|_{\mathrm{h}}\, s_2)) \mid p_1 \parallel s_1 \in \mathfrak{R}(p \parallel s) \wedge \\ & \nabla_3(p_2 \,|[\mu,\nu]|_{\mathrm{h}}\, s_2) \in \mathfrak{R}(\nabla_3(p \,|[\epsilon,\epsilon]|_{\mathrm{h}}\, s)) \wedge \\ & ((\mu = \epsilon \wedge \nu = \epsilon) \Rightarrow p_1 \parallel s_1 \leftrightarrow^{\sqcup} p_2 \parallel s_2) \bigvee \\ & \Big((\mu \neq \epsilon \wedge \nu = \epsilon) \Rightarrow \exists p_2', s_2', u \in (M_s \cup E_s)^*. \Big[p_2 \parallel s_2' \in \mathfrak{R}(p \parallel s) \wedge \\ & \qquad p_2 \parallel s_2' \xrightarrow{u} \!\!\!\!\rightarrow\, p_2' \parallel s_2 \wedge \mu = \bar{u} \wedge p_1 \parallel s_1 \leftrightarrow^{\sqcup} p_2' \parallel s_2\Big]\Big) \bigvee \\ & \Big((\mu = \epsilon \wedge \nu \neq \epsilon) \Rightarrow \exists p_2', s_2', v \in (M_p \cup E_p)^*. \Big[p_2' \parallel s_2 \in \mathfrak{R}(p \parallel s) \wedge \\ & \qquad p_2' \parallel s_2 \xrightarrow{v} \!\!\!\!\rightarrow\, p_2 \parallel s_2' \wedge \nu = \bar{v} \wedge p_1 \parallel s_1 \leftrightarrow^{\sqcup} p_2 \parallel s_2'\Big]\Big)\Big\}.\end{aligned}$$

The remainder of the proof which shows that $\mathcal{B}$ is a branching bisimulation relation can be found in Appendix B under the label Theorem B.4. □

Observe that we use strong bisimulation equivalence (instead of branching bisimulation equivalence) between the reachable states of the given concrete synchronous system $p \parallel s$ in the above definition of the relation $\mathcal{B}$ because branching bisimulation and strong bisimulation coincide in the class of concrete processes (cf. Proposition B.2). Technically, we prefer to use strong bisimulation (instead of branching bisimulation) on the reachable states of the given concrete synchronous systems in the proofs of Theorem 4.15 because its transfer conditions are simpler than the ones in branching bisimulation.

The following corollary states that well-posedness, input-determinism modulo $=$, and strong E-independent modulo $=$ are also sufficient for desynchronisability modulo $\leftrightarrow^{\sqcup}_{\mathbf{b}}$.

Corollary 4.16. *Let p, s be concrete processes. If the synchronous system $p \parallel s$ is well-posed, input-deterministic modulo $=$, and strong E-independent modulo $=$, then $p \parallel s \leftrightarrow^{\sqcup}_{\mathbf{b}} \nabla_3(p \,|[\epsilon,\epsilon]|_{\mathrm{h}}\, s)$.*

Proof. Substitute $\leftrightarrow^{\sqcup}, \leftrightarrow^{\sqcup}_{\mathbf{b}}$ by $=$ in the proof of Theorem B.4. □

Corollary 4.17. *Let p, s be concrete processes. If the synchronous system $p \parallel s$ is well-posed, locally input-deterministic modulo $=$, and locally strong E-independent modulo $=$, then $p \parallel s \leftrightarrow^{\sqcup}_{\mathbf{b}} \nabla_3(p \,|[\epsilon,\epsilon]|_{\mathrm{h}}\, s)$.*

Proof. By Lemmas 4.12,4.8 and Corollary 4.16. □

4.3.5 Contra-simulation

Recall Example 4.1 where the delaying of nondeterministic choices in an asynchronous system forbids a branching bisimulation relation between the synchronous system and its asynchronous version. However, observe that the two transition systems in Figure 4.5(b) are contra-similar, even though the synchronous system $p_1 \parallel s_1$ is not input-deterministic. In this subsection, we formally prove this observation that input-determinism is no longer required for desynchronisation modulo contra-simulation.

Unlike in the case of branching bisimulation, we have to reason with the multi-steps (reachability relation) $\_ \xrightarrow{w}\!\!\!\!\!\rightarrow \_$ (for some $w \in A^*$) due to the transfer conditions of a contra-simulation relation. In the following lemmas, we identify some properties about the asynchronous systems that are involved with the reachability relation required to prove Theorem 4.21.

Let $\sqsubset_s$ be the smallest relation on $(E \cup M)^*$ that is closed under the following rules:

$$e.e' \sqsubset_s e'.e, \quad n.e' \sqsubset_s e'.n, \quad \text{where } e \in E_s, e' \in E_p, n \in M_s.$$

Similarly, let $\sqsubset_p$ be the smallest relation on $(E \cup M)^*$ that is closed under the following rules: $e.e' \sqsubset_p e'.e, m.e' \sqsubset_p e'.m$, where $e \in E_p, e' \in E_s, m \in M_p$. Let $\sqsubseteq_s, \sqsubseteq_p$ denote the reflexive closure of the relations $\sqsubset_s, \sqsubset_p$, respectively. Intuitively, $w \sqsubseteq_x w'$ (for $x \in \{p, s\}$) denote that the sequence w' can be rewritten into the sequence w using the above rules.

Proposition 4.18. *1. Let $v \in {E_p}^*$ and $e \in E_s$. Then, $e.v \sqsubseteq_s v.e$.*

2. Let $u \in {E_s}^*$ and $e \in E_p$. Then, $e.u \sqsubseteq_p u.e$.

Lemma 4.19 (Triangle lemma)**.** *Let $p \parallel s$ be a concrete synchronous system.*

1. *If $\nabla_3(p_1 \,|[\epsilon,\epsilon]|_{\mathrm{h}}\, s_1) \xrightarrow{w}\!\!\!\!\!\rightarrow \nabla_3(p_2 \,|[\mu,\epsilon]|_{\mathrm{h}}\, s_2)$, then there exists $p_1', s_1' \in \mathbb{P}$, $u \in (E_s \cup M_s)^*$, $v \in (E_p \cup ?M_p)^*$ such that (see Figure 4.8)*

   (a) $\nabla_3(p_1 \,|[\epsilon,\epsilon]|_{\mathrm{h}}\, s_1) \xrightarrow{w_1}\!\!\!\!\!\rightarrow \nabla_3(p_1' \,|[\epsilon,\epsilon]|_{\mathrm{h}}\, s_1') \xrightarrow{w_2}\!\!\!\!\!\rightarrow \nabla_3(p_2 \,|[\mu,\epsilon]|_{\mathrm{h}}\, s_2)$, $w = w_1.w_2$, and

   (b) $s_1' \xrightarrow{!u}\!\!\!\!\!\rightarrow s_2, p_1' \xrightarrow{v}\!\!\!\!\!\rightarrow p_2$, $u.(v \uparrow E_p) \sqsubseteq_s w_2$, and $?u = (v \uparrow ?M_p).?\mu$.

2. *If $\nabla_3(p_1 \,|[\epsilon,\epsilon]|_{\mathrm{h}}\, s_1) \xrightarrow{w}\!\!\!\!\!\rightarrow \nabla_3(p_2 \,|[\epsilon,\nu]|_{\mathrm{h}}\, s_2)$, then there exists $p_1', s_1' \in \mathbb{P}$, $v \in (E_p \cup M_p)^*$, $u \in (E_s \cup ?M_s)^*$ such that*

   (a) $\nabla_3(p_1 \,|[\epsilon,\epsilon]|_{\mathrm{h}}\, s_1) \xrightarrow{w_1}\!\!\!\!\!\rightarrow \nabla_3(p_1' \,|[\epsilon,\epsilon]|_{\mathrm{h}}\, s_1') \xrightarrow{w_2}\!\!\!\!\!\rightarrow \nabla_3(p_2 \,|[\epsilon,\nu]|_{\mathrm{h}}\, s_2)$, $w = w_1.w_2$, and

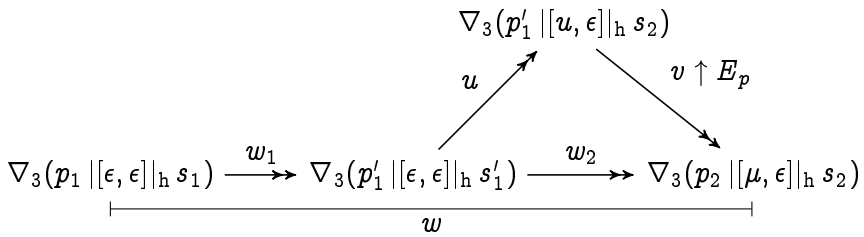


FIGURE 4.8: Illustration of Item 1 in Triangle lemma.

$$\textit{(b)}\ s_1' \xrightarrow{u} \!\!\!\rightarrow s_2, p_1' \xrightarrow{!v} \!\!\!\rightarrow p_2,\ v.(u \uparrow E_s) \sqsubseteq_p w_2,\ \textit{and}\ ?v = (u \uparrow ?M_s).?\nu.$$

Condition 1 of Triangle lemma states that if a state $\nabla_3(p_2 \,|[\mu,\epsilon]|_{\mathrm{h}}\, s_2)$ is reachable from a state $\nabla_3(p_1 \,|[\epsilon,\epsilon]|_{\mathrm{h}}\, s_1)$ with the trace w, then

1. there exists an intermediate state $\nabla_3(p_1' \,|[\epsilon,\epsilon]|_{\mathrm{h}}\, s_1')$, i.e.,

   $$\nabla_3(p_1 \,|[\epsilon,\epsilon]|_{\mathrm{h}}\, s_1) \xrightarrow{w_1} \!\!\!\rightarrow \nabla_3(p_1' \,|[\epsilon,\epsilon]|_{\mathrm{h}}\, s_1') \xrightarrow{w_2} \!\!\!\rightarrow \nabla_3(p_2 \,|[\mu,\epsilon]|_{\mathrm{h}}\, s_2)$$

   such that $w = w_1.w_2$, and

2. the target state can be reached from this intermediate state by first performing s-transitions labelled with the sequence $u \in (E_s \cup M_s)^*$ and then performing p-transitions labelled with the sequence $v \uparrow E_p$, where $v \in (E_p \cup ?M_p)^*$. This is due to the half-duplex property, which disallow the process p_1' to send any message until its input queue is empty. The sequence w_2 can be rewritten into the sequence $u.(v \uparrow E_p)$ is encoded by the condition $u.(v \uparrow E_p)w_2$.

   Finally, the condition $?u = (v \uparrow ?M_p).?\mu$ states that that a sequence of message sent by the process s_1' is equivalent to the sequence of message received by the process p_1' concatenated with the renaming messages in the input queue of the process p. Intuitively, this condition states that the input queue attached to the process p is lossless.

A similar explanation can be given for Condition 2 in Lemma 4.19.

Proof of Triangle lemma. We first prove the statement in Item 1 by induction on w. Assume, $w = w'.\alpha$ with $\nabla_3(p_1 \,|[\epsilon,\epsilon]|_{\mathrm{h}}\, s_1) \xrightarrow{w} \!\!\!\rightarrow \nabla_3(p_2 \,|[\mu,\epsilon]|_{\mathrm{h}}\, s_2) \xrightarrow{\alpha} \nabla_3(p_3 \,|[\mu_1,\epsilon]|_{\mathrm{h}}\, s_3)$. By the induction hypothesis we have there exists $p_1', s_1' \in \mathbb{P}$, $u \in (E_s \cup M_s)^*$, $v \in (E_p \cup ?M_p)^*$ such that

(a) $\nabla_3(p_1 \,|[\epsilon,\epsilon]|_{\mathrm{h}}\, s_1) \xrightarrow{w_1} \!\!\!\rightarrow \nabla_3(p_1' \,|[\epsilon,\epsilon]|_{\mathrm{h}}\, s_1') \xrightarrow{w_2} \!\!\!\rightarrow \nabla_3(p_2 \,|[\mu,\epsilon]|_{\mathrm{h}}\, s_2)$, $w = w_1.w_2$, and

(b) $s_1' \xrightarrow{!u} s_2, p_1' \xrightarrow{v} p_2$, $u.(v \uparrow E_p) \sqsubseteq_s w_2$, and $?u = (v \uparrow ?M_p).?\mu$.

Now, based on the type of α we get the following cases:

1. When $\alpha = e$, for some $e \in E_s$. Then, $p_2 = p_3$, $\mu_1 = \mu$, $s_2 \xrightarrow{e} s_3$. From above we have $s_1' \xrightarrow{!u} s_2$. Thus, $s_1' \xrightarrow{(!u).e} s_3$. And from definition of the output projection function (Proposition 2.11) we know that $(!u).e = !(u.e)$. Let $u' = u.e$. Next, we show that $?u' = (v \uparrow M_p).?\mu$.

$$\begin{aligned} ?u' &= ?(u.e) \quad \text{(from above)} \\ &= ?u.?e \\ &= (v \uparrow ?M_p).?\mu.\epsilon \quad \text{(inductive hypothesis)} \\ &= (v \uparrow ?M_p).?\mu. \end{aligned}$$

   Finally,

$$\begin{aligned} &u.(v \uparrow E_p) \sqsubseteq_s w_2 \quad \text{(inductive hypothesis)} \\ &u.(v \uparrow E_p).e \sqsubseteq_s w_2.e \\ &u.e.(v \uparrow E_p) \sqsubseteq_s w_2.e \quad (\textit{Proposition } 4.18). \end{aligned}$$

2. When $\alpha = e$, for some $e \in E_p$. Then, $p_2 \xrightarrow{e} p_3$, $\mu_1 = \mu$, $s_2 = s_3$. From above we have $p_1' \xrightarrow{v} p_2$. Thus, $p_1' \xrightarrow{v.e} p_3$. Fix, $v' = v.e$. Now we show that $?u = (v' \uparrow ?M_p).?\mu$. This is trivial because $(v.e) \uparrow ?M_p = v$. Furthermore,

$$\begin{aligned} &u.(v \uparrow E_p) \sqsubseteq_s w_2 \\ &u.(v \uparrow E_p).e \sqsubseteq_s w_2.e \\ &u.((v.e) \uparrow E_p) \sqsubseteq_s w_2.e \quad (\because (v.e) \uparrow E_p = (v \uparrow E_p).e) \\ &u.(v' \uparrow E_p) \sqsubseteq_s w_2.e. \end{aligned}$$

3. When $\alpha = n$, for some $n \in M_s$. Similar to 1.

4. When $\alpha = \tau$. Then, $p_2 \xrightarrow{?n} p_3$, $\mu = n.\mu_1$, and $s_2 = s_3$. From above we have $p_1' \xrightarrow{v} p_2$. Thus, $p_1' \xrightarrow{v.?n} p_3$. Fix, $v' = v.?n$. Now we show that $?u = (v' \uparrow ?M_p).?\mu_1$.

$$\begin{aligned} ?u &= (v \uparrow ?M_p).?\mu \quad \text{(inductive hypothesis)} \\ &= (v \uparrow ?M_p).?(n.\mu_1) \quad \text{(from above we have } \mu = n.\mu_1) \\ &= (v \uparrow ?M_p).?n.?\mu_1 \quad (\textit{Proposition } 2.11) \\ &= ((v.?n) \uparrow ?M_p).?\mu_1 \\ &= (v' \uparrow ?M_p).?\mu_1. \end{aligned}$$

$$\text{Clearly, } u.(v' \uparrow E_p) \sqsubseteq_s w_2 \text{ because } v' = v.?n \text{ and } (v.?n) \uparrow E_p = v.$$

Likewise, the statement in Item 2 can be proved. □

Lemma 4.20. *Suppose a concrete and well-posed synchronous system* $p \parallel s$ *is* E*-independent modulo* $\simeq^{\sqcup}$*. Let* $\nabla_3(p_1 \,|[\epsilon,\epsilon]|_{\mathrm{h}}\, s_1) \in \mathfrak{R}(\nabla_3(p\,|[\epsilon,\epsilon]|_{\mathrm{h}}\, s))$ *and* $\nabla_3(p_1 \,|[\epsilon,\epsilon]|_{\mathrm{h}}\, s_1) \xrightarrow{w} \!\!\!\!\rightarrow\; \nabla_3(p_2 \,|[\epsilon,\epsilon]|_{\mathrm{h}}\, s_2)$ *then,*

$$\exists p_3, s_3.\, \left[p_1 \parallel s_1 \xrightarrow{w} \!\!\!\!\rightarrow\; p_3 \parallel s_3 \wedge p_2 \parallel s_2 \leftrightarrows^{\sqcup} p_3 \parallel s_3\right].$$

Proof. See Appendix B under the label Lemma B.5. □

Theorem 4.21. *Let* p, s *be concrete processes. If the synchronous system* $p \parallel s$ *is well-posed and* E*-independent modulo* $\sim^{\sqcup}_{\mathbf{c}}$*, then* $p \parallel s \sim^{\sqcup}_{\mathbf{c}} \nabla_3(p\,|[\epsilon,\epsilon]|_{\mathrm{h}}\, s)$.

Proof. See Appendix B under the label Theorem B.6. □

The following corollary states that well-posedness and E-independent modulo $=$ are also sufficient for desynchronisability modulo $\sim^{\sqcup}_{\mathbf{c}}$.

Corollary 4.22. *Let* p, s *be concrete processes. If the synchronous system* $p \parallel s$ *is well-posed and* E*-independent modulo* $=$*, then* $p \parallel s \sim^{\sqcup}_{\mathbf{c}} \nabla_3(p\,|[\epsilon,\epsilon]|_{\mathrm{h}}\, s)$.

Corollary 4.23. *Let* p, s *be concrete processes. If the synchronous system* $p \parallel s$ *is well-posed and* E*-independent modulo* $\sim^{\sqcup}_{\mathbf{c}}$*, then* $\nabla_3(p\,|[\epsilon,\epsilon]|_{\mathrm{h}}\, s)$ *is orphan free.*

Proof. Application of Theorem 4.21 and Theorem 2.13. □

4.4 Related work

Fischer and Janssen [36] were the first to use the term 'desynchronisation' in concurrency theory. They studied the equivalence problem between a synchronous system and its asynchronous version in CSP process algebra [50], where failure equivalence is the preferred equivalence between any two processes. An asynchronous system was constructed using the abstraction scheme $\mathbf{A}_3$ with bags as buffers. Although the usage of bags as buffers is not directly mentioned in the article [36], this fact can be inferred because for each action a unique queue of unbounded size was placed between the local processes.

This work was motivated by the so-called "Foam-rubber wrapper" principle [73], borrowed from the field of delay insensitive circuits, which states that "a process and the same process connected with buffers are equivalent". The foam-rubber wrapper principle was also studied in the context of the parallel composition operator and it was shown that an extra condition, called sender domination, is required to preserve this principle. In the following, we give comparisons of our work with [36].

- We present our conditions for desynchronisability over a synchronous system $p \parallel s$ conjointly, in contrast with [36], where the check for the foam rubber wrapper principle on the processes p, s was applied separately. Although, imposing conditions on the local processes p, s allows a modular way of asserting them; however, such a formulation leads to stronger condition. Consider for instance the following 'localised' version of input-determinism modulo $\underline{\leftrightarrow}^{\sqcup}_{\mathbf{b}}$.

$$\forall p_1, p_2, p_3, n.\ \left[p_1 \in \mathfrak{R}(p) \wedge p_1 \xrightarrow{?n} p_2 \wedge p_1 \xrightarrow{?n} p_3 \Rightarrow p_2 \ \underline{\leftrightarrow}^{\sqcup}_{\mathbf{b}}\ p_3\right].$$
$$\forall s_1, s_2, s_3, m.\ \left[s_1 \in \mathfrak{R}(s) \wedge s_1 \xrightarrow{?m} s_2 \wedge s_1 \xrightarrow{?m} s_3 \Rightarrow s_2 \ \underline{\leftrightarrow}^{\sqcup}_{\mathbf{b}}\ s_3\right].$$

  Clearly, the above definition has the following advantage over Definition 4.6: one has check the above conditions on the local processes p, s, while the conditions of Definition 4.6 has to be checked on the composed system, i.e., $p \parallel s$. But, note that the above formulation is stronger than Definition 4.6 because if $p_1 \in \mathfrak{R}(p)$ then it is not necessary that this local state is also reachable in the context $p_1 \parallel \_$.

- One of the sufficient conditions [36, Lemma 5.3] required for a process p to preserve the *foam rubber wrapper principle* is: if every reachable state p' has a trace $?m.!n$ to p'', i.e., $p' \xrightarrow{?m.!n} \!\!\!\!\rightarrow\ p''$, then $p' \xrightarrow{!n.?m} \!\!\!\!\rightarrow\ p''$. We conjecture that this property can be discarded from [36] if the half-duplex mechanism is used.

- Finally, the alphabet of the synchronous systems studied in [36] do not contain external actions. This makes it impossible to desynchronise a network of synchronously communication processes (see Section 6.2).

Recently, the authors in [67] (and its companion paper [71]) also study the conditions under which synchronous interactions can be implemented using asynchronous interactions in the context of π-calculus and Petri nets. They concluded that any good[3] encoding of synchrony within a purely asynchronous setting introduces *causal dependencies*; irrespective of the representation of concurrent systems in either the π-calculus, or Petri nets.

[3] in the sense of Gorla's five criteria for language comparison, see [41] for details.

Consider the process terms $p = !m + ?n$ and $s = !n + ?m$, where $+$ is the alternative composition operator on process terms. Indeed, behaviour of the synchronous system $p \parallel s$ is a choice between the communication actions m, n, i.e., $p \parallel s = m + n$. However, to simulate this behaviour in an asynchronous setting one has to introduce a lock. This is due to the possibility of executing the alternative $!n$ (or $!m$) after the execution of the alternative $!m$ (or $!n$). In [67], the authors proposed a 'sum lock' (a syntactical approach) to simulate a synchronous interaction in the asynchronous setting, whereas we introduce causality in the asynchronous systems by using the half-duplex mechanism (a semantical approach).

Another way to establish the correctness of an asynchronous system (when queues are used as buffers) is by applying the techniques from the field of CFSM. Arguably, the most important result in this field is by Brand and Zafiropulo [22] which states that CFSM are Turing complete. Consequently, properties such as reachability of states, unspecified reception, and model checking against a temporal logic all are undecidable for a CFSM.

In the literature of CFSM, the main focus is to impose restrictions on the local automata such that the above properties becomes decidable. For instance, if the channel contents are piecewise regular then the reachability of states in an asynchronous system is decidable [40]. Informally, piecewise regular languages [40] are those recognised by nondeterministic automaton whose only nontrivial strongly connected components are states with self-loops.

Cécé and Finkel in [24] studied a subclass of CFSM's, which satisfies the following half-duplex property: every reachable state in a CFSM has atmost one nonempty queue. It was shown that two finite state automata communicating over unbounded queues, but with half-duplex property have a recognizable reachability set. In other words, reachability of states in such asynchronous systems was shown to be decidable.

Furthermore, model checking the computational tree logic CTL* [4] formulae on such systems was shown to be undecidable in [24]. Through the desynchronisation technique developed here, we have also characterised a class of asynchronous systems, where model checking the formulae of CTL*-X (computational tree logic without the next operator) is decidable. This is due to a result by De Nicola and Vaandrager [28], where it is established that a divergence blind version of stuttering equivalence on the CTL* formulae coincides with the branching bisimulation equivalence on the processes.

In process algebraic approaches [43, 51], there is yet another abstraction scheme to verify an asynchronous system. Fundamentally, this abstraction scheme assumes that the communicating processes p, s have their local input

[4] CTL* is a logic consisting of both branching time and linear time operators, see [11] for details.

and output buffers. Then, a synchronous interaction of a message (say, m) can be simulated in the following way:

1. The sender performs a send message $!m$ by updating its output buffer.
2. Then, the semantics ensures that a message m from the sender's output buffer is transferred to the receiver's input buffer.
3. Finally, a receiver can read the message $?m$ from its input buffer at its own discretion.

Furthermore, the send and receive messages are made hidden to study the equivalence between the synchronous system and its asynchronous version [43]. Note that this abstraction scheme (like the abstraction schemes $\mathbf{A}_1, \mathbf{A}_2$, and $\mathbf{A}_4$) fails to preserve deterministic choice between any two messages of a local process. Moreover, the invisible transitions induced by this abstraction scheme are non-inert modulo branching bisimulation. As an example, the interested reader is referred back to the synchronous system of Example 3.1.

In hindsight, our results indicate that the study of desynchronisability should no longer focus on the properties one needs to retain equivalence of behavior in a certain communication context, but rather should focus on changing the communication context in such a way that these properties actually become attainable.

4.5 Conclusion

We conclude this chapter by recapitulating the main results obtained here. We started out with a quest to find a weaker set of conditions for desynchronisability than obtained in Chapter 3 and observed that two factors, namely, choice of buffers and choice of abstraction schemes play a crucial role in achieving the above goal.

We discarded bags in favour of queues because bags allow more behaviour and thus, inherently lead to an extra condition called the reordering property (Definition 3.14). The abstraction scheme $\mathbf{A}_3$ was chosen because it is the abstraction scheme in which the output messages of the local processes p, s remain observable. As a result, the external choice between the messages of processes p, s remains intact. While in case of the other abstraction scheme, these external choices present in the synchronous system are not preserved by the construction of an asynchronous system as they get transformed into internal choices (see Subsection 4.1.2).

Another property which was argued to be difficult to establish in practice is the (M_p, M_s) diamond property (commonly known as diamond property

in the desynchronisation literature cf. [13, 17, 36, 43, 73]). To this end, we introduced the half-duplex mechanism that prevents the diamond property on the output messages of the distinct local processes. In summary, we showed that input-determinism modulo $\leftrightarrows^{\sqcup}_{\mathbf{b}}$, well-posedness, and strong E-independence modulo $\leftrightarrows^{\sqcup}_{\mathbf{b}}$ are sufficient and necessary for desynchronisation of any concrete synchronous systems modulo branching bisimulation (Theorem 4.15). Moreover, we also showed that well-posedness and E-independence modulo $\sim^{\sqcup}_{\mathbf{c}}$ (i.e., by dropping the input-determinism property) are sufficient and necessary for desynchronisation of any concrete synchronous systems modulo contra-simulation (Theorem 4.21).

Consequently, in the presence of abstraction scheme $\mathbf{A}_3$ and half-duplex queues, we eliminated two sufficient conditions from Chapter 3; namely, the condition M_p-singular and (M_p, M_s)-diamond property. Furthermore, it can be inferred that a synchronous system synthesised using supervisory control theory is desynchronisable modulo $\simeq^{\sqcup}$ by construction when abstraction scheme $\mathbf{A}_3$ and half-duplex queues are used. This is because the synthesis technique of Section 3.4 always result in well-posed combination of a plant and its supervisor. The following theorem states this observation in a formal way.

Theorem 4.24. *Let $p \parallel s$ be a synchronous system synthesised using the supervisory control theory of Ramadge and Wonham. Let $s' = \rho_g(p \parallel s)$, where g is the renaming function defined on Page 34. Then,*

$$p \parallel s \simeq^{\sqcup} p \parallel s' \simeq^{\sqcup} \nabla_3(p \,|[\epsilon, \epsilon]|_{\mathrm{h}}\, s').$$

Unfortunately, such a result does not hold if the restriction of determinism is dropped from the processes p, s. However, we claim that our characterisation of desynchronisation modulo $\simeq^{\sqcup}$ is effective when the given synchronous system is finite because the reachability of states and verifying $\simeq^{\sqcup}$ is decidable for finite labelled transition systems. Moreover, Corollaries 4.16, 4.17, and 4.22 can be used for desynchronisation modulo $\simeq^{\sqcup}$ in case the necessary conditions (Definitions 4.6 and 4.10) are computationally hard to establish in practice.

Chapter 5

Desynchronisation of the pusher-lift system

In this chapter, we apply the desynchronisation techniques developed of Chapter 3 and Chapter 4 to desynchronise the pusher-lift system [75]. To this end, we use the Supremica [1] tool to automatically synthesise a supervisor. Later, we use the equivalence checker from the mCRL2 tool-set [45] to verify the branching bisimulation equivalence between the synchronous system and its asynchronous version. In summary, the work-flow depicted in Figure 5.1 is followed to desynchronise the pusher-lift system. The transformations which are done manually are indicated by dashed lines in Figure 5.1.

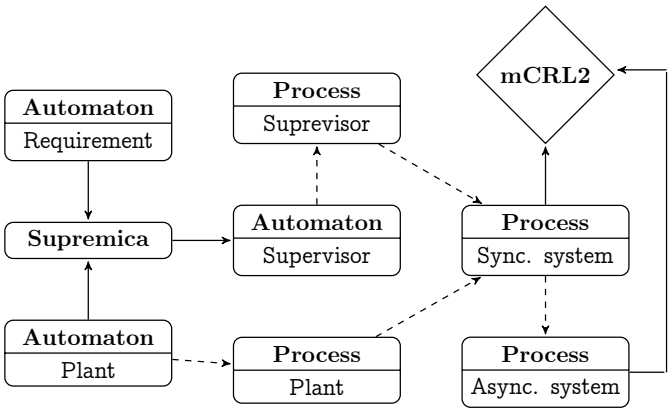


FIGURE 5.1: Work-flow followed in Chapter 5.

We begin with the modelling of a plant and a requirement in the form of automata, whose alphabets contain only controllable and uncontrollable actions.

Then, Supremica tool is used to generate a supervisor in the form of an automaton for this plant and requirement models. Subsequently, we manually transformed the plant and supervisor automata to the mCRL2 specification language as processes. At this stage, we distinguish the action labels of the plant and its supervisor as either send message, or receive message. Recall that the uncontrollable actions are modelled as send messages in a plant model and the controllable actions are modelled as send messages in a supervisor model. Then, we construct a synchronous system and its asynchronous versions as explained in Chapter 2. Finally, the transitions systems of two systems are compared with the help of equivalence checker from the mCRL2 tool-set.

5.1 The pusher-lift system

The pusher-lift system is adopted from the lecture notes on supervisory control course [75]. The overall system consists of three components: first, a lift that can go up and down; second, a pusher that can retract and extend; third, a product holder (see Figure 5.2). The incoming arrow in Figure 5.2

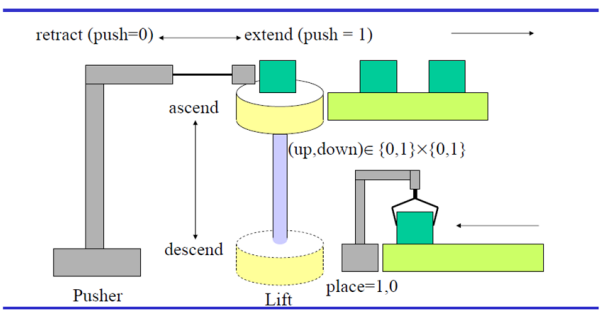


FIGURE 5.2: The pusher-lift system [75].

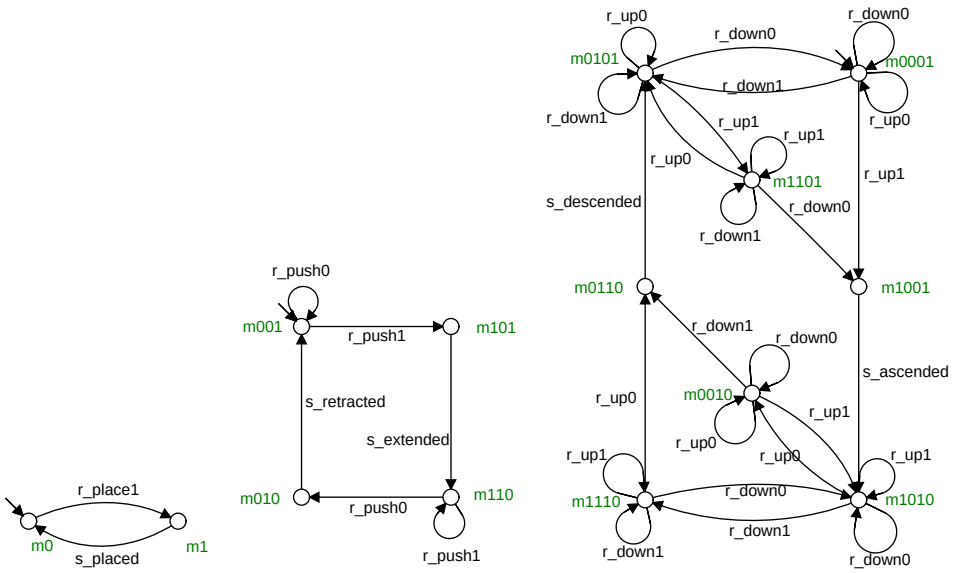
indicates the arrival of a product in the system. The product holder places it on the lift, whenever the lift is in descend position. Finally, the outgoing arrow in Figure 5.2 indicates the processed products are pushed by the pusher, whenever the lift is in ascended position.

The hardware components – the pusher, the product holder, and the lift – are to be controlled by a supervisor with the manipulation of the following variables (control signals) [75]:

- The expression $push{=}1$ signals the pusher to extend. Similarly, the expression $push{=}0$ signals the pusher to retract.
- The expression $down{=}0$ and $up = 1$ signals the lift to ascend. The expression $down{=}1$ and $up = 0$ signals the lift to descend. The lift remains in its position when $down{=}up$.

- Finally, the expression *place=1* signals the product holder to place a product on the lift.

We model the above expressions as actions by dropping the equality symbol. For instance, we specify the expression *push=1* as *push1*. The plant model of this system is the interleaving of product model (Figure 5.3(a)), pusher model (Figure 5.3(b)), and lift model (Figure 5.3(c)). In Figure 5.3, we use



(a) Product model (b) Pusher model (c) Lift model

FIGURE 5.3: The plant model of the pusher-lift system.

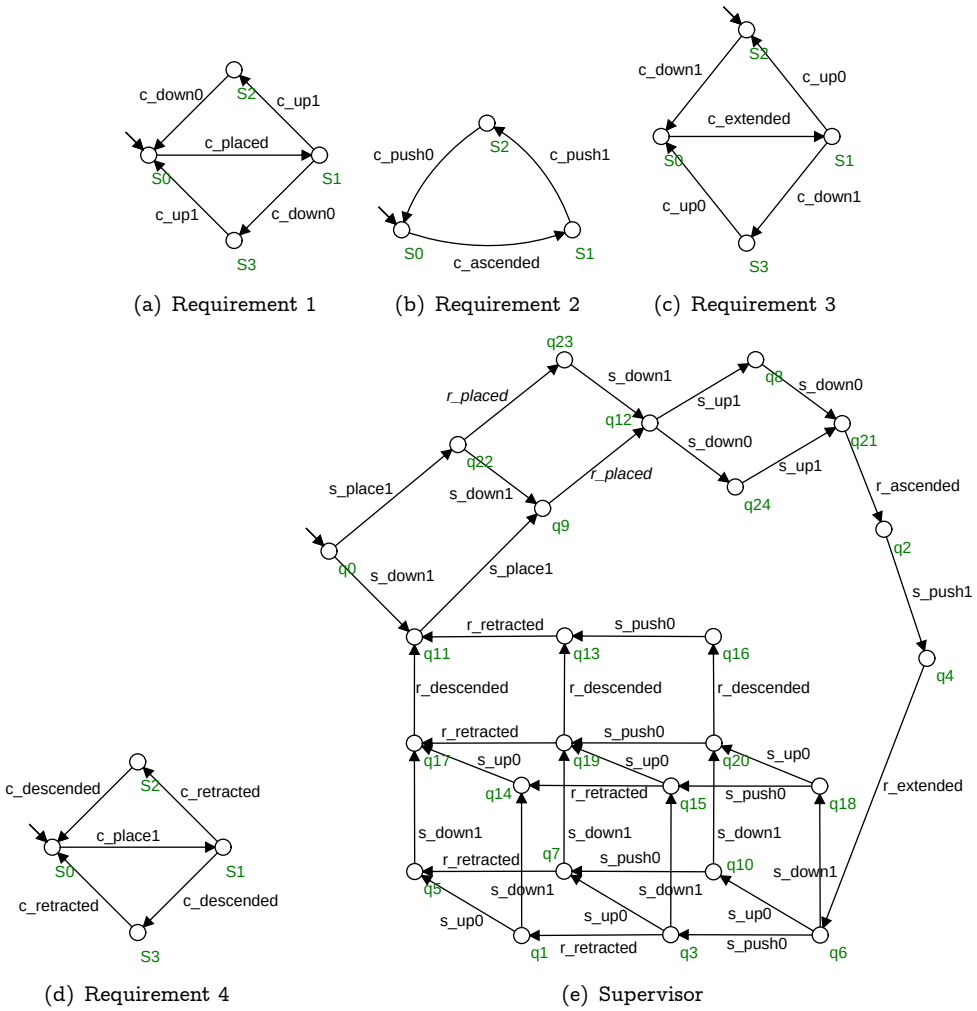
the prefixes `s_`, `r_`, and `c_` to denote a send message, a receive message, and communication of a message, respectively, because the Supremica toolset forbids the use of symbols !, ?, in the specification of an automaton. The different requirement models and the synthesised supervisor model of this system are shown in Figure 5.4.

5.1.1 Analysis

In the following, we first apply the desynchronisation techniques of Chapter 3 and then, apply the techniques of Chapter 4 on this case-study.

Chapter 3

Table 5.1 shows the results obtained when desynchronising the pusher-lift case-study using the techniques of Chapter 3. The second and third columns (from



(a) Requirement 1 (b) Requirement 2 (c) Requirement 3

(d) Requirement 4 (e) Supervisor

FIGURE 5.4: The requirement models and the supervisor model of the pusher-lift.

left) of Table 5.1 shows the number of states and the number of transitions generated by the respective processes modulo $\leftrightarrows_{\mathbf{b}}$. The column entitled as 'Size' indicates the size of the queues, or the bags used in the construction of an asynchronous system. The column with the heading $\leftrightarrows_{\mathbf{b}}$ shows whether the asynchronous systems $(\nabla_1(p\,|[\epsilon,\epsilon]|\,s)$ and $\nabla_1(p\,|\{\epsilon,\epsilon\}|\,s))$ are branching bisimilar to the synchronous system $p \parallel s$. The second column (from right) in Table 5.1 shows that the plant and the supervisor of this case-study are well-posed. This fact was established by using the "inverse-controllability" feature of Supremica tool (Theorem 3.21). The final column shows which of the systems deadlocks.

In the case of bags as buffers, the asynchronous system $\nabla_1(p\,|\{\epsilon,\epsilon\}|\,s)$ has

Processes	States	Transitions	Size	$\leftrightarrows_{\mathbf{b}}$	Well-posed	Deadlock
$p \parallel s$	25	39	-	-	Yes	No
$\nabla_1(p \,\|[\epsilon,\epsilon]\|\, s)$	31	50	2	No	-	No
$\nabla_1(p \,\|\{\varepsilon,\varepsilon\}\|\, s)$	122	251	2	No	-	Yes

TABLE 5.1: The results obtained when applying techniques of Chapter 3.

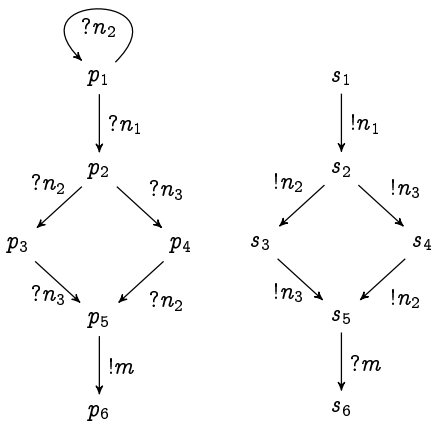


FIGURE 5.5: A toy example illustrating deadlock in $\nabla_1(p_1 \,|\{\varepsilon,\varepsilon\}|\, s_1)$.

two deadlock traces as pointed out by the tool 'lps2lts' from the mCRL2 tool-set. In Example 5.1, we simulate a situation that caused a deadlock in the asynchronous system $\nabla_1(p \,|\{\varepsilon,\varepsilon\}|\, s)$. This is possible by understanding the deadlocked situations caused by these two deadlock traces with the help of 'xsim' tool from the mCRL2 tool-set.

Example 5.1. *Consider the transition systems of a plant and its supervisor as shown in Figure 5.5. Clearly, the synchronous system $p_1 \parallel s_1$ is deadlock free; however, the trace $n_1.n_2.n_3$ leads to a deadlock in the asynchronous system $\nabla_1(p_1 \,|\{\varepsilon,\varepsilon\}|\, s_1)$. This is because the sequence of outputs $!n_1.!n_3.!n_2$ from the supervisor s_1 can be read as $?n_2.?n_1.?n_3$ by the plant p_1. Observe that the synchronous system $p_1 \parallel s_1$ does not satisfy the reordering property (Definition 3.14), which is one of the sufficient properties for desynchronisability when bags are used as buffers.*

In view of the above discussion, we claim that the synchronous system of the pusher-lift system do not satisfy the reordering property.

Furthermore, if we remove the self-loops from the lift model (Figure 5.3(c)) and keep the other models of plant and supervisor unchanged, then the modified synchronous system is desynchronisable modulo branching bisimulation in the presence of bags. However, such a result is consistent only if the old synchronous system and the modified synchronous system are branching bisimilar. Thus, the following tasks were verified using the equivalence checker of

Processes	States	Transitions	Size	$\leftrightarrow_{\mathbf{b}}$	Well-posed	Deadlock
$p \parallel s$	25	39	-	-	Yes	No
$p' \parallel s$	25	39	-	Yes	Yes	No
$\nabla_1(p' \mid [\epsilon, \epsilon] \mid s)$	31	50	2	No	-	No
$\nabla_1(p' \mid \{\varepsilon, \varepsilon\} \mid s)$	25	39	2	Yes	-	No

TABLE 5.2: The results for the modified Pusher-lift system, where p' is the modified plant model.

mCRL2 tool-set. The results obtained for the modified synchronous system is shown in Table 5.2.

