Rephrasing interoperability in terms of connector behaviours

Marco Autili, Taolue Chen, Chris Chilton, Tingting Han, Lukas Holik, Paola Inverardi, Bengt Jonsson, Marta Kwiatkowska, Marco Martinucci, Hongyang Qu, et al.

CONNECT

Emergent Connectors for

Eternal Software Intensive Networked Systems

ICT FET IP Project

Deliverable D2.3

Rephrasing interoperability in terms of connector behaviours

SEVENTH FRAMEWORK PROGRAMME

http://www.connect-forever.eu

Inria · DOCOMO Euro-Labs · Imperial College · Lancaster University · Thales · Università dell'Aquila · technische universität dortmund · University of Oxford · Uppsala Universitet

Project Number	:	231167
Project Title	:	CONNECT – Emergent Connectors for Eternal Software Intensive Networked Systems
Deliverable Type	:	Report

Deliverable Number	:	D2.3
Title of Deliverable	:	Rephrasing interoperability in terms of connector behaviours
Nature of Deliverable	:	R
Dissemination Level	:	Public
Internal Version Number	:	1.0
Contractual Delivery Date	:	17 February 2012
Actual Delivery Date	:	17 February 2012
Contributing WPs	:	WP2
Editor(s)	:	Hongyang Qu, Chris Chilton
Author(s)	:	Marco Autili, Taolue Chen, Chris Chilton, Tingting Han, Lukáš Holík, Paola Inverardi, Bengt Jonsson, Marta Kwiatkowska, Marco Martinucci, Hongyang Qu, Massimo Tivoli
Reviewer(s)	:	Valérie Issarny, Ilaria Matteucci, Bernhard Steffen

Abstract

In this deliverable we document the progress of WP2 during the third year of the CONNECT *project.*

We supply details of a compositional specification theory we have been working on for modelling connectors and components. This supports the operations of parallel composition, for examining the structural behaviour of networked systems, logical conjunction, for supporting independent development of components, and quotient for incremental development in the form of component/connector synthesis. Our framework, a portion of which is conceptually similar to interface automata, is unique in that we've provided the first definition of quotient that can act on components exhibiting non-determinism, as well as the first definition of conjunction on this flavour of model. A refinement preorder corresponding to safe-substitutivity allows for the seamless interchanging of components and connectors without violating environmental assumptions. Our formulation of refinement also ensures that the framework enjoys strong algebraic properties. These are necessary for reasoning about components and connectors in a compositional manner.

Suitability of our specification theory is demonstrated by linking synthesis in WP3 with the quotienting operation in WP2. The technique relies on formulating a specification that does not allow communication mismatches to occur, and that also ensures the components to be connected make progress without deadlocking inopportunely. Such a mediator must also satisfy the constraints imposed by the system-wide ontology (i.e., ordering of actions and equivalence of actions). We demonstrate how this works in practice through a couple of examples.

An extension of the specification theory has also been documented in which synchronisation of actions facilitates the communication of data. The models of this framework are a cross between the Logic IOLTSs of our specification theory, and the register automata used by WP4. As for the original specification theory, this extended formalism supports parallel composition, conjunction and quotient. Through a number of examples, we demonstrate how the theory can be used for synthesising mediators of systems to be connected.

Quantitative assume-guarantee verification is also presented. The key achievement in the probabilistic setting is a complete assume-guarantee technique that can generate probabilistic assumptions automatically by a new L-style learning method. This will be exploited as part of a future extended specification theory capable of handling probabilistic behaviour and non-functional properties.*

Keyword List

Specification theory, conjunction, quotient, synthesis, assume-guarantee, data, quantitative verification, functional and non-functional requirements.

Document History

Version	Type of Change	Author(s)
0.1	initial version	Hongyang
0.2	first draft for Chapter 5	Hongyang
0.3	Chapter 2	Chris
0.4	part of the introduction and conclusion	Massimo
0.5	Chapter 3	Chris
0.6	Chapter 4	Bengt
0.7	Introduction and conclusion	All
0.8	Complete version for internal review	All
0.9	All chapters: incorporated comments from the internal interview	All
1.0	All chapters: revised for the final submission	Marta

Document Review

Date	Version	Reviewer	Comment
04 Feb 2012	0.8	Ilaria Matteucci	
06 Feb 2012	0.8	Bernhard Steffen	
17 Feb 2012	0.8	Valérie Issarny	