1. The old synchronous system and the modified synchronous system were branching bisimilar (actually, the two systems are even isomorphic).
2. The modified synchronous system is desynchronisable with respect to abstraction scheme $\mathbf{A}_1$ and bags bounded by size 2.
3. Finally, the modified synchronous system is not desynchronisable with respect to abstraction scheme $\mathbf{A}_1$ and queues bounded by size 2.

Note that the column with the heading $\underline{\leftrightarrow}^{\sqcup}_{\mathbf{b}}$ in Table 5.2 shows: whether the modified synchronous system $p' \parallel s$ (or, the two asynchronous systems $\nabla_1(p' \mid [\epsilon, \epsilon] \mid s), \nabla_1(p' \mid \{\varepsilon, \varepsilon\} \mid s)$) and the old synchronous system $p \parallel s$ are branching bisimilar?

Chapter 4

While applying the desynchronisation techniques of Chapter 3, we already established that the original plant model p (or, the modified plant model p') and the supervisor model s of the pusher-lift system are well-posed. Moreover, from Theorem 4.24 we know that the synchronous systems $p \parallel s$, $p' \parallel s$ are desynchronisable under the abstraction scheme $\mathbf{A}_3$ and half-duplex queues. Thus, in the context of queues, the above experimentation supports our claim that the half-duplex mechanism and the abstraction scheme $\mathbf{A}_3$ are more reasonable restrictions to achieve desynchronisation modulo branching bisimulation than the sufficient conditions of Chapter 3.

Furthermore, we also conducted test for desynchronisability modulo branching bisimulation on the modified synchronous system $p' \parallel s$ by introducing bags and the abstraction scheme $\mathbf{A}_3$[1]. Unfortunately, the synchronous system $p' \parallel s$ was not desynchronisable in the presence of bags and the abstraction $\mathbf{A}_3$. This suggests that the abstraction scheme $\mathbf{A}_1$ outperforms $\mathbf{A}_3$ in case of bags.

[1] In order to compare with the desynchronisation result obtained in the previous subsection.

Chapter 6

Some final remarks on desynchronisation

So far we have developed a reasonable desynchronisation technique for synchronous systems that consists of two communicating concrete processes. In this chapter, we extend this technique in three orthogonal directions.

Admittedly, the choice for half-duplex communication is an odd one from the perspective of efficiency. The half-duplex protocol essentially makes components wait for each other, which makes communication slow. In Section 6.1, we sketch a first step to remedy this by recognizing when actions are independent of each other. Intuitively, independent messages satisfy the diamond property and can therefore be processed in a full-duplex way.

Often the model of embedded system contain more than two communicating processes. In Section 6.2, we show how to desynchronise a network of synchronously communicating processes in a compositional way.

Finally, we consider the desynchronisation of synchronous systems whose alphabet contains the invisible action τ. In Section 6.3, we show that the conditions of Chapter 4 are also sufficient for desynchronisability of non-concrete synchronous systems.

6.1 A mix of half-duplex and full-duplex mechanisms

The goal of this section is to answer: how can a synchronous system be desynchronised by relaxing the half-duplex restriction on certain output messages of a sender without introducing any further sufficient conditions? Observe that

this question can be reformulated as: when is it safe to execute a send message by a sender in an asynchronous system while its input queue is non-empty? Here, by the word 'safe' we mean preserving the branching bisimulation equivalence between a synchronous system and its asynchronous version.

To address this issue, consider the equation $p_1 \parallel s_1 \leftrightarrows^{\sqcup}_{\mathbf{b}} \nabla_3(p_1 \,|[\epsilon,\epsilon]|\, s_1)$ with the transitions $p_1 \xrightarrow{!m} p_2$ and $s_1 \xrightarrow{!n} s_2$. Then, by the semantics of an asynchronous system we can derive the following transitions:

$$\nabla_3(p_1 \,|[\epsilon,\epsilon]|\, s_1) \xrightarrow{m} \nabla_3(p_2 \,|[\epsilon,m]|\, s_1) \xrightarrow{n} \nabla_3(p_2 \,|[n,m]|\, s_2),$$
$$\nabla_3(p_1 \,|[\epsilon,\epsilon]|\, s_1) \xrightarrow{n} \nabla_3(p_1 \,|[n,\epsilon]|\, s_2) \xrightarrow{m} \nabla_3(p_2 \,|[n,m]|\, s_2).$$

We notice that the messages m, n, which are output with respect to processes p, s execute in an independent way and form a diamond of transitions labelled with m, n. Furthermore, if the synchronous system $p_1 \parallel s_1$ is branching bisimilar with its asynchronous version, such a diamond must also exist in the synchronous system $p_1 \parallel s_1$. Lemma 6.2 formally proves this fact.

Definition 6.1. Let $m \in M_p, n \in M_s$. A process $p_1 \parallel s_1$ satisfies the predicate $\Diamond(m,n)$, notation $p_1 \parallel s_1 \models \Diamond(m,n)$, iff the following condition holds:

$$\forall p_2, s_2, p_3, s_3.\, \Big[\Big(p_1 \parallel s_1 \xrightarrow{m} p_2 \parallel s_2 \wedge p_1 \parallel s_1 \xrightarrow{n} p_3 \parallel s_3\Big) \Rightarrow$$
$$\exists p_4, s_4, p_5, s_5.\, \Big[p_2 \parallel s_2 \xrightarrow{n} p_4 \parallel s_4 \wedge p_3 \parallel s_3 \xrightarrow{m} p_5 \parallel s_5 \wedge p_4 \parallel s_4 \leftrightarrows^{\sqcup}_{\mathbf{b}} p_5 \parallel s_5\Big]\Big].$$

Observe the difference between the above condition and Definition 3.9. In essence, the above formulation is nothing but the diamond property (Definition 3.9) modulo branching bisimulation.

Lemma 6.2. *Let p, s be concrete processes such that $p \parallel s \leftrightarrows^{\sqcup}_{\mathbf{b}} \nabla_3(p \,|[\epsilon,\epsilon]|\, s)$. Suppose, $m \in M_p, n \in M_s$. Then,*

$$\forall p_1, s_1, m, n.\Big[p_1 \parallel s_1 \in \mathfrak{R}(p \parallel s) \Rightarrow p_1 \parallel s_1 \models \Diamond(m,n)\Big].$$

Proof. We need to show that if $p_1 \parallel s_1 \in \mathfrak{R}(p \parallel s)$, $p_1 \parallel s_1 \xrightarrow{m} p_2 \parallel s_2$ and $p_1 \parallel s_1 \xrightarrow{n} p_3 \parallel s_3$, then $\exists p_4, s_4.\, \Big[p_2 \parallel s_2 \xrightarrow{n} p_4 \parallel s_4 \wedge p_3 \parallel s_3 \xrightarrow{m} p_4 \parallel s_4\Big]$. So assume the antecedent in the above implication. Then, from semantics we have $p_1 \xrightarrow{!m} p_2$, $s_1 \xrightarrow{?m} s_2$, $p_1 \xrightarrow{?n} p_3$, and $s_1 \xrightarrow{!n} s_3$. From Lemma 4.4 we know that $p_1 \parallel s_1 \leftrightarrows^{\sqcup}_{\mathbf{b}} \nabla_3(p_1 \,|[\epsilon,\epsilon]|\, s_1)$. Using the above derived transitions we get the transitions depicted as solid lines in Figure 6.1.

Again, applying Lemma 4.4 we get $p_i \parallel s_i \leftrightarrows^{\sqcup}_{\mathbf{b}} \nabla_3(p_i \,|[\epsilon,\epsilon]|\, s_i)$, for $i \in \{2,3\}$. And from Lemma 4.2 we know that the invisible transitions are inert modulo $\leftrightarrows^{\sqcup}_{\mathbf{b}}$ whenever the local processes p, s are concrete. Thus,

$$p_2 \parallel s_2 \leftrightarrows^{\sqcup}_{\mathbf{b}} \nabla_3(p_2 \,|[\epsilon,m]|\, s_1) \text{ and } p_3 \parallel s_3 \leftrightarrows^{\sqcup}_{\mathbf{b}} \nabla_3(p_1 \,|[n,\epsilon]|\, s_3).$$

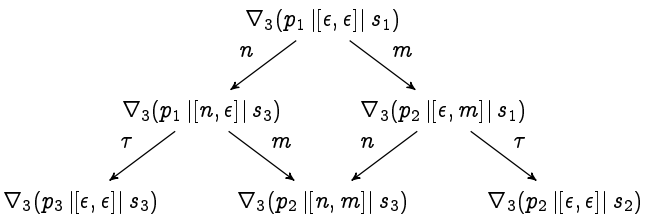


FIGURE 6.1: Partial transition system used in Lemma 6.2.

And, from Proposition 2.7 we get

$$\exists p_4, s_4.\left[p_2 \parallel s_2 \xrightarrow{n} p_4 \parallel s_4 \wedge p_4 \parallel s_4 \leftrightarrows^{\sqcup}_{\mathbf{b}} \nabla_3(p_2 \,|[n,m]|\, s_3)\right].$$

Likewise, $\exists p_5, s_5.\left[p_3 \parallel s_3 \xrightarrow{m} p_5 \parallel s_5 \wedge p_5 \parallel s_5 \leftrightarrows^{\sqcup}_{\mathbf{b}} \nabla_3(p_2 \,|[n,m]|\, s_3)\right]$. Finally, from transitivity we get the desired result $p_4 \parallel s_4 \leftrightarrows^{\sqcup}_{\mathbf{b}} p_5 \parallel s_5$. □

In view of Lemma 6.2, sending a message m from one local process should be considered safe in an asynchronous system, whenever it does not disable the sending of a message n from the other local process. Moreover, any order of execution, i.e., traces $m.n$ and $n.m$, leads to behaviourally equivalent states. This leads to the following definition, which identifies pair of independent messages in a synchronous system that can be safely executed.

Definition 6.3. Let $p \parallel s$ be a synchronous system. Define an *independence* relation $\mathcal{I} \subseteq M_s \times M_p$:

$$\mathcal{I} = \Big\{(n,m) \mid \forall p_1, s_1.[p_1 \parallel s_1 \in \mathfrak{R}(p \parallel s) \Rightarrow p_1 \parallel s_1 \models \Diamond(m,n)]\Big\}.$$

On the contrary, suppose in a synchronous system the above condition between messages m, n is impossible to satisfy then from Chapter 4 we know that these messages should be executed in half-duplex way in the asynchronous system.

Thus, the above two opposing observations demands a semi-duplex buffering strategy for an efficient implementation of an asynchronous system from the perspective of performance.

To this end, we propose that at any time it is safe to send a message m by a local process in an asynchronous system if the message m and the messages in the input queue of the local process are independent in the sense of Definition 6.3. Formally, we lift the relation $\mathcal{I}$ of independent actions to a relation $\widehat{\mathcal{I}} \subseteq {M_s}^* \times {M_p}^*$ of sequences of independent actions:

$$(\mu,\nu) \in \widehat{\mathcal{I}} = \forall \alpha_1, \alpha_2.\Big[(\alpha_1 \in \mu \wedge \alpha_2 \in \nu) \Rightarrow (\alpha_1,\alpha_2) \in \mathcal{I}\Big].$$

$$\frac{p \xrightarrow{!m} p',\ (\mu, m) \in \widehat{\mathcal{I}}}{p\,|[\mu,\nu]|_{\mathcal{I}}\, s \xrightarrow{!m} p'\,|[\mu,\nu.m]|_{\mathcal{I}}\, s} \quad (22) \qquad \frac{s \xrightarrow{!n} s',\ (n, \nu) \in \widehat{\mathcal{I}}}{p\,|[\mu,\nu]|_{\mathcal{I}}\, s \xrightarrow{!n} p\,|[\mu,n]|_{\mathcal{I}}\, s'} \quad (23)$$

$$\frac{p_1\,|[\mu_1,\nu_1]|\, s_1 \xrightarrow{\alpha} p_2\,|[\mu_2,\nu_2]|\, s_2,\ \alpha \notin !M_p \cup !M_s}{p_1\,|[\mu_1,\nu_1]|_{\mathcal{I}}\, s_1 \xrightarrow{\alpha} p_2\,|[\mu_2,\nu_2]|_{\mathcal{I}}\, s_2} \quad (24)$$

TABLE 6.1: SOS rules for asynchronous parallel composition with a mix of half-duplex and full-duplex communication mechanisms.

Now, a buffered system with a mix of half-duplex and full-duplex mechanisms is denoted by $p\,|[\mu,\nu]|_{\mathcal{I}}\, s$, where $\mu \in {M_s}^*, \nu \in {M_p}^*$. The operational rules for the family of operators $\_\,|[\mu,\nu]|_{\mathcal{I}}\,\_$, parameterised by $\mu \in {M_s}^*, \nu \in {M_p}^*$, are given in Table 6.1. Rule 22 states that a sender (say, p) can execute an output message $!m$, whenever its input queue content μ and the message m are independent, i.e, $(\mu, m) \in \widehat{\mathcal{I}}$; otherwise, the output message $!m$ is executed in the half-duplex way. Rule 23 is analogous to Rule 22. Rule 24 states that the remaining rules of Table 2.2 are just inherited.

Lemma 6.4. *Let p_1, s_1 be any two concrete and well-posed processes. Suppose $p_1 \parallel s_1$ is input-deterministic modulo $\leftrightarrow^{\sqcup}_{\mathbf{b}}$ and strong E-independent modulo $\leftrightarrow^{\sqcup}_{\mathbf{b}}$. If $p_1 \parallel s_1 \xrightarrow{u}\!\!\!\!\rightarrow p_2 \parallel s_2$, $p_1 \parallel s_1 \xrightarrow{v}\!\!\!\!\rightarrow p_3 \parallel s_3$, $u \in (M_s \cup E_s)^*$, $v \in (M_p \cup E_p)^*$, and $(\bar{u}, \bar{v}) \in \widehat{\mathcal{I}}$, then*

$$\exists p_4, s_4, p_5, s_5.\ \left[p_2 \parallel s_2 \xrightarrow{v}\!\!\!\!\rightarrow p_4 \parallel s_4 \wedge p_3 \parallel s_3 \xrightarrow{u}\!\!\!\!\rightarrow p_5 \parallel s_5 \wedge p_4 \parallel s_4 \leftrightarrow^{\sqcup} p_5 \parallel s_5\right].$$

Proof. We first prove the following claim if $p_1 \parallel s_1 \xrightarrow{u}\!\!\!\!\rightarrow p_2 \parallel s_2$, $p_1 \parallel s_1 \xrightarrow{\alpha} p_3 \parallel s_3$, and $(\bar{u}, \bar{\alpha}) \in \widehat{\mathcal{I}}$, then

$$\exists p_4, s_4, p_5, s_5.\ \left[p_2 \parallel s_2 \xrightarrow{\alpha} p_4 \parallel s_4 \wedge p_3 \parallel s_3 \xrightarrow{u}\!\!\!\!\rightarrow p_5 \parallel s_5 \wedge p_4 \parallel s_4 \leftrightarrow^{\sqcup} p_5 \parallel s_5\right].$$

Without loss of generality, assume that $p_1 \parallel s_1 \xrightarrow{u}\!\!\!\!\rightarrow p_2' \parallel s_2' \xrightarrow{\alpha'} p_2 \parallel s_2$. Then, by induction hypothesis we have

$$\exists p_4', s_4', p_5', s_5'.\ \left[p_2' \parallel s_2' \xrightarrow{\alpha} p_4' \parallel s_4' \wedge p_3 \parallel s_3 \xrightarrow{u}\!\!\!\!\rightarrow p_5' \parallel s_5' \wedge p_4' \parallel s_4' \leftrightarrow^{\sqcup} p_5' \parallel s_5'\right].$$

We identify the following cases based on the types of α, α'.

1. Let $\alpha = e$, for some $e \in E_p$, $\alpha' = e'$, for some $e' \in E_s$. Trivial!
2. Let $\alpha = e$, for some $e \in E_p$, $\alpha' = n$, for some $n \in M_s$. The single step transitions from the above inductive hypothesis are shown as solid lines

in Figure 6.2. Note that $s_2' = s_4'$ because of Rule 3. From the transition $p_2' \parallel s_2' \xrightarrow{n} p_2 \parallel s_2$ we have $s_2' \xrightarrow{!n} s_2$. But, p_1, s_1 are well-posed processes,

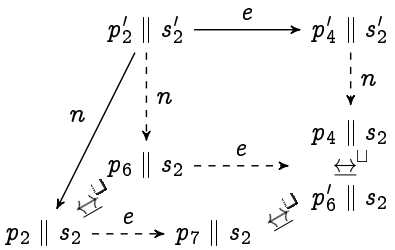


FIGURE 6.2: Case 2 in Lemma 6.4.

so assume a well-posed relation $\mathcal{W}$ such that $(p_1, s_1) \in \mathcal{W}$. And from Proposition 3.6 we have $(p_4', s_2') \in \mathcal{W}$. By applying Definition 3.5 we get $\exists p_4. \left[p_4' \xrightarrow{?n} p_4\right]$. Thus, we get (see Figure 6.2) $p_4' \parallel s_2' \xrightarrow{n} p_4 \parallel s_2$.

Now, applying strong version of Definition 4.10 at the state $p_2' \parallel s_2'$ we get (see Figure 6.2) $\exists p_6, p_6'. \left[p_2' \parallel s_2' \xrightarrow{n} p_6 \parallel s_2 \xrightarrow{e} p_6' \parallel s_2 \wedge p_6' \parallel s_2 \leftrightarrow^{\sqcup} p_4 \parallel s_2\right]$. Under concreteness assumption input determinism gives us $p_2 \parallel s_2 \leftrightarrow^{\sqcup} p_6 \parallel s_2$. And from transfer conditions of strong bisimulation we get

$$\exists p_7. \left[p_2 \parallel s_2 \xrightarrow{e} p_7 \parallel s_2 \wedge p_6' \parallel s_2 \leftrightarrow^{\sqcup} p_7 \parallel s_2\right].$$

Thus, by transitivity we get $p_7 \parallel s_2 \leftrightarrow^{\sqcup} p_4 \parallel s_2$. Recall from induction hypothesis that $p_4' \parallel s_2' \leftrightarrow^{\sqcup} p_5' \parallel s_5'$. Again, from the transfer conditions of strong bisimulation we get $\exists p_5, s_5. \left[p_5' \parallel s_5' \xrightarrow{n} p_5 \parallel s_5 \wedge p_4 \parallel s_2 \leftrightarrow^{\sqcup} p_5 \parallel s_5\right]$. Finally, by transitivity we get the desired result $p_7 \parallel s_2 \leftrightarrow^{\sqcup} p_5 \parallel s_5$.

3. Let $\alpha = m$, for some $m \in M_p$, $\alpha' = e'$, for some $e' \in E_s$. Similar to the previous case.

4. Let $\alpha = m$, for some $m \in M_p$, $\alpha' = n$, for some $n \in M_s$. Similar to Case 2, use Definitions 6.1, 6.3 instead of Definition 4.10.

Likewise, the main statement can be proved by assuming $v = v'.\alpha$ for some $\alpha \in E_p \cup M_p$. □

The next theorem states that a synchronous system satisfying well-posedness, input-determinism, and strong E-independence properties is desynchronisable modulo branching bisimulation even if semi-duplex buffering strategy is used instead of half-duplex mechanism.

Theorem 6.5. *Let p, s be concrete processes. If the synchronous system $p \parallel s$ is well-posed, input-deterministic modulo $\leftrightarrow^{\sqcup}_{\mathbf{b}}$, and strong E-independent modulo $\leftrightarrow^{\sqcup}_{\mathbf{b}}$, then $p \parallel s \leftrightarrow^{\sqcup}_{\mathbf{b}} \nabla_3(p \,|[\epsilon, \epsilon]|_{\mathcal{I}}\, s)$.*

Proof. See Theorem C.1. □

Like the previous theorems on desynchronisability of Chapter 4, the above theorem is also independent of the size of queues.

Corollary 6.6. *Let p, s be concrete processes. If the synchronous system $p \parallel s$ is well-posed, input-deterministic modulo $=$, and strong E-independent modulo $=$, then $p \parallel s \leftrightarrows^{\sqcup}_{\mathbf{b}} \nabla_3(p\,|[\epsilon,\epsilon]|_{\mathcal{I}}\, s)$.*

Corollary 6.7. *Let p, s be concrete processes. If the synchronous system $p \parallel s$ is well-posed, input-deterministic modulo $\leftrightarrows^{\sqcup}_{\mathbf{b}}$, and strong E-independent modulo $\leftrightarrows^{\sqcup}_{\mathbf{b}}$, then $p \parallel s \leftrightarrows^{\sqcup}_{\mathbf{b}} \nabla_3(p\,|[\epsilon,\epsilon]|_{\mathrm{h}}\, s) \leftrightarrows^{\sqcup}_{\mathbf{b}} \nabla_3(p\,|[\epsilon,\epsilon]|_{\mathcal{I}}\, s)$.*

6.2 Desynchronising a network of synchronously communicating processes

Often, a model of an embedded system is built by the synchronous parallel composition of more than two local processes. Consider a network of three processes communicating synchronously $s_1 \parallel p \parallel s_2$, as shown in Figure 6.3. We assume that the set of external actions of the processes p, s_1, s_2 are pairwise disjoint, i.e., $E_p \cap E_{s_1} = \emptyset$, $E_p \cap E_{s_2} = \emptyset$, and $E_{s_1} \cap E_{s_2} = \emptyset$. Likewise, we assume the set of messages of the processes p, s_1, s_2 are also pairwise disjoint.

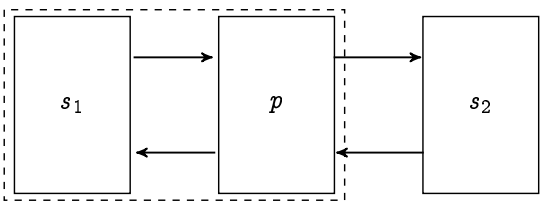


FIGURE 6.3: An illustration of a network of three processes communicating synchronously.

Conceptually, this network can be desynchronised in three steps:

- First, desynchronise the synchronous system $p_1 \parallel s_1$ captured by dashed lines in Figure 6.3, where $p_1 = \rho_{f_1}(p)$ and

$$f_1(\alpha) = \begin{cases} e_{!m}, & \text{if } \alpha = !m \wedge ?m \in \mathrm{Alph}(s_2), \\ e_{?n}, & \text{if } \alpha = ?n \wedge !n \in \mathrm{Alph}(s_2). \end{cases} . \tag{6.1}$$

  Intuitively, with the renaming function f_1 we mask the interactions between the process p and s_2. At this stage, we just want to decouple the synchronous interactions between the process p and s_1. Now, check for desynchronisability on the synchronous system $p_1 \parallel s_1$.

- Second, construct a new synchronous system $p_2 \parallel s_2$, where $p_2 = \rho_{f_2}(p_1 \parallel s_1)$ and

$$f_2(\alpha) = \begin{cases} f_1^{-1}(\alpha), & \text{if } \alpha = e_{!m} \vee \alpha = e_{?n}, \\ e_m, & \text{if } \alpha = m \wedge m \in \text{Alph}(p_1 \parallel s_1). \end{cases} . \tag{6.2}$$

  Informally, the renaming function f_2 enables the masked interactions between the process p, s_2, and renames the communication actions into external actions, which are generated by the synchronous execution of the processes p_1 and s_1.

- Finally, desynchronisation of the complete network can be attained as shown in the following derivation, where f is the renaming function defined as $f(e_m) = m$.

$$\begin{aligned} & (s_1 \parallel p) \parallel s_2 \\ \leftrightarrow\!\!\!\!\!\!\sqcup_{\mathbf{b}} \ & \rho_f(\rho_{f_2}(s_1 \parallel p_1) \parallel s_2), \text{ (Lemma 6.8)} \\ \leftrightarrow\!\!\!\!\!\!\sqcup_{\mathbf{b}} \ & \rho_f(\rho_{f_2}(\nabla_3(p_1 \,|[\epsilon,\epsilon]|_{\mathrm{h}}\, s_1)) \parallel s_2), \text{ (Desynchronisation of } s_1 \parallel p_1) \\ \leftrightarrow\!\!\!\!\!\!\sqcup_{\mathbf{b}} \ & \rho_f(p_2 \parallel s_2), \ \Big(p_2 = \rho_{f_2}(p_1 \parallel s_1) \leftrightarrow\!\!\!\!\!\!\sqcup_{\mathbf{b}} \rho_{f_2}(\nabla_3(p_1 \,|[\epsilon,\epsilon]|_{\mathrm{h}}\, s_1))\Big), \\ \leftrightarrow\!\!\!\!\!\!\sqcup_{\mathbf{b}} \ & \rho_f(\nabla_3(p_2 \,|[\epsilon,\epsilon]|_{\mathrm{h}}\, s_2)) \text{ (Desynchronisation of } p_2 \parallel s_2). \end{aligned}$$

  Note that in the last step of the above derivation we can also substitute the process $\nabla_3(p_2 \,|[\epsilon,\epsilon]|_{\mathrm{h}}\, s_2)$ by the process $\nabla_3(p_2 \,|[\epsilon,\epsilon]|_{\mathcal{I}}\, s_2)$ because of Corollary 6.7. As a result, on the one hand, we introduce between half-duplex queues between the processes p, s_1. On the other hand, we introduce a mix of half-duplex and full-duplex queues between the processes p, s_2. Such a choice in the design of this network can result in an efficient implementation of this network.

Alternatively, one can also first desynchronise the parallel composition of the processes p and s_2. Later, this desynchronised process under suitable renaming can be composed with the process s_1 to desynchronise the network $s_1 \parallel p \parallel s_2$.

Lemma 6.8. *Let p, s_1, s_2 be any three concrete processes with the process $s_1 \parallel p \parallel s_2$ representing the network in Figure 6.3. Let f_1, f_2 be the renaming functions as defined in Equation 6.1 and Equation 6.2, respectively. Then, $s_1 \parallel p \parallel s_2 \leftrightarrow\!\!\!\!\!\!\sqcup\ \rho_f(\rho_{f_2}(s_1 \parallel \rho_{f_1}(p)) \parallel s_2)$.*

Proof. Define a binary relation $\mathcal{S}$ in the following way:

$$\mathcal{S} = \Big\{ \Big(s_3 \parallel p_1 \parallel s_4, \rho_f(\rho_{f_2}(s_3 \parallel \rho_{f_1}(p_1)) \parallel s_4) \Big) \mid s_3 \parallel p_1 \parallel s_4 \in \mathfrak{R}(s_1 \parallel p \parallel s_2) \Big\}.$$

Next, we show that $\mathcal{S}$ is a bisimulation relation.

1. Let $s_3 \parallel p_1 \parallel s_4 \xrightarrow{\alpha} s_5 \parallel p_2 \parallel s_6$, $(s_3 \parallel p_1 \parallel s_4, \rho_f(\rho_{f_2}(s_3 \parallel \rho_{f_1}(p_1)) \parallel s_4)) \in \mathcal{S}$. Now, based on the type of α we get the following cases:

   (a) Let $\alpha \in E_p \cup E_{s_1} \cup E_{s_2}$. Trivial!

   (b) Let $\alpha = m$, for some $m \in M_p$. Then, this communication action is due to the synchronous interaction between the processes p_1 and s_3 (or s_4) because of the network topology in Figure 6.3.

      i. Interaction with s_3. Then, we have $p_1 \xrightarrow{!m} p_2$, $s_3 \xrightarrow{?m} s_5$, and $s_4 = s_6$. By definition of f_1 we get $\rho_{f_1}(p_1) \xrightarrow{!m} \rho_{f_1}(p_2)$ because $?m \in ?M_{s_1}$, and $?M_{s_1} \cap ?M_{s_2} = \emptyset$. Thus, $s_3 \parallel \rho_{f_1}(p_1) \xrightarrow{m} s_5 \parallel \rho_{f_1}(p_2)$. And, from definition of f_2 we get $\rho_{f_2}(s_3 \parallel \rho_{f_1}(p_1)) \xrightarrow{e_m} \rho_{f_2}(s_5 \parallel \rho_{f_1}(p_2))$. From Rule 3 we get $\rho_{f_2}(s_3 \parallel \rho_{f_1}(p_1)) \parallel s_4 \xrightarrow{e_m} \rho_{f_2}(s_3 \parallel \rho_{f_1}(p_1)) \parallel s_4$. Furthermore, applying definition of f we get $\rho_f(\rho_{f_2}(s_3 \parallel \rho_{f_1}(p_1)) \parallel s_4) \xrightarrow{m} \rho_f(\rho_{f_2}(s_5 \parallel \rho_{f_1}(p_2)) \parallel s_4)$. Finally, from the construction of $\mathcal{S}$ we conclude that

$$(s_5 \parallel p_2 \parallel s_4, \rho_f(\rho_{f_2}(s_5 \parallel \rho_{f_1}(p_2)) \parallel s_4)) \in \mathcal{S}.$$

      ii. Interaction with s_4. Then, we have $p_1 \xrightarrow{!m} p_2$, $s_3 = s_5$, and $s_4 \xrightarrow{?m} s_6$. By definition of f_1 and Rule 8 we get $\rho_{f_1}(p_1) \xrightarrow{e_{!m}} \rho_{f_1}(p_2)$ because $?m \in ?M_{s_2}$ and $?M_{s_1} \cap ?M_{s_2} = \emptyset$. And from Rule 4 we get $s_3 \parallel \rho_{f_1}(p_1) \xrightarrow{e_{!m}} s_3 \parallel \rho_{f_1}(p_2)$. Now, by definition of f_2 and Rule 8 we get $\rho_{f_2}(s_3 \parallel \rho_{f_1}(p_1)) \xrightarrow{!m} \rho_{f_2}(s_3 \parallel \rho_{f_1}(p_2))$. Furthermore, from Rule 1 we get

$$\rho_{f_2}(s_3 \parallel \rho_{f_1}(p_1)) \parallel s_4 \xrightarrow{m} \rho_{f_2}(s_3 \parallel \rho_{f_1}(p_2)) \parallel s_6.$$

      Thus, $\rho_f(\rho_{f_2}(s_3 \parallel \rho_{f_1}(p_1)) \parallel s_4) \xrightarrow{m} \rho_f(\rho_{f_2}(s_3 \parallel \rho_{f_1}(p_2)) \parallel s_6)$. Finally, by the construction of $\mathcal{S}$ we conclude that

$$(s_3 \parallel p_2 \parallel s_6, \rho_{f_2}(s_3 \parallel \rho_{f_1}(p_2)) \parallel s_6) \in \mathcal{S}.$$

   (c) Let $\alpha = n$, for some $n \in M_{s_1} \cup M_{s_2}$. Similar to the previous case.

2. Let $(s_3 \parallel p_1 \parallel s_4)\sqcup$ and $(s_3 \parallel p_1 \parallel s_4, \rho_{f_2}(s_3 \parallel \rho_{f_1}(p_1)) \parallel s_4) \in \mathcal{S}$. Trivial.

3. Let $\rho_{f_2}(s_3 \parallel \rho_{f_1}(p_1) \parallel s_4) \xrightarrow{\alpha} \rho_{f_2}(s_5 \parallel \rho_{f_1}(p_2) \parallel s_6)$ and $(s_3 \parallel p_1 \parallel s_4, \rho_{f_2}(s_3 \parallel \rho_{f_1}(p_1)) \parallel s_4) \in \mathcal{S}$. Similar to Case 1.

4. Let $(\rho_{f_2}(s_3 \parallel \rho_{f_1}(p_1) \parallel s_4))\sqcup$ and $(s_3 \parallel p_1 \parallel s_4, \rho_{f_2}(s_3 \parallel \rho_{f_1}(p_1)) \parallel s_4) \in \mathcal{S}$. Trivial. □

Unfortunately, the above technique fails to desynchronise a network, if the network topology contains a cycle. Consider a network of three synchronously communicating processes as shown in Figure 6.4. Suppose we begin with

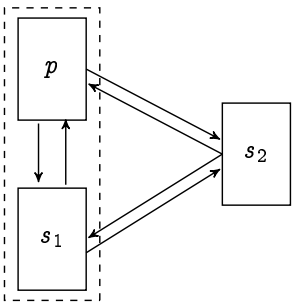


FIGURE 6.4: An illustration of a network with a cycle.

desynchronisation of the synchronous system depicted as the dashed rectangle in Figure 6.4. At this stage one should not only mask the interactions between the processes p, s_2, but, also the interactions between the processes s_1, s_2. For this reason, we define the following renaming functions f_3, f_4:

$$f_3(\alpha) = \begin{cases} e_{!m}, & \text{if } \alpha = !m, !m \in \text{Alph}(p) \cup \text{Alph}(s_1), ?m \in \text{Alph}(s_2), \\ e_{?n}, & \text{if } \alpha = ?n, ?n \in \text{Alph}(p) \cup \text{Alph}(s_1), !n \in \text{Alph}(s_2). \end{cases} . \quad (6.3)$$

$$f_4(\alpha) = \begin{cases} f_3^{-1}(\alpha), & \text{if } \alpha = e_{!m} \vee \alpha = e_{?n}, \\ e_m, & \text{if } \alpha = m \wedge m \in \text{Alph}(p_1 \parallel s_1). \end{cases} . \quad (6.4)$$

Now, desynchronisability of the complete network can be shown by the following derivation, where f is the renaming function defined as $f(e_m) = m$.

$$\begin{aligned}
& s_1 \parallel p \parallel s_2 \\
\overset{\sqcup}{\leftrightarrow}_{\mathbf{b}} \; & \rho_f(\rho_{f_4}(\rho_{f_3}(s_1) \parallel \rho_{f_3}(p)) \parallel s_2) \text{ (Lemma 6.9)}, \\
\overset{\sqcup}{\leftrightarrow}_{\mathbf{b}} \; & \rho_f(\rho_{f_4}(\nabla_3(\rho_{f_3}(p) \,|[\epsilon, \epsilon]|_{\mathrm{h}}\, \rho_{f_3}(s_1))) \parallel s_2) \\
& \qquad \text{(Desynchronisation of } \rho_{f_3}(s_1) \parallel \rho_{f_3}(p)), \\
\overset{\sqcup}{\leftrightarrow}_{\mathbf{b}} \; & \rho_f(\rho_{f_4}(p_2 \parallel s_2)) \\
& \qquad (p_2 = \rho_{f_4}(\rho_{f_3}(s_1) \parallel \rho_{f_3}(p)), \text{ and the above desynchronisation}), \\
\overset{\sqcup}{\leftrightarrow}_{\mathbf{b}} \; & \rho_f(\rho_{f_4}(\nabla_3(p_2 \,|[\epsilon, \epsilon]|_{\mathrm{h}}\, s_2))).
\end{aligned}$$

Lemma 6.9. *Let p, s_1, s_2 be any three concrete processes such that the process $s_1 \parallel p \parallel s_2$ represents the network in Figure 6.4. Let f_3, f_4 be the renaming functions as defined in Equation 6.3 and Equation 6.4, respectively. Then, $s_1 \parallel p \parallel s_2 \overset{\sqcup}{\leftrightarrow} \rho_f(\rho_{f_4}(\rho_{f_3}(s_1) \parallel \rho_{f_3}(p)) \parallel s_2)$.*

Proof. Define a binary relation $\mathcal{S}$ in the following way:

$$\mathcal{S} = \Big\{ \Big(s_3 \parallel p_1 \parallel s_4, \rho_f(\rho_{f_4}(\rho_{f_3}(s_3) \parallel \rho_{f_3}(p_1)) \parallel s_4) \Big) \mid s_3 \parallel p_1 \parallel s_4 \in \mathfrak{R}(s_1 \parallel p \parallel s_2) \Big\}.$$

Next, we show that the relation $\mathcal{S}$ is a bisimulation relation.

1. Let $s_3 \parallel p_1 \parallel s_4 \xrightarrow{e} s_5 \parallel p_2 \parallel s_6$, for some $e \in E_{s_3} \cup E_{p_1} \cup E_{s_4}$, and $\Big(s_3 \parallel p_1 \parallel s_4, \rho_f(\rho_{f_4}(\rho_{f_3}(s_3) \parallel \rho_{f_3}(p_1)) \parallel s_4)\Big) \in \mathcal{S}$. Trivial.

2. Let $s_3 \parallel p_1 \parallel s_4 \xrightarrow{m} s_5 \parallel p_2 \parallel s_6$, for some $m \in M_p \cup M_{s_1} \cup M_{s_2}$, and $\Big(s_3 \parallel p_1 \parallel s_4, \rho_f(\rho_{f_4}(\rho_{f_3}(s_3) \parallel \rho_{f_3}(p_1)) \parallel s_4)\Big) \in \mathcal{S}$.

   (a) Let $m \in M_{s_1}$. Then, from the network topology we know that this communication action is due to the interaction between either processes p_1, s_3, or s_3, s_4. Thus, we get the following cases:

      i. Let $p_1 \xrightarrow{?m} p_2$, $s_3 \xrightarrow{!m} s_5$, and $s_4 = s_6$. Similar to Case 1(b)i of Lemma 6.8.

      ii. Let $p_1 = p_2$, $s_3 \xrightarrow{!m} s_5$, and $s_4 \xrightarrow{?m} s_6$. Then, we have

      $\rho_{f_3}(s_3) \xrightarrow{e!m} \rho_{f_3}(s_5)$ (Rule 8, $s_3 \xrightarrow{!m} s_5$),
      $\rho_{f_3}(s_3) \parallel \rho_{f_3}(p_1) \xrightarrow{e!m} \rho_{f_3}(s_5) \parallel \rho_{f_3}(p_1)$ (Rule 3),
      $\rho_{f_4}(\rho_{f_3}(s_3) \parallel \rho_{f_3}(p_1)) \xrightarrow{!m} \rho_{f_4}(\rho_{f_3}(s_5) \parallel \rho_{f_3}(p_1))$
      (Rule 8, Definition of f_4),
      $\rho_{f_4}(\rho_{f_3}(s_3) \parallel \rho_{f_3}(p_1)) \parallel s_4 \xrightarrow{m} \rho_{f_4}(\rho_{f_3}(s_5) \parallel \rho_{f_3}(p_1)) \parallel s_6$,
      $\rho_f(\rho_{f_4}(\rho_{f_3}(s_3) \parallel \rho_{f_3}(p_1)) \parallel s_4) \xrightarrow{m} \rho_f(\rho_{f_4}(\rho_{f_3}(s_5) \parallel \rho_{f_3}(p_1)) \parallel s_6)$.

      Finally, from the construction of $\mathcal{S}$ we conclude that

      $$\Big(s_5 \parallel p_1 \parallel s_6, \rho_f(\rho_{f_4}(\rho_{f_3}(s_5) \parallel \rho_{f_3}(p_1)) \parallel s_6)\Big) \in \mathcal{S}.$$

   (b) Let $m \in M_p$. Similar to Case 2a.

   (c) Let $m \in M_{s_2}$. Similar to Case 2a.

3. Let $(s_3 \parallel p_1 \parallel s_4)\sqcup$, $\Big(s_3 \parallel p_1 \parallel s_4, \rho_f(\rho_{f_4}(\rho_{f_3}(s_3) \parallel \rho_{f_3}(p_1)) \parallel s_4)\Big) \in \mathcal{S}$. Trivial!

4. Let $\rho_f(\rho_{f_4}(\rho_{f_3}(s_3) \parallel \rho_{f_3}(p_1)) \parallel s_4) \xrightarrow{\alpha} \rho_f(\rho_{f_4}(\rho_{f_3}(s_5) \parallel \rho_{f_3}(p_2)) \parallel s_6)$, and $\Big(s_3 \parallel p_1 \parallel s_4, \rho_f(\rho_{f_4}(\rho_{f_3}(s_3) \parallel \rho_{f_3}(p_1)) \parallel s_4)\Big) \in \mathcal{S}$. Similar to Case 2.

5. Let $\rho_f(\rho_{f_4}(\rho_{f_3}(s_3) \parallel \rho_{f_3}(p_1)) \parallel s_4)\sqcup$, and $\Big(s_3 \parallel p_1 \parallel s_4, \rho_f(\rho_{f_4}(\rho_{f_3}(s_3) \parallel \rho_{f_3}(p_1)) \parallel s_4)\Big) \in \mathcal{S}$. Trivial! □

Ideally, one would like to generalise the above two desynchronisation techniques for a network of three processes to any arbitrary number k of processes, for $k \geq 3$. However, we refrain from doing this because the renaming functions

which are required to construct modular synchronous systems from a given network depends upon the network topology.

6.3 Synchronous systems with invisible transitions

In certain situations, it is necessary to consider some activity of processes as internal and unobservable. Such activities can be modelled by the so-called invisible transitions. The purpose of invisible transitions in a synchronous system can be either to hide certain details, or to add impurities in the behaviour of a synchronous system (for instance, a plant entering into an undesirable state without informing its supervisor).

However, the introduction of invisible transitions in a synchronous system contradicts with our assumption on a synchronous system. In particular, it is impossible to know from the semantics (Table 2.1) whether the process p_1, or[1] s_1 performed an invisible transition, whenever the synchronous system $p_1 \parallel s_1$ executes invisible transition. Such information is vital in the definition of witnessing branching bisimulation, or contra-simulation relations between a synchronous system, and its asynchronous version.

One way to circumvent this problem is by renaming the label τ of every invisible transitions present in the processes p and s by the labels τ_p and τ_s, respectively. Furthermore, by assuming that the labels τ_p, τ_s are present in the external actions of the local processes p, s, respectively, the conditions of Theorem 4.15 (Theorem 6.5) can still be used to assert whether a non-concrete synchronous system is desynchronisable or not. However, despite this soundness result, more research is required in order to examine to what extent are these conditions necessary in the absence of concreteness assumption.

[1]The word 'or' is used in the exclusive sense.

Chapter 7

Hierarchical compositional interchange format

In this chapter, we focus on the issue of refinement of states in a model of a hybrid system that contains both discrete and continuous dynamics. The idea is to facilitate the top-down development of a hybrid system in the Compositional Interchange Format (CIF) [7], just like the statecharts [47] and the hierarchical automata [61] are used in the development of discrete systems.

CIF is a modeling language [14, 15] based on hybrid automata [48], which aims to establish interoperability among a wide range of formalisms and associated tools for the specification of hybrid and timed systems. This is accomplished by means of model transformations to and from CIF to avoid the implementation of many bilateral translators. The CIF language contains the following features.

- A CIF automaton contains three kind of predicates at every location. First, the initial conditions in an automaton are specified by the initialisation predicate, written as init. Second, the time can progress predicate, written as tcp, that specifies the conditions under which time can progress in a location. Finally, the predicate inv which specifies the invariant that must hold in a location of a CIF automaton.
- Communication among CIF automata is done by common action labels and shared variables.
- The variable scope operator is used to specify local variables in a CIF automaton and the action scope operator defines which actions are to be made hidden in a CIF automaton.

- An initialization operator for restricting the initial conditions of variables. This allows initialization on a more global level as compared to the init predicate of a CIF automaton.
- A synchronization operator ensures an action is executed synchronously in a parallel composition.
- An urgency operator for declaring actions as urgent.

Although there exists many frameworks [3, 29, 30, 46, 60] that allow hierarchical description of a hybrid system; however, these frameworks either do not have formal semantics or the formal semantics of these frameworks are not based on the Structural Operational Semantics (SOS) [68]. There are two reasons for having the formal semantics based on SOS.

First, this allows the semantics of our new language to be inline with the semantics of CIF. As a result, the existing simulation tools for validating a model of an hybrid system can be reused upon eliminating the hierarchy. Second, usually, the model transformations to and from CIF are not only to be executed on 'complete' models, but also on components of bigger models. Thus, it is crucial that bisimulation equivalence is a congruence for all the constructs of the CIF. This is guaranteed by the process-tyft format on the SOS rules of Mousavi et al. [62].

For this purpose, we define the semantics of a hierarchical hybrid automaton in a compositional manner, by referring only to the transition system of the substructures and not to their syntactic representation. This compositional introduction of hierarchy allows us to keep the semantics of the Hierarchical CIF (HCIF) operators almost unchanged with respect to their CIF versions. Note that the previous works [27, 54, 61] on the semantics of hierarchical automata requires tree-structure on the set of locations. Consequently, additional concepts from tree-structures, like least common ancestors, children of a location, etc., complicate the semantics and thus, bringing considerable differences between the semantics of CIF and HCIF [20].

This chapter is organised as follows. In Section 7.1 we describe the formal syntax of HCIF formalism. Section 7.2, we describe our semantic framework consisting of hybrid transition system, valuation, trajectories, etc. in detail. In Section 7.3, we give the SOS rules for all the constructs of HCIF and explain it intuitively with examples. In Section 7.4, we give linearisation procedure that eliminates hierarchy from a subclass of HCIF models. Lastly, in Section 7.5, we describe the patient support system in HCIF, which is used in medical diagnosis to position a patient in a Magnetic Resonance Scanner (MRI).

7.1 Syntax of HCIF

In this section, we describe the mathematical syntax of HCIF and illustrate it by modelling a controller of a simplified patient support table attached to a MRI scanner, which is discussed in more detail in Section 7.5. Note that in this section we give an incomplete description of the controller model to illustrate the various concepts involved in the definition of a hierarchical automaton.

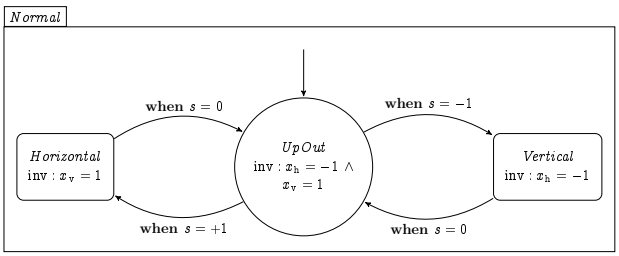


FIGURE 7.1: Movement control.

Figure 7.1 gives an informal, graphical representation of a HCIF automaton, which models a controller of a patient support table. The control operates in one of the following three modes:

1. *Horizontal*: horizontal movement of the table.
2. *UpOut*: table fully up and out.
3. *Vertical*: vertical movement of the table.

Every location has an initialization predicate, an *invariant* predicate, and a *time can progress predicate* associated to it. The initialization predicate of a location l describes the constraints that the initial values of variables must satisfy for an execution to start in l. Such locations for which the initialization predicate is true are called *active locations*. The *invariant* of a location ℓ is a predicate that must hold as long as the location ℓ is one of the active locations of the system. The *time can progress* predicate of a location ℓ is a predicate that must hold during time delays when ℓ is an active location.

In Figure 7.1[1], the location *UpOut* has true as the initialization predicate, and the time can progress predicate and the predicate $x_h = -1 \wedge x_v = 1$ as the invariant. Time can progress predicates are in general useful for triggering the execution of an action from a location within a certain period of time. For instance, an action a must be executed when the clock value has reached 2 units of time (see Figure 7.4(a)).

[1]Table 7.1 describes the conventions followed throughout this chapter.

Edges represent discrete changes in the computational state of a system. An edge has a *source* and a *target* location whose execution results in a change of active location (unless the edge is a self loop). The automaton of Figure 7.1 has four edges in total among the locations *Horizontal*, *UpOut*, and *Vertical*.

Every edge contains a predicate called *guard* that determines when an action can be executed, a set of *jumping variables* that specify the variables that are changed by the action, and a predicate called *update* that determines how these model variables can change. Edges are labeled by *actions* that may be used to synchronize the behavior of automata in a parallel composition. In Figure 7.1, the edge from the location *UpOut* to the location *Horizontal* has the guard $s = 1$, update predicate `true`, empty set of jumping variables, and the silent action label τ.

Formally, the set of locations in a HCIF automaton is denoted by L and its alphabet is denoted as usual by A. In HCIF, there are three types of variables: regular variables, denoted by the set V; the dotted versions of those variables, which belong to the set $\dot{V} = \{\dot{x} \mid x \in V\}$; and the step variables, which belong to the set $\{x^+ \mid x \in V \cup \dot{V}\}$. The notation x^+ is used to refer to the value of the variable x after the execution of an action. Furthermore, the variables can be classified according to their evolution (i.e. how their values change during time delays). In particular, we distinguish between discrete variables (such as s in Figure 7.1), whose values remain constant during time delays, so that the values of their dotted versions are always 0; and continuous variables (such as x_h and x_v in Figure 7.1), whose values evolve as a continuous function of time during delays and their dotted versions represent their derivatives.

The values of the variables belong to the set Λ that contains, among others, the sets $\mathbb{B}$ (booleans) and $\mathbb{R}$ (reals). The predicates representing the guards, time can progress, invariant, and initialization predicates are taken from the set P and the predicates representing the update predicates are taken from the set P^+. The exact syntax and semantics of predicates are defined in [7]. The predicates P and P^+ are the terms of the language of predicate logic [70], where for P the variables are taken from the set $V \cup \dot{V}$ and for P^+ the variables are taken from the set $V \cup \dot{V} \cup \{x^+ \mid x \in V \cup \dot{V}\}$. Lastly, Expr denotes the set of all expressions over variables $V \cup \dot{V}$.

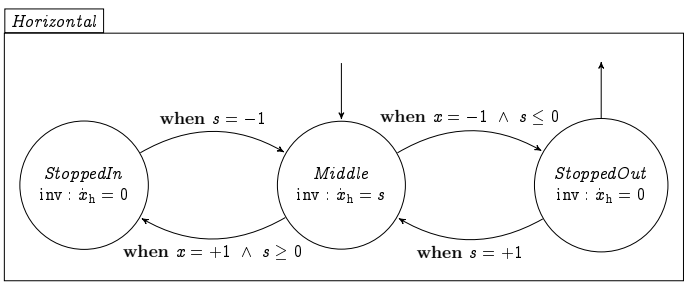


FIGURE 7.2: Horizontal movement.

Locations can contain other automata (or compositions of them, as we show in Section 7.5). In Figure 7.1 the location *Horizontal* contains the automaton shown in Figure 7.2, that defines the horizontal movement of the controller in more detail. Automata that are contained inside other locations are referred to as *sub-automata*, or *sub-structure*, and the containing automata are referred to as *super-automata*, or *super-structure*. In a HCIF automaton, there are two types of edges: *non-disruptive* edges (for brevity, we refer to a non-disruptive edge as an edge) and *disruptive* edges. Intuitively, an edge is executed from a location if the sub-structure at that location is terminating, while a disruptive edge can be executed even if the sub-structure at that location is non-terminating. Note that the conditions under which an edge, or a disruptive edge can be executed depend on several other factors, which are defined in Section 7.3.

In addition to the initialization, the invariant, and the time can progress predicates, each location has a *termination predicate* which defines when execution can terminate in that location. Termination predicates are useful to specify when the super-structure can perform a transition. In the automaton shown in Figure 7.1, the τ transition from the location *Horizontal* to the location *UpOut* can be executed only if the guard $s = 0$ holds, the automaton (Figure 7.2) inside the location *Horizontal* has *StoppedOut* as its active location and the termination predicate holds.

Additional components of an automaton (not shown in the example presented here) include: *control variables*, *synchronizing actions*, and *dynamic type* mappings. Intuitively, controlled variables are those variables that can only be modified by the automaton that declares them, and they do not change arbitrarily after performing an action. The set of synchronizing actions is used to specify which actions are to be synchronized when the automaton is composed in parallel. The concept of dynamic types [56] is used to model the constraints in the joint evolution of a variable and its dotted version. In CIF, a dynamic type is a set containing pairs of functions, whose domain is a closed range of the form $[0, t]$, with $t \in T$. Notation T is used to refer to the set of all time points, which is nothing but the set of all positive real numbers.

Definition 7.1 (Hierarchical automata)**.** A hierarchical automaton β is a tuple

$$(L, \text{init}, \text{inv}, \text{tcp}, E, D, \text{cvar}^2, \text{act}_S, \text{dtype}, \text{term}, h), \text{ where}$$

- L is a set of locations,
- initialization predicate init, invariant predicate inv, time-can-progress predicate tcp, and termination predicate term are of the type $L \to P$,

[2]Some of the notations differs from the original publication on HCIF [64] to prevent any conflict with the notations of Chapter 2.

- a set of edges $E \subseteq L \times P \times A_\tau \times (2^{V \cup \dot{V}} \times P^+) \times L$. Intuitively, an edge of the form $(\ell, g, a, (J, \mathrm{up}), \ell')$ specifies:
  1. The source and the target locations are represented by $\ell, \ell' \in L$, respectively.
  2. The guard is represented by the predicate $g \in P$.
  3. The label of this transition is denoted by $a \in A_\tau$.
  4. The set $J \subseteq V$ represents the set of jumping variables.
  5. The update predicate, denoted as $\mathrm{up} \in P^+$, determines how the model variables change after executing this edge.
- $D \subseteq E$ is the set of *disruptive* edges of the automaton.
- $\mathrm{cvar} \subseteq 2^V$ is the set of controlled variables.
- $\mathrm{act}_S \subseteq 2^A$ is the set of synchronizing actions.
- $\mathrm{dtype} : V \rightharpoonup 2^{(T \to \Lambda) \times (T \to \Lambda)}$ is the dynamic type mapping.
- $h : L \rightharpoonup \mathbb{C}$ is a partial function that associates to some locations a sub-structure. Here, $\mathbb{C}$ is the set of all compositions in HCIF (see Definition 7.2).

Using operators more complex models, referred to as compositions (Definition 7.2), can be constructed. The semantics of the operators is presented in Section 7.3, with the exception of the semantics of the action and variable scope operators. The semantics of these operators is unchanged with respect to the semantics of these operators in CIF, as defined in [7, 63].

Definition 7.2. The set of *compositions* $\mathbb{C}$ in the HCIF formalism is recursively defined by the grammar below, where $x \in V$, $e \in \mathrm{Expr}$.

$c, c' \in \mathbb{C} ::=$	β		hierarchical automaton
	$c : \beta$		automaton postfix operator
	$c \parallel c'$		parallel composition
	$\lvert[_{\mathrm{V}}\, x = e, \dot{x} = e :: c\,]\rvert$		variable scope operator
	$\lvert[_{\mathrm{A}}\, a :: c\,]\rvert$		action scope operator, $a \in A$
	$\upsilon_a(c)$		urgency operator, $a \in A_\tau$.

Throughout this chapter, the textual and graphical conventions given in Table 7.1 are followed.

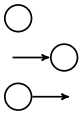
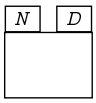
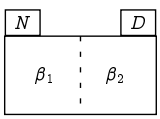
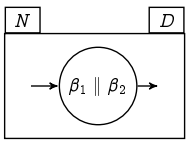
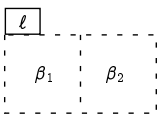
Graphical representation	**Meaning**
	Location without any sub-structure Initial location with init predicate true Final location with termination predicate true
	Location containing a sub-structure
when g **act** a **do** $x := e$	Edge $(g, a, (\{x\}, x^+ = e))$
$\lightning$ **when** g **act** a **do** $x := e$	Disruptive edge $(g, a, (\{x\}, x^+ = e))$
when g **act** a	Edge $(g, \tau, (\emptyset, \mathtt{true}))$ Edge $(\mathtt{true}, a, (\emptyset, \mathtt{true}))$
	Automaton N with declarations D
	
	AND superstate ℓ containing parallel composition $\beta_1 \parallel \beta_2$

TABLE 7.1: Textual and graphical conventions in HCIF.

7.2 Semantic framework

In this section, the semantic framework is set up to properly explain the semantics of HCIF. First, we present the concepts of variable valuations and flow trajectories. Second, we describe informally hybrid transitions systems, which are induced by the different SOS rules of HCIF compositions. Finally, a formal definition of this semantic model is given.

7.2.1 Concepts involved with the semantics of HCIF

The execution of a hybrid automaton can be regarded as a sequence of discrete or continuous changes in the values of variables from the set $V \cup \dot{V}$. This phenomenon of discrete changes in the values of variables is achieved by a valuation. On the other hand, the phenomenon of continuous changes in the values of variables is realized by a variable trajectory. In this way, we can

describe the semantics in terms of the changes in valuation that an automaton may cause in a given initial valuation.

A *valuation* $\sigma\colon (V \cup \dot{V}) \to \Lambda$ is a function that for each variable returns its corresponding value. We use the notation $\Sigma = \{\sigma \mid \sigma : (V \cup \dot{V}) \to \Lambda\}$ to refer to the set of all valuations. Having defined valuations, we introduce the concept of satisfiability. Even though predicates are abstract entities, we assume that a satisfaction relation $\models \subseteq \Sigma \times P$ is defined, that expresses a predicate evaluates to true in a valuation. For a valuation σ, we define $\sigma^+(v^+) = \sigma(v)$.

A *variable trajectory* is a partial function $\varrho\colon T \rightharpoonup \Sigma$ that returns the valuations of the variables at each time point. In other words, $\varrho(t)(x)$ is the value of variable x at time $t \in T$. We assume the domain of variable trajectories to be closed intervals, i.e. intervals of the form $[0, t]$, for some $t \in T$ to model that t time unit has been elapsed.

7.2.2 Hybrid transition systems

The semantics of HCIF compositions is given in terms of SOS rules, which induce a hybrid transition system (HTS) [26]. The states of the HTS are of the form $\langle c, \sigma \rangle$, where $c \in \mathbb{C}$ and $\sigma \in \Sigma$ is a valuation. There are three kinds of transition in the HTS, namely, *action transitions*, *time transitions*, and *environment transitions*.

Action transitions are of the form $\langle c, \sigma \rangle \xrightarrow{a,b,X} \langle c', \sigma' \rangle$. Intuitively, this transition models the execution of an action a by a composition c in a valuation σ, which results in a new composition c and a new valuation σ'. Label b is a boolean that indicates whether action a is synchronizing or not, and label X is the set of controlled variables defined by the environment.

Time behavior is captured by *time transitions*. Time transitions are of the form $\langle c, \sigma \rangle \xrightarrow{\varrho,A',\theta,\omega} \langle c', \sigma' \rangle$. They model the passage of time in a composition c in a valuation σ, which results in a composition c' and a valuation σ'. Label $A' \subseteq A$ contains the set of synchronizing actions of c and c'. Function $\varrho : T \rightharpoonup \Sigma$ is the variable trajectory. Function $\theta : T \rightharpoonup 2^A$ is called *guard trajectory*. It models the evolution of enabled actions during time delays. For each time point $t \in \mathrm{dom}(\theta)$, the function application $\theta(t)$ yields the set of enabled actions of the composition c at time t.

Consider the automaton as shown in Figure 7.3(a) and assume that $0 < k_0 < k_1$. Then, the set of enabled actions at the active location will depend on the function e^x as illustrated in Figure 7.3(b). In other words,

- the set of enabled actions is $\{a\}$, whenever $0 \leq x < \ln(k_0)$,
- the set of enabled actions is $\{a, b\}$, whenever $\ln(k_0) \leq x < \ln(k_1)$,

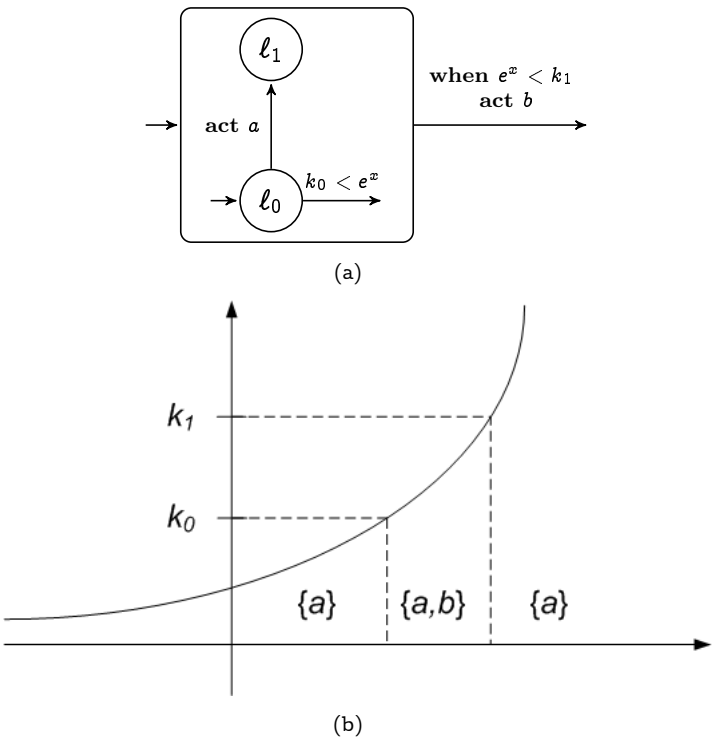


FIGURE 7.3: An illustration of the set of enabled actions over a time period.

- the set of enabled actions is $\{a\}$, whenever $x \geq \ln(k_1)$.

Lastly, the function $\omega : T \to \mathbb{B}$ is called *termination trajectory*. It models the evolution of termination (Definition 7.3) during time delays: for each time point $t \in \mathrm{dom}(\omega)$, composition c' is terminating at time t if and only if $\omega(t) = \mathrm{true}$. For all time transitions, it is assumed that the domain of variable trajectory ϱ is a non-empty closed interval, i.e., $\mathrm{dom}(\varrho) = [0, t]$, for some $t > 0$, and $\mathrm{dom}(\varrho) = \mathrm{dom}(\theta) = \mathrm{dom}(\omega)$.

Definition 7.3. Given a valuation σ, we define *termination* as follows:

- An automaton $(L, \mathrm{init}, \mathrm{inv}, \mathrm{tcp}, E, D, \mathrm{cvar}, \mathrm{act}_S, \mathrm{dtype}, \mathrm{term}, h)$ is terminating in σ, if there is a location $\ell \in L$ such that $\sigma \models \mathrm{init}(\ell)$, $\sigma \models \mathrm{inv}(\ell)$, $\sigma \models \mathrm{term}(\ell)$, and if $\ell \in \mathrm{Dom}(h)$ then $h(\ell)$ is terminating in σ.
- The composition $c_1 \parallel c_2$ is terminating in the valuation σ, if the compositions c_1 and c_2 are terminating in the valuation σ.
- For the remaining operators, termination is defined point-wise.

Environment transitions are of the form $\langle c, \sigma \rangle \overset{A', b}{\dashrightarrow} \langle c', \sigma' \rangle$. They are used in the semantics to enforce restrictions posed by the environment of a composition on the action behavior of the composition. More specifically, a transition $\langle c, \sigma \rangle \overset{A', b}{\dashrightarrow} \langle c', \sigma' \rangle$ expresses that the composition c is consistent (Definition 7.4) in the valuation σ and the composition c' is consistent in the valuation σ'. In addition, the role of the environment transitions is to indicate that a composition c can initialize to become a composition c' in which an active location is fixed for each (active) substructure. Furthermore, the boolean b indicates whether the initialized substructure can terminate and thus, give back the control to its super-structure. As before, label A' contains the set of synchronizing actions of compositions c and c'.

Definition 7.4. Given a valuation σ, we define *consistency* as follows.

- An automaton $(L, \text{init}, \text{inv}, \text{tcp}, E, D, \text{cvar}, \text{act}_S, \text{dtype}, \text{term}, h)$ is consistent in σ, if there is a location $\ell \in L$ such that $\sigma \models \text{init}(\ell)$, $\sigma \models \text{inv}(\ell)$, and if $\ell \in \text{Dom}(h)$ then $h(\ell)$ is consistent in σ.
- The composition $c \parallel c'$ is consistent in valuation σ, if the compositions c and c' are consistent in the valuation σ.
- For the remaining operators, consistency is defined point-wise.

We use notation $\sigma \models c$ to denote that composition c is consistent in the valuation σ. Alternatively, we say that σ is consistent with c.

Definition 7.5 formalizes the hybrid transition system induced by the SOS rules presented in the next sections.

Definition 7.5. A *hybrid transition system* (HTS) is a five-tuple of the form $(\mathbb{C} \times \Sigma, A, \rightarrow, \mapsto, \dashrightarrow)$, where

1. $\rightarrow \subseteq (\mathbb{C} \times \Sigma) \times (A_\tau \times \mathbb{B} \times 2^V) \times (\mathbb{C} \times \Sigma)$,
2. $\mapsto \subseteq (\mathbb{C} \times \Sigma) \times \Big((T \rightharpoonup \Sigma) \times 2^A \times (T \rightharpoonup 2^A) \times (T \to \mathbb{B})\Big) \times (\mathbb{C} \times \Sigma)$, and
3. $\dashrightarrow \subseteq (\mathbb{C} \times \Sigma) \times (2^A \times \mathbb{B}) \times (\mathbb{C} \times \Sigma)$.

7.3 Semantics

In this section we explain the semantics of HCIF, both informally by means of examples and formally by means of SOS rules.

7.3.1 Hierarchical automata

In the following, we use the notation $\beta[\ell]$ to refer to the automaton:

$$(L, \mathrm{id}_\ell, \mathrm{inv}, \mathrm{tcp}, E, D, \mathrm{cvar}, \mathrm{act}_S, \mathrm{dtype}, \mathrm{term}, h),$$

where $\mathrm{Dom}(\mathrm{id}_\ell) = L$, $\mathrm{id}_\ell(\ell') = \mathrm{true}$ for $\ell = \ell'$, and false otherwise. Intuitively, the notation $\beta[\ell]$ denotes that ℓ is the active location of the automaton β. Furthermore, we use notation $f \uparrow A'$ to refer to the domain restriction of function f to the set A'.

Action transitions

In the absence of hierarchy, an automaton β can perform an action in a location ℓ and a valuation σ, if there is an edge $(\ell, g, a, (J, \mathrm{up}), \ell')$ such that the following conditions are met:

- Location ℓ is active in the valuation σ, i.e., $\sigma \models \mathrm{init}(\ell)$.
- Guard g and the invariant at the location ℓ holds in the valuation σ, i.e., $\sigma \models g \wedge \sigma \models \mathrm{inv}(\ell)$.
- It is possible to find a new valuation σ' such that:
  - The invariant of the new location ℓ' holds in the valuation σ', i.e., $\sigma' \models \mathrm{inv}(\ell')$.
  - The update predicate up is satisfied in the valuation $\sigma \cup \sigma'^{+}$, i.e., $\sigma \cup \sigma'^{+} \models \mathrm{up}$. Note that we do not write $\sigma' \models \mathrm{up}$ since, in general, the predicate up can refer to the next values of the variables in up, which are contained in σ'^{+}.
  - The values of the control variables of the automaton (cvar) and of the environment (X) remains the same in σ and σ', except those variables that belong to the set of jumping variables J. This is encoded in the equation $\sigma \uparrow ((X \cup \mathrm{cvar}) \setminus J) = \sigma' \uparrow ((X \cup \mathrm{cvar}) \setminus J)$.

The above conditions are summarized in $\sigma, \sigma' \models_\beta \langle \ell, g, a, (J, \mathrm{up}), \ell', \mathrm{cvar}, X \rangle$, which is syntactically equivalent to:

$$(\ell, g, a, \mathrm{up}, \ell') \in E \wedge\ \sigma \models \mathrm{init}(\ell) \wedge\ \sigma \models g\ \wedge \sigma \models \mathrm{inv}(\ell) \wedge\ \sigma' \models \mathrm{inv}(\ell') \wedge$$
$$\sigma'^{+} \cup \sigma \models \mathrm{up} \wedge \sigma \uparrow ((X \cup \mathrm{cvar}) \setminus J) = \sigma' \uparrow ((X \cup \mathrm{cvar}) \setminus J).$$

Actually, the conditions in the predicate $\sigma, \sigma' \models_\beta \langle \ell, g, a, (J, \mathrm{up}), \ell', \mathrm{cvar}, X \rangle$ are *exactly* the conditions required to execute an action in a CIF automaton.

The first line in the premise of Rule H-1 asserts the conditions that are required to execute an action transition in a flat CIF automaton. The second line in the premise of Rule H-1 asserts that if the edge is not disruptive, it is necessary to check that the substructure of the initial location, if any, is terminating. This is expressed by the condition $(\ell, g, a, (J, \mathrm{up}), \ell') \notin D \Rightarrow (\langle h(\ell), \sigma\rangle \overset{A_0,\mathrm{true}}{\dashrightarrow} \langle c, \sigma\rangle \vee \ell \notin \mathrm{dom}(h)))$. Finally, after the action is performed, the substructure in the target location, if present, must be initialized $(\langle h(\ell'), \sigma'\rangle \overset{A_1,b}{\dashrightarrow} \langle c' : \beta[\ell'], \sigma'\rangle)$. The choice of selecting active locations of substructure $h(\ell')$ is made upon entering location ℓ'. Consistency of the substructures in the target location ℓ' is taken care of by the above environment transition.

$$\frac{\begin{array}{c}\sigma, \sigma' \models_\beta \langle \ell, g, a, (J, \mathrm{up}), \ell', \mathrm{cvar}, X\rangle,\\ (\ell, g, a, (J, \mathrm{up}), \ell') \notin D \Rightarrow \left(\langle h(\ell), \sigma\rangle \overset{A_0,\mathrm{true}}{\dashrightarrow} \langle c, \sigma\rangle \vee \ell \notin \mathrm{dom}(h)\right),\\ \ell' \in \mathrm{dom}(h), \quad \langle h(\ell'), \sigma'\rangle \overset{A_1,b}{\dashrightarrow} \langle c', \sigma'\rangle\end{array}}{\langle \beta, \sigma\rangle \xrightarrow{a, a \in \mathrm{act}_S, X} \langle c' : \beta[\ell'], \sigma'\rangle} \quad \text{(H-1)}$$

Remark 7.6. Consider the controller automaton in Figure 7.1, assuming *Up-Out* is an active location with a valuation σ. The edge labelled **when** $s = +1$ can be executed if there exists a valuation σ' such that σ satisfies the invariant of the location *UpOut* ($\sigma(x_\mathrm{h}) = -1 \wedge \sigma(x_\mathrm{v}) = 1$), σ' satisfies the invariant of the location *Horizontal* ($\sigma'(x_\mathrm{v}) = 1$), and the valuation σ' is consistent with the automaton shown in Figure 7.2. The consistency of the valuation σ' implies that the active location of the automaton in Figure 7.2 is *Middle* such that $\sigma' \models \dot{x}_\mathrm{h} = s$.

Now consider the active location of the controller to be *Horizontal* and the active location of the automaton in Figure 7.2 to be *Stopped-in*. In this case, the edge labelled **when** $s = 0$ in Figure 7.1 cannot be executed even if the guard $s = 0$ is true. This is due to the sub-automaton inside the location *Horizontal* is non-terminating. The edge labelled **when** $s = 0$, can be executed only if either the sub-automaton is terminating, or the edge is specified as disruptive.

Rule H-1 requires as a condition that there is an active substructure in the target location $\ell' \in \mathrm{dom}(h)$. If this is not the case then no active substructure is prefixed to $\beta[\ell]$, as expressed by Rule H-2.

$$\frac{\begin{array}{c}\sigma, \sigma' \models_\beta \langle \ell, g, a, (J, \mathrm{up}), \ell', \mathrm{cvar}, X\rangle, \quad \ell' \notin \mathrm{dom}(h),\\ (\ell, g, a, (J, \mathrm{up}), \ell') \notin D \Rightarrow \left(\langle h(\ell), \sigma\rangle \overset{A_0,\mathrm{true}}{\dashrightarrow} \langle c, \sigma\rangle \vee \ell \notin \mathrm{dom}(h)\right)\end{array}}{\langle \beta, \sigma\rangle \xrightarrow{a, a \in \mathrm{act}_S, X} \langle \beta[\ell'], \sigma'\rangle} \quad \text{(H-2)}$$

Besides the action transitions triggered by the edges of a hierarchical automaton, the action transitions can also be triggered by their substructures. Given a valuation σ, an action transition triggered by a substructure can be executed at a super-state ℓ if the following conditions hold:

- ℓ is the active location in the valuation σ, i.e., $\sigma \models \text{init}(\ell)$.
- The invariant associated with location ℓ is satisfied by σ, i.e., $\sigma \models \text{inv}(\ell)$.
- The action performed by the substructure of ℓ results in a new valuation σ' such that the valuation σ' also satisfies the invariant of ℓ, i.e., $\sigma' \models \text{inv}(\ell)$.

Rule H-3 formalizes this. In the conclusion, $c : \beta[\ell]$ reflects that an initial location ℓ is chosen in a hierarchical automaton β if the substructure $h(\ell)$ performs an action transition. In the premise of this rule, we ignore the boolean b that indicates whether the action a is synchronizing. As a result, the superstructure decides on which actions it wants to synchronize. In other words, the superstructure defines the set of synchronizing actions, independently of the sub-structures.

$$\frac{\begin{array}{c}\sigma \models \text{init}(\ell), \quad \sigma \models \text{inv}(\ell), \quad \sigma' \models \text{inv}(\ell), \\ \ell \in \text{dom}(h), \qquad \langle h(\ell), \sigma\rangle \xrightarrow{a,b,X\cup\text{cvar}} \langle c, \sigma'\rangle\end{array}}{\langle \beta, \sigma\rangle \xrightarrow{a,a\in\text{act}_S,X} \langle c : \beta[\ell], \sigma'\rangle} \text{(H-3)}$$

Transition $\langle h(\ell), \sigma\rangle \xrightarrow{a,b,X\cup\text{cvar}} \langle c, \sigma'\rangle$ in the premise of the above rule ensures that the control variables inherited from the environment (X) and the control variables (cvar) of the automaton β will not jump arbitrarily when the action is executed by the substructure.

Remark 7.7. Again consider the model of the controller as given in Figures 7.1 and 7.2. Assume that the active location is *Horizontal* and the active location of the sub-structure is *Middle*. The edge labelled **when** $x \geq 1 \wedge s \geq 0$ can be executed from the location *Middle* only if there exists a new valuation σ' such that it satisfies the invariant of the locations *Horizontal* and *StoppedIn*.

Time transitions

In HCIF, a time delay $t > 0$ is possible in an active location ℓ, if there exists a variable trajectory ϱ such that

1. the invariant associated with the active locations is satisfied in the interval $[0, t]$,
2. the time can progress predicate is satisfied in the interval $[0, t)$, and

3. the dynamic type constraints specified by dtype are satisfied by the variable trajectory ϱ.

We formalise the above conditions as a predicate $\varrho \models \langle t, \ell, \text{init}, \text{inv}, \text{tcp}, \text{dtype}\rangle$, which is syntactically equivalent to the following condition:

$$\begin{aligned}
&\varrho(0) \models \text{init}(\ell) \wedge \text{dom}(\varrho) = [0, t] \wedge 0 < t \wedge \\
&\forall t' \in [0, t).\Big[\varrho(t') \models \text{tcp}(\ell)\Big] \wedge \forall t' \in [0, t].\Big[\varrho(t') \models \text{inv}(\ell)\Big] \wedge \\
&\forall x \in \text{dom}(\text{dtype}).\Big[(\varrho \uparrow x, \varrho \uparrow \dot{x}) \in \text{dtype}(x)\Big].
\end{aligned}$$

In addition, for a time delay t, the substructure (if present) must perform a time transition with the same variable trajectory. Thus, the invariant and the time can progress predicates of the active location of the automaton and, recursively, of the substructure are satisfied simultaneously for the time delay t. In other words, an automaton and its active substructure synchronize on time delays. Rule H-4 models this fact, where $\text{dom}(\omega) = \text{dom}(\varrho), \text{dom}(\theta) = \text{dom}(\varrho)$,

$$\begin{aligned}
&\forall t' \in [0, t].\Big[\omega(t') = (\omega_0(t') \wedge \varrho(t') \models \text{term}(\ell))\Big], \text{ and} \\
&\forall t' \in [0, t].\Big[\theta(t') = \theta_0(t') \cup \{a \mid (\ell, g, a, (J, \text{up}), \ell') \in E \wedge \varrho(t') \models g \wedge \omega_0(t')\}\Big].
\end{aligned}$$

The above condition states that the guard trajectory θ and the termination trajectory ω are constructed by using the corresponding trajectories generated by the time transition in the substructure. Intuitively, the equation $\omega(t') = (\omega_0(t') \wedge \varrho(t') \models \text{term}(\ell))$ states that a hierarchical automaton β terminates at a time instant t' if the substructure terminates at the time instant t' and the termination predicate of the active location ℓ is satisfied at the time instant t'. Similarly, the equation

$$\theta(t') = \theta_0(t') \cup \{a \mid (\ell, g, a, (J, \text{up}), \ell') \in E \wedge \varrho(t') \models g \wedge \omega_0(t')\}$$

states that an action is enabled at a time instant t' by a HCIF automaton if

- either, the action is enabled by the substructure at a time instant t',
- or, the action is present as a label in an edge of the hierarchical automaton with its corresponding guard satisfied at the time instant t' and the substructure terminates at the time instant t'.

$$\frac{\begin{array}{c}\varrho \models \langle t, \ell, \text{init}, \text{inv}, \text{tcp}, \text{dtype}\rangle, \quad \ell \in \text{dom}(h), \\ \langle h(\ell), \varrho(0)\rangle \xrightarrow{\varrho, A, \theta_0, \omega_0} \langle c, \varrho(t)\rangle\end{array}}{\langle \beta, \varrho(0)\rangle \xrightarrow{\varrho, \text{act}_S, \theta, \omega} \langle c : \alpha[\ell], \varrho(t)\rangle} \text{ (H-4)}$$

The set of synchronizing actions only takes into account the set act_S of the superstructure, since the set of synchronizing actions in the substructure does not influence the action synchronizing behavior of its parent. The same approach is taken when computing the set of synchronizing actions in the environment transition in Rule H-6.

Rule H-5 deals with the case that an initial location ℓ does not contain a substructure, where $\text{dom}(\omega) = \text{dom}(\varrho), \text{dom}(\theta) = \text{dom}(\varrho)$,

$$\forall t' \in [0,t].\Big[\omega(t') = (\varrho(t') \models \text{term}(\ell))\Big], \text{ and}$$
$$\forall t' \in [0,t].\Big[\theta(t') = \{a \mid (\ell, g, a, (J, \text{up}), \ell') \in E \wedge \varrho(t') \models g\}\Big].$$

$$\frac{\varrho \models \langle t, \ell, \text{init}, \text{inv}, \text{tcp}, \text{dtype}\rangle, \quad \ell \notin \text{dom}(h)}{\langle \beta, \varrho(0)\rangle \xmapsto{\varrho, \text{act}_S, \theta, \omega} \langle \beta[\ell], \varrho(t)\rangle} \text{ (H-5)}$$

Environment transitions

In HCIF, if an automaton performs an environment transition, a unique active location is chosen, and the substructure (if present) is also initialized. The environment transition ensures that the active location contains a consistent hierarchical structure (Definition 7.4). This is expressed by Rule H-6. The initialized composition c becomes the active substructure of $\beta[\ell]$ and the automaton is terminating if the active location and the active substructure both are terminating. Rule H-7 deals with the case where there is no substructure.

$$\frac{\begin{array}{c}\sigma \models \text{init}(\ell), \quad \sigma \models \text{inv}(\ell), \quad \sigma' \models \text{inv}(\ell), \\ \sigma \uparrow \text{cvar} = \sigma' \uparrow \text{cvar}, \\ \ell \in \text{dom}(h), \quad \langle h(\ell), \sigma\rangle \overset{A', b}{\dashrightarrow} \langle c, \sigma'\rangle\end{array}}{\langle \beta, \sigma\rangle \overset{\text{act}_S, \sigma \models \text{term}(\ell) \wedge b}{\dashrightarrow} \langle c : \beta[\ell], \sigma'\rangle} \text{ (H-6)}$$

$$\frac{\begin{array}{c}\sigma \models \text{init}(\ell), \quad \sigma \models \text{inv}(\ell), \quad \sigma' \models \text{inv}(\ell), \\ \sigma \uparrow \text{cvar} = \sigma' \uparrow \text{cvar}, \quad \ell \notin \text{dom}(h)\end{array}}{\langle \beta, \sigma\rangle \overset{\text{act}_S, \sigma \models \text{term}(\ell)}{\dashrightarrow} \langle \beta[\ell], \sigma'\rangle} \text{ (H-7)}$$

7.3.2 Automaton postfix operator

We now define the SOS rules for the automaton postfix operator, which helps in defining the overall behavior of a hierarchical automaton. Intuitively, $c : \beta$ means that composition c is the active substructure of some initial location $\ell \in L$ in the automaton β. Note that whenever the composition $c : \beta$ is the

result of a previous transition in β, this initial location is always uniquely defined.

The semantics of $c : \beta$ is reminiscent of the mode transfer (disrupt) operator of process algebra [4], i.e., the composition c in $c : \beta$ will perform the action transition until it is terminating and no disruptive edges in β are enabled. However, in the overall evolution of $c : \beta$, the automaton β can perform its disruptive edges. The difference between the two operators is due to the passage of time caused by these operators and how the termination is handled in them.

- In the automaton postfix operator, the passage of time is synchronized between the first, and second component. In the mode transfer operator, the passage of time is not only synchronized between the two components; but, if right component is unable to match the time transitions generated by the left component, then the overall composition behaves as the left component [4].
- In the automaton postfix operator, $c : \beta$ terminates whenever the composition c and β terminates. In the mode transfer operator, the overall composition terminates, whenever one of the components terminates [4].