Table of Contents

List of Figures

List of Acronyms

AG : Assume-Guarantee

BIP : Behavior-Interaction-Priority

CCS : Calculus of Communicating Systems

CMC : Constraint Markov Chain

CSL : Continuous Stochastic Logic

CSP : Communicating Sequential Processes

CTIMC : Continuous-Time Interactive Markov Chain

CTMC : Continuous-Time Markov chain

CTMDP : Continuous-Time Markov Decision Process

DFA : Deterministic Finite Automaton

DRA : Deterministic Rabin Automata

DTIMC : Discrete-time interactive Markov Chain

DTMC : Discrete-Time Markov chain

EC : End Component

EMF : Eclipse Modeling Framework

IA : Interface Automata

IMC : Interactive Markov Chain

LP : Linear Programming

LTL : Linear Temporal Logic

M2T : Model-to-Text

MDD : Model Driven Development

MDP : Markov Decision Process

NS : Networked System

PA : Probabilistic Automata

PCTL : Probabilistic Computational Tree Logic

PIA : Probabilistic Interface Automata

QMO : Quantitative Multi-Objective

Ticc : Tool for Interface Compatibility and Composition

1 Introduction

The CONNECT project aims to develop a novel network infrastructure to allow heterogeneous networked systems to freely communicate with each other via on-the-fly synthesis of emergent connectors. The role of Work Package 2 (WP2) is to investigate the foundations and verification methods for composable connectors, so that support is provided for composition of networked systems, whilst enabling automated learning, reasoning and synthesis.

This document provides an overview of the work undertaken by WP2 in the third year. The work is organised into two streams, (i) *compositional theory of connector behaviours* and (ii) *compositional assume-guarantee verification for probabilistic automata*, with the view to integrate these together.

For (i), we have formulated a specification theory for modelling components and connectors. The framework admits a substitutive refinement preorder, and provides the operations of parallel composition, for examining the structural behaviour of systems, conjunction (or shared refinement), for independent development, and quotient to synthesise components, for supporting incremental development. An extension with data has been provided for this specification theory. Integration with synthesis in WP3 is demonstrated by linking the algorithmic approach to connector synthesis developed in that work package with the quotient operator of the specification theory.

For (ii), we continued our work on assume-guarantee verification for probabilistic automata, extending the multi-objective approach to allow probabilistic liveness and expectations, which will be integrated with the quantitative specification theory over the course of the next year.

This deliverable addresses part of the issues concerning the work within Task 2.3, as described in the *Description of Work* (DoW). These issues are related to the definition of techniques for interoperability checking and protocol matching even in the presence of non-functional properties. Since Task 2.3 will end with the project, the rest of the work in Task 2.3 is left as future work. We recall that this work concerns the use of negotiation for achieving interoperability while still satisfying specified QoS properties and the evaluation of the related offline and online verification techniques.

In this chapter, we first recall the role of WP2 in the project (Section 1.1), and then address the review recommendations (Section 1.2). Next, we explain the research challenges and achievements of the third year (Section 1.3), before providing an outline structure of the deliverable (Section 1.4).

1.1 The role of work package WP2

The role of WP2 is to investigate the foundations and verification methods for composable connectors, so that support is provided for composition of networked systems, whilst enabling automated learning, reasoning and synthesis. The expected outcomes are formalisms, methods and software tools that can be used for the specification, design and development of connectors, allowing for both functional and non-functional properties to be expressed and verified. WP2 thus provides the theoretical underpinning for the work carried out in the other workpackages, in the sense that connectors specified in WP2 can be instantiated in WP3 (synthesis), WP4 (learning) and WP5 (dependability analysis).

The remit of WP2 is to develop compositional specification and verification techniques into a situation where they can be successfully applied to the modelling and reasoning of connector behaviours in a compositional manner. To achieve this goal, WP2 is structured into the four tasks:

- **Task 2.1**. Capturing functional and non-functional connector behaviours. This task aims to guide the project by formalising the notions of connector and component, characterising the types of interaction and identifying a verification approach, capable of capturing non-functional properties.

- **Task 2.2**. Compositional connector operators. The main thrust here is to formulate a compositional modelling and reasoning framework for components and connectors.

- **Task 2.3**. Rephrasing interoperability in terms of connector behaviours. The aim is to formulate techniques for interoperability checking, in the presence of dynamic behaviours and non-functional properties.

- **Task 2.4**. Reasoning toolset. The focus here is on a quantitative verification framework for connectors and components, capable of handling dynamic scenarios and non-functional properties, which includes algorithms and prototype implementations.

We have organised the work into two parallel streams, respectively aimed at formulating (i) a *compositional theory of connector behaviours*, and (ii) *compositional assume-guarantee verification for probabilistic automata*. Such an approach facilitates progress across the full breadth of the tasks, while at the same time delivering reasoning methods and tools that are directly relevant to other workpackages, which rely on automata-like formalisms. The goal is to integrate the two streams together, which we are now beginning to do.

Work stream (i) is aimed at formulating a comprehensive framework for modelling, specifying, composing and reasoning about connector behaviours. The CONNECT consortium has selected a variant of interface automata as the underlying (stateful) model for interactive behaviours, which we refer to as Logic IOLTSs. A purely abstract specification notation consisting of traces is also provided, which is a stepping-stone to defining the substitutive trace-based refinement preorder. Our deviations from interface automata are motivated by the goal to provide a general theory with a range of operations, which can also support extensions with non-functional properties, and liveness. For each of the defined operators (parallel, conjunction, and quotient) we demonstrate congruence-like compositionality results, in addition to compositionality results linking the operational models to the abstract trace-based models. For maximum flexibility, we are concentrating on modelling arbitrary component behaviours, where connectors correspond to particular subclasses. To capture non-functional properties, we must generalise the theory to allow for quantitative behaviours. Such an extension will be coupled with the assume-guarantee framework for performing compositional verification.

For work stream (ii), we settled on probabilistic model checking as a means to formally analyze non-functional properties. In the deliverable D5.2 [25], we have demonstrated an application of these techniques to a CONNECT scenario. To support independent development of connectors, we are developing compositional assume-guarantee verification for probabilistic automata, which can support a range of non-functional properties. Compositional verification works by breaking a verification problem down into manageable sub-tasks, based on the structure of the system being analysed. In the *assume-guarantee* paradigm, in which properties (*guarantees*) of individual system components are verified under *assumptions* about their environment, desired properties of the combined system are then obtained by combining separate verification results using proof rules. We have adopted a compositional assume-guarantee framework for probabilistic automata based on the multi-objective approach. A broad range of non-functional properties can be expressed and verified in this framework.

The integration of the two streams will be achieved through integrating probabilistic automata and Logic IOLTSs, combined with an enhancement of the specification theory with assume-guarantee reasoning proof rules.

1.2 Review recommendations

- **The relation between the calculus developed in WP2 and the synthesis work in WP3 is unclear.**

  We can use the theory of WP2 to prove technical results about the CONNECTors generated by WP3.

  The framework proposed in WP2 supports modelling and reasoning about complex systems of components that interact with one another, whereas the work of WP3 is focused on generating actual mediators on-the-fly to support interoperability of networked systems. In WP2 we have used a directed model of communication, in which each interaction is under the control of one component, thus distinguishing fundamentally between input and output. This model is suitable for modelling asynchronous communication in distributed systems. WP3, on the other hand, has utilised a concurrent model whereby all components handshake on communications. Such an assumption simplifies, although certainly does not trivialise, the problem of generating mediators on-the-fly.

  Although the frameworks of WP2 and WP3 have different communication semantics, the models used by WP3 can be ported across to the WP2 framework by explicitly representing the handshaking. In Chapter 3 of this deliverable, we demonstrate a mapping from components modelled as LTSs

of WP3 (see Chapter 3 of [23]) to semantically equivalent models in the WP2 notation. As part of our work in WP2, we develop a quotienting operator for synthesising mediators of systems, where the systems can exhibit non-determinism. We prove that our mediators are free from communication mismatches, and moreover make progress so that the mediator guarantees meaningful communication takes place. Our synthesised mediator is also the most general. Through this, we can relate the models of WP3 to the technical results we have derived in WP2. As a result, we can show whether a mediator synthesised *à la* WP3 is the most general mediator, and check whether it is free from errors by using the theory built up in WP2. Thus in this deliverable, we demonstrate how the theory of WP2 can be used for justifying the practical aspects of WP3 concerned with mediator synthesis.

- **It is unclear how the compositionality addressed in WP2 fits with the compositionality potentially needed for connector synthesis.**

  We have introduced a quotienting operation in the specification theory, for linking the inherent compositionality results of WP2 (stated in Chapter 2) to the synthesis of mediators in WP3. Through assume-guarantee reasoning, we will infer properties of synthesised mediators from the properties of the components to be mediated; this is, to some extent, what we mean by compositionality. Utilising such compositional properties relies on the semantic preserving mapping we introduce between models of WPs 2 and 3 (see Chapter 3).