Rule H-8 models the action transition taken by automaton β when the active substructure is terminating or when the chosen edge is disruptive, and the target location has a substructure.

$$\frac{\begin{array}{c}\sigma, \sigma' \models_\beta \langle \ell, g, a, (J, \mathrm{up}), \ell', \mathrm{cvar}, X\rangle, \\ (\ell, g, a, (J, \mathrm{up}), \ell') \notin D \Rightarrow \left(\langle c_1, \sigma\rangle \xrightarrow{A_0, \mathrm{true}} \langle c_1', \sigma\rangle \vee \ell \notin \mathrm{dom}(h) \right), \\ \ell' \in \mathrm{dom}(h), \quad \langle h(\ell'), \sigma'\rangle \xrightarrow{A_1, b'} \langle c_2, \sigma'\rangle \end{array}}{\langle c_1 : \beta, \sigma\rangle \xrightarrow{a, a \in \mathrm{act}_S, X} \langle c_2 : \beta[\ell'], \sigma'\rangle} \quad \text{(H-8)}$$

Rule H-9 differs from Rule H-8 only in that the target location does not have a substructure.

$$\frac{\begin{array}{c}\sigma, \sigma' \models_\beta \langle \ell, g, a, (J, \mathrm{up}), \ell', \mathrm{cvar}, X\rangle, \quad \ell' \notin \mathrm{dom}(h), \\ (\ell, g, a, (J, \mathrm{up}), \ell') \notin D \Rightarrow \left(\langle c, \sigma\rangle \xrightarrow{A_0, \mathrm{true}} \langle c', \sigma\rangle \vee \ell \notin \mathrm{dom}(h) \right) \end{array}}{\langle c : \beta, \sigma\rangle \xrightarrow{a, a \in \mathrm{act}_S, X} \langle \beta[\ell'], \sigma'\rangle} \quad \text{(H-9)}$$

Rule H-10 models the action transition that results from execution of the substructure. Transition $\langle c, \sigma\rangle \xrightarrow{a, b, X \cup \mathrm{cvar}} \langle c', \sigma'\rangle$ in the premise of Rule H-10 ensures that the control variables inherited from the environment (X) and the control variables (cvar) of the automaton β will not jump arbitrarily when the action is executed by the substructure.

$$\frac{\langle c, \sigma\rangle \xrightarrow{a, b, X \cup \mathrm{cvar}} \langle c', \sigma'\rangle}{\langle c : \beta, \sigma\rangle \xrightarrow{a, a \in \mathrm{act}_S, X} \langle c' : \beta, \sigma'\rangle} \text{ (H-10)}$$

$$\frac{\begin{array}{c}\varrho \models \langle t, \ell, \mathrm{init}, \mathrm{inv}, \mathrm{tcp}, \mathrm{dtype}\rangle, \\ \langle c, \varrho(0)\rangle \xmapsto{\varrho, A, \theta_0, \omega_0} \langle c', \varrho(t)\rangle\end{array}}{\langle c : \beta, \varrho(0)\rangle \xmapsto{\varrho, \mathrm{act}_S, \theta, \omega} \langle c' : \beta[\ell], \varrho(t)\rangle} \text{ (H-11)}$$

Rule H-11 models the passage of time in an automaton postfix such that the timed transitions are (recursively) synchronized in every level of hierarchy of $c : \beta$, where, $\mathrm{dom}(\omega) = \mathrm{dom}(\varrho), \mathrm{dom}(\theta) = \mathrm{dom}(\varrho)$,

$$\forall t' \in [0, t].\Big[\omega(t') = (\omega_0(t') \wedge \varrho(t') \models \mathrm{term}(\ell))\Big], \text{ and}$$

$$\forall t' \in [0, t].\Big[\theta(t') = (\theta_0(t') \cup \{a \mid (\ell, g, a, (J, \mathrm{up}), \ell') \in E \wedge \varrho(t') \models g \wedge \omega_0(t')\})\Big].$$

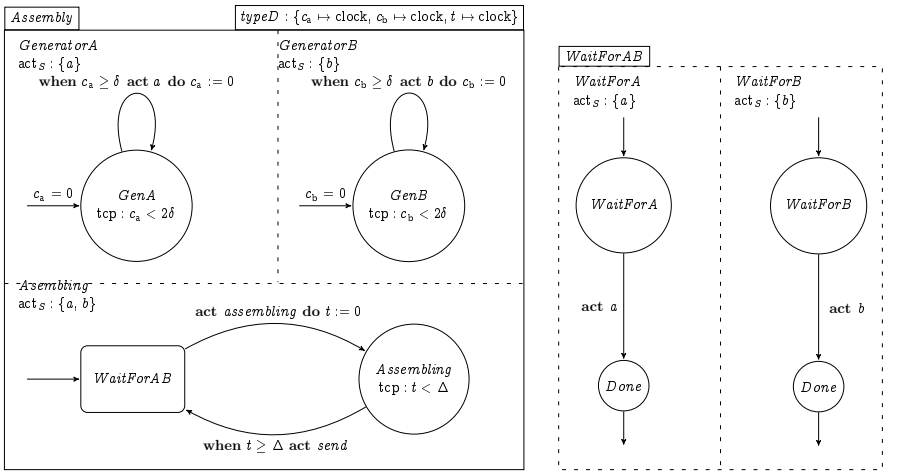
Finally, Rule H-12 models the execution of an environment transition in an automaton postfix.

$$\frac{\begin{array}{c}\sigma \models \mathrm{init}(\ell), \quad \sigma \models \mathrm{inv}(\ell), \quad \sigma' \models \mathrm{inv}(\ell), \\ \sigma \uparrow \mathrm{cvar} = \sigma' \uparrow \mathrm{cvar}, \qquad \langle c, \sigma\rangle \overset{A', b}{\dashrightarrow} \langle c', \sigma'\rangle\end{array}}{\langle c : \alpha, \sigma\rangle \overset{\mathrm{act}_S, \sigma \models \mathrm{term}(\ell) \wedge b}{\dashrightarrow} \langle c' : \alpha[\ell], \sigma'\rangle} \text{ (H-12)}$$

7.3.3 Parallel composition

The parallel composition operator allows concurrent execution of HCIF compositions. The semantics of parallel composition is almost unchanged with respect to the parallel composition of CIF. Action behavior is not affected by the addition of hierarchy. The rules for time and environment transitions are updated to reflect that a parallel composition is terminating only if both components are.

As an illustration, consider the assembly process shown in Figure 7.4(a), henceforth referred to as *Assembly*, such that its location *WaitForAB* contains the parallel composition shown in Figure 7.4(b). The assembly process initially is in the *WaitForAB* location and according to the semantics of the atomic automata, it can trigger action *assembling* only if its sub-structure terminates. Since the sub-structure is a parallel composition of two automata, namely *WaitForA* and *WaitForB* (see Figure 7.4(b)), the sub-structure $h(\mathit{WaitForAB})$ can terminate after actions a and b have both been executed; i.e., both automata *WaitForA* and *WaitForB* can terminate. This



(a) Assembly process ($Assembly$). (b) Receive process ($h(WaitForAB)$).

FIGURE 7.4: An assembly line.

pattern, in which an action is triggered after a series of parallel processes terminate, can be expressed succinctly using hierarchy. Without support for hierarchy and termination it is necessary to rewrite the parallel processes into a flat automaton.

The addition of hierarchy facilitates inter-level synchronization. As an example consider the generator process $GeneratorA$ shown in Figure 7.4(a), which enables an action a every δ time units, when $c_a \geq \delta$. The action a from the generator synchronizes with action a specified as synchronizing in the automaton $WaitForA$, which is part of the substructure of location $WaitForAB$. This synchronizing behavior is obtained by inclusion of action a in the set of synchronizing actions act_S of $GeneratorA$ ($\{a\}$) and in the set of synchronizing actions of automaton $Assembly$ ($\{a, b\}$). Note that strictly speaking, action a need not be defined as synchronizing for the automaton $WaitForA$.

Formally, Rule H-13 states that two synchronizing actions with the same label can execute in parallel only if they share the same initial and final valuation, and if the action is synchronizing in both compositions. The set of control variables X, is propagated from the conclusions to the premises since the control variables in the scope of a parallel composition are shared by both partners. The resulting action transition is also synchronizing which allows action a to synchronise with more than two compositions.

$$\frac{\langle c_1, \sigma\rangle \xrightarrow{a,\mathrm{true},X} \langle c_1', \sigma'\rangle, \quad \langle c_2, \sigma\rangle \xrightarrow{a,\mathrm{true},X} \langle c_2', \sigma'\rangle}{\langle c_1 \parallel c_2, \sigma\rangle \xrightarrow{a,\mathrm{true},X} \langle c_1' \parallel c_2', \sigma'\rangle} \text{ (H-13)}$$

Rule H-14 models interleaving behavior of two compositions when executed in parallel. In these rules, an action can be performed in one of the components (c_1) only if the initial and final valuations are consistent with the other composition (c_2); and if this action is not synchronizing in the other component, which is expressed by the condition $a \notin A'$. The environment transition $(c_2, \sigma) \overset{A',b'}{\dashrightarrow} (c_2', \sigma')$ is used to obtain the set of synchronizing action labels in composition c_2, to ensure that the initial valuation σ is consistent with the active invariants and initialization conditions of c_2.

$$\frac{\langle c_1, \sigma\rangle \xrightarrow{a,b,X} \langle c_1', \sigma'\rangle, \quad \langle c_2, \sigma\rangle \overset{A',b'}{\dashrightarrow} \langle c_2', \sigma'\rangle, \ a \notin A'}{\begin{array}{c}\langle c_1 \parallel c_2, \sigma\rangle \xrightarrow{a,b,X} \langle c_1' \parallel c_2', \sigma'\rangle \\ \langle c_2 \parallel c_1, \sigma\rangle \xrightarrow{a,b,X} \langle c_2' \parallel c_1', \sigma'\rangle\end{array}} \text{(H-14)}$$

Rule H-15 models the fact that if two compositions are put in parallel, t time units can elapse, whenever it is allowed by both partners, where

$$\forall t' \in [0,t].\big[\theta_{01}(t') = (\theta_0(t') \cap \theta_1(t')) \cup (\theta_0(t') \setminus A_1) \cup (\theta_1(t') \setminus A_0)\,\big], \text{ and}$$
$$\forall t' \in [0,t].\big[\omega_{01}(t') = \omega_0(t') \wedge \omega_1(t')\big]$$

The equation $\theta_{01}(t') = (\theta_0(t') \cap \theta_1(t')) \cup (\theta_0(t') \setminus A_1) \cup (\theta_1(t') \setminus A_0)$ states that

1. if an action a is enabled in the composition c_1 and c_2 at a time point $t' \in [0,t]$, then the action a is also enabled in the parallel composition at the time point t'.

2. if an action a is enabled in c_1 (or c_2) at a time point $t' \in [0,t]$ and a is not synchronising in the other component c_2 (or c_1), then the actions a is also enabled in the parallel composition at the time point t'.

The termination trajectory in the parallel composition at a given time point t' is the conjunction of the termination trajectories of the respective components at the same time instant t'. This is encoded in $\forall t' \in [0,t].\big[\omega_{01}(t') = \omega_0(t') \wedge \omega_1(t')\big]$.

$$\frac{\langle c_1, \varrho(0)\rangle \overset{\varrho,A_0,\theta_0,\omega_0}{\longmapsto} \langle c_1', \varrho(t)\rangle, \quad \langle c_2, \varrho(0)\rangle \overset{\varrho,A_1,\theta_1,\omega_1}{\longmapsto} \langle c_2', \varrho(t)\rangle}{\langle c_1 \parallel c_2, \varrho(0)\rangle \overset{\varrho,A_0\cup A_1,\theta_{01},\omega_{01}}{\longmapsto} \langle c_1' \parallel c_2', \varrho(t)\rangle} \text{(H-15)}$$

Rule H-16 defines the environment transition behavior for parallel composition. The resulting set of synchronizing actions is the union of the synchronizing actions of c_1 and c_2. The conjunction $b_0 \wedge b_1$ models the fact that a parallel composition is terminating if its components are. Note that the end

valuations of all transitions match.

$$\frac{\langle c_1, \sigma\rangle \overset{A_0, b_0}{\dashrightarrow} \langle c_1', \sigma'\rangle, \langle c_2, \sigma\rangle \overset{A_1, b_1}{\dashrightarrow} \langle c_2', \sigma'\rangle}{\langle c_1 \parallel c_2, \sigma\rangle \overset{A_0 \cup A_1, b_0 \wedge b_1}{\dashrightarrow} \langle c_1' \parallel c_2', \sigma'\rangle} \text{ (H-16)}$$

7.3.4 Urgency operator

By means of the urgency operator it is possible to declare actions as urgent. This means that time cannot pass if an urgent action is enabled. However, urgent actions do not have priority over regular (non-urgent) actions.

For example, consider the model of the controller of Figure 7.1 with active location *UpOut*. When a user demands the controller to operate in the horizontal mode, it should react as soon as possible. In other words, the action τ in the labelled edge **when** $s = +1$ between the locations *UpOut* and *Horizontal* must be urgent. This ensures that time does not pass in the location *UpOut* from the instant when the guard $s = +1$ is enabled.

Rule H-17 specifies that the urgent action operator restricts the time behavior of a composition in such a way that time can pass as long as no urgent action is enabled.

$$\frac{\langle c, \sigma\rangle \overset{\varrho, A', \theta, \omega}{\longmapsto} \langle c', \sigma'\rangle, \quad \forall t' \in [0, t). \left[a \notin \theta(t')\right]}{\langle \upsilon_a(c), \sigma\rangle \overset{\varrho, A', \theta, \omega}{\longmapsto} \langle \upsilon_a(c'), \sigma'\rangle} \text{ (H-17)}$$

The urgency operator affects only the time behavior. Action and environment transitions remain unchanged as expressed by Rules H-18 and H-19.

$$\frac{\langle c, \sigma\rangle \xrightarrow{\ell, b, X} \langle c', \sigma'\rangle}{\langle \upsilon_a(c), \sigma\rangle \xrightarrow{\ell, b, X} \langle \upsilon_a(c'), \sigma'\rangle} \text{ (H-18)} \qquad \frac{\langle c, \sigma\rangle \overset{A', b}{\dashrightarrow} \langle c', \sigma'\rangle}{\langle \upsilon_a(c), \sigma\rangle \overset{A', b}{\dashrightarrow} \langle \upsilon_a(c'), \sigma'\rangle} \text{ (H-19)}$$

Stateless bisimulation

It is clear from the definition of a HTS (Definition 4.1) that a state in a transition system consists of a process part (a behavioural entity) and a data part (valuation). Furthermore, we know that the stateless bisimulation is the most robust equivalence for the transition systems whose states contains data [25, 62]. In this section, we show that the semantics of HCIF is compositional with respect to stateless bisimulation [62].

Definition 7.8. A binary symmetric relation $\mathcal{R} \subseteq \mathbb{C} \times \mathbb{C}$ is called a *stateless bisimulation* relation iff the following conditions are satisfied.

1. $\forall c_1, c_2, c_1', \sigma, \sigma', X, a, b. \Big[\langle c_1, \sigma\rangle \xrightarrow{a,b,X} \langle c_1', \sigma'\rangle \wedge (c_1, c_2) \in \mathcal{R} \Rightarrow \exists c_2'. \Big[\langle c_2, \sigma\rangle \xrightarrow{a,b,X} \langle c_2', \sigma'\rangle \wedge (c_1', c_2') \in \mathcal{R}\Big]\Big]$.

2. $\forall c_1, c_2, c_1', \sigma, \sigma', \varrho, A', \theta, \omega. \Big[\langle c_1, \sigma\rangle \xmapsto{\varrho,A',\theta,\omega} \langle c_1', \sigma'\rangle \wedge (c_1, c_2) \in \mathcal{R} \Rightarrow \exists c_2'. \Big[\langle c_2, \sigma\rangle \xmapsto{\varrho,A',\theta,\omega} \langle c_2', \sigma'\rangle \wedge (c_1', c_2') \in \mathcal{R}\Big]\Big]$.

3. $\forall c_1, c_2, c_1', \sigma, \sigma', A', b. \Big[\langle c_1, \sigma\rangle \dashrightarrow^{A',b} \langle c_1', \sigma'\rangle \wedge (c_1, c_2) \in \mathcal{R} \Rightarrow \exists c_2'. \Big[\langle c_2, \sigma\rangle \dashrightarrow^{A',b} \langle c_2', \sigma'\rangle \wedge (c_1', c_2') \in \mathcal{R}\Big]\Big]$.

Two compositions c_1, c_2 are said to be stateless bisimilar, notation $c_1 \leftrightarrows_{\mathrm{ss}} c_1$, iff there exists a stateless bisimulation relation $\mathcal{R}$ such that $(c_1, c_2) \in \mathcal{R}$.

Theorem 7.9. *Stateless bisimulation is a congruence for all the constructs of HCIF.*

Proof. The SOS rules of HCIF are in the *process-tyft* format [62], which guarantees the congruence for stateless bisimilarity. □

7.4 Flattening of HCIF models

In this section, we present a technique that converts a hierarchical automaton into a flat automaton such that they are stateless bisimilar. That is, we define a procedure that eliminates the hierarchy function of a hierarchical automaton and produces an equivalent flat automaton modulo stateless bisimulation. Such techniques in the field of process algebras (see [6]) are known as *linearisation*, or as *elimination* of operators. The advantage of *flattening* a hierarchical automaton is to allow the usage of existing tools such as simulators of the CIF language.

Definition 7.10. The *depth* $D(c)$ of a composition $c \in \mathbb{C}$ is recursively defined:

$$\begin{aligned} D(\beta) &= 1 + \max_{\ell \in \mathrm{dom}(h)} D(h(\ell)) \\ D(c_1 \parallel c_2) &= \max(D(c_1), D(c_2)) \\ D(\square(c)) &= D(c) \quad \square \in \{[\![_{\mathrm{V}}\ \_ :: \_\ ]\!], [\![_{\mathrm{A}}\ \_ :: \_\ ]\!], \upsilon\_(\_)\}. \end{aligned}$$

An automaton has a *finite* depth, whenever its depth is defined. An automaton of depth 1 (i.e., $\mathrm{dom}(h) = \emptyset$) is called a *flat* automaton.

We write $\beta.X$ to access the component X of an automaton β that is of the form $(L, \text{init}, \text{inv}, \text{tcp}, E, D, \text{cvar}, \text{act}_S, \text{dtype}, \text{term}, h)$. For example, the notation $\beta.L$ denotes the set of locations L of the automaton β. For a given predicate $g \in P$, we assume the existence of the operator $\_^+ : P \to P^+$, i.e., all the variables in g^+ belong to the set $V^+ \cup \dot{V}^+$.

7.4.1 Flattening of HCIF compositions

Suppose that we have a procedure Ψ that turns any composition of CIF into a stateless bisimilar CIF automaton (see, [32] for instance). Note that a CIF automaton is a flat automaton. Then we can lift this procedure to any finite HCIF compositions $c \in \mathbb{C}$, by first applying a flattening procedure (Ψ_f, see Definition 7.12) to all the components of c before applying the procedure Ψ to the result. Formally, for any $c, c_1, c_2, \beta \in \mathbb{C}$ we have,

$$
\begin{array}{lcl}
\Psi_\text{f}(c_1 \parallel c_2, \varsigma) & = & \Psi(\Psi_\text{f}(c_1, \varsigma) \parallel \Psi_\text{f}(c_2, \varsigma)) \\
\Psi_\text{f}(\llbracket_\text{A}\, a :: c \,\rrbracket, \varsigma) & = & \Psi(\llbracket_\text{A}\, a :: \Psi_\text{f}(c, \varsigma) \,\rrbracket) \\
\Psi_\text{f}(\upsilon_a(c), \varsigma) & = & \Psi(\upsilon_a(\Psi_\text{f}(c, \varsigma))) \\
\Psi_\text{f}(\llbracket_\text{V}\, x = e_0, \dot{x} = e_1 :: c \,\rrbracket, \varsigma) & = & |[_V x = \bot, \dot{x} = \bot :: \\
 & & \quad \Psi_\text{f}(c, \varsigma \cup \{x \mapsto e_0, \dot{x} \mapsto e_1\})]| \\
\Psi_\text{f}(\llbracket_\text{V}\, x = e_0, \dot{x} = e_1 :: \beta \,\rrbracket, \varsigma) & = & \llbracket_\text{V}\, x = \bot, \dot{x} = \bot :: \beta' \,\rrbracket
\end{array}
$$

where

- $\beta = (L, \text{init}, \text{inv}, \text{tcp}, E, D, \text{cvar}, \text{act}_S, \text{dtype}, \text{term}, h)$,
- $\beta' = \Psi_\text{f}((L, \text{init}', \text{inv}, \text{tcp}, E, D, \text{cvar}, \text{act}_S, \text{dtype}, \text{term}, h))$, and
- init' is defined for all $\ell \in L$ by:

$$\text{init}'(\ell) = \text{init}(\ell) \wedge x = e_0 \wedge \dot{x} = e_1 \wedge \bigwedge_{y \in \text{Dom}(\varsigma)} (y = \varsigma(y) \wedge \dot{y} = \varsigma(y)) \,.$$

Note that for flattening the compositions contained in a variable scope, as in $\Psi_\text{f}(\llbracket_\text{V}\, x = e_0, \dot{x} = e_1 :: c \,\rrbracket, \varsigma)$, we assume that all variables are unique. This can be easily realized by means of variable renaming.

7.4.2 Requirements for flattening

Unfortunately, a HCIF automaton cannot be linearised to a CIF automaton in general. This is because the calculation of the control variables and the dynamic types in a HCIF automaton depends on the active locations of a HCIF automaton and thus, also on the substructures inside these active locations. So

a HCIF automaton can be flattened only if the *global set of control variables* does not depend on the active location because the set of control variables is constant for a flat automaton. The global set of control variables of an automaton is defined as the union of its own set of control variables and the sets of control variables of all active substructures.

Let $\mathbb{A}$ denote the set of all HCIF automata. To establish whether the global set of control variables of a hierarchical automaton is independent of the active locations of the automaton, a *control variable range* function $\mathrm{cvr} : \mathbb{A} \to 2^{2^V}$ is introduced. This functions returns the sets of control variables for all combinations of active locations of the automaton and its substructures. If this set contains exactly one element, then the set of control variables of the automaton is constant; thus, in principle a HCIF automaton can be flattened.

Definition 7.11. The *control variable range* $\mathrm{cvr} : \mathbb{A} \to 2^{2^V}$ is defined in the following way.

$$\begin{aligned}
\mathrm{cvr}(\bot) &= \emptyset \\
\mathrm{cvr}(c_1 \parallel c_2) &= \mathrm{cvr}(c_1) \cup \mathrm{cvr}(c_2) \\
\mathrm{cvr}(\square(c)) &= \mathrm{cvr}(c) \qquad \square \in \{[\![_{\mathrm{V}}\ _ :: _\ ]\!], [\![_{\mathrm{A}}\ _ :: _\ ]\!], \upsilon_(_)\} \\
\mathrm{cvr}(\beta) &= \bigcup_{\ell \in \beta.L} \{\mathrm{cvr}((\beta.h)(\ell)) \cup \mathrm{cvar}\}.
\end{aligned}$$

Likewise, the *dynamic type range* $\mathrm{dtr} : \mathbb{A} \to 2^{V \to 2^{(T \to \Lambda) \times (T \to \Lambda)}}$ is a function that returns the set of global dynamic types of a finite hierarchical automaton.

$$\begin{aligned}
\mathrm{dtr}(\bot) &= \emptyset \\
\mathrm{dtr}(c_1 \parallel c_2) &= \mathrm{dtr}(c_1) \cap \mathrm{dtr}(c_2) \\
\mathrm{dtr}(\square(c)) &= \mathrm{dtr}(c) \qquad \square \in \{[\![_{\mathrm{V}}\ _ :: _\ ]\!], [\![_{\mathrm{A}}\ _ :: _\ ]\!], \upsilon_(_)\} \\
\mathrm{dtr}(\beta) &= \bigcup_{\ell \in \alpha.L} \{\mathrm{dtr}((\alpha.h)(\ell)) \cap \mathrm{dtype}\}.
\end{aligned}$$

A HCIF automaton $\alpha \in \mathbb{A}$ is *linearisable* iff $|\mathrm{cvr}(\alpha)| = 1$ and $|\mathrm{dtr}(\alpha)| = 1$.

Definition 7.12. Let β be a linearisable automaton of the form

$$(L, \mathrm{init}, \mathrm{inv}, \mathrm{tcp}, E, D, \mathrm{cvar}, \mathrm{act}_S, \mathrm{dtype}, \mathrm{term}, h).$$

We define, if $\mathrm{Dom}(h) = \emptyset$, then

$$\Psi_{\mathrm{f}}(\beta) \triangleq (L, \mathrm{init}, \mathrm{inv}, \mathrm{tcp}, E, D, \mathrm{cvar}, \mathrm{act}_S, \mathrm{dtype}, \mathrm{term}, \emptyset),$$

and otherwise

$$\Psi_{\mathrm{f}}(\beta) \triangleq (\hat{L}, \hat{\mathrm{init}}, \hat{\mathrm{inv}}, \hat{\mathrm{tcp}}, \hat{E}, \hat{D}, \hat{\mathrm{cvar}}, \mathrm{act}_S, \hat{\mathrm{dtype}}, \hat{\mathrm{term}}, \emptyset, \hat{X}).$$

Here, we have used the shorthand notation

$$(L, \mathrm{init}, \mathrm{inv}, \mathrm{tcp}, E, D, \mathrm{cvar}, \mathrm{act}_S, \mathrm{dtype}, \mathrm{term}, h, X),$$

that extends an automaton β with a list of variables $X = [x_0, \cdots, x_n]$. It is defined as the following composition:

$$|[_{\mathrm{V}}\ x_0 = \bot, \dot{x}_0 = \bot :: |[_{\mathrm{V}}\ \cdots :: |[_{\mathrm{V}}\ x_n = \bot, \dot{x}_n = \bot :: \beta\ ]|\ \cdots\ ]|]| \ .$$

The elements of the flattened automaton $\Psi_{\mathrm{f}}(\alpha)$ are defined below.

- The set $\hat{L}$ is defined by:

  $$\hat{L} = \{(\ell, \bot) \mid \ell \in L, h(\ell) = \bot\} \cup \{(\ell, \ell') \mid \ell \in L, h(\ell) \neq \bot, \ell' \in \Psi_{\mathrm{f}}(h(\ell)).L\}.$$

  Note that the set $\hat{L}$ is also the domain of the functions $\hat{Op}$ where, $Op \in \{\mathrm{init}, \mathrm{term}, \mathrm{inv}, \mathrm{tcp}\}$.

- Let $\mathrm{Op} \in \{\mathrm{init}, \mathrm{term}, \mathrm{inv}, \mathrm{tcp}\}$. The functions Op which return the predicates are defined in the following way for all $(\ell, \ell') \in \hat{L}$:

  $$\hat{\mathrm{Op}}((\ell, \ell')) = \begin{cases} \mathrm{Op}(\ell) & \text{if } \ell' = \bot \\ \mathrm{Op}(\ell) \ \wedge \ \Psi_{\mathrm{f}}(h(\ell)).\mathrm{Op}(\ell') & \text{if } \ell' \neq \bot \end{cases}.$$

- The set of control variables and the dynamic type for the flat automaton is defined by:

  $$\hat{\mathrm{cvar}} \triangleq x \quad \text{where, } \mathrm{cvr}(\alpha) = \{x\}$$
  $$\hat{\mathrm{dtype}} \triangleq x \quad \text{where, } \mathrm{dtr}(\alpha) = \{x\}.$$

- The set of local variables X is defined as:

  $$X = \bigcup_{\ell \in \mathrm{dom}(h)} \Psi_{\mathrm{f}}(h(\ell)).X \ .$$

- Finally, the set of edges $\hat{E}$ is defined in the following way.

  $$\hat{E} = \Big\{ \Big((\ell_0, \ell_0'), \hat{g}, a, (\hat{J}, \hat{\mathrm{up}}), (\ell_1, \ell_1')\Big) \mid$$
  $$(\ell_0, \ell_0') \in \hat{L} \wedge (\ell_1, \ell_1') \in \hat{L} \wedge (\ell_0, g, a, (J, \mathrm{up}), \ell_1) \in \beta.E \ \wedge$$
  $$\hat{g} = \begin{cases} g, & \text{if } (\ell_0, g, a, (J, \mathrm{up}), \ell_1) \in \beta.D \\ \Psi_{\mathrm{f}}(h(\ell_0)).\mathrm{term}(w_0) \wedge g, & \text{if } (\ell_0, g, a, (J, \mathrm{up}), \ell_1) \in \beta.E \setminus \beta.D \end{cases} \wedge$$
  $$\hat{J} = J \cup \Psi_{\mathrm{f}}(h(\ell_1)).X \ \wedge$$
  $$\hat{\mathrm{up}} = \mathrm{up} \wedge (\Psi_{\mathrm{f}}(h(\ell_1)).\mathrm{init})(\ell_1') \Big\} \bigcup$$

$$\left\{\Big((\ell_0,\ell_0'),\hat{g},a,(\hat{J},\hat{\mathrm{up}}),(\ell_0,\ell_1')\Big) \mid (\ell_0',\hat{g},a,(\hat{J},\hat{\mathrm{up}}),\ell_1') \in \Psi_{\mathrm{f}}(h(\ell_0)).E\right\}.$$

It is assumed that all the references to initialisation predicates, or termination predicates to be true when a location $\ell \in L$ has no substructure. Thus, $(\Psi_{\mathrm{f}}(h(\ell_0)).\mathrm{term})(\ell_0') = \mathrm{true}$ and $(\Psi_{\mathrm{f}}(h(\ell_1).\mathrm{init})(\ell_1')^{+} = \mathrm{true}$ in the above conditions, whenever $\ell_0 \notin \mathrm{dom}(h)$ and $\ell_1 \notin \mathrm{dom}(h)$, respectively. Likewise, the set of local variables $\Psi_{\mathrm{f}}(h(\ell_1)).X = \emptyset$, whenever $\ell_1 \notin \mathrm{dom}(h)$. In this way, we get definitions for the simpler cases derived from the above one.

7.5 Case-study: Patient Support System

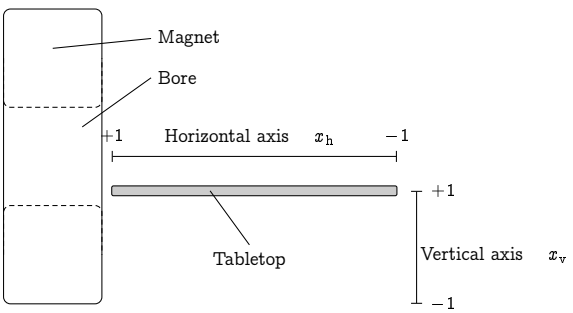


FIGURE 7.5: Patient Support System.

The patient support system (see Figure 7.5) is used in medical diagnosis to position a patient in a MRI scanner [72]. The system can be operated in the following modes: vertical mode, horizontal mode, and user interface mode. In the vertical mode, the table top on which a patient resides can only move vertically between the bounds depicted in Figure 7.5. Similarly, in the horizontal mode, the table top can be moved in or out of the bore, either manually or by means of a motor drive. Furthermore, the system is equipped with a table top release switch for emergency situations. This system is controlled via a user interface that contains a tumble switch to control the movement (both horizontally and vertically) of the table, and a button to enable the start of an initialization sequence. The position of the tumble switch is represented by variable s which can have the values $+1$, 0, and -1. The continuous variables x_{h} and x_{v} represent the horizontal and vertical position of the table top, respectively.

The objective of this case-study is to design a controller that satisfies the following requirements. The table should move up and down, or in and out of the bore, by operating the tumble switch. The table should not move beyond the boundaries shown in Figure 7.5. The case-study is specified using a top-down design methodology. In other words, we first model the overall

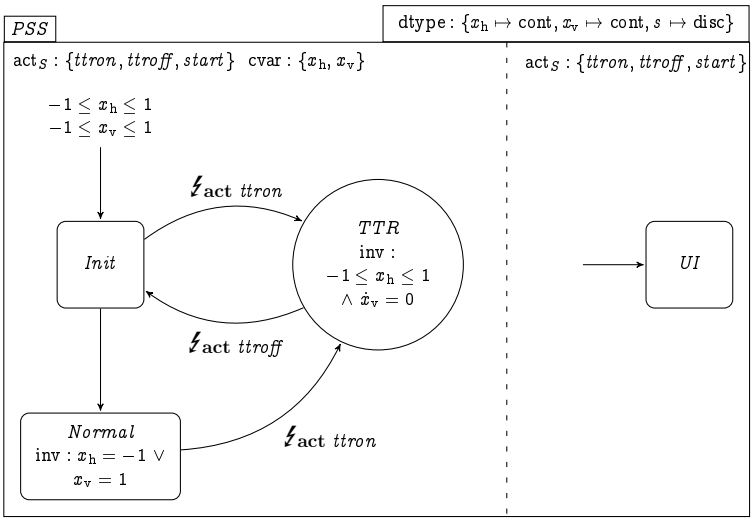


FIGURE 7.6: Patient Support System.

system at a higher level of abstraction in which we identify that the system consists of a controller and a user interface. Furthermore, a controller can run in the following three modes: *Init* mode in which the controller should place the table in the initial position; *Normal* mode in which the controller synchronises with the events of the tumble switch; *TTR* (Table Top Release) mode in which an operator is allowed to override the normal execution of the controller. Figure 7.6 shows the model of the system at this level of abstraction. Throughout the complete description of this case-study, it is assumed that only the τ action is urgent. All other actions, which are the actions generated by the user interface, are non-urgent. This is modeled by $v_\tau(PSS)$ where *PSS* represents the automaton *PSS* shown in Figure 7.6.

User interface The user interface consists of three input devices: the tumble switch, the table top release switch and the start button (see Figure 7.7). *MvUpOrIn*, *Neutral*, and *MvDownOrOut* are the three positions of the tumble switch. The *MvUpOrIn* position is used to move the table either up or into the bore, the *MvDownOrOut* position is used to move the table down or out of the bore. When the switch is released, it returns to the neutral position, which enforces actuated (motorized) movement of the table to stop.

The TTR switch can be used to release the table top from the horizontal motor. When the switch is active, the horizontal movement of the table is uncontrolled by the system, so that an operator can manually move the table freely in the horizontal direction. The start button is used to allow initialization of the system.

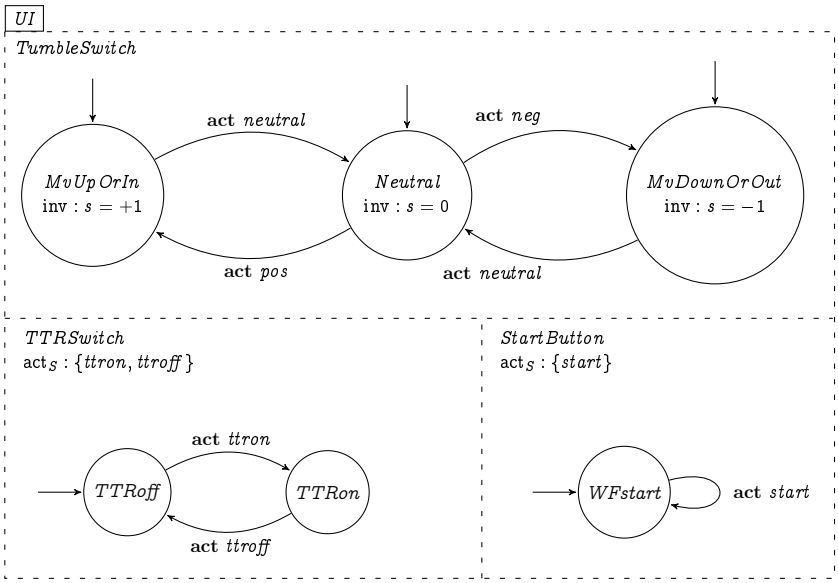


FIGURE 7.7: User interface.

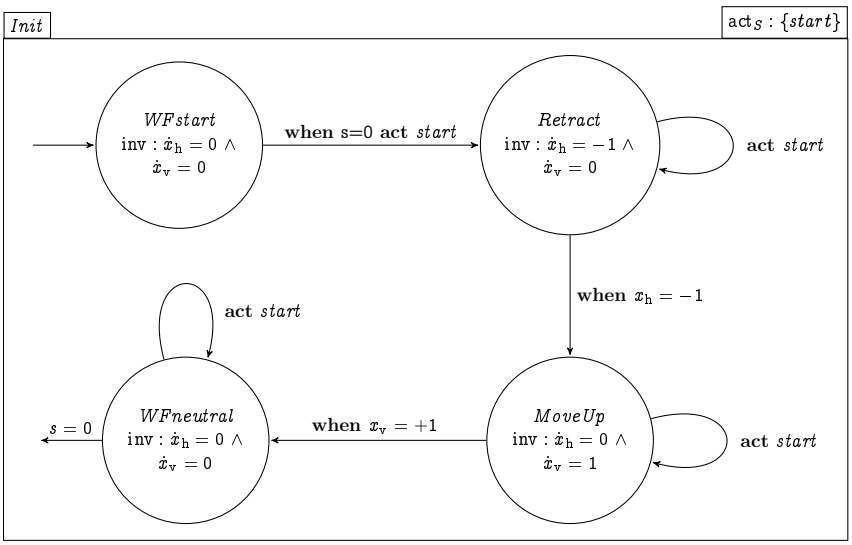


FIGURE 7.8: Initialization.

Initialization In the *Init* mode, the position of the patient support system is initialized (Figure 7.8). The position of the tumble switch needs to be neutral before initialization begins and the movement is triggered by pressing the start button. The desired final position of the table is fully retracted and fully up. First, the table is retracted since this is always a safe movement. Then, when the table is fully retracted, the table is moved up until it reaches the top position. The initialization is complete when the tumble switch is

in the neutral position, to prevent that the table starts moving immediately after initialization.

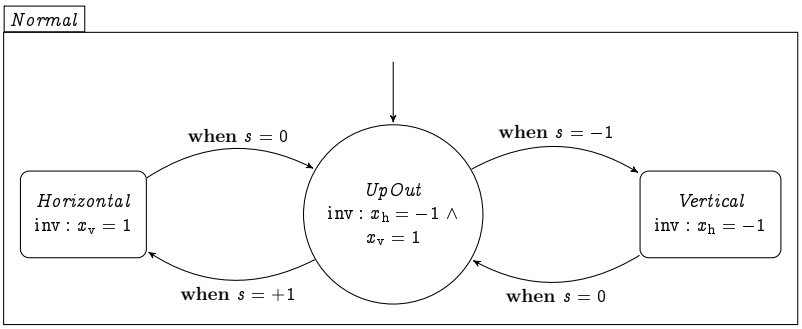


FIGURE 7.9: Normal movement control.

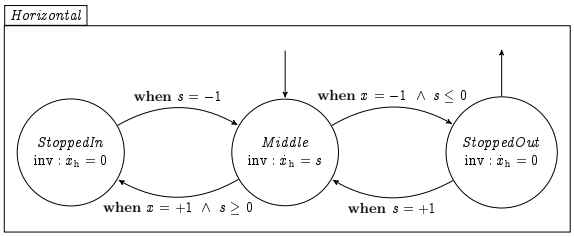
Normal mode Initially, the system enters the normal mode with the table fully up and retracted, so in an up and out position (Figure 7.9). In this intersection point between moving the table horizontally or vertically, holding the tumble switch in the *MvUpOrIn* position triggers horizontal movement of the table into the bore, whereas holding it in a *MvDownOrOut* position triggers vertical, downward movement.

A system requirement is that between switching from horizontal to vertical movement, and vice versa, the position of the tumble switch must be neutral. This to prevent the table from continuing movement unexpectedly in a different direction. Figures 7.10(a),and 7.10(b) show the horizontal and vertical movement of the system in more detail.

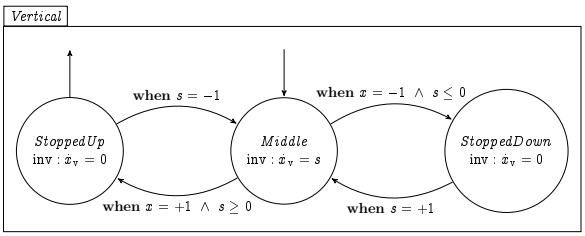
7.6 Related work

In the past, the following techniques were proposed in order to accommodate hierarchy in other formalisms.

- Action refinement [77]. In this approach an action in the alphabet of a process or an automaton is substituted by another process/automaton. However, the setting of action refinement is incompatible with the interleaving models of concurrency (see, e.g. [74]). Since CIF and its hierarchical extensions are based on interleaving models of concurrency, we disregard the technique of action refinement.
- Statecharts [47]. Statecharts were the first formalism that extended finite state machines with the concept of hierarchy. Conventionally, the semantics of statecharts requires a tree-structure on the set of locations



(a) Horizontal movement control.



(b) Vertical movement control.

FIGURE 7.10: Horizontal and vertical movements of the controller.

of a statechart. Consequently, additional concepts from tree-structures, like least common ancestors, children of a location, etc., complicate the semantics. In [20], it was shown that these additional concepts are unnecessary when reverting to structural operational semantics [68]. We only need to introduce the notion of a substructure, which we handle through the automaton postfix operator.

$$((c_1 : \beta_1) \parallel (c_2 : \beta_2)) : \beta_0$$

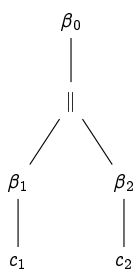


FIGURE 7.11: Relation between automaton postfix operator and state tree structures

Consider a state $\langle c, \sigma \rangle$ in the HTS, it becomes clear how the postfix operator helps us to mimick the state-tree structures used in the semantics of statecharts [47]. Figure 7.11 shows that a composition c in essence is a *tree*, where the postfix operator represents the edges of the tree and the parallel compositions represent the branching points in the tree. The root of this tree is the active location of the hierarchical automaton we described, while the leaves are the active substructures where the control over actions currently lies. Indeed, an informal comparison of our

semantics to that of statecharts suggest that the AND-superstates and the OR-superstates are represented by the *parallel compositions* and the *multiple initial locations* of substructures, respectively. The concepts of history retention (not considered in this work) is not supported directly in HCIF, although it can be emulated.

- Hierarchical timed automata [27, Chapter 4.] are extensions of statecharts with a finite set of clock variables modeling real time. Again the semantics of this formalism is based on the concepts of tree structures and for this reason we also disregard this approach. However, there is a common intuition about the passage of time in [27] with the current work. The time can pass in a hierarchical structure only if the time can pass in all the levels of hierarchy, i.e., time transitions must synchronise in all the levels of hierarchy of a hierarchical automaton.

- State refinement operator [74]. State refinement is a binary operator originating from the field of process algebras, denoted as $p[q]$, where p, q are arbitrary process terms. Informally, it means that p is a state with the substructure q. In other words, a location of an automaton is allowed to contain another automaton representing its substructure. Furthermore, it was also stated [74] that the above way of introducing hierarchy is compatible with the interleaving models of concurrency. Thus, the current work is motivated by [74], even though the basic entity in our formalism is an automaton rather than an action.