  From another perspective, the intention of WP2 is to develop a notation for modelling and reasoning about component and connector behaviours. It is therefore likely to have features and functionality that lie outside the main workflow of WPs 3-5.

- **Choice of a theory in WP2 should be guided by its usefulness for CONNECT.**

  The specification theory is centred on the notion of components/connectors with operations on components to support composition, as well as independent (conjunction) and incremental (quotient) development. In Chapter 3, we have demonstrated the applicability of our theory to synthesis in WP3. An extension with data in Chapter 4 supports modelling in both WPs 3 and 4. The eventual probabilistic extension of the specification theory will link closely to modelling in WP5. Consequently, the theory is well-suited to the project's needs.

- **Weakness of the algebra.**

  As part of our work in D2.2 [22], we tried to show that a basic collection of primitive connectors can be built up into complex connectors that cover the class of mediators. However, our work proved not to be so fruitful on this front, as in particular it was difficult to prevent the constructed mediator from generating communication mismatches. However, our preliminary findings of the algebra have led us to a greater understanding of how we can use such a collection of primitives, in the form of building ontological constraints.

  In Chapter 3, we introduce a quotienting operator that can be used to generate abstract mediators in the notation of our specification theory. The purpose of the mediator is to allow communication between the supplied components that is free of communication mismatches and guarantees that suitable progress is made (i.e. the system does not deadlock prematurely). We achieve this by building a special component that captures the constraints imposed by the system ontology, which relates actions of components and may impose ordering constraints on actions. Although we have not elaborated on it in this deliverable, these ontological constraints can be built as the parallel composition of the algebraic primitives we introduced in D2.2 [22], treating them as observers (that is, all actions are inputs). It is then the responsibility of the quotienting operator to guarantee that the mediator will not generate any communication mismatches.

  Thus in D2.2 [22] we adopted an approach where the algebra was used to specify and build mediators, whereas we have shifted direction now to the extent that primitives of the algebra are used for building constraints on what the mediator must do, while the quotient operator takes care of the actual synthesis.

  Another comment by the reviewers concerned the fact that the underlying model of our algebra, namely interface automata, was weak in that it did not permit non-determinism or hidden actions. These restrictions were imposed at the time because the known definition of quotient, as given

in [9], could only be applied to deterministic systems. Our new specification theory relaxes these constraints, which has been made possible by devising a quotient operator that can be applied to non-deterministic systems.

- **Develop stronger AG rules for probabilistic reasoning.**

  In the third year, we focused on completeness of assume-guarantee reasoning for probabilistic systems. The AG rules for reasoning about safety and liveness properties in D2.1 [21] and D2.2 [22] are incomplete, as it is not always possible to find assumptions to apply those AG rules to reason about probabilistic systems. In this deliverable, we proposed a complete compositional reasoning technique by modelling assumptions as probabilistic finite automata (PFAs). Furthermore, PFAs assumptions can be automatically constructed in a similar fashion to the learning technique described in D2.2 [22].

1.3 Challenges and achievements for Year 3

The key challenge for Year 3 has been to develop a compositional theory for modelling and reasoning about components, whilst ensuring that it is fully integrated and usable by the remainder of the project. Despite being a non-trivial objective, we have developed a specification theory that aligns well with work packages 3, 4, and 5. Concerning the alignment with WP3, we have demonstrated through Chapter 3 a clean semantic preserving mapping between our theory in WP2 and the notation of extended LTSs used as part of synthesis in WP3 [23]. Concerning alignment with WP4, we have developed an extension of our specification theory with data in Chapter 4: the process model defined there has the same form as the register automata defined in Deliverable 4.3 [24]. The differences are that for learning, register automata should be deterministic, and that inputs and outputs do not play the same roles in learning. The link with WP5 [26] will come to fruition in the fourth year, when we produce a quantitative extension of our specification theory.