7.7 Conclusions

In this chapter, we have presented the syntax and semantics of HCIF, which extends CIF with hierarchy in a compositional manner, so that only the SOS rules for an automaton and for the time transitions of parallel composition need to be adapted.

We conjecture that we are able to transform a HCIF composition into a bisimilar CIF specification on the condition that the global set of controlled variables and the global dynamic type of the variables is independent of the active locations of the automata.This condition is needed because the dynamic type of variables and the set of control variables can change per location due to the presence of the substructures at different locations. These substructures, in principle, may have different dynamic types and different control variables.

Chapter 8

Conclusions

As outlined in Chapter 1, the model-based engineering approach allows an incremental development of an embedded system. In this approach, the process of refining a system model is key in obtaining a more exact description of system behaviour. Although, a model of an embedded system can be refined in a number of ways depending on the application domain, this thesis focussed only on refinement of communication and on refinement of states.

8.1 Refinement of communication

The part on the refinement of communication dealt with correct implementation of synchronous systems using asynchronous communication because synchronous systems are easier to understand and design than asynchronous systems, and often their implementation are asynchronous in nature [33, 36]. To address this issue, our objective was to render a synchronous system and its asynchronous version equivalent, by a suitable equivalence relation from the van Glabbeek spectrum [76]. Roughly, a synchronous system that is unaltered by the addition of communication buffers is called desynchronisable.

Another reason to study this refinement of synchrony into asynchrony is that the presence of buffers makes ensuring the correctness (via state exploration techniques) of an asynchronous system a non-trivial task. In general, if the buffers are modelled to have infinite capacity, such systems are known to be Turing complete as shown by Brand and Zafiropulo [22]. But also, if the buffers are modeled to have finite capacity, we may still face the state-space explosion problem. In this respect, it helps to separate concerns by first designing a correct synchronous system and then desynchronising it.

In general, the equivalence between any synchronous system and its asynchronous version cannot be established because addition of the buffers (bags, or queues) introduces more behaviour. For instance, deadlocks in the asynchronous system, the introduction of new traces in the asynchronous system, the reordering of traces present in the synchronous system, or delaying of choices present in the synchronous system.

Thus, we investigated the conditions in this thesis that are required to desynchronise a given synchronous system without building the transition system of the corresponding asynchronous system, which may have an infinite number of states. We noticed that sufficient conditions for desynchronisability depends upon the equivalence notion, the type of buffers, and the abstraction schemes used to construct an asynchronous system from a given synchronous system. In this thesis, desynchronisability was studied up to either branching bisimilarity or contra-similarity, buffers were unbounded and lossless queues or bags, and four abstraction schemes were proposed to ensure that the alphabet of the asynchronous system is identical to its synchronous version.

Our results indicate that (when comparing the conditions of Chapter 3 and Chapter 4) better desynchronisability conditions can be obtained by changing the properties of the communication protocol, rather than just focusing on the properties that the communicating processes should have to ensure desynchronisability. For instance, half-duplex queues avoid the diamond property from the sufficient conditions of desynchronisability (Theorems 4.15, 4.21). This leads to weaker conditions for desynchronisability than the sufficient conditions obtained in the past (cf. [13, 36, 73, 80]) for weaker equivalences. Moreover, we also showed that a reasonable characterisation of desynchronisation (at-least for concrete communicating processes, see Chapter 4) can be obtained even for the finest equivalence in the van Glabbeek spectrum.

A question which rises before applying the techniques developed in this thesis is: which of the abstraction schemes lead to a reasonable restriction on a synchronous system for desynchronisation? We believe that answer to this question depends upon the choice of buffers.

In particular, in the case of (half-duplex) queues, we argued in Chapter 4 that the abstraction scheme $\mathbf{A}_3$ leads to less restrictive conditions for desynchronisability than the other abstraction schemes $\mathbf{A}_1$, $\mathbf{A}_2$, or $\mathbf{A}_4$. Recall that the abstraction scheme $\mathbf{A}_3$ was the only abstraction scheme that preserves the external choice between the messages sent from the communicating processes in a synchronous system. On the contrary, the abstraction scheme $\mathbf{A}_1$ and $\mathbf{A}_2$ fail to preserve the external choices between the messages sent by the processes p and s, respectively. Lastly, the abstraction scheme $\mathbf{A}_4$ fails to preserve the external choices between the messages sent by both the process p and s (see the examples from Subsection 4.1.2 and Subsection 4.1.3).

However, in the case of bags, this thesis does not provide such a concrete answer that helps in selecting one abstraction scheme over the others. Note that, in general, the above combination of the half-duplex mechanism and the abstraction scheme $\mathbf{A}_3$ does not ensure desynchronisation in the presence of bags. In particular, the half-duplex bags introduces deadlocks in the asynchronous system; even though if a synchronous system is desynchronisable modulo $\leftrightarrows_{\mathbf{b}}$ under the abstraction scheme $\mathbf{A}_1$ and full-duplex bags (recall, the modified synchronous system of the Pusher-lift system from Chapter 5).

Another issue which we faced is that the half-duplex mechanism reduces the degree of parallelism in an asynchronous system because it restricts a sender to proceed with the output messages, while its input queue is non-empty. For this purpose, in Chapter 6, we studied the conditions under which it is possible to relax the half-duplex restriction on certain send messages. Our results indicate that a tradeoff between the full-duplex and the half-duplex buffering strategies is required in order to obtain a correct and an efficient implementation of an asynchronous system. Furthermore, the techniques of Chapter 4 were also extended to desynchronise a non-concrete synchronous system and a network of synchronously communicating processes.

We conclude the work on refinement of communication by sketching directions for future research.

1. An obvious extension, especially given the nature of embedded systems, is the desynchronisation of timed/hybrid synchronous systems modulo *timed branching bisimulation* [5]. However, there is no common understanding of such a notion in the literature of timed process algebras [9]. This is due to the following open problem: whether two branching bisimilar process must have exactly matching time behaviour? Furthermore, the current focus of researchers from the field of hybrid systems/cyber-physical systems is on the development of non-exact/approximate versions of behavioural relations (see, for instance, [35, 49]).

2. For deterministic supervisory control, we showed that it is possible to synthesize a controller that satisfies the well-posedness property by construction (Theorem 3.18). For other systems, however, this may not be so easy. Therefore, it would be beneficial if tools for model checking asynchronous systems, like mCRL2 [45] and CADP [59], could be optimized to check for well-posedness as well.

3. Well-posedness was the only property, which we were unable to remove from the sufficient conditions of desynchronisability presented in Chapter 3. A way to relax the conditions of well-posedness is by allowing a special kind of messages, called *negative acknowledgements* [43]. Intuitively, a receiver executes a negative acknowledgement message to inform the sender that it is unable to perform the output message of the sender. The introduction of negative acknowledgements raises the

following question: How to handle the messages in the buffer in case the receiver sends a negative acknowledgement? The answer to this question is crucial for giving the semantics to an asynchronous system, which is a prerequisite before establishing a desynchronisation result.

4. Throughout, the development of desynchronisation techniques in this thesis we assume that our buffers were lossless. However, in practice "real" buffers are lossy. Usually, the specification of a lossless buffer and a lossy buffer differs in the τ-transitions that forgets the elements from the buffer contents. In this respect, the conventional specification of a lossy buffer are sometimes considered too pessimistic and inadequate for verifying liveness or fairness properties (see [12]). In such cases, the message losses are associated with probabilities that leads to a more realistic model of an asynchronous system. Thus, this issue should be addressed in a desynchronisation study that allow buffers to be lossy before adopting probabilistic transition system as the computational model.

5. Finally, from Chapter 3 we know that an extra condition called reordering property arises while desynchronising a synchronous system in the presence of bags. A way to relax this condition is by augmenting multiple queues in an asynchronous system, rather than having a single bag for one direction. Recall the deadlock situation which arises in Example 5.1 (Chapter 5) due to bags. However, if we attached two input queues Q_1, Q_2 at the plant-side such that the messages n_1, n_2 travel via queue Q_1 and the message n_3 travel via Q_2. Then the modified asynchronous system does not have a deadlock because the self-loop labelled as $?n_2$ at p_1 is never executed as it will be guarded by the reception of the message n_1. As a consequence, we expect a weaker collection of sufficient conditions for desynchronisability in the presence of multiple queues than in the presence of unidirectional bags.

8.2 Refinement of states

The work on refinement of states focussed on the top-down development of models in the Compositional Interchange Format (CIF). CIF is a modelling language based on hybrid automata, which inherits some of the process algebraic operators to construct larger models from smaller models.

In his seminal paper [47], Harel argued that hierarchy, concurrency, and communication are sufficient to describe behaviour of a discrete system in an economical way and at the same time being formal and rigorous to do computerized simulation. In this respect, CIF language was lacking the notion of hierarchy, while the notions of concurrency and communication were already present in CIF since its inception (see [16]). Thus, in Chapter 7, we

extended CIF with the notion of hierarchy, which resulted in the Hierarchical Compositional Interchange Format (HCIF).

Syntactically, we introduced hierarchy in a CIF automaton by a hierarchy function h that associates a substructure to some locations of the CIF automaton. Semantically, a hierarchical automaton is defined in a compositional manner by referring only to the transition system of the substructures and not to their syntactic representation. This allowed us to describe the SOS rules of HCIF in the process-tyft format of Mousavi et al. [62]; thus, leading to stateless bisimulation as a congruence for all the constructs of HCIF for free.

For this purpose, an auxiliary operator called the automaton postfix operator was introduced to define the overall behaviour of a HCIF automaton. Thanks to this operator, we were able to give the semantics of a HCIF automaton without using the tree-structure (and its associated operations) on the set of locations. This is how we obtained a concise semantics for hierarchy even in the hybrid setting in contrast to the previous works [27, 54, 61]. It was these concepts of tree-structure that would have otherwise bring the considerable differences between the semantics of CIF and HCIF. Furthermore, we anticipate that the already well-established semantics of (timed) hierarchical automata [27] and statecharts [54, 61] can be made more simpler and concise in this regard.

Nevertheless, there were subtle changes in the SOS rules that triggers the time transitions in the various HCIF operators (w.r.t CIF). In CIF semantics, there was no need of guard trajectory and termination trajectory because there was no notion of termination in CIF. However, the notion of termination is an important one in HCIF semantics because it intuitively tells us when a non-disruptive edge can be executed in a HCIF automaton. For instance, an action of a non-disruptive edge is enabled in a location at a time instant t if the substructure at that location terminates at t. Such an information cannot be left out while giving semantics to a HCIF automaton (or even in case of a hierarchical timed automaton [27]).

Future work includes proving that HCIF is more expressive than CIF and defining the subset of HCIF that can be translated to CIF. Although, an initial prototype (see [37] for details) of Definition 7.12 has been implemented to flatten the hierarchical models of Patient-support case-study (Section 7.5). These transformations are important to reuse existing tools for CIF, including model transformations that will allow model checking of HCIF models via tools like SpaceEx [39] or Uppaal [53].

Appendix A

Proofs of main theorems in Chapter 3

Theorem A.1 (Proof of Theorem 3.11). *Let $p \parallel s$ be a synthesised synchronous system such that p, s are well-posed. If $p \parallel s$ is M_p-singular and satisfies the (M_p, M_s)-diamond property, then*

$$p \parallel s \leftrightarrows_{\mathbf{b}} \nabla_1(p \,|[\epsilon, \epsilon]|\, s).$$

Proof. Define a relation $\mathcal{B}$ in the following way:

$$\begin{aligned}\mathcal{B} = \Big\{ (p_1 \parallel s_1, \nabla_1(p_2 \,|[\mu,\nu]|\, s_1)) \;\Big|\; & p_1 \parallel s_1 \in \mathfrak{R}(p \parallel s) \wedge \\ & \nabla_1(p_2 \,|[\mu,\nu]|\, s_1) \in \mathfrak{R}(\nabla_1(p \,|[\epsilon,\epsilon]|\, s)) \wedge \\ & \exists p_2', s_2', p_2, s_2.\, \Big[p_2' \parallel s_2' \xrightarrow{\mu} \!\!\!\!\rightarrow p_1 \parallel s_1 \wedge p_2' \parallel s_2' \xrightarrow{\nu} \!\!\!\!\rightarrow p_2 \parallel s_2 \Big] \Big\}.\end{aligned}$$

Next, we show that the relation $\mathcal{B}$ is a branching bisimulation relation.

1. Let $p_1 \parallel s_1 \xrightarrow{\alpha} p_4 \parallel s_4$ and $(p_1 \parallel s_1, \nabla_1(p_2 \,|[\mu,\nu]|\, s_1)) \in \mathcal{B}$. From the construction of $\mathcal{B}$ we have

$$\exists p_2', s_2', p_2, s_2.\, \Big[p_2' \parallel s_2' \xrightarrow{\mu} \!\!\!\!\rightarrow p_1 \parallel s_1 \wedge p_2' \parallel s_2' \xrightarrow{\nu} \!\!\!\!\rightarrow p_2 \parallel s_2. \Big] \tag{A.1}$$

   Since, $E_p = \emptyset$, $E_s = \emptyset$ and the processes p, s are concrete we know that either, $\alpha \in M_p$, or $\alpha \in M_s$.

(a) Let $\alpha \in M_p$. Then, we have

$$\exists m.\left[\alpha = m \wedge p_1 \xrightarrow{!m} p_4 \wedge s_1 \xrightarrow{?m} s_4\right]. \tag{A.2}$$

i. Either, $\nu = \epsilon$. Then, due to the concreteness assumption we have $p_2 \parallel s_2 = p_2' \parallel s_2'$. Then, the transition $p_2' \parallel s_2' \xrightarrow{\mu} p_1 \parallel s_1$ in Equation A.1 is of the form $p_2 \parallel s_2 \xrightarrow{\mu} p_1 \parallel s_1$. And from Proposition 2.11 we get $p_2 \xrightarrow{?\mu} p_1$. Using this transition at state $\nabla_1(p_2 \,|[\mu,\epsilon]|\, s_1)$ we get

$$\nabla_1(p_2 \,|[\mu,\epsilon]|\, s_1) \longrightarrow \nabla_1(p_1 \,|[\epsilon,\epsilon]|\, s_1).$$

From the construction of $\mathcal{B}$ we get $(p_1 \parallel s_1, \nabla_1(p_1 \,|[\epsilon,\epsilon]|\, s_1)) \in \mathcal{B}$. Using the transitions $p_1 \xrightarrow{!m} p_4, s_1 \xrightarrow{?m} s_4$ (Equation A.2) we get $\nabla_1(p_1 \,|[\epsilon,\epsilon]|\, s_1) \xrightarrow{\tau} \nabla_1(p_4 \,|[\epsilon,m]|\, s_1) \xrightarrow{m} \nabla_1(p_4 \,|[\epsilon,\epsilon]|\, s_4)$. Using the transition $p_1 \parallel s_1 \xrightarrow{m} p_4 \parallel s_4$ in the construction of $\mathcal{B}$ we get $(p_1 \parallel s_1, \nabla_1(p_4 \,|[\epsilon,m]|\, s_1)) \in \mathcal{B}$. Clearly, from the construction of $\mathcal{B}$ we have $(p_4 \parallel s_4, \nabla_1(p_4 \,|[\epsilon,\epsilon]|\, s_4)) \in \mathcal{B}$.

ii. Or, $\nu \neq \epsilon$. Then, $\nu = \alpha'.\nu'$, for some $\alpha' \in M_p$, and $\nu' \in {M_p}^*$. Since $\nu = \alpha'.\nu'$, the transition $p_2' \parallel s_2' \xrightarrow{\nu} p_2 \parallel s_2$ (Equation A.1) can be written in the following way (see Figure A.1):

$$p_2' \parallel s_2' \xrightarrow{\alpha'} p_3' \parallel s_3' \xrightarrow{\nu'} p_2 \parallel s_2, \text{ for some } p_3', s_3' \in \mathbb{P}.$$

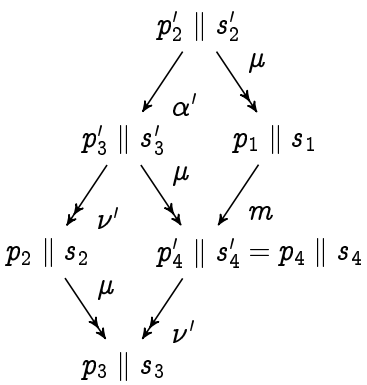


FIGURE A.1: Transitions derived in Case 1(a)ii.

Applying Corollary 3.10 at the state $p_2' \parallel s_2'$ we get (see Figure A.1): $\exists p_4', s_4'.\left[p_1 \parallel s_1 \xrightarrow{\alpha'} p_4' \parallel s_4' \wedge p_3' \parallel s_3' \xrightarrow{\mu} p_4' \parallel s_4'\right]$. If $\alpha' = m$, then by determinism we have $p_4' \parallel s_4' = p_4 \parallel s_4$. If $\alpha' \neq m$, then by M_p-singularity we have $\alpha' = m$. And, again by determinism we have $p_4' \parallel s_4' = p_4 \parallel s_4$. In any case, we have

$p_1 \parallel s_1 \xrightarrow{m} p_4 \parallel s_4$ and $p_3' \parallel s_3' \xrightarrow{\mu} p_4 \parallel s_4$. Now, applying Corollary 3.10 at the state $p_3' \parallel s_3'$ we get (see Figure A.1):

$$\exists p_3, s_3. \left[p_2 \parallel s_2 \xrightarrow{\mu} p_3 \parallel s_3 \wedge p_4 \parallel s_4 \xrightarrow{\nu'} p_3 \parallel s_3 \right].$$

Thus, $\nu = m.\nu' \wedge p_1 \parallel s_1 \xrightarrow{m} p_4 \parallel s_4 \xrightarrow{\nu'} p_3 \parallel s_3$. Furthermore, using the transition $s_1 \xrightarrow{?m} s_4$ (Equation A.2) and the fact $\nu = m.\nu'$ we get $\nabla_1(p_2 \,|[\mu, \nu]|\, s_1) \xrightarrow{m} \nabla_1(p_2 \,|[\mu, \nu']|\, s_4)$. Finally, using the transitions $p_3' \parallel s_3' \xrightarrow{\nu'} p_2 \parallel s_2$ and $p_3' \parallel s_3' \xrightarrow{\mu} p_4 \parallel s_4$ in the construction we have $(p_4 \parallel s_4, \nabla_1(p_2 \,|[\mu, \nu']|\, s_4)) \in \mathcal{B}$.

(b) Let $\alpha \in M_s$. Then, we have

$$\exists n. \left[\alpha = n \wedge p_1 \xrightarrow{?n} p_4 \wedge s_1 \xrightarrow{!n} s_4 \right].$$

Using the transition $s_1 \xrightarrow{!n} s_4$ at the state $\nabla_1(p_2 \,|[\mu, \nu]|\, s_1)$ we get

$$\nabla_1(p_2 \,|[\mu, \nu]|\, s_1) \xrightarrow{n} \nabla_1(p_2 \,|[\mu.n, \nu]|\, s_4).$$

Clearly, we have $p_2' \parallel s_2' \xrightarrow{\mu.n} p_4 \parallel s_4$ and $p_2' \parallel s_2' \xrightarrow{\nu} p_2 \parallel s_2$. Using the above transitions in the construction of $\mathcal{B}$ we conclude that $(p_4 \parallel s_4, \nabla_1(p_2 \,|[\mu.n, \nu]|\, s_4)) \in \mathcal{B}$.

2. Let $\nabla_1(p_2 \,|[\mu, \nu]|\, s_1) \xrightarrow{\tau} \nabla_1(p_4 \,|[\mu', \nu']|\, s_4)$, $(p_1 \parallel s_1, \nabla_1(p_2 \,|[\mu, \nu]|\, s_1)) \in \mathcal{B}$. From the construction of $\mathcal{B}$ we have

$$\exists p_2', s_2', p_2, s_2. \left[p_2' \parallel s_2' \xrightarrow{\mu} p_1 \parallel s_1 \wedge p_2' \parallel s_2' \xrightarrow{\nu} p_2 \parallel s_2 \right]. \quad \text{(A.3)}$$

Since, the given processes p, s are concrete and the abstraction scheme $\mathbf{A}_1$ is used we know that the transition

$$\nabla_1(p_2 \,|[\mu, \nu]|\, s_1) \xrightarrow{\tau} \nabla_1(p_4 \,|[\mu', \nu']|\, s_4)$$

is either, due to the removal of an element from μ, or due to the addition of an element in ν.

(a) Removal of an element from μ. Then, $\mu = n.\mu'$, $\nu' = \nu$, $p_2 \xrightarrow{?n} p_4$, and $s_1 = s_4$. Since, $\mu = n.\mu'$ we know that the transition $p_2' \parallel s_2' \xrightarrow{\mu} p_1 \parallel s_1$ (Equation A.3) can be written in the following way: $p_2' \parallel s_2' \xrightarrow{n} p_3' \parallel s_3' \xrightarrow{\mu'} p_1 \parallel s_1$, for some $p_3', s_3' \in \mathbb{P}$. Applying Corollary 3.10 at the state $p_2' \parallel s_2'$ we get

$$\exists p_4', s_4'. \left[p_3' \parallel s_3' \xrightarrow{\nu} p_4' \parallel s_4' \wedge p_2 \parallel s_2 \xrightarrow{n} p_4' \parallel s_4' \right].$$

Thus, $p_2 \xrightarrow{?n} p_4'$ and $s_2 \xrightarrow{!n} s_4'$. But, from above we have $p_2 \xrightarrow{?n} p_4$. Since the plant process is deterministic, we have $p_4 = p_4'$.

Finally, using the transition $p_3' \parallel s_3' \xrightarrow{\nu} p_4 \parallel s_4'$ and the transition $p_3' \parallel s_3' \xrightarrow{\mu'} p_1 \parallel s_1$ in the construction of $\mathcal{B}$ we conclude that $(p_1 \parallel s_1, \nabla_1(p_4 \,|[\mu', \nu]|\, s_1)) \in \mathcal{B}$.

(b) Addition of an element in ν. Then, $\nu = \nu'.m$, $\mu' = \mu$, $p_2 \xrightarrow{!m} p_4$, and $s_1 = s_4$. Since, the given p, s are well-posed, so assume a well-posedness relation $\mathcal{W}$ such that $(p, s) \in \mathcal{W}$. From Proposition 3.6 we have $(p_2, s_2) \in \mathcal{W}$. And from Definition 3.5 we get $p_2 \xrightarrow{!m} p_4 \wedge (p_2, s_2) \in \mathcal{W} \Rightarrow \exists s_4'.\left[s_2 \xrightarrow{?m} s_4'\right]$. Thus, $p_2 \parallel s_2 \xrightarrow{m} p_4 \parallel s_4'$. Hence, $p_2' \parallel s_2' \xrightarrow{\nu.m} p_4 \parallel s_4'$. Finally, using this transition together with the transition $p_2' \parallel s_2' \xrightarrow{\mu} p_1 \parallel s_1$ in the construction of $\mathcal{B}$ we conclude that $(p_1 \parallel s_1, \nabla_1(p_4 \,|[\mu, \nu.m]|\, s_1)) \in \mathcal{B}$.

3. Let $\nabla_1(p_2 \,|[\mu, \nu]|\, s_1) \xrightarrow{\alpha} \nabla_1(p_4 \,|[\mu', \nu']|\, s_4)$, $(p_1 \parallel s_1, \nabla_1(p_2 \,|[\mu, \nu]|\, s_1)) \in \mathcal{B}$, and $\alpha \neq \tau$. From the construction of $\mathcal{B}$ we have

$$\exists p_2', s_2', p_2, s_2.\left[p_2' \parallel s_2' \xrightarrow{\mu} p_1 \parallel s_1 \wedge p_2' \parallel s_2' \xrightarrow{\nu} p_2 \parallel s_2\right].$$

Since, $E_p = \emptyset$, $E_s = \emptyset$, and the processes p, s are concrete we know that either, $\alpha \in M_p$, or $\alpha \in M_s$.

(a) Let $\alpha \in M_p$. Then, due to the abstraction scheme $\mathbf{A}_1$ we know that from the above transition we have $s_1 \xrightarrow{?m} s_4$, $p_2 = p_4$, $\mu' = \mu$, and $\nu = m.\nu'$. So the transition $p_2' \parallel s_2' \xrightarrow{\nu} p_2 \parallel s_2$ can be written in the following way: $p_2' \parallel s_2' \xrightarrow{m} p_3' \parallel s_3' \xrightarrow{\nu'} p_2 \parallel s_2$, for some $p_3', s_3' \in \mathbb{P}$. Now, applying Corollary 3.10 at the state $p_2' \parallel s_2'$ we get $\exists p_4', s_4'.\left[p_1 \parallel s_1 \xrightarrow{m} p_4' \parallel s_4' \wedge p_3' \parallel s_3' \xrightarrow{\mu} p_4' \parallel s_4'\right]$.

Thus, $p_1 \xrightarrow{!m} p_4'$ and $s_1 \xrightarrow{?m} s_4'$. But, $s_1 \xrightarrow{?m} s_4$. Since, the supervisor s is deterministic we get $s_4' = s_4$. Thus, $p_1 \parallel s_1 \xrightarrow{m} p_4' \parallel s_4$.

Using the transitions $p_3' \parallel s_3' \xrightarrow{\nu'} p_2 \parallel s_2$ and $p_3' \parallel s_3' \xrightarrow{\mu} p_4' \parallel s_4$ we get $(p_4' \parallel s_4, \nabla_1(p_2 \,|[\mu, \nu']|\, s_4)) \in \mathcal{B}$.

(b) Let $\alpha \in M_s$. Then, $s_1 \xrightarrow{!n} s_4$, $\mu' = \mu.n$, $p_2 = p_4$, and $\nu = \nu'$. Since, the given processes p, s are well-posed, so assume a well-posedness relation $\mathcal{W}$ such that $(p, s) \in \mathcal{W}$. From Proposition 3.6 we have $(p_1, s_1) \in \mathcal{W}$. And by Definition 3.5 we get $s_1 \xrightarrow{!n} s_4 \wedge (p_1, s_1) \in \mathcal{W} \Rightarrow \exists p_4'.\left[p_1 \xrightarrow{?n} p_4'\right]$. Thus, $p_1 \parallel s_1 \xrightarrow{n} p_4' \parallel s_4$. Combining this transition with $p_2' \parallel s_2' \xrightarrow{\mu} p_1 \parallel s_1$ we get $p_2' \parallel s_2' \xrightarrow{\mu.n} p_4' \parallel s_4$.

Finally, using this transition together with $p_2' \parallel s_2' \xrightarrow{\nu\!\!\rightarrow} p_2 \parallel s_2$ we conclude that $(p_4' \parallel s_4, \nabla_1(p_2 \,|[\mu.n, \nu]|\, s_4)) \in \mathcal{B}$. □

Theorem A.2 (Proof of Theorem 3.15). *Let $p \parallel s$ be a synthesised synchronous system such that p, s are well-posed. If $p \parallel s$ satisfies the $(M_p, M_p \cup M_s)$ diamond property and the reordering property then*

$$p \parallel s \leftrightarrows_{\mathbf{b}} \nabla_1(p \,|\{\varepsilon, \varepsilon\}|\, s).$$

Proof. Define a relation $\mathcal{B}$ in the following way:

$$\begin{aligned}
\mathcal{B} = \Big\{ (p_1 \parallel s_1, \nabla_1(p_2 \,|\{\xi, \zeta\}|\, s_1)) \;\Big|\; & p_1 \parallel s_1 \in \mathfrak{R}(p \parallel s) \wedge \\
& \nabla_1(p_2 \,|\{\xi, \zeta\}|\, s_1) \in \mathfrak{R}(\nabla_1(p \,|\{\varepsilon, \varepsilon\}|\, s)) \wedge \\
& \exists p_2', s_2', p_2, s_2, \mu, \nu. \Big[\mu \in \mathbf{S}(\xi) \wedge p_2' \parallel s_2' \xrightarrow{\mu\!\!\rightarrow} p_1 \parallel s_1 \wedge \\
& \qquad \nu \in \mathbf{S}(\zeta) \wedge p_2' \parallel s_2' \xrightarrow{\nu\!\!\rightarrow} p_2 \parallel s_2 \Big] \bigvee \qquad \text{(C1)} \\
& \exists p_2', s_2', p_2, s_2, \mu, \nu. \Big[\mu \in \mathbf{S}(\xi) \wedge p_2 \parallel s_2 \xrightarrow{\mu\!\!\rightarrow} p_2' \parallel s_2' \wedge \\
& \qquad \nu \in \mathbf{S}(\zeta) \wedge p_1 \parallel s_1 \xrightarrow{\nu\!\!\rightarrow} p_2' \parallel s_2' \Big] \Big\}. \qquad \text{(C2)}
\end{aligned}$$

We show that the relation $\mathcal{B}$ is a branching bisimulation relation.

1. Let $p_1 \parallel s_1 \xrightarrow{\alpha} p_4 \parallel s_4$ and $(p_1 \parallel s_1, \nabla_1(p_2 \,|\{\xi, \zeta\}|\, s_1)) \in \mathcal{B}$ due to Condition C1. From the construction of $\mathcal{B}$ we have

$$\begin{aligned}
\exists p_2', s_2', p_2, s_2, \mu, \nu. \Big[& \mu \in \mathbf{S}(\xi) \wedge p_2' \parallel s_2' \xrightarrow{\mu\!\!\rightarrow} p_1 \parallel s_1 \wedge \\
& \nu \in \mathbf{S}(\zeta) \wedge p_2' \parallel s_2' \xrightarrow{\nu\!\!\rightarrow} p_2 \parallel s_2 \Big].
\end{aligned}$$

   Since, $E_p = \emptyset$, $E_s = \emptyset$, and the processes p, s are concrete we know that either, $\alpha \in M_p$, or $\alpha \in M_s$.

   (a) Let $\alpha \in M_p$. Then, from disjointness of the sets M_p, M_s we have

$$\exists m. \left[\alpha = m \wedge p_1 \xrightarrow{!m} p_4 \wedge s_1 \xrightarrow{?m} s_4 \right].$$

      i. Either, $\zeta = \varepsilon$. Similar to Case 1(a)i of Theorem 3.11.

      ii. Or, $\zeta \neq \varepsilon$. Note that the state $\nabla_1(p_1 \,|\{\xi, \zeta\}|\, s_1)$ can perform the action m, if $m \in \zeta$. Otherwise, the plant process p_1 will have to update its output's bag content by adding m. Thus, we have two possibilities.

         A. Either, $m \in \zeta$. Then, using the transition $s_1 \xrightarrow{?m} s_4$ we get $\nabla_1(p_2 \,|\{\xi, \zeta\}|\, s_1) \xrightarrow{m} \nabla_1(p_2 \,|\{\xi, \zeta \ominus m\}|\, s_4)$. Applying,

Corollary 3.10 at state $p_2' \parallel s_2'$ we get (see Figure A.2)

$$\exists p_3, s_3. \left[p_2 \parallel s_2 \xrightarrow{\mu} p_3 \parallel s_3 \wedge p_1 \parallel s_1 \xrightarrow{\nu} p_3 \parallel s_3 \right].$$

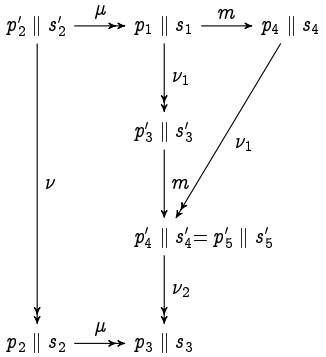


FIGURE A.2: Transitions derived in Case 1(a)iiA.

Since, $m \in \zeta$ we have $m \in \nu$. Thus, the transition can $p_1 \parallel s_1 \xrightarrow{\nu} p_3 \parallel s_3$ can be rewritten as (see Figure A.2):

$$p_1 \parallel s_1 \xrightarrow{\nu_1} p_3' \parallel s_3' \xrightarrow{m} p_4' \parallel s_4' \xrightarrow{\nu_2} p_3 \parallel s_3,$$

for some $p_3', s_3', p_4', s_4', \nu_1, \nu_2$ such that $\nu = \nu_1.m.\nu_2$ and $m \notin \nu_1$. Applying Corollary 3.10 at $p_1 \parallel s_1$ we get (see Figure A.2) $\exists p_5', s_5'. \left[p_4 \parallel s_4 \xrightarrow{\nu_1} p_5' \parallel s_5' \wedge p_3' \parallel s_3' \xrightarrow{m} p_5' \parallel s_5' \right]$. Since the synchronous system is deterministic we have $p_4' \parallel s_4' = p_5' \parallel s_5'$. Thus, we have $p_4 \parallel s_4 \xrightarrow{\nu_1.\nu_2} p_3 \parallel s_3$. Note that $\nu_1.m.\nu_2 \in \mathbf{S}(\zeta) \Rightarrow \nu_1.\nu_2 \in \mathbf{S}(\zeta \ominus m)$. Using the above transition with the transition $p_2 \parallel s_2 \xrightarrow{\mu} p_3 \parallel s_3$ in Condition C2 we get $(p_4 \parallel s_4, \nabla_1(p_2 |\{\xi, \zeta \ominus m\}| s_4)) \in \mathcal{B}$.

B. Or, $m \notin \zeta$. Applying, Corollary 3.10 at the states $p_2' \parallel s_2'$, $p_1 \parallel s_1$ we get

$$\exists p_3, s_3. \left[p_2 \parallel s_2 \xrightarrow{\mu} p_3 \parallel s_3 \wedge p_1 \parallel s_1 \xrightarrow{\nu} p_3 \parallel s_3 \right].$$
$$\exists p_5, s_5. \left[p_3 \parallel s_3 \xrightarrow{m} p_5 \parallel s_5 \wedge p_4 \parallel s_4 \xrightarrow{\nu} p_5 \parallel s_5 \right].$$

Applying Proposition 2.11 on the transition $p_2 \parallel s_2 \xrightarrow{\mu} p_3 \parallel s_3$ we get $p_2 \xrightarrow{?\mu} p_3$. Thus, $\nabla_1(p_2 |\{\xi, \zeta\}| s_1) \longrightarrow \nabla_1(p_3 |\{\varepsilon, \zeta\}| s_1)$. Using the transition $p_1 \parallel s_1 \xrightarrow{\nu} p_3 \parallel$

s_3, $\nu \in \mathbf{S}(\zeta)$ in Condition C1 we get

$$(p_1 \parallel s_1, \nabla_1(p_3 \,|\{\varepsilon, \zeta\}|\, s_1)) \in \mathcal{B}.$$

Also, from the transition $p_3 \parallel s_3 \xrightarrow{m} p_5 \parallel s_5$ we have $p_3 \xrightarrow{!m} p_5$. Using this transition we have

$$\nabla_1(p_3 \,|\{\varepsilon, \zeta\}|\, s_1) \xrightarrow{\tau} \nabla_1(p_5 \,|\{\varepsilon, \zeta \oplus m\}|\, s_1).$$

Using the transition $p_1 \parallel s_1 \xrightarrow{\nu.m} p_5 \parallel s_5$, $\nu.m \in \mathbf{S}(\zeta \oplus m)$ in Condition C1 we get $(p_1 \parallel s_1, \nabla_1(p_5 \,|\{\varepsilon, \zeta \oplus m\}|\, s_1)) \in \mathcal{B}$. Furthermore, from the transition $s_1 \xrightarrow{?m} s_4$ we get $\nabla_1(p_5 \,|\{\varepsilon, \zeta \oplus m\}|\, s_1) \xrightarrow{m} \nabla_1(p_5 \,|\{\varepsilon, \zeta\}|\, s_4)$. Finally, using the transition $p_4 \parallel s_4 \xrightarrow{\nu} p_5 \parallel s_5$ and the fact $\nu \in \mathbf{S}(\zeta)$ in C1 we conclude that $(p_4 \parallel s_4, \nabla_1(p_5 \,|\{\varepsilon, \zeta\}|\, s_4)) \in \mathcal{B}$.

(b) Let $\alpha \in M_s$. Similar to Case 1b of Theorem 3.11.

2. Let $p_1 \parallel s_1 \xrightarrow{\alpha} p_4 \parallel s_4$ and $(p_1 \parallel s_1, \nabla_1(p_2 \,|\{\xi, \zeta\}|\, s_1)) \in \mathcal{B}$ due to Condition C2. Similar to the previous case.

3. Let $\nabla_1(p_2 \,|\{\xi, \zeta\}|\, s_1) \xrightarrow{\tau} \nabla_1(p_4 \,|\{\xi', \zeta'\}|\, s_4)$, $(p_1 \parallel s_1, \nabla_1(p_2 \,|\{\xi, \zeta\}|\, s_1)) \in \mathcal{B}$ due to Condition C1. Then, from the construction of $\mathcal{B}$ we have

$$\exists p_2', s_2', p_2, s_2, \mu, \nu. \Big[\, \mu \in \mathbf{S}(\xi) \wedge p_2' \parallel s_2' \xrightarrow{\mu} p_1 \parallel s_1 \;\wedge\; \nu \in \mathbf{S}(\zeta) \wedge p_2' \parallel s_2' \xrightarrow{\nu} p_2 \parallel s_2 \,\Big].$$

Applying Corollary 3.10 at the state $p_2' \parallel s_2'$ we get

$$\exists p_3, s_3. \left[p_2 \parallel s_2 \xrightarrow{\mu} p_3 \parallel s_3 \wedge p_1 \parallel s_1 \xrightarrow{\nu} p_3 \parallel s_3 \right].$$

Since, the given processes p, s are concrete and the abstraction scheme $\mathbf{A}_1$ is used we know that this transition is either, due to the removal of an element from ξ, or due to the addition of an element in ζ.

(a) Removal of an element from ξ. Then, $\xi' = \xi \ominus n$, for some $n \in A$, $\zeta' = \zeta$, $p_2 \xrightarrow{?n} p_4$, $s_1 = s_4$. Then, $n \in \xi$. Thus, the transition $p_2 \parallel s_2 \xrightarrow{\mu} p_3 \parallel s_3$ can be rewritten as:

$$p_2 \parallel s_2 \xrightarrow{\mu_1} p_3' \parallel s_3' \xrightarrow{n} p_4' \parallel s_4' \xrightarrow{\mu_2} p_3 \parallel s_3,$$

for some $p_3', p_4', s_3', s_4', \mu_1, \mu_2$ such that $\mu = \mu_1.n.\mu_2$. And from reordering property we get

$$\exists s_5', \mu_1'. \left[p_2 \parallel s_2 \xrightarrow{n} p_4 \parallel s_5' \xrightarrow{\mu_1'} p_4' \parallel s_4' \wedge \mu_1' =_\pi \mu_1 \right].$$

Thus, $p_4 \parallel s_5' \xrightarrow{\mu_1'.\mu_2} p_3 \parallel s_3$. Using Proposition 2.4 we have $\mu_1.n.\mu_2 \in \mathbf{S}(\xi) \Rightarrow n.\mu_1'.\mu_2 \in \mathbf{S}(\xi) \Rightarrow \mu_1'.\mu_2 \in \mathbf{S}(\xi \ominus n)$. Using the transitions $p_4 \parallel s_5' \xrightarrow{\mu_1'.\mu_2} p_3 \parallel s_3$ and $p_1 \parallel s_1 \xrightarrow{\nu} p_3 \parallel s_3$ in Condition C2 we get $(p_1 \parallel s_1, \nabla_1(p_4 \,|\{\xi \ominus m, \zeta\}|\, s_1)) \in \mathcal{B}$.

(b) Addition of an element in ζ. Similar to Case 2b of Theorem 3.11.

4. Let $\nabla_1(p_2 \,|\{\xi, \zeta\}|\, s_1) \xrightarrow{\tau} \nabla_1(p_4 \,|\{\xi', \zeta'\}|\, s_4)$, $(p_1 \parallel s_1, \nabla_1(p_2 \,|\{\xi, \zeta\}|\, s_1)) \in \mathcal{B}$ due to Condition C2. Similar to the previous case.

5. Let $\nabla_1(p_2 \,|\{\xi, \zeta\}|\, s_1) \xrightarrow{\alpha} \nabla_1(p_4 \,|\{\xi', \zeta'\}|\, s_4)$, $(p_1 \parallel s_1, \nabla_1(p_2 \,|\{\xi, \zeta\}|\, s_1)) \in \mathcal{B}$ due to Condition C1, $\alpha \neq \tau$. Then, from the construction of $\mathcal{B}$ we get

$$\exists p_2', s_2', p_2, s_2, \mu, \nu. \Big[\, \mu \in \mathbf{S}(\xi) \wedge p_2' \parallel s_2' \xrightarrow{\mu} p_1 \parallel s_1 \;\wedge\; \nu \in \mathbf{S}(\zeta) \wedge p_2' \parallel s_2' \xrightarrow{\nu} p_2 \parallel s_2 \,\Big].$$

Since, $E_p = \emptyset$, $E_s = \emptyset$, and the processes p, s are concrete we know that either, $\alpha \in M_p$, or $\alpha \in M_s$.

(a) Either, $\alpha \in M_p$. Then, due to the abstraction scheme $\mathbf{A}_1$ we know that from the above transition we have $s_1 \xrightarrow{?m} s_4$, $p_2 = p_4$, $\xi' = \xi$, and $\zeta = \zeta' \ominus m$, for some $m \in A$ such that $\alpha = m$. Applying Corollary 3.10 at state $p_2' \parallel s_2'$ we get

$$\exists p_3, s_3. \left[p_2 \parallel s_2 \xrightarrow{\mu} p_3 \parallel s_3 \wedge p_1 \parallel s_1 \xrightarrow{\nu} p_3 \parallel s_3 \right].$$

Clearly, we have $m \in \zeta$ and thus $m \in \nu$. Thus, the transition $p_1 \parallel s_1 \xrightarrow{\nu} p_3 \parallel s_3$ can be rewritten as: $p_1 \parallel s_1 \xrightarrow{\nu_1} p_3' \parallel s_3' \xrightarrow{m} p_4' \parallel s_4' \xrightarrow{\nu_2} p_3 \parallel s_3$, for some $p_3', p_4', s_3', s_4', \nu_1, \nu_2$ such that $\nu = \nu_1.m.\nu_2$. Applying reordering property at the state $p_1 \parallel s_1$ we get $\exists p_5, \nu_1'. \left[p_1 \parallel s_1 \xrightarrow{m} p_5 \parallel s_4 \xrightarrow{\nu_1'} p_4' \parallel s_4' \wedge \nu_1' =_\pi \nu_1 \right]$. Thus, $p_5 \parallel s_4 \xrightarrow{\nu_1'.\nu_2} p_3 \parallel s_3$. Using Proposition 2.4 we have $\nu_1.m.\nu_2 \in \mathbf{S}(\zeta) \Rightarrow m.\nu_1'.\nu_2 \in \mathbf{S}(\zeta) \Rightarrow \nu_1'.\nu_2 \in \mathbf{S}(\zeta \ominus m)$. Using the above derived transition with the transition $p_2 \parallel s_2 \xrightarrow{\mu} p_3 \parallel s_3$ in Condition C2 we conclude that $(p_5 \parallel s_4, \nabla_1(p_2 \,|\{\xi, \zeta \ominus m\}|\, s_4)) \in \mathcal{B}$.

(b) Or, $\alpha \in M_s$. Similar to Case 3b of Theorem 3.11. □

Appendix B

Proofs of main theorems in Chapter 4

Definition B.1. A symmetric binary relation $\mathcal{S} \subseteq \mathbb{P} \times \mathbb{P}$ is called a *strong bisimulation* (simply, bisimulation) relation [66] iff the following transfer condition is satisfied.

$$\forall q_1, q_1', q_2, \alpha.\ \Big[q_1 \xrightarrow{\alpha} q_1' \wedge (q_1, q_2) \in \mathcal{S} \Rightarrow \exists q_2'.\ \Big[q_2 \xrightarrow{\alpha} q_2' \wedge (q_1', q_2') \in \mathcal{S}\Big]\Big].$$

Furthermore, a relation $\mathcal{S} \subseteq \mathbb{P} \times \mathbb{P}$ is an $\sqcup$-*sensitive bisimulation* (pronounced, emptiness sensitive bisimulation) relation iff $\mathcal{S}$ is a bisimulation relation satisfying the following transfer condition:

$$\forall q_1, q_2.\big[(q_1, q_2) \in \mathcal{S} \wedge q_1 \sqcup \Rightarrow q_2 \sqcup\big].$$

Two processes q_1, q_2 are ($\sqcup$-*sensitive*) *bisimilar*, denoted $(q_1 \leftrightarrows^{\sqcup} q_2)\ q_1 \leftrightarrows q_2$, if there exists a (an $\sqcup$-sensitive) bisimulation relation $\mathcal{S}$ such that $(q_1, q_2) \in \mathcal{S}$.

The following proposition states that for the class of concrete processes both branching bisimilarity and contra-similiarity coincides with bisimilarity.

Proposition B.2. *Suppose q_1, q_2 are any two concrete processes. Then,*

$$q_1 \leftrightarrows^{\sqcup} q_2 \Leftrightarrow q_1 \simeq^{\sqcup} q_2.$$

Proof. For branching bisimulation, we know the result $q_1 \leftrightarrows^{\sqcup} q_2 \Leftrightarrow q_1 \leftrightarrows^{\sqcup}_{\mathbf{b}} q_2$ holds, whenever q_1, q_2 are concrete processes ([77]). So we show the result for contra-similarity. It is easy to see that $q_1 \leftrightarrows^{\sqcup} q_2 \Rightarrow q_1 \sim^{\sqcup}_{\mathbf{c}} q_2$ because $\leftrightarrows^{\sqcup}$ is the finest equivalence in van Glabbeek's spectrum [76]. So we prove the other

direction, i.e., if $q_1 \sim_{\mathbf{c}}^{\sqcup} q_2$, then $q_1 \leftrightarrows^{\sqcup} q_2$. Define a binary relation $\mathcal{S}$:

$$\mathcal{S} = \{(q_3, q_4), (q_4, q_3) \mid q_3 \sim_{\mathbf{c}}^{\sqcup} q_4 \wedge q_3 \in \mathfrak{R}(q_1) \wedge q_4 \in \mathfrak{R}(q_2)\}.$$

Next, we need to show that $\mathcal{S}$ is a bisimulation relation.

1. Let $q_3 \xrightarrow{\alpha} q_5$, $\alpha \neq \tau$ ($\because q_1$ is concrete), and $(q_3, q_4) \in \mathcal{S}$. Then, by the construction of $\mathcal{S}$ we have $q_3 \sim_{\mathbf{c}}^{\sqcup} q_4$. Now, using the transfer condition of Definition 2.9 we get $\exists q_6. \left[q_4 \xrightarrow{\alpha} \!\!\!\!\rightarrow\, q_6 \wedge q_6 \preccurlyeq^{\sqcup} q_5\right]$. Since q_1 is a concrete processes, so is q_5. Thus, $q_5 \not\xrightarrow{\tau}$. And from Proposition 2.10 we get $q_5 \preccurlyeq^{\sqcup} q_6$. Thus, $q_5 \sim_{\mathbf{c}}^{\sqcup} q_6$. Also, q_2 is a concrete process, so $q_4 \xrightarrow{\alpha} \!\!\!\!\rightarrow\, q_6$ implies $q_4 \xrightarrow{\alpha} q_6$. Finally, from the construction of $\mathcal{S}$ we conclude that $(q_5, q_6) \in \mathcal{S}$ and $(q_6, q_5) \in \mathcal{S}$.

2. Let $q_3\!\sqcup$ and $(q_3, q_4) \in \mathcal{S}$. Then, by the construction of $\mathcal{S}$ we have $q_3 \sim_{\mathbf{c}}^{\sqcup} q_4$. Now, using the transfer condition of Definition 2.9 we get

   $$\exists q_6. \left[q_4 \longrightarrow\!\!\!\!\rightarrow\, q_6 \wedge q_6 \sqcup \wedge\, q_6 \preccurlyeq^{\sqcup} q_3\right].$$

   Since q_1 is a concrete processes, then q_3 is also a concrete process. Thus, $q_3 \not\xrightarrow{\tau}$. And from Proposition 2.10 we get $q_3 \preccurlyeq^{\sqcup} q_6$. Thus, $q_3 \sim_{\mathbf{c}}^{\sqcup} q_6$. Also, q_2 is a concrete process, so $q_4 \longrightarrow\!\!\!\!\rightarrow\, q_6$ implies $q_4 = q_6$. Hence, $q_4\!\sqcup$.

3. Let $q_2 \xrightarrow{\alpha} q_2'$, $\alpha \neq \tau$ and $(q_1, q_2) \in \mathcal{S}$. Similar to Case 1.

4. Let $q_2\!\sqcup$ and $(q_1, q_2) \in \mathcal{S}$. Similar to Case 2.

Hence, the desired result $q_1 \leftrightarrows^{\sqcup} q_2 \Leftrightarrow q_1 \simeq^{\sqcup} q_2$ follows. □

Lemma B.3 (Proof of Lemma 4.3). *Let p_1, s_1, p_2, s_2 be any four concrete processes. Then, $p_1 \parallel s_1 \simeq^{\sqcup} \nabla_3(p_2 \,|[\epsilon, \epsilon]|\, s_2) \Rightarrow p_1 \parallel s_1 \simeq^{\sqcup} p_2 \parallel s_2$.*

Proof. It is easier to show $p_1 \parallel s_1 \simeq^{\sqcup} \nabla_3(p_2 \,|[\epsilon, \epsilon]|\, s_2) \Rightarrow p_1 \parallel s_1 \leftrightarrows^{\sqcup} p_2 \parallel s_2$ than $p_1 \parallel s_1 \simeq^{\sqcup} \nabla_3(p_2 \,|[\epsilon, \epsilon]|\, s_2) \Rightarrow p_1 \parallel s_1 \simeq^{\sqcup} p_2 \parallel s_2$. Note that the desired result follows directly from Proposition B.2. To see the former, define the following relations $\mathcal{S}_{\simeq^{\sqcup}}$ (for $\simeq^{\sqcup} \in \{\leftrightarrows_{\mathbf{b}}^{\sqcup}, \sim_{\mathbf{c}}^{\sqcup}\}$):

$$\mathcal{S}_{\simeq^{\sqcup}} = \Big\{(p_3 \parallel s_3, p_4 \parallel s_4), (p_4 \parallel s_4, p_3 \parallel s_3) \Big| \; p_3 \parallel s_3 \in \mathfrak{R}(p_1 \parallel s_1) \wedge$$
$$\nabla_3(p_4 \,|[\epsilon, \epsilon]|\, s_4) \in \mathfrak{R}(\nabla_3(p_2 \,|[\epsilon, \epsilon]|\, s_2)) \wedge p_3 \parallel s_3 \simeq^{\sqcup} \nabla_3(p_4 \,|[\epsilon, \epsilon]|\, s_4)\Big\}.$$

Next, we need to show that $\mathcal{S}_{\simeq^{\sqcup}}$ is a bisimulation relation.

1. When $p_3 \parallel s_3 \xrightarrow{\alpha} p_5 \parallel s_5$ and $(p_3 \parallel s_3, p_4 \parallel s_4) \in \mathcal{S}_{\simeq^{\sqcup}}$. Now, perform case-distinction based on the type of α, we get the following cases:

(a) Let $\alpha = e$, for some $e \in E_s$. Then $p_3 = p_5$ and $s_3 \xrightarrow{e} s_5$. And, by the construction of $\mathcal{S}_{\simeq^{\sqcup}}$ we have $p_3 \parallel s_3 \simeq^{\sqcup} \nabla_3(p_4 \,|[\epsilon,\epsilon]|\, s_4)$.

- Let $\simeq^{\sqcup} = \underline{\leftrightarrow}^{\sqcup}_{\mathbf{b}}$. Then, from the transfer conditions of branching bisimulation we get $\exists q.[\nabla_3(p_4 \,|[\epsilon,\epsilon]|\, s_4) \longrightarrow\!\!\!\!\!\rightarrow\ q \xrightarrow{e} q' \wedge q \,\underline{\leftrightarrow}^{\sqcup}_{\mathbf{b}}\, p_3 \parallel s_3 \wedge p_3 \parallel s_5 \,\underline{\leftrightarrow}^{\sqcup}_{\mathbf{b}}\, q']$. Since concrete processes are used to construct the asynchronous systems we have $q = \nabla_3(p_4 \,|[\epsilon,\epsilon]|\, s_4)$. Since the sets E_p, E_s are disjoint, so from the semantics we have $q' = \nabla_3(p_4 \,|[\epsilon,\epsilon]|\, s_6)$ such that $s_4 \xrightarrow{e} s_6$, for some $s_6 \in \mathbb{P}$. Using the fact $p_3 \parallel s_5 \,\underline{\leftrightarrow}^{\sqcup}_{\mathbf{b}}\, \nabla_3(p_4 \,|[\epsilon,\epsilon]|\, s_6)$ in the construction of $\mathcal{S}_{\underline{\leftrightarrow}^{\sqcup}_{\mathbf{b}}}$ we get $(p_3 \parallel s_5, p_4 \parallel s_6) \in \mathcal{S}_{\underline{\leftrightarrow}^{\sqcup}_{\mathbf{b}}}$.
- Let $\simeq^{\sqcup} = \sim^{\sqcup}_{\mathbf{c}}$. Then, from the transfer conditions of contrasimulation we get $\exists q.[\nabla_3(p_4 \,|[\epsilon,\epsilon]|\, s_4) \longrightarrow\!\!\!\!\!\rightarrow\ q \wedge q \preccurlyeq^{\sqcup} p_3 \parallel s_5]$. Since concrete processes are used to construct the asynchronous systems and the external actions do not update the contents of queues we have $q = \nabla_3(p_6 \,|[\epsilon,\epsilon]|\, s_6)$ such that

$$\nabla_3(p_4 \,|[\epsilon,\epsilon]|\, s_4) \xrightarrow{e} \nabla_3(p_6 \,|[\epsilon,\epsilon]|\, s_6) \text{ for some } p_6, s_6 \in \mathbb{P}.$$

  Since the sets E_p, E_s are disjoint, so from the semantics we have $p_4 = p_6$ and $s_4 \xrightarrow{e} s_6$.
  Also, p_1, s_1 are concrete processes, so is $p_1 \parallel s_1$. Thus, $p_3 \parallel s_5$ is also concrete and $p_3 \parallel s_5 \xrightarrow{\tau}\!\!\!\!\!/\ $. From Proposition 2.10 we get $p_3 \parallel s_5 \preccurlyeq^{\sqcup} \nabla_3(p_4 \,|[\epsilon,\epsilon]|\, s_6)$. Thus $p_3 \parallel s_5 \sim^{\sqcup}_{\mathbf{c}} \nabla_3(p_4 \,|[\epsilon,\epsilon]|\, s_6)$. Clearly, $p_4 \parallel s_4 \xrightarrow{e} p_4 \parallel s_6$. Using $p_4 \parallel s_4 \sim^{\sqcup}_{\mathbf{c}} \nabla_3(p_4 \,|[\epsilon,\epsilon]|\, s_6)$ in the construction of $\mathcal{S}_{\sim^{\sqcup}_{\mathbf{c}}}$ we get $(p_3 \parallel s_5, p_4 \parallel s_6) \in \mathcal{S}_{\sim^{\sqcup}_{\mathbf{c}}}$.

(b) Let $\alpha = e$, for some $e \in E_p$. Similar to Case (a).

(c) Let $\alpha = n$, for some $n \in M_s$. Then, $p_3 \xrightarrow{?n} p_5$ and $s_3 \xrightarrow{!n} s_5$. And by the construction of $\mathcal{S}^{\sqcup}_{\simeq}$ we have $p_3 \parallel s_3 \simeq^{\sqcup} \nabla_3(p_4 \,|[\epsilon,\epsilon]|\, s_4)$.

- When $\simeq^{\sqcup} = \underline{\leftrightarrow}^{\sqcup}_{\mathbf{b}}$. Using the transfer conditions of branching bisimulation we get there exists q, q' such that

$$\nabla_3(p_4 \,|[\epsilon,\epsilon]|\, s_4) \longrightarrow\!\!\!\!\!\rightarrow\ q \xrightarrow{n} q' \wedge q \,\underline{\leftrightarrow}^{\sqcup}_{\mathbf{b}}\, p_3 \parallel s_3 \wedge q' \,\underline{\leftrightarrow}^{\sqcup}_{\mathbf{b}}\, p_5 \parallel s_5.$$

  Due to concreteness assumption, we have $q = \nabla_3(p_4 \,|[\epsilon,\epsilon]|\, s_4)$. Since the sets M_p, M_s are disjoint, so the sets M_p, M_s are also disjoint. Using this fact in the semantics we know that $q' = \nabla_3(p_4 \,|[n,\epsilon]|\, s_6)$ such that $s_4 \xrightarrow{!n} s_6$, for some $s_6 \in \mathbb{P}$. But, $\nabla_3(p_4 \,|[n,\epsilon]|\, s_6)\!\downarrow$ and from the transfer conditions of the predicate $\sqcup$ we get

$$\exists q''.\ [\nabla_3(p_4 \,|[n,\epsilon]|\, s_6) \longrightarrow\!\!\!\!\!\rightarrow\ q'' \wedge q'' \sqcup \wedge\ q'' \,\underline{\leftrightarrow}^{\sqcup}_{\mathbf{b}}\, p_5 \parallel s_5]\ .$$

  Due to concreteness assumption, we have $q'' = \nabla_3(p_6 \,|[\epsilon,\epsilon]|\, s_6)$ such that $\nabla_3(p_4 \,|[n,\epsilon]|\, s_6) \xrightarrow{\tau} \nabla_3(p_6 \,|[\epsilon,\epsilon]|\, s_6)$, for some $p_6 \in$

$\mathbb{P}$. Thus, we have the transitions $s_4 \xrightarrow{!n} s_6$ and $p_4 \xrightarrow{?n} p_6$. Clearly, $p_4 \parallel s_4 \xrightarrow{n} p_6 \parallel s_6$. Finally, using the fact $p_5 \parallel s_5 \underline{\leftrightarrow}^{\sqcup}_{\mathbf{b}} \nabla_3(p_6 \,|[\epsilon,\epsilon]|\, s_6)$ we conclude that $(p_5 \parallel s_5, p_6 \parallel s_6) \in \mathcal{S}_{\underline{\leftrightarrow}^{\sqcup}_{\mathbf{b}}}$.

- Let $\simeq^{\sqcup} = \sim^{\sqcup}_{\mathbf{c}}$. Using the transfer conditions of contra-simulation we get $\exists q.\left[\nabla_3(p_4 \,|[\epsilon,\epsilon]|\, s_4) \xrightarrow{n}\!\!\!\!\rightarrow q \wedge q \preccurlyeq^{\sqcup} p_5 \parallel s_5\right]$. Again, due to concreteness assumption we have $p_5 \parallel s_5 \not\xrightarrow{} \tau$. And, from Proposition 2.10 we have $p_5 \parallel s_5 \preccurlyeq^{\sqcup} q$. Thus, $p_5 \parallel s_5 \sim^{\sqcup}_{\mathbf{c}} q$.
  Now consider the transition $\nabla_3(p_4 \,|[\epsilon,\epsilon]|\, s_4) \xrightarrow{n}\!\!\!\!\rightarrow q$. Note that there are two situations possible: either, n is consumed, or n is not consumed.
  - Suppose n is consumed. Again, due to concreteness assumption and the fact that the sets M_p, M_s are disjoint and $n \in M_s$ we have $q = \nabla_3(p_6 \,|[\epsilon,\epsilon]|\, s_6)$ such that
    $$\nabla_3(p_4 \,|[\epsilon,\epsilon]|\, s_4) \xrightarrow{n} \nabla_3(p_4 \,|[n,\epsilon]|\, s_6) \xrightarrow{\tau} \nabla_3(p_6 \,|[\epsilon,\epsilon]|\, s_6),$$
    for some $p_6, s_6 \in \mathbb{P}$. Thus, $s_4 \xrightarrow{!n} s_6$ and $p_4 \xrightarrow{?n} p_6$. Clearly, $p_4 \parallel s_4 \xrightarrow{n} p_6 \parallel s_6$. And from the construction of $\mathcal{S}_{\sim^{\sqcup}_{\mathbf{c}}}$ we get $(p_5 \parallel s_5, p_6 \parallel s_6) \in \mathcal{S}_{\sim^{\sqcup}_{\mathbf{c}}}$.
  - Suppose n is not consumed. Since, concrete processes are used to construct the asynchronous systems, $n \in M_s$, and the sets M_p, M_s are disjoint we have $q = \nabla_3(p_4 \,|[n,\epsilon]|\, s_6)$ such that
    $$\nabla_3(p_4 \,|[\epsilon,\epsilon]|\, s_4) \xrightarrow{n} \nabla_3(p_4 \,|[n,\epsilon]|\, s_6),$$
    for some $s_6 \in \mathbb{P}$. But $\nabla_3(p_4 \,|[n,\epsilon]|\, s_6)\!\downarrow$ and from above we have $p_5 \parallel s_5 \sim^{\sqcup}_{\mathbf{c}} \nabla_3(p_4 \,|[n,\epsilon]|\, s_6)$ ($\because p_5 \parallel s_5 \sim^{\sqcup}_{\mathbf{c}} q$). So using the transfer conditions of the predicate $\sqcup$ we get
    $$\exists q'.\,[\nabla_3(p_4 \,|[n,\epsilon]|\, s_6) \longrightarrow\!\!\!\!\rightarrow q' \wedge q'\sqcup \wedge\, q' \preccurlyeq^{\sqcup} p_5 \parallel s_5].$$
    Due to concreteness assumption we have $q' = \nabla_3(p_6 \,|[\epsilon,\epsilon]|\, s_6)$ such that $\nabla_3(p_4 \,|[n,\epsilon]|\, s_6) \xrightarrow{\tau} \nabla_3(p_6 \,|[\epsilon,\epsilon]|\, s_6)$, for some $p_6 \in \mathbb{P}$. Also, from above we have $p_5 \parallel s_5 \not\xrightarrow{} \tau$; thus, from Proposition 2.10 we get $p_5 \parallel s_5 \preccurlyeq^{\sqcup} \nabla_3(p_6 \,|[\epsilon,\epsilon]|\, s_6)$. Thus, $p_5 \parallel s_5 \sim^{\sqcup}_{\mathbf{c}} \nabla_3(p_6 \,|[\epsilon,\epsilon]|\, s_6)$. Note that we derived the transition $p_4 \xrightarrow{?n} p_6$ and $s_4 \xrightarrow{!n} s_6$. Thus, $p_4 \parallel s_4 \xrightarrow{n} p_6 \parallel s_6$. Finally, from the construction of $\mathcal{S}_{\sim^{\sqcup}_{\mathbf{c}}}$ we conclude that $(p_5 \parallel s_5, p_6 \parallel s_6) \in \mathcal{S}_{\sim^{\sqcup}_{\mathbf{c}}}$.

(d) Let $\alpha = m$, for some $m \in M_p$. Similar to Case 1c.

2. Let $p_3 \parallel s_3\sqcup$ and $(p_3 \parallel s_3, p_4 \parallel s_4) \in \mathcal{S}_{\simeq^{\sqcup}}$. Trivial.

3. Let $p_4 \parallel s_4 \xrightarrow{\alpha} p_6 \parallel s_6$ and $(p_3 \parallel s_3, p_4 \parallel s_4) \in \mathcal{S}_{\simeq^{\sqcup}}$. Similar to Case 1.

4. Let $p_4 \parallel s_4 \sqcup$ and $(p_3 \parallel s_3, p_4 \parallel s_4) \in \mathcal{S}_{\simeq \sqcup}$. Trivial. □

Theorem B.4 (Proof of Theorem 4.15). *Let p, s be concrete processes. If the synchronous system $p \parallel s$ is well-posed, input-deterministic modulo $\underline{\leftrightarrow}^{\sqcup}_{\mathbf{b}}$, and strong E-independent modulo $\underline{\leftrightarrow}^{\sqcup}_{\mathbf{b}}$, then $p \parallel s \,\underline{\leftrightarrow}^{\sqcup}_{\mathbf{b}}\, \nabla_3(p \,|[\epsilon,\epsilon]|_{\mathrm{h}}\, s)$.*

Proof. Define a relation $\mathcal{B}$ in the following way:

$$
\begin{aligned}
\mathcal{B} = \Big\{ & (p_1 \parallel s_1, \nabla_3(p_2 \,|[\mu,\nu]|_{\mathrm{h}}\, s_2)) \mid p_1 \parallel s_1 \in \mathfrak{R}(p \parallel s) \wedge \\
& \nabla_3(p_2 \,|[\mu,\nu]|_{\mathrm{h}}\, s_2) \in \mathfrak{R}(\nabla_3(p \,|[\epsilon,\epsilon]|_{\mathrm{h}}\, s)) \wedge \\
& ((\mu = \epsilon \wedge \nu = \epsilon) \Rightarrow p_1 \parallel s_1 \,\underline{\leftrightarrow}^{\sqcup}\, p_2 \parallel s_2) \;\bigvee \\
& \Big((\mu \neq \epsilon \wedge \nu = \epsilon) \Rightarrow \exists p_2', s_2', u \in (M_s \cup E_s)^*. \Big[p_2 \parallel s_2' \in \mathfrak{R}(p \parallel s) \wedge \\
& \qquad p_2 \parallel s_2' \xrightarrow{u} \!\!\!\!\rightarrow\; p_2' \parallel s_2 \wedge \mu = \bar{u} \wedge p_1 \parallel s_1 \,\underline{\leftrightarrow}^{\sqcup}\, p_2' \parallel s_2\Big]\Big) \;\bigvee \\
& \Big((\mu = \epsilon \wedge \nu \neq \epsilon) \Rightarrow \exists p_2', s_2', v \in (M_p \cup E_p)^*. \Big[p_2' \parallel s_2 \in \mathfrak{R}(p \parallel s) \wedge \\
& \qquad p_2' \parallel s_2 \xrightarrow{v} \!\!\!\!\rightarrow\; p_2 \parallel s_2' \wedge \nu = \bar{v} \wedge p_1 \parallel s_1 \,\underline{\leftrightarrow}^{\sqcup}\, p_2 \parallel s_2'\Big]\Big)\Big\}.
\end{aligned}
$$

Note that in the above definition there is no implication with the antecedent $\mu \neq \epsilon, \nu \neq \epsilon$ because of half-duplex mechanism. Next, we need to show that $\mathcal{B}$ is a branching bisimulation relation.

1. Let $p_1 \parallel s_1 \xrightarrow{\alpha} p_3 \parallel s_3$ and $(p_1 \parallel s_1, \nabla_3(p_2 \,|[\mu,\nu]|_{\mathrm{h}}\, s_2)) \in \mathcal{B}$. Based on the type of α we get the following cases:
   (a) When $\alpha = e$, for some $e \in E_s$. Then, $p_1 = p_3$ and $s_1 \xrightarrow{e} s_3$. Now, applying structural induction on μ, ν we get the following cases:
      i. Let $\mu = \epsilon, \nu = \epsilon$. Similar to the next case.
      ii. When $\mu \neq \epsilon, \nu = \epsilon$. Then, we have

$$
\exists p_2', s_2', u \in (M_s \cup E_s)^*. \Big[p_2 \parallel s_2' \xrightarrow{u} \!\!\!\!\rightarrow\; p_2' \parallel s_2 \wedge \mu = \bar{u} \wedge p_1 \parallel s_1 \,\underline{\leftrightarrow}^{\sqcup}\, p_2' \parallel s_2\Big].
$$

Consider the transition $p_1 \parallel s_1 \xrightarrow{e} p_1 \parallel s_3$ and applying the transfer conditions of strong bisimulation we get

$$
\exists p_4', s_4. \left[p_2' \parallel s_2 \xrightarrow{e} p_4' \parallel s_4 \wedge p_1 \parallel s_3 \,\underline{\leftrightarrow}^{\sqcup}\, p_4' \parallel s_4\right].
$$

Since, the sets E_p, E_s are disjoint, thus, from the semantics we have $p_4' = p_2'$ and $s_2 \xrightarrow{e} s_4$. Using this transition we get $\nabla_3(p_2 \,|[\mu,\epsilon]|_{\mathrm{h}}\, s_2) \xrightarrow{e} \nabla_3(p_2 \,|[\mu,\epsilon]|_{\mathrm{h}}\, s_4)$. Note that $\overline{(u.e)} = \mu$

because $\bar{u}.e = \bar{u}$ and $\bar{u} = \mu$. From the construction of $\mathcal{B}$, we get $(p_1 \parallel s_3, \nabla_3(p_2 \,|[\mu,\epsilon]|_{\mathrm{h}}\, s_4)) \in \mathcal{B}$.

iii. Let $\mu = \epsilon, \nu \neq \epsilon$. Then, we have

$$\exists p_2', s_2', v \in (M_p \cup E_p)^*. \Big[p_2' \parallel s_2 \xrightarrow{v}\!\!\!\!\rightarrow p_2 \parallel s_2' \wedge \nu = \bar{v} \wedge p_1 \parallel s_1 \leftrightarrows^{\sqcup} p_2 \parallel s_2'\Big].$$

Consider the transition $p_1 \parallel s_1 \xrightarrow{e} p_1 \parallel s_3$ and applying the transfer conditions of strong bisimulation we get

$$\exists p_4, s_4'. \left[p_2 \parallel s_2' \xrightarrow{e} p_4 \parallel s_4' \wedge p_1 \parallel s_3 \leftrightarrows^{\sqcup} p_4 \parallel s_4'\right]. \tag{B.1}$$

Since, the sets E_p, E_s are disjoint, thus, from the semantics we have $p_2 = p_4$ and $s_2' \xrightarrow{e} s_4'$. From above we have $p_2' \parallel s_2 \xrightarrow{v}\!\!\!\!\rightarrow p_2 \parallel s_2'$. And, Proposition 2.11 we get $s_2 \xrightarrow{?v}\!\!\!\!\rightarrow s_2'$. Note that $?v = ?\nu$, since $\nu = \bar{v}$. Thus, we have $\nabla_3(p_2 \,|[\epsilon,\nu]|_{\mathrm{h}}\, s_2) \longrightarrow\!\!\!\!\rightarrow \nabla_3(p_2 \,|[\epsilon,\epsilon]|_{\mathrm{h}}\, s_2')$. But, $p_1 \parallel s_1 \leftrightarrows^{\sqcup} p_2 \parallel s_2'$, so from the construction of $\mathcal{B}$ we have $(p_1 \parallel s_1, \nabla_3(p_2 \,|[\epsilon,\epsilon]|_{\mathrm{h}}\, s_2')) \in \mathcal{B}$. From the transition $s_2' \xrightarrow{e} s_4'$ we get $\nabla_3(p_2 \,|[\epsilon,\epsilon]|_{\mathrm{h}}\, s_2') \xrightarrow{e} \nabla_3(p_2 \,|[\epsilon,\epsilon]|_{\mathrm{h}}\, s_4')$. Applying the fact $p_1 \parallel s_3 \leftrightarrows^{\sqcup} p_4 \parallel s_4'$ from Equation B.1 in the construction of $\mathcal{B}$, we get $(p_1 \parallel s_3, \nabla_3(p_2 \,|[\epsilon,\epsilon]|_{\mathrm{h}}\, s_4')) \in \mathcal{B}$.

iv. Let $\mu, \nu \neq \epsilon$. Inapplicable, due to half-duplex mechanism!

(b) Let $\alpha = e$, for some $e \in E_p$. Similar to the previous case.

(c) Let $\alpha = m$, for some $m \in M_p$. Similar to Case 1a.

(d) Let $\alpha = n$, for some $n \in M_s$. Similar to Case 1a.

2. Let $p_1 \parallel s_1 \sqcup$ and $(p_1 \parallel s_1, \nabla_3(p_2 \,|[\mu,\nu]|_{\mathrm{h}}\, s_2)) \in \mathcal{B}$. Now applying structural induction on μ, ν we get the following cases:

(a) Let $\mu = \epsilon, \nu = \epsilon$. Trivial.

(b) Let $\mu \neq \epsilon, \nu = \epsilon$. Then, from the construction we have:

$$\exists p_2', s_2', u \in (M_s \cup E_s)^*. \Big[p_2 \parallel s_2' \xrightarrow{u}\!\!\!\!\rightarrow p_2' \parallel s_2 \wedge \mu = \bar{u} \wedge p_1 \parallel s_1 \leftrightarrows^{\sqcup} p_2' \parallel s_2\Big].$$

From Proposition 2.11 we have $p_2 \xrightarrow{?u}\!\!\!\!\rightarrow p_2'$. Note that $\mu = \bar{u}$; thus, $?u = ?\mu$. Thus, $\nabla_3(p_2 \,|[\mu,\epsilon]|_{\mathrm{h}}\, s_2) \longrightarrow\!\!\!\!\rightarrow \nabla_3(p_2' \,|[\epsilon,\epsilon]|_{\mathrm{h}}\, s_2)$. Clearly, $\nabla_3(p_2 \,|[\epsilon,\epsilon]|_{\mathrm{h}}\, s_2)\sqcup$. And from the construction of $\mathcal{B}$ we get $(p_1 \parallel s_1, \nabla_3(p_2' \,|[\epsilon,\epsilon]|_{\mathrm{h}}\, s_2)) \in \mathcal{B}$.

(c) Let $\mu = \epsilon, \nu \neq \epsilon$. Similar to the previous case.

(d) Let $\mu \neq \epsilon, \nu \neq \epsilon$. Inapplicable, due to the half-duplex mechanism!

3. Let $\nabla_3(p_2 \,|[\mu,\nu]|_{\mathrm{h}}\, s_2) \xrightarrow{\alpha} \nabla_3(p_4 \,|[\mu',\nu']|_{\mathrm{h}}\, s_4)$, $(p_1 \parallel s_1, \nabla_3(p_2 \,|[\mu,\nu]|_{\mathrm{h}}\, s_2)) \in \mathcal{B}$. Based on the type of α we get the following cases:

   (a) Let $\alpha = e$, for some $e \in E_s$. Then, $p_2 = p_4$, $\mu' = \mu$, $\nu' = \nu$, and $s_2 \xrightarrow{e} s_4$. Now applying structural induction on μ, ν we get:

      i. Let $\mu = \epsilon, \nu = \epsilon$. Similar to the next case.

      ii. Let $\mu \neq \epsilon, \nu = \epsilon$. Then,

$$\exists p_2', s_2', u \in (M_s \cup E_s)^*. \Big[p_2 \parallel s_2' \xrightarrow{u}\!\!\!\!\rightarrow p_2' \parallel s_2 \wedge \mu = \bar{u} \wedge p_1 \parallel s_1 \leftrightarrow\!\!\!\!\!\!\sqcup\; p_2' \parallel s_2\Big].$$

      Using the transition $s_2 \xrightarrow{e} s_4$ we get $p_2' \parallel s_2 \xrightarrow{e} p_2' \parallel s_4$. Now from the transfer conditions of strong bisimulation we get

$$\exists p_3, s_3. \left[p_1 \parallel s_1 \xrightarrow{e} p_3 \parallel s_3 \wedge p_3 \parallel s_3 \leftrightarrow\!\!\!\!\!\!\sqcup\; p_2' \parallel s_4\right].$$

      Using the above derived facts and $\overline{u.e} = \mu$ in the construction of $\mathcal{B}$ we conclude that $(p_1 \parallel s_3, \nabla_3(p_2 \,|[\mu,\epsilon]|_{\mathrm{h}}\, s_4)) \in \mathcal{B}$.

      iii. Let $\mu = \epsilon, \nu \neq \epsilon$. Then,

$$\exists p_2', s_2', v \in (M_p \cup E_p)^*. \Big[p_2' \parallel s_2 \xrightarrow{v}\!\!\!\!\rightarrow p_2 \parallel s_2' \wedge \nu = \bar{v} \wedge p_1 \parallel s_1 \leftrightarrow\!\!\!\!\!\!\sqcup\; p_2 \parallel s_2'\Big].$$

      By applying Proposition 2.11 we get $p_2' \xrightarrow{!v}\!\!\!\!\rightarrow p_2$. Using the transition $s_2 \xrightarrow{e} s_4$ we get $p_2' \parallel s_2 \xrightarrow{e} p_2' \parallel s_4$. But, the given synchronous system is well-posed. By applying Lemma 3.7 we get $p_2' \parallel s_4 \in \mathfrak{R}(p \parallel s) \wedge p_2' \xrightarrow{!v}\!\!\!\!\rightarrow p_2 \Rightarrow \exists s_4'. \left[p_2' \parallel s_4 \xrightarrow{v}\!\!\!\!\rightarrow p_2 \parallel s_4'\right]$. Now applying the strong version of Definition 4.10 at $p_2' \parallel s_2$ we get $\exists s_6, s_6'. \left[p_2' \parallel s_2 \xrightarrow{v}\!\!\!\!\rightarrow p_2 \parallel s_6' \xrightarrow{e} p_2 \parallel s_6 \wedge p_2 \parallel s_6 \leftrightarrow\!\!\!\!\!\!\sqcup\; p_2 \parallel s_4'\right]$. Now, from input-determinism we have $p_2 \parallel s_2' \leftrightarrow\!\!\!\!\!\!\sqcup\; p_2 \parallel s_6'$. From above, we have $p_1 \parallel s_1 \leftrightarrow\!\!\!\!\!\!\sqcup\; p_2 \parallel s_2'$ and by transitivity we get $p_1 \parallel s_1 \leftrightarrow\!\!\!\!\!\!\sqcup\; p_2 \parallel s_6'$. Consider $p_2 \parallel s_6' \xrightarrow{e} p_2 \parallel s_6$ and applying the transfer conditions of strong bisimulation we get: $\exists p_3, s_3. \left[p_1 \parallel s_1 \xrightarrow{e} p_3 \parallel s_3 \wedge p_3 \parallel s_3 \leftrightarrow\!\!\!\!\!\!\sqcup\; p_2 \parallel s_6\right]$. Also, from above we have $p_2 \parallel s_6 \leftrightarrow\!\!\!\!\!\!\sqcup\; p_2 \parallel s_4'$; by transitivity we get $p_3 \parallel s_3 \leftrightarrow\!\!\!\!\!\!\sqcup\; p_2 \parallel s_4'$. Using the transitions $p_2' \parallel s_4 \xrightarrow{v}\!\!\!\!\rightarrow p_2 \parallel s_4'$ and the fact $p_3 \parallel s_3 \leftrightarrow\!\!\!\!\!\!\sqcup\; p_2 \parallel s_4'$ in the construction of $\mathcal{B}$ we conclude that $(p_3 \parallel s_3, \nabla_3(p_2 \,|[\epsilon,\nu]|_{\mathrm{h}}\, s_4)) \in \mathcal{B}$.

      iv. When $\mu \neq \epsilon, \nu \neq \epsilon$. Inapplicable due to half-duplex mechanism!

   (b) Let $\alpha = e$, for some $e \in E_p$. Similar to the previous case.

(c) Let $\alpha = n$, for some $n \in M_s$. Then, $p_2 = p_4$, $\mu' = \mu.n$, $\nu' = \nu$, $s_2 \xrightarrow{!n} s_4$. Now, applying structural induction on μ, ν we get the following cases:

i. When $\mu = \epsilon, \nu = \epsilon$. Similar to the next case.

ii. When $\mu \neq \epsilon, \nu = \epsilon$. Then,

$$\exists p_2', s_2', u \in (M_s \cup E_s)^*. \Big[p_2 \parallel s_2' \xrightarrow{u} \!\!\!\!\rightarrow\, p_2' \parallel s_2 \wedge \mu = \bar{u} \wedge p_1 \parallel s_1 \leftrightarrows^{\sqcup} p_2' \parallel s_2\Big].$$

Since, the processes p, s are well-posed. So by Lemma 3.7 we have $\exists p_4'. \left[p_2' \parallel s_2 \xrightarrow{n} p_4' \parallel s_4\right]$. But, $p_1 \parallel s_1 \leftrightarrows^{\sqcup} p_2' \parallel s_2$. So from the transfer conditions of strong bisimulation we get

$$\exists p_3, s_3. \left[p_1 \parallel s_1 \xrightarrow{n} p_3 \parallel s_3 \wedge p_4' \parallel s_4 \leftrightarrows^{\sqcup} p_3 \parallel s_3\right].$$

Using the above facts and $\overline{u.n} = \mu.n$ in the construction of $\mathcal{B}$ we conclude that $(p_3 \parallel s_3, \nabla_3(p_2 \,|[\mu.n, \epsilon]|_{\mathrm{h}}\, s_4)) \in \mathcal{B}$.

iii. Let $\mu = \epsilon, \nu \neq \epsilon$. Inapplicable due to half-duplex mechanism!

iv. Let $\mu \neq \epsilon, \nu \neq \epsilon$. Inapplicable due to half-duplex mechanism!

(d) Let $\alpha = m$, for some $m \in M_p$. Similar to the previous case.

(e) Let $\alpha = \tau$. Since, the local processes are concrete so the transition $\nabla_3(p_2 \,|[\mu, \nu]|_{\mathrm{h}}\, s_2) \xrightarrow{\tau} \nabla_3(p_4 \,|[\mu', \nu']|_{\mathrm{h}}\, s_4)$ is either due to Rule 12 or Rule 14.

- Application of Rule 12. Now perform structural induction on μ, ν we get the following cases:

i. Let $\mu = \epsilon, \nu = \epsilon$. Inapplicable, because μ is nonempty when Rule 12 is applied!

ii. Let $\mu \neq \epsilon, \nu = \epsilon$. Then, $p_2 \xrightarrow{?n} p_4$, $\mu = n.\mu'$, $\nu' = \nu$, and $s_2 = s_4$. Also, from the construction we have

$$\exists p_2', s_2', u \in (M_s \cup E_s)^*. \Big[p_2 \parallel s_2' \xrightarrow{u} \!\!\!\!\rightarrow\, p_2' \parallel s_2 \wedge \mu = \bar{u} \wedge p_1 \parallel s_1 \leftrightarrows^{\sqcup} p_2' \parallel s_2\Big].$$

Since $\mu = \bar{u}$ and $\mu = n.\mu'$, we can then decomposed the above transition as:

$$p_2 \parallel s_2' \xrightarrow{u_1} \!\!\!\!\rightarrow\, p_2 \parallel s_2'' \xrightarrow{n} p_4' \parallel s_4' \xrightarrow{u_2} \!\!\!\!\rightarrow\, p_2' \parallel s_2,$$

for some $u_1, u_2 \in (E_s \cup M_s)^*$ such that $\bar{u}_1 = \epsilon$, $\bar{u}_2 = \mu'$, and $s_2'', p_4', s_4' \in \mathbb{P}$. Thus, we get $p_2 \xrightarrow{?n} p_4'$, $s_2'' \xrightarrow{!n} s_4'$, $s_4' \xrightarrow{!u_2} \!\!\!\!\rightarrow\, s_2$. Also, from above we have $p_2 \xrightarrow{?n} p_4$.

Then, we get $p_2 \parallel s_2'' \xrightarrow{n} p_4 \parallel s_4'$. Since the given synchronous system is well-posed, so from Lemma 3.7 we get $\exists p_6.\left[p_4 \xrightarrow{?u_2} p_6\right]$. Thus, $p_4 \parallel s_4' \xrightarrow{u_2} p_6 \parallel s_2$.

And from input-determinism we get $p_2' \parallel s_2 \leftrightarrow^{\sqcup}_{\mathbf{b}} p_6 \parallel s_2$. Since the synchronous system is concrete, from Proposition B.2 we have $p_2' \parallel s_2 \leftrightarrow^{\sqcup} p_6 \parallel s_2$. But, $p_1 \parallel s_1 \leftrightarrow^{\sqcup} p_2' \parallel s_2$ and from transitivity we get $p_1 \parallel s_1 \leftrightarrow^{\sqcup} p_6 \parallel s_2$. Finally, using the transition $p_4 \parallel s_4' \xrightarrow{u_2} p_6 \parallel s_2$ and the facts $\bar{u}_2 = \mu', p_1 \parallel s_1 \leftrightarrow^{\sqcup} p_6 \parallel s_2$ in the construction of $\mathcal{B}$ we get $(p_1 \parallel s_1, \nabla_3(p_4 \,|[\mu', \epsilon]|_{\mathrm{h}}\, s_2)) \in \mathcal{B}$.

iii. Let $\mu = \epsilon, \nu \neq \epsilon$. Inapplicable, same reason as in Case3(e)i!

iv. Let $\mu, \nu \neq \epsilon$. Inapplicable, due to half-duplex mechanism!

- Application of Rule 14. Similar to the previous case.

4. Let $\nabla_3(p_2 \,|[\mu, \nu]|_{\mathrm{h}}\, s_2)\sqcup$ and $(p_1 \parallel s_1, \nabla_3(p_2 \,|[\mu, \nu]|_{\mathrm{h}}\, s_2)) \in \mathcal{B}$. Then, from semantics we have $\mu = \epsilon, \nu = \epsilon$, and from the construction of $\mathcal{B}$ we get $p_1 \parallel s_1 \leftrightarrow^{\sqcup} p_2 \parallel s_2$. Clearly, from the semantics we have $p_1 \parallel s_1 \sqcup$. □

Lemma B.5 (Proof of Lemmma 4.20)**.** *Suppose a concrete and well-posed synchronous system $p \parallel s$ is E-independent modulo $\simeq^{\sqcup}$. If $\nabla_3(p_1 \,|[\epsilon, \epsilon]|_{\mathrm{h}}\, s_1) \in \mathfrak{R}(\nabla_3(p \,|[\epsilon, \epsilon]|_{\mathrm{h}}\, s))$ and $\nabla_3(p_1 \,|[\epsilon, \epsilon]|_{\mathrm{h}}\, s_1) \xrightarrow{w} \nabla_3(p_2 \,|[\epsilon, \epsilon]|_{\mathrm{h}}\, s_2)$ then,*

$$\exists p_3, s_3.\left[p_1 \parallel s_1 \xrightarrow{w} p_3 \parallel s_3 \wedge p_2 \parallel s_2 \leftrightarrow^{\sqcup} p_3 \parallel s_3\right].$$

Proof. We prove this lemma by **strong induction** on w. When $w = \epsilon$ then the lemma holds trivially due to concreteness assumption. So suppose otherwise $w = w'.\alpha$ such that $\nabla_3(p_1 \,|[\epsilon, \epsilon]|_{\mathrm{h}}\, s_1) \xrightarrow{w'} \nabla_3(p_2' \,|[\mu, \nu]|_{\mathrm{h}}\, s_2') \xrightarrow{\alpha} \nabla_3(p_2 \,|[\epsilon, \epsilon]|_{\mathrm{h}}\, s_2)$. Based on the type of α we get the following cases:

1. Let $\alpha = e$, for some $e \in E_s$. Then, from the semantics we know that $p_2' = p_2$, $\mu, \nu = \epsilon$, and $s_2' \xrightarrow{e} s_2$. And by inductive hypothesis we have

$$\exists p_3, s_3.\left[p_1 \parallel s_1 \xrightarrow{w'} p_3 \parallel s_3 \wedge p_2 \parallel s_2' \leftrightarrow^{\sqcup} p_3 \parallel s_3\right].$$

From the transition $s_2' \xrightarrow{e} s_2$, we get $p_2 \parallel s_2' \xrightarrow{e} p_2 \parallel s_2$. Using the transfer conditions of strong bisimulation we get

$$\exists p_4, s_4.\left[p_3 \parallel s_3 \xrightarrow{e} p_4 \parallel s_4 \wedge p_2 \parallel s_2 \leftrightarrow^{\sqcup} p_4 \parallel s_4\right].$$

2. Let $\alpha = e$, for some $e \in E_p$. Then, from the semantics we know that $p_2' \xrightarrow{e} p_2$, $\mu, \nu = \epsilon$, and $s_2' = s_2$. The remainder of the proof is similar to the previous case.

3. Let $\alpha = m$, or $\alpha = n$, for some $m \in M_p, n \in M_s$. Inapplicable, because the target state contains empty queue contents!

4. Let $\alpha = \tau$. Note that this transition can be either due to Rule 12 or Rule 14. Then, $w = w'$ by definition of reachability relation $\longrightarrow\!\!\!\!\!\rightarrow$.

   (a) Application of Rule 12. Then, $p_2' \xrightarrow{?n} p_2$, $\mu = n$, $\nu = \epsilon$, $s_2' = s_2$, for some $n \in M_s$. Then, the transition $\nabla_3(p_2' \,|[\mu,\nu]|_{\mathrm{h}}\, s_2') \xrightarrow{\alpha} \nabla_3(p_2 \,|[\epsilon,\epsilon]|_{\mathrm{h}}\, s_2)$ is of the form $\nabla_3(p_2' \,|[n,\epsilon]|_{\mathrm{h}}\, s_2) \xrightarrow{\tau} \nabla_3(p_2 \,|[\epsilon,\epsilon]|_{\mathrm{h}}\, s_2)$. And from triangle lemma (Lemma 4.19) we have there exists $p_1', s_1' \in \mathbb{P}$, $u \in (E_s \cup M_s)^*$, $v \in (E_p \cup ?M_p)^*$ such that

      i. $\nabla_3(p_1 \,|[\epsilon,\epsilon]|_{\mathrm{h}}\, s_1) \xrightarrow{w_1} \!\!\!\!\!\rightarrow\ \nabla_3(p_1' \,|[\epsilon,\epsilon]|_{\mathrm{h}}\, s_1') \xrightarrow{w_2} \!\!\!\!\!\rightarrow\ \nabla_3(p_2' \,|[n,\epsilon]|_{\mathrm{h}}\, s_2)$, $w = w_1.w_2$, and

      ii. $s_1' \xrightarrow{!u} \!\!\!\!\!\rightarrow\ s_2, p_1' \xrightarrow{v} \!\!\!\!\!\rightarrow\ p_2'$, $?u = (v \uparrow ?M_p).?n$ and $u.(v \uparrow E_p) \sqsubseteq_s w_2$.

   And, from the strong induction hypothesis we have

$$\exists p_3, s_3.\Big[p_1 \parallel s_1 \xrightarrow{w_1} \!\!\!\!\!\rightarrow\ p_3 \parallel s_3 \wedge p_1' \parallel s_1' \leftrightarrow^{\sqcup} p_3 \parallel s_3\Big]. \tag{B.2}$$

   Next, we show that

$$\begin{aligned}\nabla_3(p_1' \,|[\epsilon,\epsilon]|_{\mathrm{h}}\, s_1') \xrightarrow{w_2} \!\!\!\!\!\rightarrow\ \nabla_3(p_2 \,|[\epsilon,\epsilon]|_{\mathrm{h}}\, s_2) \Rightarrow \\ \exists p_4, s_4.\Big[p_1' \parallel s_1' \xrightarrow{w_2} \!\!\!\!\!\rightarrow\ p_4 \parallel s_4 \wedge p_2 \parallel s_2 \leftrightarrow^{\sqcup} p_4 \parallel s_4\Big].\end{aligned} \tag{B.3}$$

   Note that if the above condition holds then the desired result:

$$\exists p_5, s_5.\Big[p_3 \parallel s_3 \xrightarrow{w_2} \!\!\!\!\!\rightarrow\ p_5 \parallel s_5 \wedge p_5 \parallel s_5 \leftrightarrow^{\sqcup} p_4 \parallel s_4\Big]$$

   follows directly from the transfer conditions of bisimilarity. And by transitivity of $\leftrightarrow^{\sqcup}$ we get $p_5 \parallel s_5 \leftrightarrow^{\sqcup} p_2 \parallel s_2$.
   In the remainder we show the condition in Equation B.3 holds. To this end, we rewrite the transitions $s_1' \xrightarrow{!u} \!\!\!\!\!\rightarrow\ s_2$ as (for some $j \geq 1$):

$$s_1^{\triangle} \xrightarrow{u_1} \!\!\!\!\!\rightarrow\ s_1^{\triangledown} \xrightarrow{!n_1} s_2^{\triangle} \cdots \xrightarrow{u_j} \!\!\!\!\!\rightarrow\ s_j^{\triangledown} \xrightarrow{!n_j} s_{j+1}^{\triangle},$$

   where $u_1.!n_1.\cdots.u_j.!n_j.u_{j+1} = !u$, $u_i \in {E_s}^*$ (for all $i \in [1,j]$), $s_1^{\triangle} = s_1'$, $s_{j+1}^{\triangle} = s_2$, and $!n_j = !n$.

   Similarly, we rewrite the transitions $p_1' \xrightarrow{?v.?n} \!\!\!\!\!\rightarrow\ p_2$ as:

$$p_1^{\triangle} \xrightarrow{v_1} \!\!\!\!\!\rightarrow\ p_1^{\triangledown} \xrightarrow{?n_1} p_2^{\triangle} \cdots \xrightarrow{v_j} \!\!\!\!\!\rightarrow\ p_j^{\triangledown} \xrightarrow{?n_j} p_{j+1}^{\triangle},$$

   where $v_1.?n_1.\cdots.v_j.?n_j.v_{j+1} = v$, $v_i \in {E_p}^*$ (for all $i \in [1,j]$), $p_1^{\triangle} = p_1'$, $p_j^{\triangledown} = p_2'$, $p_{j+1}^{\triangle} = p_2$, $?n_j = ?n$. Thus, for every $i \in [1,j]$

we have:

$$\begin{aligned} s_i^{\triangle} &\xrightarrow{u_i} \!\!\!\!\rightarrow\ s_i^{\triangledown} \xrightarrow{!n_i} s_{i+1}^{\triangle}, \\ p_i^{\triangle} &\xrightarrow{v_i} \!\!\!\!\rightarrow\ p_i^{\triangledown} \xrightarrow{?n_i} p_{i+1}^{\triangle}. \end{aligned} \tag{B.4}$$

A consequence of the above encoding is: if w_3 is a prefix of w_2 and $u_1.n_1.\cdots.u_i.n_i.v_1.\cdots.v_i \sqsubseteq_s w_3$ (for $i \leq j$), then there exists q such that $\nabla_3(p_1^{\triangle} \,|[\epsilon,\epsilon]|_{\mathrm{h}}\, s_1^{\triangle}) \xrightarrow{w_3} \!\!\!\!\rightarrow\ q$ and $q = \nabla_3(p_{i+1}^{\triangle} \,|[\epsilon,\epsilon]|_{\mathrm{h}}\, s_{i+1}^{\triangle})$. Next, we **show** that for every sequence w_3 such that the sequence w_3 is a prefix of the sequence w_2, $u_1.n_1.\cdots.u_i.n_i.v_1.\cdots.v_i \sqsubseteq_s w_3$, (for $i \leq j$) the following condition hold:

$$\begin{aligned} &\nabla_3(p_1^{\triangle} \,|[\epsilon,\epsilon]|_{\mathrm{h}}\, s_1^{\triangle}) \xrightarrow{w_3} \!\!\!\!\rightarrow\ \nabla_3(p_{i+1}^{\triangle} \,|[\epsilon,\epsilon]|_{\mathrm{h}}\, s_{i+1}^{\triangle}) \Rightarrow \\ &\quad \exists p', s'.\Big[p_1^{\triangle} \parallel s_1^{\triangle} \xrightarrow{w_3} \!\!\!\!\rightarrow\ p' \parallel s' \wedge p' \parallel s' \leftrightarrows^{\sqcup} p_{i+1}^{\triangle} \parallel s_{i+1}^{\triangle}\Big]. \end{aligned} \tag{B.5}$$

Observe that the condition in Equation B.3 follows directly from the condition in Equation B.5 by substituting $w_3 = w_2$, $p_1^{\triangle} = p_1'$, $s_1^{\triangle} = s_1'$, $p_{j+1}^{\triangle} = p_2$, and $s_{j+1}^{\triangle} = s_2$.

Let w_3 be a prefix of w_2 such that $u_1.\cdots.u_i.n_i.v_1.\cdots.v_i \sqsubseteq_s w_3$ and the condition in Equation B.5 holds. We show by induction the above result holds for the sequence $w_3.w_3'$, where $u_{i+1}.n_{i+1}.v_{i+1} \sqsubseteq_s w_3'$. By the induction hypothesis, we have $\exists p', s'.\Big[p_1^{\triangle} \parallel s_1^{\triangle} \xrightarrow{w_3} \!\!\!\!\rightarrow\ p' \parallel s' \wedge p' \parallel s' \leftrightarrows^{\sqcup} p_{i+1}^{\triangle} \parallel s_{i+1}^{\triangle}\Big]$. Now, using the transitions in Equation B.4 we get the transitions depicted in Figure B.1.

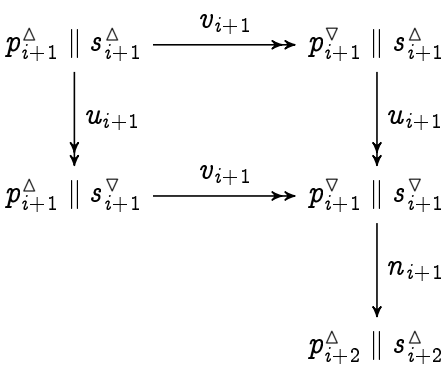


FIGURE B.1: Case 4(a) of Lemma 4.20.

Since, $u_{i+1}.n_{i+1}.v_{i+1} \sqsubseteq_s w_3'$ then the sequence w_3' is in one of the following forms:

i. Either, $w_3' = w_4'.n_{i+1}$ such that $u_{i+1}.v_{i+1} \sqsubseteq_s w_4'$. Consider the transitions $p_{i+1}^{\triangle} \parallel s_{i+1}^{\triangle} \xrightarrow{u_{i+1}.v_{i+1}} \!\!\!\!\rightarrow\ p_{i+1}^{\triangledown} \parallel s_{i+1}^{\triangledown}$ (see Figure B.1). Since $u_{i+1}.v_{i+1} \sqsubseteq_s w_4'$ and the sequences of external actions

$u_{i+1} \in {E_s}^*, v_{i+1} \in {E_p}^*$ can be executed in any arbitrary order by the processes $p^{\triangle}_{i+1}, s^{\triangle}_{i+1}$ in the synchronous system; thus, we have $p^{\triangle}_{i+1} \parallel s^{\triangle}_{i+1} \xrightarrow{w'_4} p^{\triangledown}_{i+1} \parallel s^{\triangledown}_{i+1}$. Hence,

$$p^{\triangle}_{i+1} \parallel s^{\triangle}_{i+1} \xrightarrow{w'_4.n_{i+1}} p^{\triangle}_{i+2} \parallel s^{\triangle}_{i+2}.$$

ii. Or, $w'_3 = w'_4.n_{i+1}.v''_{i+1}$ such that $u_{i+1}.v'_{i+1} \sqsubseteq_s w'_4$ and $v_{i+1} = v'_{i+1}.v''_{i+1}$. Since $p^{\triangle}_{i+1} \xrightarrow{v_{i+1}} p^{\triangledown}_{i+1}$ (Equation B.4) and $v_{i+1} = v'_{i+1}.v''_{i+1}$, then there exists $p^{\diamond}_{i+1} \in \mathbb{P}$ such that

$$p^{\triangle}_{i+1} \xrightarrow{v'_{i+1}} p^{\diamond}_{i+1} \xrightarrow{v''_{i+1}} p^{\triangledown}_{i+1}.$$

Thus, using the transitions in Equation B.4 we get the transitions at the state $p^{\triangle}_{i+1} \parallel s^{\triangle}_{i+1}$ depicted as solid lines in Figure B.2. By applying Definition 4.10 at the state $p^{\diamond}_{i+1} \parallel s^{\triangledown}_{i+1}$ we get the dashed transition of Figure B.2. Since $u_{i+1}.v'_{i+1} \sqsubseteq_s$

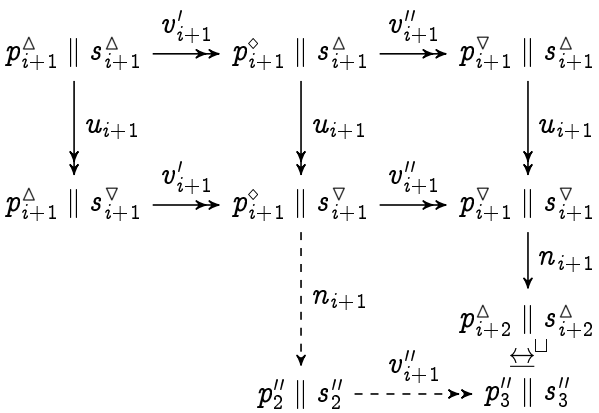


FIGURE B.2: Case 4(a) of Lemma 4.20.

w'_4 and the sequences of external actions $u_{i+1} \in {E_s}^*, v'_{i+1} \in {E_p}^*$ can be executed in any arbitrary order by the processes $p^{\triangle}_{i+1}, s^{\triangle}_{i+1}$ in the synchronous system; thus, we have $p^{\triangle}_{i+1} \parallel s^{\triangle}_{i+1} \xrightarrow{w'_4} p^{\diamond}_{i+1} \parallel s^{\triangledown}_{i+1}$. Hence, $p^{\triangle}_{i+1} \parallel s^{\triangle}_{i+1} \xrightarrow{w'_3} p''_3 \parallel s''_3$, where $p''_3 \parallel s''_3 \leftrightarrow^{\sqcup} p^{\triangle}_{i+2} \parallel s^{\triangle}_{i+2}$. Thus, the condition in Equation B.5 holds.

(b) Application of Rule 14. Similar to the previous case. □

Theorem B.6 (Proof of Theorem 4.21). *Let p, s be concrete processes. If the synchronous system $p \parallel s$ is well-posed and E-independent modulo $\sim_{\mathbf{c}}$, then $p \parallel s \sim_{\mathbf{c}} \nabla_3(p \,|[\epsilon, \epsilon]|_{\mathrm{h}}\, s)$.*

Proof. For brevity we fix the type of u, v as $(M_s \cup E_s)^*, (M_p \cup E_p)^*$, respectively. Recall the definition of the relation $\mathcal{B}$ from Theorem B.4. Now, define a pair of relations $\mathcal{C}_1, \mathcal{C}_2$ in the following way:

$$\mathcal{C}_1 = \mathcal{B},\ \mathcal{C}_2 = \{(\nabla_3(p_2 \,|[\epsilon,\epsilon]|_{\mathrm{h}}\, s_2), p_1 \parallel s_1) \mid p_1 \parallel s_1 \leftrightarrows p_2 \parallel s_2\}.$$

Next, we show that $\mathcal{C} = \mathcal{C}_1 \cup \mathcal{C}_2$ is a contra-simulation relation. Note that by construction we have $\mathcal{C}_1 \cap \mathcal{C}_2 = \emptyset$. In the following, without mentioning this reason we write $(q_s, q_a) \in \mathcal{C}_1$ whenever $(q_s, q_a) \in \mathcal{C}$ and $(q_a, q_s) \in \mathcal{C}_2$ whenever $(q_a, q_s) \in \mathcal{C}$, where $q_s \in \mathfrak{R}(p \parallel s)$ and $q_a \in \mathfrak{R}(\nabla_3(p \,|[\epsilon,\epsilon]|_{\mathrm{h}}\, s))$.

1. Let $p_1' \parallel s_1' \xrightarrow{w} p_3 \parallel s_3$ and $(p_1' \parallel s_1', \nabla_3(p_2' \,|[\mu,\nu]|_{\mathrm{h}}\, s_2')) \in \mathcal{C}$. By structural induction on w we get following cases:

   - Let $w = \epsilon$. Then, due to concreteness assumption we have $p_1' = p_3, s_1' = s_3$. Now, apply structural induction on μ, ν we get the following cases:
     (a) Let $\mu = \epsilon, \nu = \epsilon$. Similar to the next case.
     (b) Let $\mu \neq \epsilon, \nu = \epsilon$. Then, we have

     $$\exists p_2'', s_2'', u.\Big[p_2' \parallel s_2'' \xrightarrow{u} p_2'' \parallel s_2' \wedge \mu = \bar{u} \wedge p_1' \parallel s_1' \leftrightarrows p_2'' \parallel s_2'\Big].$$

     From Proposition 2.11 we get $p_2' \xrightarrow{?u} p_2''$. Thus,

     $$\nabla_3(p_2' \,|[\mu,\epsilon]|_{\mathrm{h}}\, s_2') \longrightarrow\!\!\!\!\!\rightarrow \nabla_3(p_2'' \,|[\epsilon,\epsilon]|_{\mathrm{h}}\, s_2').$$

     Furthermore, using the fact $p_1' \parallel s_1' \leftrightarrows p_2'' \parallel s_2'$ in the construction of $\mathcal{C}_2$ we conclude that $(\nabla_3(p_2'' \,|[\epsilon,\epsilon]|_{\mathrm{h}}\, s_2'), p_1' \parallel s_1') \in \mathcal{C}_2$.
     (c) Let $\mu = \epsilon, \nu \neq \epsilon$. Similar to the previous case.
     (d) Let $\mu, \nu \neq \epsilon$. Inapplicable, due to half-duplex mechanism!
   - Let $w = w'.\alpha$, for some $w' \in A^*, \alpha \in A$. Then, assume that $p_1' \parallel s_1 \xrightarrow{w'} p_1 \parallel s_1 \xrightarrow{\alpha} p_3 \parallel s_3$, for some $p_1, s_1 \in \mathbb{P}$. By the induction hypothesis we have there exists $p_2, s_2 \in \mathbb{P}$ such that

     $$\nabla_3(p_2' \,|[\mu,\nu]|_{\mathrm{h}}\, s_2') \xrightarrow{w'} \nabla_3(p_2 \,|[\mu',\nu']|_{\mathrm{h}}\, s_2),$$

     and $(\nabla_3(p_2 \,|[\mu',\nu']|_{\mathrm{h}}\, s_2), p_1 \parallel s_1) \in \mathcal{C}$. Thus, $(\nabla_3(p_2 \,|[\mu',\nu']|_{\mathrm{h}}\, s_2), p_1 \parallel s_1) \in \mathcal{C}_2$ and from the construction of $\mathcal{C}_2$ we know that $\mu' = \epsilon, \nu' = \epsilon$. The remainder of the proof is similar to the Case 1 of Theorem B.4.

2. Let $\nabla_3(p_3 \,|[\mu_3,\nu_3]|_{\mathrm{h}}\, s_3) \xrightarrow{w} \nabla_3(p_4 \,|[\mu_4,\nu_4]|_{\mathrm{h}}\, s_4)$, $(\nabla_3(p_3 \,|[\mu_3,\nu_3]|_{\mathrm{h}}\, s_3), p_3' \parallel s_3') \in \mathcal{C}$. From the construction of $\mathcal{C}_2$ we have $\mu_3 = \epsilon, \nu_3 = \epsilon$, i.e.,

   $$(\nabla_3(p_3 \,|[\epsilon,\epsilon]|_{\mathrm{h}}\, s_3), p_3' \parallel s_3') \in \mathcal{C}_2. \tag{B.6}$$

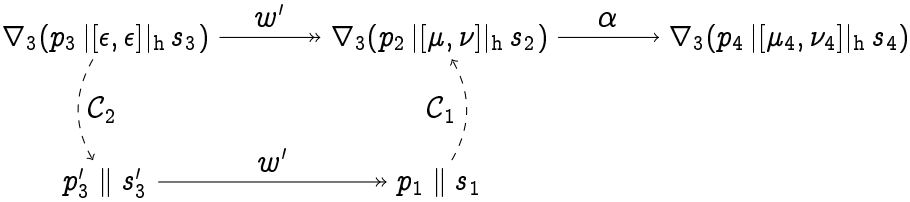


FIGURE B.3: Hypothesis in Case 2 of Theorem B.6.

When $w = \epsilon$, the proof is trivial. So consider otherwise, $w = w'.\alpha$ such that $\nabla_3(p_3 \,|[\epsilon,\epsilon]|_{\mathrm{h}}\, s_3) \xrightarrow{w'} \nabla_3(p_2 \,|[\mu,\nu]|_{\mathrm{h}}\, s_2) \xrightarrow{\alpha} \nabla_3(p_4 \,|[\mu_4,\nu_4]|_{\mathrm{h}}\, s_4)$, for some $\mu \in {M_s}^*, \nu \in {M_p}^*$. By inductive hypothesis we have (see Figure B.3) $\exists p_1, s_1.\left[p_3' \parallel s_3' \xrightarrow{w'} p_1 \parallel s_1 \wedge (p_1 \parallel s_1, \nabla_3(p_2 \,|[\mu,\nu]|_{\mathrm{h}}\, s_2)) \in \mathcal{C}\right]$. Thus, $(p_1 \parallel s_1, \nabla_3(p_2 \,|[\mu,\nu]|_{\mathrm{h}}\, s_2)) \in \mathcal{C}_1$. Based on the type of α we get the following cases:

(a) Let $\alpha = e$, for some $e \in E_s$. Then, from semantics we have $p_2 = p_4$, $s_2 \xrightarrow{e} s_4$, $\mu = \mu_4$, and $\nu = \nu_4$. Now, applying structural induction on μ, ν we get the following cases:

i. Let $\mu = \epsilon$, $\nu = \epsilon$. Similar to the next case.

ii. Let $\mu \neq \epsilon$, $\nu = \epsilon$. Then, from the construction of $\mathcal{C}_1$ we have

$$\exists p_2', s_2', u.\left[p_2 \parallel s_2' \xrightarrow{u} p_2' \parallel s_2 \wedge \mu = \bar{u} \wedge p_1 \parallel s_1 \leftrightarrows^{\sqcup} p_2' \parallel s_2\right].$$

Using the transition $s_2 \xrightarrow{e} s_4$ we get $p_2' \parallel s_2 \xrightarrow{e} p_2' \parallel s_4$. Note that $\overline{(u.e)} = \bar{u}.e = \mu$. Since $p_1 \parallel s_1 \leftrightarrows p_2' \parallel s_2$, so applying the transfer condition of strong bisimulation we get

$$\exists p_1', s_1'.\left[p_1 \parallel s_1 \xrightarrow{e} p_1' \parallel s_1' \wedge p_1' \parallel s_1' \leftrightarrows^{\sqcup} p_2' \parallel s_4\right].$$

Using the above facts in the construction of $\mathcal{C}_1$ we conclude that $(p_1' \parallel s_1', \nabla_3(p_2 \,|[\mu,\epsilon]|_{\mathrm{h}}\, s_4)) \in \mathcal{C}_1$.

iii. Let $\mu = \epsilon$, $\nu \neq \epsilon$. Then, from the construction of $\mathcal{C}_1$ we have

$$\exists p_2', s_2', v.\left[p_2' \parallel s_2 \xrightarrow{v} p_2 \parallel s_2' \wedge \nu = \bar{v} \wedge p_1 \parallel s_1 \leftrightarrows^{\sqcup} p_2 \parallel s_2'\right].$$

From Proposition 2.11 we get $p_2' \xrightarrow{!v} p_2$. And, using the transition $s_2 \xrightarrow{e} s_4$ we get $p_2' \parallel s_2 \xrightarrow{e} p_2' \parallel s_4$. Since, the processes p, s are well-posed, so applying Lemma 3.7 we get

$$\exists s_4'.\left[p_2' \parallel s_4 \xrightarrow{v} p_2 \parallel s_4'\right].$$

Again, from Proposition 2.11 we get $s_4 \xrightarrow{?v} s_4'$. Thus,

$$\nabla_3(p_2 \,|[\epsilon,\nu]|_{\mathrm{h}}\, s_4) \longrightarrow\!\!\!\!\!\rightarrow \nabla_3(p_2 \,|[\epsilon,\epsilon]|_{\mathrm{h}}\, s_4').$$

Recall the hypothesis and observe that we have derived the transition $\nabla_3(p_3 \,|[\epsilon,\epsilon]|_{\mathrm{h}}\, s_3) \xrightarrow{w'.e} \nabla_3(p_2 \,|[\epsilon,\epsilon]|_{\mathrm{h}}\, s_4')$. Now, applying Lemma 4.20 we get

$$\exists p_5, s_5.\left[p_3 \parallel s_3 \xrightarrow{w'.e} p_5 \parallel s_5 \wedge p_5 \parallel s_5 \leftrightarrows^{\sqcup} p_2 \parallel s_4'\right].$$

Thus, using the transition $p_2' \parallel s_4 \xrightarrow{v} p_2 \parallel s_4'$ and the equation $p_5 \parallel s_5 \leftrightarrows^{\sqcup} p_2 \parallel s_4'$ in the construction of $\mathcal{C}_1$ we conclude that (see Figure B.4) $(p_5 \parallel s_5, \nabla_3(p_2 \,|[\epsilon,\nu]|_{\mathrm{h}}\, s_4)) \in \mathcal{C}_1$.

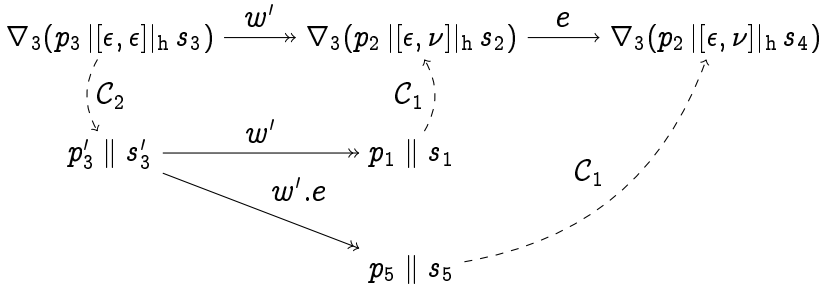


FIGURE B.4: Case 2(a)iii in hindsight.

iv. Let $\mu \neq \epsilon$, $\nu \neq \epsilon$. Inapplicable due to half-duplex mechanism!

(b) Let $\alpha = e$, for some $e \in E_p$. Similar to the previous case.

(c) Let $\alpha = n$, for some $n \in M_s \cup M_p$. Similar to Case 3(c)ii of Theorem B.4.

(d) Let $\alpha = \tau$. Since, the given processes p, s are concrete, so the transition $\nabla_3(p_2 \,|[\mu,\nu]|_{\mathrm{h}}\, s_2) \xrightarrow{\alpha} \nabla_3(p_4 \,|[\mu_4,\nu_4]|_{\mathrm{h}}\, s_4)$ (in the hypothesis) is either, due to the Rule 12, or Rule 14.

i. Application of Rule 12. Then, we have $p_2 \xrightarrow{?n} p_4$, $\mu = n.\mu_4$, $\nu = \nu_4$, and $s_2 = s_4$, for some $n \in M_s$. Now, applying structural induction on μ, ν we get the following cases:

A. Let $\mu = n, \nu = \epsilon$. Similar to the next case.

B. Let $\mu = n.\mu_4$, $\nu = \epsilon$. Then, from the construction of $\mathcal{C}_1$ we have

$$\exists p_2', s_2', u.\left[p_2 \parallel s_2' \xrightarrow{u} p_2' \parallel s_2 \wedge \mu = \bar{u} \wedge p_1 \parallel s_1 \leftrightarrows^{\sqcup} p_2' \parallel s_2\right].$$

Since $\bar{u} = \mu = n.\mu_4$, then the above transitions can be decomposed in the following way: there exists $p_4', s_2'', s_4' \in$

$\mathbb{P}$, $u_1 \in {E_s}^*, u_2 \in (E_s \cup M_s)^*$ such that $\bar{u}_1 = \epsilon$, $\bar{u}_2 = \mu_4$, and

$$p_2 \parallel s_2' \xrightarrow{u_1} \!\!\!\!\rightarrow\; p_2 \parallel s_2'' \xrightarrow{n} p_4' \parallel s_4' \xrightarrow{u_2} \!\!\!\!\rightarrow\; p_2' \parallel s_2.$$

Thus, $s_2'' \xrightarrow{!n} s_4' \xrightarrow{!u_2} \!\!\!\!\rightarrow\; s_2$. But, from above we have $p_2 \xrightarrow{?n} p_4$. Thus, we get $p_2 \parallel s_2'' \xrightarrow{n} p_4 \parallel s_4'$. Since, the given processes p, s are well-posed, so applying Lemma 3.7 we get $\exists p_6. \left[p_4 \parallel s_4' \xrightarrow{u_2} \!\!\!\!\rightarrow\; p_6 \parallel s_2\right]$. And from Proposition 2.11 we get $p_4 \xrightarrow{?u_2} \!\!\!\!\rightarrow\; p_6$. Since $\bar{u}_2 = \mu_4$, we have

$$\nabla_3(p_4 \,|[\mu_4, \epsilon]|_{\mathrm{h}}\, s_2) \longrightarrow\!\!\!\!\rightarrow\; \nabla_3(p_6 \,|[\epsilon, \epsilon]|_{\mathrm{h}}\, s_2).$$

Recall the hypothesis and observe that we derived the transition $\nabla_3(p_3 \,|[\epsilon, \epsilon]|_{\mathrm{h}}\, s_3) \xrightarrow{w} \!\!\!\!\rightarrow\; \nabla_3(p_6 \,|[\epsilon, \epsilon]|_{\mathrm{h}}\, s_2)$ (note, $w' = w$ when $\alpha = \tau$). Now, applying Lemma 4.20 we get

$$\exists p_5, s_5. \left[p_3 \parallel s_3 \xrightarrow{w} \!\!\!\!\rightarrow\; p_5 \parallel s_5 \wedge p_5 \parallel s_5 \leftrightarrows^{\sqcup} p_6 \parallel s_2\right].$$

Thus, using the transition $p_4 \parallel s_4' \xrightarrow{u_2} \!\!\!\!\rightarrow\; p_6 \parallel s_2$ and the equation $p_5 \parallel s_5 \leftrightarrows^{\sqcup} p_6 \parallel s_2$ in the construction of C_1, we have (see Figure B.5) $(p_5 \parallel s_5, \nabla_3(p_4 \,|[\mu_4, \epsilon]|_{\mathrm{h}}\, s_2)) \in C_1$.

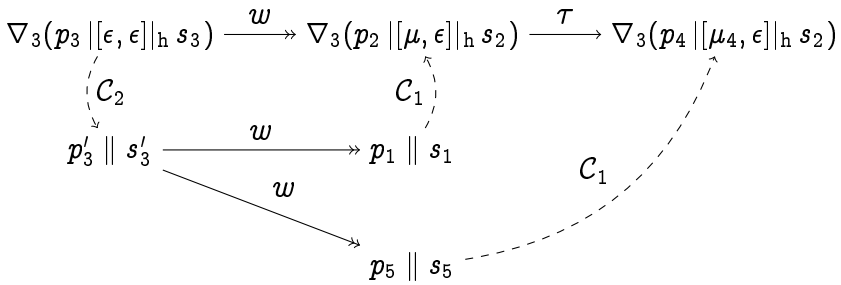


FIGURE B.5: Case 2(d)iB in hindsight.

C. Let $\mu = n.\mu_4$, $\nu \neq \epsilon$. Inapplicable due to half-duplex mechanism.

ii. Application of Rule 14. Similar to the previous case. □

Appendix C

Proofs of Theorem 6.5

Theorem C.1. *Let p, s be concrete processes. If the synchronous system $p \parallel s$ is well-posed, input-deterministic modulo $\underline{\leftrightarrow}^{\sqcup}_{\mathbf{b}}$, and strong E-independent modulo $\underline{\leftrightarrow}^{\sqcup}_{\mathbf{b}}$, then $p \parallel s \underline{\leftrightarrow}^{\sqcup}_{\mathbf{b}} \nabla_3(p \,|[\epsilon,\epsilon]|_{\mathcal{I}}\, s)$.*

Proof. Define a binary relation $\mathcal{B}$ in the following way:

$$\begin{aligned}\mathcal{B} = \Big\{ & (p_1 \parallel s_1, \nabla_3(p_2 \,|[\mu,\nu]|_{\mathcal{I}}\, s_2)) \mid \\ & p_1 \parallel s_1 \in \mathfrak{R}(p \parallel s) \wedge \nabla_3(p_2 \,|[\mu,\nu]|_{\mathcal{I}}\, s_2) \in \mathfrak{R}(\nabla_3(p \,|[\epsilon,\epsilon]|_{\mathcal{I}}\, s)) \wedge \\ & \exists p_2', s_2', q, q', u, v. \Big[u \in (M_s \cup E_s)^* \wedge v \in (M_p \cup E_p)^* \wedge \bar{u} = \mu \wedge \\ & \quad \bar{v} = \nu \wedge p_2 \parallel s_2' \in \mathfrak{R}(p \parallel s) \wedge p_2' \parallel s_2 \in \mathfrak{R}(p \parallel s) \wedge \\ & \quad p_2 \parallel s_2' \xrightarrow{u} \!\!\!\!\rightarrow\; q \underline{\leftrightarrow}^{\sqcup} p_1 \parallel s_1 \underline{\leftrightarrow}^{\sqcup} q' \xleftarrow{v} \!\!\!\!\rightarrow\; p_2' \parallel s_2 \Big] \Big\},\end{aligned}$$

Next, we show that $\mathcal{B}'$ is a branching bisimulation relation.

One can easily verify that if $(p_1 \parallel s_1, \nabla_3(p_2 \,|[\mu,\nu]|_{\mathcal{I}}\, s_2)) \in \mathcal{B}$, then every transition originating from the state $p_1 \parallel s_1$ is simulated by the state $\nabla_3(p_2 \,|[\mu,\nu]|_{\mathcal{I}}\, s_2)$ because the transitions $p_2 \parallel s_2' \xrightarrow{u} \!\!\!\!\rightarrow\; q, p_2' \parallel s_2 \xrightarrow{v} \!\!\!\!\rightarrow\; q'$ in the construction of $\mathcal{B}$ allows the processes p_2, s_2 to read the contents of their input queues such that an asynchronous state branching bisimilar to $p_1 \parallel s_1$ is reached.

The proof for the other direction is an intricate one and we distinguish the following cases.

1. Let $\nabla_3(p_2 \,|[\mu,\nu]|_{\mathcal{I}}\, s_2) \xrightarrow{\alpha} \nabla_3(p_4 \,|[\mu',\nu']|_{\mathcal{I}}\, s_4)$, $\alpha \in E_s$ (the case when $\alpha \in E_p$ is symmetric), and $(p_1 \parallel s_1, \nabla_3(p_2 \,|[\mu,\nu]|_{\mathcal{I}}\, s_2)) \in \mathcal{B}$. Then, by semantics we have $p_2 = p_4, s_2 \xrightarrow{e} s_4, \mu' = \mu, \nu' = \nu$. Since $(p_1 \parallel$

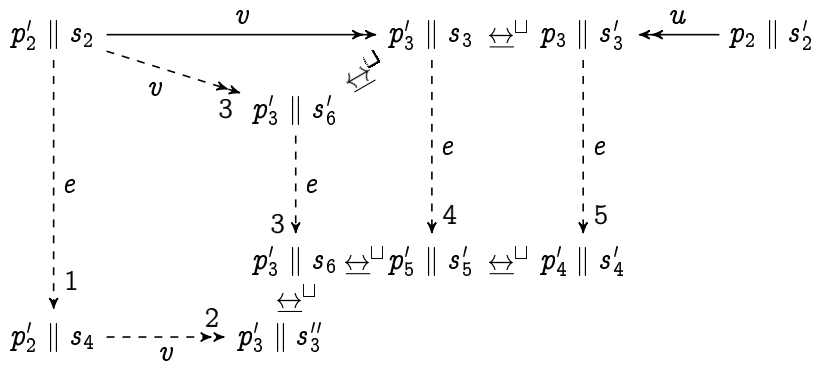


FIGURE C.1: Case 1 of Theorem C.1.

$s_1, \nabla_3(p_2 \,|[\mu,\nu]|_{\mathcal{I}}\, s_2)) \in \mathcal{B}$, so from the construction of $\mathcal{B}$ we get the solid transitions depicted in Figure C.1. Using the transition $s_2 \xrightarrow{e} s_4$ we get the dashed transition (1) in Figure C.1.

Consider the transition $p_2' \parallel s_2 \xrightarrow{v\!\!\rightarrow} p_3' \parallel s_3$ then from Proposition 2.11 we have $p_2' \xrightarrow{!v\!\!\rightarrow} p_3'$. By generalised well-posedness we get the dashed transition (2) in Figure C.1. By applying Definition 4.10 we get the two dashed transitions labeled as (3) in Figure C.1. And, from input-determinism under concreteness assumption, we get $p_3' \parallel s_6' \leftrightarrow^{\sqcup} p_3' \parallel s_3$ (see Figure C.1).

Furthermore, from the conditions of strong bisimulation we get the remaining dashed transitions (4) and (5). And, from the construction of $\mathcal{B}$ we have $p_3 \parallel s_3' \leftrightarrow^{\sqcup} p_1 \parallel s_1$. Also, from conditions of strong bisimulation we get there exists p_1', s_1' such that $p_1 \parallel s_1 \xrightarrow{e} p_1' \parallel s_1'$ and $p_1' \parallel s_1' \leftrightarrow^{\sqcup} p_4' \parallel s_4'$. Finally, using the transitions $p_2 \parallel s_2' \xrightarrow{u.e\!\!\rightarrow} p_4' \parallel s_4'$, $p_2' \parallel s_4 \xrightarrow{v\!\!\rightarrow} p_3' \parallel s_3''$ and the fact $p_3' \parallel s_3'' \leftrightarrow^{\sqcup} p_4' \parallel s_4' \leftrightarrow^{\sqcup} p_1' \parallel s_1'$ in the construction of $\mathcal{B}$ we conclude that $(p_1' \parallel s_1', \nabla_3(p_2 \,|[\mu,\nu]|_{\mathcal{I}}\, s_4)) \in \mathcal{B}$.

2. Let $\nabla_3(p_2 \,|[\mu,\nu]|_{\mathcal{I}}\, s_2) \xrightarrow{\alpha} \nabla_3(p_4 \,|[\mu',\nu']|_{\mathcal{I}}\, s_4)$, $\alpha \in M_s$ (the case when $\alpha \in M_p$ is symmetric), and $(p_1 \parallel s_1, \nabla_3(p_2 \,|[\mu,\nu]|_{\mathcal{I}}\, s_2)) \in \mathcal{B}$. Then, $p_2 = p_4, \mu' = \mu.n, \nu' = \nu, s_2 \xrightarrow{!n} s_4$, where $\alpha = n$, for $n \in M_s$. The remainder of the proof is similar to the previous case; except use Lemma 6.4 instead of applying Definition 4.10.

3. Let $\nabla_3(p_2 \,|[\mu,\nu]|_{\mathcal{I}}\, s_2) \xrightarrow{\tau} \nabla_3(p_4 \,|[\mu',\nu']|_{\mathcal{I}}\, s_4)$, $(p_1 \parallel s_1, \nabla_3(p_2 \,|[\mu,\nu]|_{\mathcal{I}}\, s_2)) \in \mathcal{B}$. Since the communicating components are concrete, so the above transition is due to the removal of an element either, from μ or ν. Thus,

   (a) Either, an element is remove from μ. Then, $p_2 \xrightarrow{?n} p_4, \mu = n.\mu', \nu = \nu', s_2 = s_4$, for some $n \in M_s$. Since $(p_1 \parallel s_1, \nabla_3(p_2 \,|[\mu,\nu]|_{\mathcal{I}}\, s_2)) \in \mathcal{B}$, so from the construction of $\mathcal{B}$ we have the transitions $p_2 \parallel s_2' \xrightarrow{u\!\!\rightarrow}$

$p_3 \parallel s_3'$ and $p_2' \parallel s_2 \xrightarrow{v} p_3' \parallel s_3$. But, $\bar{u} = \mu = n.\mu'$, we can then decompose the transition $p_2 \parallel s_2' \xrightarrow{u} p_3 \parallel s_3'$ as shown by the solid lines in Figure C.2, where $\bar{u}_1 = \epsilon$, and $\bar{u}_2 = \mu'$.

Also, $p_2 \xrightarrow{?n} p_4$; thus, we get the dashed transition (1) in Figure C.2. Since the given synchronous system is well-posed, so from generalised well-posedness (Lemma 3.7) we get the dashed transition (2) in Figure C.2. By input-determinism we have $p_5 \parallel s_3' \underline{\leftrightarrow}^{\sqcup} p_3 \parallel s_3'$. Finally, by using the transitions $p_4 \parallel s_4'' \xrightarrow{u_2} p_5 \parallel s_3'$, $p_2' \parallel s_2 \xrightarrow{v} p_3' \parallel s_3$ and the facts $\bar{u}_2 = \mu'$ and $p_5 \parallel s_3' \underline{\leftrightarrow}^{\sqcup} p_3' \parallel s_3 \underline{\leftrightarrow}^{\sqcup} p_1 \parallel s_1$ in the construction of $\mathcal{B}$ we conclude that $(p_1 \parallel s_1, \nabla_3(p_4 \,|[\mu', \nu]|_{\mathcal{I}}\, s_2)) \in \mathcal{B}$.

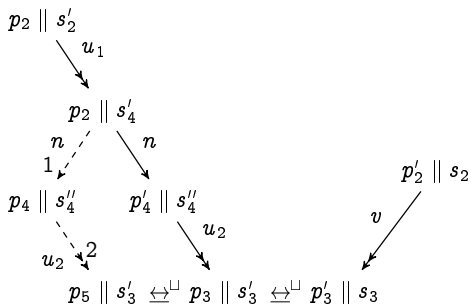


FIGURE C.2: Case 3 of Theorem C.1.

(b) Or, an element is remove from ν. Symmetric to the previous case.

4. Let $\nabla_3(p_2 \,|[\mu, \nu]|_{\mathcal{I}}\, s_2)\sqcup$ and $(p_1 \parallel s_1, \nabla_3(p_2 \,|[\mu, \nu]|_{\mathcal{I}}\, s_2)) \in \mathcal{B}$. Trivial! □

Bibliography

[1] Åkesson, K.; Fabian, M., and Flordal, H. *Supremica in a nutshell*. Chalmers University of Technology, SE-412 96 Göteborg, Sweden, 2007.

[2] Alur, R. Formal verification of hybrid systems. In *Proceedings of the 9th ACM international conference on Embedded software*, EMSOFT '11, pages 273–278, New York, NY, USA, 2011.

[3] Alur, R.; Grosu, R.; Lee, I., and Sokolsky, O. Compositional modeling and refinement for hierarchical hybrid systems. *Journal of Logic and Algebraic Programming*, 68(1-2):105 – 128, 2006.

[4] Baeten, J. C. M. and Bergstra, J. A. Mode Transfer in Process Algebra. Computing science reports ISSN 0926-4515, Eindhoven university of technology, Eindhoven, The Netherlands, 2000.

[5] Baeten, J. C. M.; Middelburg, C. A., and Middelburg, K. *Process Algebra with Timing*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2002. ISBN 354043447X.

[6] Baeten, J. C. M.; Basten, T., and Reniers, M. *Process Algebra: Equational Theories of Communicating Processes*. Cambridge University Press, 2009.

[7] Baeten, J. C. M.; van Beek, D.A.; Hendriks, D.; Hofkamp, A. T.; Nadales Agut, D. E.; Rooda, J. E., and Schiffelers, R. R. H. Report describing the extended CIF functionality. http://cms.multiform.bci.tu-dortmund.de/images/stories/multiform/deliverables/multiform_d112.pdf, 2010.

[8] Baeten, J. C. M.; van Beek, B.; van Hulst, A., and Markovski, J. A process algebra for supervisory coordination. In Aceto, L. and Mousavi, M. R., editors, *PACO*, volume 60 of *EPTCS*, pages 36–55, 2011.

[9] Baeten, J.C.M. A brief history of process algebra. *Theoretical Computer Science*, 335(2-3):131 – 146, 2005. ISSN 0304-3975. doi: 10.1016/j.tcs.2004.07.036.

[10] Baeten, J.C.M.; Beek, D.A. v.; Luttik, S.P.; Markovski, J., and Rooda, J.E. Partial bisimulation. SE Report 2010-04, Eindhoven University of Technology, Systems Engineering Group, Department of Mechanical Engineering, Eindhoven, The Netherlands, 2010.

[11] Baier, C. and Katoen, J. P. *Principles of Model Checking*. The MIT Press, 2008.

[12] Baier, C.; Bertrand, N., and Schnoebelen, Ph. Symbolic verification of communicating systems with probabilistic message losses: liveness and fairness. In *Proceedings of the 26th IFIP WG 6.1 international conference on Formal Techniques for Networked and Distributed Systems*, FORTE'06, pages 212–227, Berlin, Heidelberg, 2006. Springer-Verlag.

[13] Balemi, S. *Control of Discrete Event Systems: Theory And Application*. PhD thesis, Swiss Federal Institute of Technology, Automatic Control Laboratory, ETH Zurich, May 1992.

[14] Beek, D. A. van; Reniers, M. A.; Schiffelers, R. R. H., and Rooda, J. E. Foundations of an interchange format for hybrid systems. In Bemporad, A.; Bicchi, A., and Butazzo, G., editors, *Hybrid Systems: Computation and Control, 10th International Workshop*, volume 4416 of *Lecture Notes in Computer Science*, pages 587–600, Pisa, 2007. Springer-Verlag.

[15] Beek, D. A. van; Collins, P.; Nadales, D. E.; Rooda, J.E., and Schiffelers, R. R. H. New concepts in the abstract format of the compositional interchange format. In Giua, A.; Mahuela, C.; Silva, M., and Zaytoon, J., editors, *3rd IFAC Conference on Analysis and Design of Hybrid Systems*, pages 250–255, Zaragoza, Spain, 2009.

[16] Beek, D.A. van; Reniers, M. A.; Schiffelers, R. R. H., and Rooda, J. E. Foundations of a compositional interchange format for hybrid systems. SE Report 2006-05, Eindhoven University of Technology, Systems Engineering Group, Department of Mechanical Engineering, Eindhoven, The Netherlands, 2006.

[17] Beohar, H. and Cuijpers, P. J. L. Desynchronizability of (partial) synchronous closed loop systems. *Scientific Annals of Computer Science*, 21(1):5–38, 2011.

[18] Beohar, H. and Cuijpers, P. J. L. Avoiding diamonds in desynchronisation. Accepted in proceedings of Formal Aspects of Compoent Software (FACS), September 2012.

[19] Beohar, H. and Cuijpers, P.J.L. A theory of desynchronisable closed loop systems. In *Proceedings of Interaction and Concurrency Experience (ICE'10)*, 2010.

[20] Beohar, H.; Nadales Agut, D. E.; van Beek, D. A., and Cuijpers, P. J. L. Hierarchical states in the Compositional Interchange Format. In Aceto, L. and Sobocinski, P., editors, *Proceedings of the 7th Workshop on Structural Operational Semantics, SOS 2010.*, volume 32 of *EPTCS*, pages 42–56, 2010.

[21] Boehm, B. and Basili, R. V. Software defect reduction top 10 list. *Computer*, 34(1):135–137, January 2001.

[22] Brand, D. and Zafiropulo, P. On communicating finite-state machines. *J. ACM*, 30:323–342, April 1983. ISSN 0004-5411.

[23] Braspenning, N. C. W. M. *Model-based Integration and Testing of High-tech Multidisciplinary Systems*. PhD thesis, Eindhoven University of Technology, Eindhoven, The Netherlands, 2008.

[24] Cécé, G. and Finkel, A. Verification of programs with half-duplex communication. *Inf. Comput.*, 202:166–190, November 2005.

[25] Cuijpers, P. J. L. and Reniers, M. A. Hybrid process algebra. *Journal of Logic and Algebraic Programming*, 62(2):191–245, 2005.

[26] Cuijpers, P. J. L.; Reniers, M. A., and Heemels, W. P. M. H. Hybrid transition systems. Technical Report CS-Report 02-12, Eindhoven University of Technology, Department of Computer Science, The Netherlands, 2002.

[27] David, A. *Hierarchical Modeling and Analysis of Timed Systems*. PhD thesis, Uppsala University, November 2003.

[28] De Nicola, R. and Vaandrager, F. Three logics for branching bisimulation. *J. ACM*, 42:458–487, March 1995. ISSN 0004-5411.

[29] Deshpande, A.; Göllü, A., and Varaiya, P. Shift: A formalism and a programming language for dynamic networks of hybrid automata. In *Hybrid Systems IV*, pages 113–133, London, UK, UK, 1997. Springer-Verlag. ISBN 3-540-63358-8.

[30] Eker, J.; Janneck, J. W.; Lee, E. A.; Liu, J.; Liu, X.; Ludvig, J.; Neuendorffer, S.; Sachs, S., and Xiong, Y. Taming heterogeneity - the Ptolemy approach. *Proceedings of the IEEE*, 91(1):127–144, January 2003.

[31] Engelfriet, J. Determinancy $\rightarrow$ (observation equivalence = trace equivalence). *Theoretical Computer Science*, 36(0):21 – 25, 1985.

[32] Engels, T. P. CIF to CIF model transformations. Master's thesis, Eindhoven university of technology, System engineering group, Dept. of Mechanical Engineering, 2010.

[33] Fabian, M. and Hellgren, A. PLC-based implementation of supervisory control for discrete event systems. In *Proceedings of the 37th IEEE Conference on Decision and Control, 1998.*, volume 3, pages 3305–3310, 1998.

[34] Fabian, M. and Lennartson, B. On non-deterministic supervisory control. In *Proceedings of the 35th IEEE Decision and Control*, volume 2, pages 2213 –2218, December 1996.

[35] Fahrenberg, U.; Legay, A., and Thrane, C. The Quantitative Linear-Time–Branching-Time Spectrum. In Chakraborty, S. and Kumar, A., editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2011)*, volume 13 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 103–114, Dagstuhl, Germany, 2011. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. ISBN 978-3-939897-34-7.

[36] Fischer, C. and Janssen, W. Synchronous development of asynchonous systems. In *Proceedings of the 7th International Conference on Concurrency Theory*, CONCUR '96, pages 735–750, London, UK, UK, 1996. Springer-Verlag.

[37] Fonteijn, J. AND/OR superstate refinement in Hierarchical Compositional Interchange Format. SE 420640, Eindhoven University of Technology, System Engieering Group, Department of Mechanical Engineering, Eindhoven, March 2011.

[38] Forschelen, S. T. J. Supervisory control of theme park vehicles. Master's thesis, Eindhoven University of Technology, System Engineering Group, Dept. of Mechanical Engineering, 2010.

[39] Frehse, G.; Le Guernic, C.; Donzé, A.; Cotton, S.; Ray, R.; Lebeltel, O.; Ripado, R.; Girard, A.; Dang, T., and Maler, O. Spaceex: Scalable verification of hybrid systems. In Gopalakrishnan, G. and Qadeer, S., editors, *Proc. 23rd International Conference on Computer Aided Verification (CAV)*, LNCS. Springer, 2011.

[40] Ghafari, N.; Gurfinkel, A.; Klarlund, N., and Trefler, R. Reachability problems in piecewise fifo systems. *ACM Trans. Comput. Logic*, 13(1): 1–33, January 2012.

[41] Gorla, D. Comparing communication primitives via their relative expressive power. *Inf. Comput.*, 206:931–952, August 2008. ISSN 0890-5401.

[42] Gouda, M. G. and Yu, Y. T. Protocol validation by maximal progress state exploration. Computer science report TR-211, University of Texas at Austin, Austin, TX, USA, 1982.

[43] Groote, J. F. Implementation of events in LOTOS-specifications. Master's thesis, Technical Report 009/88EN, Philips CFT, Eindhoven, 1988.

[44] Groote, J. F. and Sellink, M. P. A. Confluence for process verification. *Theor. Comput. Sci.*, 170(1-2):47–81, December 1996.

[45] Groote, J. F.; Mathijssen, A.; Reniers, M.; Usenko, Y., and van Weerdenburg, M. The formal specification language mCRL2. In *MMOSS'06*, 2006.

[46] Grosu, R. and Stauner, T. Modular and visual specification of hybrid systems: An introduction to hycharts. *Form. Methods Syst. Des.*, 21 (1):5–38, July 2002.

[47] Harel, D. Statecharts: A visual formalism for complex systems. *Sci. Comput. Program.*, 8(3):231–274, June 1987.

[48] Henzinger, T. A. The theory of hybrid automata. In *Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science*, LICS '96, Washington, DC, USA, 1996. IEEE Computer Society.

[49] Henzinger, T.A. Two challenges in embedded systems design: Predictability and robustness. *Philosophical Transactions of the Royal Society A*, 366:3727–3736, 2008.

[50] Hoare, C. A. R. *Communicating sequential processes*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1985. ISBN 0-13-153271-5.

[51] Kouzapas, D.; Yoshida, N., and Honda, K. On asynchronous session semantics. In *Proceedings of the joint 13th IFIP WG 6.1 and 30th IFIP WG 6.1 international conference on Formal techniques for distributed systems*, FMOODS'11/FORTE'11, pages 228–243, Berlin, Heidelberg, 2011. Springer-Verlag. ISBN 978-3-642-21460-8.

[52] Kumar, R. and Shayman, M. A. Non-blocking Supervisory Control of Nondeterministic Systems via Prioritized Synchronization. *IEEE Transactions on Automatic Control*, 41(8):1160–1175, 1996.

[53] Larsen, K. G.; Pettersson, P., and Yi, W. Uppaal in a nutshell. *International Journal on Software Tools for Technology Transfer (STTT)*, 1:134–152, 1997. ISSN 1433-2779.

[54] Latella, D.; Majzik, I., and Massink, M. Towards a Formal Operational Semantics of UML Statechart Diagrams. In *Proceedings of the IFIP TC6/WG6.1 Third International Conference on Formal Methods for Open Object-Based Distributed Systems (FMOODS)*, Deventer, The Netherlands, 1999. Kluwer, B.V.

[55] Lozes, É. and Villard, J. Reliable contracts for unreliable half-duplex communications. In Carbone, M. and Petit, J-M., editors, *International Workshop on Web Services and Formal Methods*, Lecture Notes in Computer Science. Springer, 2011.

[56] Lynch, N.; Segala, R., and Vaandrager, F. Hybrid I/O Automata Revisited. In *Proceedings Fourth International Workshop on Hybrid Systems: Computation and Control (HSCC'01*, pages 403–417. Springer-Verlag, 2001.

[57] Markovski, J. A synthesis-based framework for systems engineering. Presented in 1st Workshop on Synthesis (SYNT), 2012.

[58] Marwedel, P. *Embedded System Design*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[59] Mateescu, R. Specification and analysis of asynchronous systems using cadp. In *Modeling and Verification of Real-Time Systems*, pages 141–169. ISTE, 2010.

[60] MathWorks, . StateChart - Stateflow - Simulink. `http://www.mathworks.nl/products/stateflow`.

[61] Mikk, E.; Lakhnech, Y., and Siegel, M. Hierarchical automata as model for statecharts. In *Proceedings of the Third Asian Computing Science Conference on Advances in Computing Science*, ASIAN '97, pages 181–196, London, UK, 1997. Springer-Verlag.

[62] Mousavi, M. R.; Reniers, M. A., and Groote, J. F. Notions of bisimulation and congruence formats for SOS with data. *Information and Computation*, 200(1):107–147, 2005.

[63] Nadales Agut, D. E. *A Compositional Interchange Format for Hybrid Systems: Design and Implementation*. PhD thesis, Eindhoven univsersity of technology, In preparation, 2012.

[64] Nadales Agut, D. E.; Beek, D. A. v.; Beohar, H.; Cuijpers, P. J. L., and Fonteijn, J. The Hierarchical Compositional Interchange Format. In Aichernig, B.; de Boer, F., and Bonsangue, M., editors, *Formal Methods for Components and Objects*, volume 6957 of *Lecture Notes in Computer Science*, pages 316–335. Springer Berlin / Heidelberg, 2012.

[65] Overkamp, A. Supervisory Control Using Partial Failure Semantics and Partial Specifications. *IEEE Trans. on Automatic Control*, 42(4):498–510, 1997.

[66] Park, D. Concurrency and automata on infinite sequences. In *Proceedings of the 5th GI-Conference on Theoretical Computer Science*, pages 167–183, London, UK, 1981. Springer-Verlag.

[67] Peters, K.; Schicke, J.-W., and Nestmann, U. Synchrony vs causality in asynchronous pi-calculus. In Luttik, S. P. and Valencia, F., editors, *18th International Workshop on Expressiveness in Concurrency, EXPRESS*, volume 64 of *EPTCS*, pages 89–103, 2011.

[68] Plotkin, G. D. A Structural Approach to Operational Semantics. Technical Report DAIMI FN-19, Computer science department, University of Aarhus, 1981. Also published in: Journal of Logic and Algebraic Programming, 60-61:17-140, 2004.

[69] Ramadge, P. J. and Wonham, W. M. Supervisory control of a class of discrete event processes. *SIAM Journal on Control and Optimization*, 25(1):206–230, 1987.

[70] Reynolds, John C. *Theories of programming languages*. Cambridge University Press, New York, NY, USA, 1999. ISBN 0-521-59414-6.

[71] Schicke, J.-W.; Peters, K., and Goltz, U. Synchrony vs causality in asynchronous petri nets. In Luttik, S. P. and Valencia, F., editors, *18th International Workshop on Expressiveness in Concurrency, EXPRESS*, volume 64 of *EPTCS*, pages 119–131, 2011.

[72] Theunissen, R. J. M.; Petreczky, M.; Schiffelers, R. R. H.; Beek, D. A. van, and Rooda, J. E. Application of supervisory control synthesis to MRI scanners: improving evolvability. SE Report 2010-06, System Engineering Group, Department of Mechanical Engineering, Eindhoven university of technology, Eindhoven, 2010.

[73] Udding, J. T. *Classification and Composition of Delay-Insensitive Circuits*. PhD thesis, Eindhoven University of Technology, Eindhoven, 1984.

[74] Uselton, Andrew E. and Smolka, Scott A. State Refinement in Process Algebra. Technical report, Stony Brook university, NY, 1993.

[75] van de Mortel-Fronczak, J.M. and Su, Rong. Advanced supervisory control - 4k460. `http://se.wtb.tue.nl/sewiki/\_media/4k420/pusherlift-overview.pdf`. Dept. of Mechanical Engineering, Eindhoven University of Technology.

[76] van Glabbeek, R. J. The linear time - branching time spectrum ii. In *Proceedings of the 4th International Conference on Concurrency Theory*, CONCUR '93, pages 66–81, London, UK, 1993. Springer-Verlag.

[77] van Glabbeek, R. J. and Weijland, W. P. Branching time and abstraction in bisimulation semantics. *J. ACM*, 43(3):555–600, May 1996.

[78] Voorhoeve, M. and Mauw, S. Impossible futures and determinism. *Inf. Process. Lett.*, 80(1):51–58, October 2001.

[79] Wonham, W. M. Supervisory control of discrete-event systems. Monograph ECE 1636F/1637S, University of Toronto, Dept. of Electrical & Computer Engineering, 2008.

[80] Xu, S. and Kumar, R. Asynchronous implementation of synchronous discrete event control. In *Proceedings of the 9th International Workshop on Discrete Event Systems, WODES 2008.*, pages 181–186, May 2008.

Samenvatting

Verfijning van Communicatie en Toestanden in Modellen van Embedded Systemen

Dit proefschrift behandeld twee aspecten van het model-gebaseerd ontwerpen van embedded systemen, namelijk: verfijning van communicatie en verfijning van toestandsmodellen.

Het deel over verfijning van communicatie behandelt de implementatie van een synchroon ontwerp met behulp van asynchrone primitieven, op zodanige manier dat de twee systemen equivalent zijn in hun gedrag. Het resultaat hiervan is dat de correctheid van een asynchrone implementatie gegarandeerd kan worden door eerst de correctheid van het synchrone ontwerp vast te stellen. Dit laatste kost in het algemeen minder rekenkracht. Het doel van het onderzoek was om condities te vinden onder welke het toevoegen van buffers het gedrag van een gegeven synchroon ontwerp niet zou veranderen. We laten zien dat het mogelijk is om betere desynchronisatie condities te vinden (zelfs voor fijnere equivalenties dan vertakkende bisimulatie) door eigenschappen van het communicatie protocol aan te passen. Dit staat in contrast met eerder werk, waar de aandacht uitsluitend uitging naar het beperken van het gedrag van de communicerende componenten zelf.

Het deel over verfijning van toestanden behandelt de stapsgewijze ontwikkeling van hybride systeemmodellen. Een dergelijk concept was nog niet aanwezig in het "Compositional Interchange Format (CIF)", een modelleer taal voor embedded systemen, gebaseerd op hybride automaten in combinatie met een aantal proces-algebraïsche operatoren. Het doel van het onderzoek was in dit geval om een compositionele operationele semantiek te ontwikkelen voor CIF uitgebreid met hierarchie (HCIF). We laten zien dat, door slechts te verwijzen naar de transitie-systemen van substructuren (in tegenstelling tot hun syntactische representatie), de semantiek van HCIF operatoren vrijwel onveranderd kan blijven ten opzichte van hun oorspronkelijke CIF variant. Bovendien wordt er een methode gepresenteerd om de hiërarchie in HCIF modellen te elimineren. Daardoor kunnen bestaande simulatie-pakketten en transformatie-algoritmen voor andere getimede en hybride talen hergebruikt worden voor HCIF modellen waar de hiërarchie uit geëlimineerd is.

Curriculum Vitae

Harsh Beohar was born on the 18th of April 1984 in Jabalpur, India. He studied computer science at the NMAM Institute of Technology, Nitte, India and obtained the degree of Bachelor in Computer Science and Engineering in June, 2006. In August 2008 he obtained a M.Tech. degree in Software Engineering from Manipal Institute of Technology, India and a M.Sc. degree in Computer Science from Eindhoven University of Technology, The Netherlands. In October 2008 he became a Ph.D. student at the Formal Methods Group (now Formal System Analysis Group), Department of Mathematics and Computer Science, Eindhoven University of Technology, The Netherlands.

Titles in the IPA Dissertation Series since 2007

H.A. de Jong. *Flexible Heterogeneous Software Systems*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2007-01

N.K. Kavaldjiev. *A run-time reconfigurable Network-on-Chip for streaming DSP applications*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2007-02

M. van Veelen. *Considerations on Modeling for Early Detection of Abnormalities in Locally Autonomous Distributed Systems*. Faculty of Mathematics and Computing Sciences, RUG. 2007-03

T.D. Vu. *Semantics and Applications of Process and Program Algebra*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2007-04

L. Brandán Briones. *Theories for Model-based Testing: Real-time and Coverage*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2007-05

I. Loeb. *Natural Deduction: Sharing by Presentation*. Faculty of Science, Mathematics and Computer Science, RU. 2007-06

M.W.A. Streppel. *Multifunctional Geometric Data Structures*. Faculty of Mathematics and Computer Science, TU/e. 2007-07

N. Trčka. *Silent Steps in Transition Systems and Markov Chains*. Faculty of Mathematics and Computer Science, TU/e. 2007-08

R. Brinkman. *Searching in encrypted data*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2007-09

A. van Weelden. *Putting types to good use*. Faculty of Science, Mathematics and Computer Science, RU. 2007-10

J.A.R. Noppen. *Imperfect Information in Software Development Processes*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2007-11

R. Boumen. *Integration and Test plans for Complex Manufacturing Systems*. Faculty of Mechanical Engineering, TU/e. 2007-12

A.J. Wijs. *What to do Next?: Analysing and Optimising System Behaviour in Time*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2007-13

C.F.J. Lange. *Assessing and Improving the Quality of Modeling: A Series of Empirical Studies about the UML*. Faculty of Mathematics and Computer Science, TU/e. 2007-14

T. van der Storm. *Component-based Configuration, Integration and Delivery*. Faculty of Natural Sciences, Mathematics, and Computer Science,UvA. 2007-15

B.S. Graaf. *Model-Driven Evolution of Software Architectures.* Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2007-16

A.H.J. Mathijssen. *Logical Calculi for Reasoning with Binding.* Faculty of Mathematics and Computer Science, TU/e. 2007-17

D. Jarnikov. *QoS framework for Video Streaming in Home Networks.* Faculty of Mathematics and Computer Science, TU/e. 2007-18

M. A. Abam. *New Data Structures and Algorithms for Mobile Data.* Faculty of Mathematics and Computer Science, TU/e. 2007-19

W. Pieters. *La Volonté Machinale: Understanding the Electronic Voting Controversy.* Faculty of Science, Mathematics and Computer Science, RU. 2008-01

A.L. de Groot. *Practical Automaton Proofs in PVS.* Faculty of Science, Mathematics and Computer Science, RU. 2008-02

M. Bruntink. *Renovation of Idiomatic Crosscutting Concerns in Embedded Systems.* Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2008-03

A.M. Marin. *An Integrated System to Manage Crosscutting Concerns in Source Code.* Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2008-04

N.C.W.M. Braspenning. *Model-based Integration and Testing of High-tech Multi-disciplinary Systems.* Faculty of Mechanical Engineering, TU/e. 2008-05

M. Bravenboer. *Exercises in Free Syntax: Syntax Definition, Parsing, and Assimilation of Language Conglomerates.* Faculty of Science, UU. 2008-06

M. Torabi Dashti. *Keeping Fairness Alive: Design and Formal Verification of Optimistic Fair Exchange Protocols.* Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2008-07

I.S.M. de Jong. *Integration and Test Strategies for Complex Manufacturing Machines.* Faculty of Mechanical Engineering, TU/e. 2008-08

I. Hasuo. *Tracing Anonymity with Coalgebras.* Faculty of Science, Mathematics and Computer Science, RU. 2008-09

L.G.W.A. Cleophas. *Tree Algorithms: Two Taxonomies and a Toolkit.* Faculty of Mathematics and Computer Science, TU/e. 2008-10

I.S. Zapreev. *Model Checking Markov Chains: Techniques and Tools.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-11

M. Farshi. *A Theoretical and Experimental Study of Geometric Networks*. Faculty of Mathematics and Computer Science, TU/e. 2008-12

G. Gulesir. *Evolvable Behavior Specifications Using Context-Sensitive Wildcards*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-13

F.D. Garcia. *Formal and Computational Cryptography: Protocols, Hashes and Commitments*. Faculty of Science, Mathematics and Computer Science, RU. 2008-14

P. E. A. Dürr. *Resource-based Verification for Robust Composition of Aspects*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-15

E.M. Bortnik. *Formal Methods in Support of SMC Design*. Faculty of Mechanical Engineering, TU/e. 2008-16

R.H. Mak. *Design and Performance Analysis of Data-Independent Stream Processing Systems*. Faculty of Mathematics and Computer Science, TU/e. 2008-17

M. van der Horst. *Scalable Block Processing Algorithms*. Faculty of Mathematics and Computer Science, TU/e. 2008-18

C.M. Gray. *Algorithms for Fat Objects: Decompositions and Applications*. Faculty of Mathematics and Computer Science, TU/e. 2008-19

J.R. Calamé. *Testing Reactive Systems with Data - Enumerative Methods and Constraint Solving*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-20

E. Mumford. *Drawing Graphs for Cartographic Applications*. Faculty of Mathematics and Computer Science, TU/e. 2008-21

E.H. de Graaf. *Mining Semi-structured Data, Theoretical and Experimental Aspects of Pattern Evaluation*. Faculty of Mathematics and Natural Sciences, UL. 2008-22

R. Brijder. *Models of Natural Computation: Gene Assembly and Membrane Systems*. Faculty of Mathematics and Natural Sciences, UL. 2008-23

A. Koprowski. *Termination of Rewriting and Its Certification*. Faculty of Mathematics and Computer Science, TU/e. 2008-24

U. Khadim. *Process Algebras for Hybrid Systems: Comparison and Development*. Faculty of Mathematics and Computer Science, TU/e. 2008-25

J. Markovski. *Real and Stochastic Time in Process Algebras for Performance Evaluation*. Faculty of Mathematics and Computer Science, TU/e. 2008-26

H. Kastenberg. *Graph-Based Software Specification and Verification.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-27

I.R. Buhan. *Cryptographic Keys from Noisy Data Theory and Applications.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-28

R.S. Marin-Perianu. *Wireless Sensor Networks in Motion: Clustering Algorithms for Service Discovery and Provisioning.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-29

M.H.G. Verhoef. *Modeling and Validating Distributed Embedded Real-Time Control Systems.* Faculty of Science, Mathematics and Computer Science, RU. 2009-01

M. de Mol. *Reasoning about Functional Programs: Sparkle, a proof assistant for Clean.* Faculty of Science, Mathematics and Computer Science, RU. 2009-02

M. Lormans. *Managing Requirements Evolution.* Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2009-03

M.P.W.J. van Osch. *Automated Model-based Testing of Hybrid Systems.* Faculty of Mathematics and Computer Science, TU/e. 2009-04

H. Sozer. *Architecting Fault-Tolerant Software Systems.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2009-05

M.J. van Weerdenburg. *Efficient Rewriting Techniques.* Faculty of Mathematics and Computer Science, TU/e. 2009-06

H.H. Hansen. *Coalgebraic Modelling: Applications in Automata Theory and Modal Logic.* Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2009-07

A. Mesbah. *Analysis and Testing of Ajax-based Single-page Web Applications.* Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2009-08

A.L. Rodriguez Yakushev. *Towards Getting Generic Programming Ready for Prime Time.* Faculty of Science, UU. 2009-9

K.R. Olmos Joffré. *Strategies for Context Sensitive Program Transformation.* Faculty of Science, UU. 2009-10

J.A.G.M. van den Berg. *Reasoning about Java programs in PVS using JML.* Faculty of Science, Mathematics and Computer Science, RU. 2009-11

M.G. Khatib. *MEMS-Based Storage Devices. Integration in*

Energy-Constrained Mobile Systems. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2009-12

S.G.M. Cornelissen. *Evaluating Dynamic Analysis Techniques for Program Comprehension*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2009-13

D. Bolzoni. *Revisiting Anomaly-based Network Intrusion Detection Systems*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2009-14

H.L. Jonker. *Security Matters: Privacy in Voting and Fairness in Digital Exchange*. Faculty of Mathematics and Computer Science, TU/e. 2009-15

M.R. Czenko. *TuLiP - Reshaping Trust Management*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2009-16

T. Chen. *Clocks, Dice and Processes*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2009-17

C. Kaliszyk. *Correctness and Availability: Building Computer Algebra on top of Proof Assistants and making Proof Assistants available over the Web*. Faculty of Science, Mathematics and Computer Science, RU. 2009-18

R.S.S. O'Connor. *Incompleteness & Completeness: Formalizing Logic and Analysis in Type Theory*. Faculty of Science, Mathematics and Computer Science, RU. 2009-19

B. Ploeger. *Improved Verification Methods for Concurrent Systems*. Faculty of Mathematics and Computer Science, TU/e. 2009-20

T. Han. *Diagnosis, Synthesis and Analysis of Probabilistic Models*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2009-21

R. Li. *Mixed-Integer Evolution Strategies for Parameter Optimization and Their Applications to Medical Image Analysis*. Faculty of Mathematics and Natural Sciences, UL. 2009-22

J.H.P. Kwisthout. *The Computational Complexity of Probabilistic Networks*. Faculty of Science, UU. 2009-23

T.K. Cocx. *Algorithmic Tools for Data-Oriented Law Enforcement*. Faculty of Mathematics and Natural Sciences, UL. 2009-24

A.I. Baars. *Embedded Compilers*. Faculty of Science, UU. 2009-25

M.A.C. Dekker. *Flexible Access Control for Dynamic Collaborative Environments*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2009-26

J.F.J. Laros. *Metrics and Visualisation for Crime Analysis and Genomics*. Faculty of Mathematics and Natural Sciences, UL. 2009-27

C.J. Boogerd. *Focusing Automatic Code Inspections*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2010-01

M.R. Neuhäußer. *Model Checking Nondeterministic and Randomly Timed Systems*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2010-02

J. Endrullis. *Termination and Productivity*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2010-03

T. Staijen. *Graph-Based Specification and Verification for Aspect-Oriented Languages*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2010-04

Y. Wang. *Epistemic Modelling and Protocol Dynamics*. Faculty of Science, UvA. 2010-05

J.K. Berendsen. *Abstraction, Prices and Probability in Model Checking Timed Automata*. Faculty of Science, Mathematics and Computer Science, RU. 2010-06

A. Nugroho. *The Effects of UML Modeling on the Quality of Software*. Faculty of Mathematics and Natural Sciences, UL. 2010-07

A. Silva. *Kleene Coalgebra*. Faculty of Science, Mathematics and Computer Science, RU. 2010-08

J.S. de Bruin. *Service-Oriented Discovery of Knowledge - Foundations, Implementations and Applications*. Faculty of Mathematics and Natural Sciences, UL. 2010-09

D. Costa. *Formal Models for Component Connectors*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2010-10

M.M. Jaghoori. *Time at Your Service: Schedulability Analysis of Real-Time and Distributed Services*. Faculty of Mathematics and Natural Sciences, UL. 2010-11

R. Bakhshi. *Gossiping Models: Formal Analysis of Epidemic Protocols*. Faculty of Sciences, Department of Computer Science, VUA. 2011-01

B.J. Arnoldus. *An Illumination of the Template Enigma: Software Code Generation with Templates*. Faculty of Mathematics and Computer Science, TU/e. 2011-02

E. Zambon. *Towards Optimal IT Availability Planning: Methods and Tools*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2011-03

L. Astefanoaei. *An Executable Theory of Multi-Agent Systems Refinement*. Faculty of Mathematics and Natural Sciences, UL. 2011-04

J. Proença. *Synchronous coordination of distributed components*.

Faculty of Mathematics and Natural Sciences, UL. 2011-05

A. Moralı. *IT Architecture-Based Confidentiality Risk Assessment in Networks of Organizations*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2011-06

M. van der Bijl. *On changing models in Model-Based Testing*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2011-07

C. Krause. *Reconfigurable Component Connectors*. Faculty of Mathematics and Natural Sciences, UL. 2011-08

M.E. Andrés. *Quantitative Analysis of Information Leakage in Probabilistic and Nondeterministic Systems*. Faculty of Science, Mathematics and Computer Science, RU. 2011-09

M. Atif. *Formal Modeling and Verification of Distributed Failure Detectors*. Faculty of Mathematics and Computer Science, TU/e. 2011-10

P.J.A. van Tilburg. *From Computability to Executability – A process-theoretic view on automata theory*. Faculty of Mathematics and Computer Science, TU/e. 2011-11

Z. Protic. *Configuration management for models: Generic methods for model comparison and model co-evolution*. Faculty of Mathematics and Computer Science, TU/e. 2011-12

S. Georgievska. *Probability and Hiding in Concurrent Processes*. Faculty of Mathematics and Computer Science, TU/e. 2011-13

S. Malakuti. *Event Composition Model: Achieving Naturalness in Runtime Enforcement*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2011-14

M. Raffelsieper. *Cell Libraries and Verification*. Faculty of Mathematics and Computer Science, TU/e. 2011-15

C.P. Tsirogiannis. *Analysis of Flow and Visibility on Triangulated Terrains*. Faculty of Mathematics and Computer Science, TU/e. 2011-16

Y.-J. Moon. *Stochastic Models for Quality of Service of Component Connectors*. Faculty of Mathematics and Natural Sciences, UL. 2011-17

R. Middelkoop. *Capturing and Exploiting Abstract Views of States in OO Verification*. Faculty of Mathematics and Computer Science, TU/e. 2011-18

M.F. van Amstel. *Assessing and Improving the Quality of Model Transformations*. Faculty of Mathematics and Computer Science, TU/e. 2011-19

A.N. Tamalet. *Towards Correct Programs in Practice*. Faculty of

Science, Mathematics and Computer Science, RU. 2011-20

H.J.S. Basten. *Ambiguity Detection for Programming Language Grammars.* Faculty of Science, UvA. 2011-21

M. Izadi. *Model Checking of Component Connectors.* Faculty of Mathematics and Natural Sciences, UL. 2011-22

L.C.L. Kats. *Building Blocks for Language Workbenches.* Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2011-23

S. Kemper. *Modelling and Analysis of Real-Time Coordination Patterns.* Faculty of Mathematics and Natural Sciences, UL. 2011-24

J. Wang. *Spiking Neural P Systems.* Faculty of Mathematics and Natural Sciences, UL. 2011-25

A. Khosravi. *Optimal Geometric Data Structures.* Faculty of Mathematics and Computer Science, TU/e. 2012-01

A. Middelkoop. *Inference of Program Properties with Attribute Grammars, Revisited.* Faculty of Science, UU. 2012-02

Z. Hemel. *Methods and Techniques for the Design and Implementation of Domain-Specific Languages.* Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2012-03

T. Dimkov. *Alignment of Organizational Security Policies: Theory and Practice.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2012-04

S. Sedghi. *Towards Provably Secure Efficiently Searchable Encryption.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2012-05

F. Heidarian Dehkordi. *Studies on Verification of Wireless Sensor Networks and Abstraction Learning for System Inference.* Faculty of Science, Mathematics and Computer Science, RU. 2012-06

K. Verbeek. *Algorithms for Cartographic Visualization.* Faculty of Mathematics and Computer Science, TU/e. 2012-07

D.E. Nadales Agut. *A Compositional Interchange Format for Hybrid Systems: Design and Implementation.* Faculty of Mechanical Engineering, TU/e. 2012-08

H. Rahmani. *Analysis of Protein-Protein Interaction Networks by Means of Annotated Graph Mining Algorithms.* Faculty of Mathematics and Natural Sciences, UL. 2012-09

S.D. Vermolen. *Software Language Evolution.* Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2012-10

L.J.P. Engelen. *From Napkin Sketches to Reliable Software*. Faculty of Mathematics and Computer Science, TU/e. 2012-11

F.P.M. Stappers. *Bridging Formal Models – An Engineering Perspective*. Faculty of Mathematics and Computer Science, TU/e. 2012-12

W. Heijstek. *Software Architecture Design in Global and Model-Centric Software Development*. Faculty of Mathematics and Natural Sciences, UL. 2012-13

C. Kop. *Higher Order Termination*. Faculty of Sciences, Department of Computer Science, VUA. 2012-14

A. Osaiweran. *Formal Development of Control Software in the Medical Systems Domain*. Faculty of Mathematics and Computer Science, TU/e. 2012-15

W. Kuijper. *Compositional Synthesis of Safety Controllers*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2012-16

H. Beohar. *Refinement of Communication and States in Models of Embedded Systems*. Faculty of Mathematics and Computer Science, TU/e. 2013-01