One of the main achievements of WP2 for Year 3 is the formalization of a component, and connector, specification theory. The theory is described in Chapter 2. It allows us to rephrase interoperability between heterogeneous protocols in terms of connector behaviours, hence providing a solution to issues related to Task 2.3. A suitable interoperability notion has been defined within the work of WP3. In particular, by referring to Deliverable D3.3 [23], with *interoperability*, we mean the ability of protocols to correctly communicate and coordinate, i.e., to correctly synchronize. In other words, two systems successfully interoperate if they correctly exchange compatible conversations or compatible traces. We focus on protocols that can potentially interoperate despite showing some differences. That is, communication and coordination between such protocols is possible in principle since they are semantically equivalent and complementary, but cannot be achieved seamlessly because of heterogeneity or diversities. Synchronization between protocols, thus, can be achieved under mediation, i.e., through a mediator that while managing mismatches, allows protocols to effectively exchange compatible traces (sequences of actions). A mediator is then a protocol that allows communication and coordination among compatible protocols by mediating their differences. Therefore, such a mediator serves as the locus where semantically equivalent and complementary actions are correctly synchronized thus enabling a *mediated interoperability* among protocols. Semantic equivalence of (heterogeneous) actions is managed by the use of ontologies.

With this notion of mediated interoperability in mind, our specification theory provides a declarative notation for describing components and connectors by means of trace sets, and its related operational semantics in terms of transition systems. The theory is equipped with both a suitable refinement relation hence allowing for safe replacing of components/connectors and composition operators for the construction of composite components/connectors out of smaller ones. Furthermore, a suitable *quotient* operator is defined hence supporting incremental development. Our specification theory also allows to represent specific ontology abstractions as “observers” (which are simply just components), so that ontology abstractions can be taken into account for connector synthesis purposes.

As it is detailed in Chapter 3, the quotient operator is particularly important in order to characterize the connector synthesis problem faced by WP3 as a *quotient problem*, hence showing, within the work of WP2, how we can use our theory for synthesizing connectors that act as component mediators. By reasoning in terms of quotient, we state correctness properties of the mediator synthesis such as *soundness*

and *generality* of the synthesized mediators, i.e., a synthesized mediator does not introduce inconsistent behavior, and any other equivalent mediator is a refinement of the synthesized one, respectively.

As described in Chapter 3, and in particular in Section 3.8, the mediator synthesis approach developed within WP3 fits within the theory formalized in this deliverable. In Section 3.8, we show that there exists a mapping between the theory of mediators described in [23] and our specification theory. This allows us to state correctness of the WP3 mediator synthesis as inherited, under the described mapping, by the soundness and generality properties stated for mediators which are synthesized by means of quotient in our specification theory. Furthermore, we recall that the WP3 mediator theory uses ontologies for checking semantic equivalence among heterogenous sequences of actions during the mediator synthesis process. This is similar to our quotient procedure, that makes use of ontology abstractions. It is worthwhile to mention that such ontology abstractions can be represented as a composition of terms from the connector algebra defined in Deliverable D2.2 [22], under minor technical changes to the algebra that we avoid explicitly discussing here since they are not crucial for the purposes of this deliverable. For instance, in the example described in Section 3.6, a specific part of the ontology description could be given as a *Plug* "*observer*" connector of two *Trans* primitives by using the algebra notation. This means that, within the specification theory formalized in this deliverable, we can use the connector algebra (representing one of the main outcomes of WP2 during Year 2) to express the kind of ontological constraints that are needed to enable automated mediator synthesis.

A further link between WPs 2, 3, and 4, is established by the data-extension of our specification theory, described in Chapter 4. This extension links with the work on register automata (see Deliverable 4.3 [24]), in that register automata can be seen as a restricted form of processes with data. The extension also links with the work on synthesis in WP3, in that it allows to take constraints on data into account. Adding data constraints to the synthesis problem can in some cases provide an alternative to the use of ontologies for the purpose of matching actions between components: we illustrate this point by a couple of examples in Chapter 4.

Summing up, the specification theory that we describe in this deliverable, beyond providing a solution to issues related to Task 2.3, i.e., rephrasing interoperability in terms of connector behaviours, allows us to explicitly show the relationships between some previous work of WP2 (i.e., the connector algebra), current work of WP2 (i.e., a compositional specification theory for components and connectors suitable for automated mediator synthesis), and current work of WP3 (i.e., revised implementation of the mediator theory and related synthesis method). That is, the specification theory described in this deliverable allows us to address also the following reviewer comments from the second review: (i) "*The relation between the calculus developed in WP2 based on interface automata targeted at verification and the synthesis work in WP3 is unclear.*" and (ii) "*It is not obvious to the reviewers that this algebra can be exploited in WP3 which seems to be progressing well with the use of LTSs and does not seem to require this algebra.*"

Completeness is an important issue in compositional reasoning, which affects the usefulness of a compositional reasoning technique. A complete technique allows us to check the validity of a property by using this technique only. In this sense, the assume-guarantee rules presented in D2.1 [21] and D2.2 [22] are incomplete. In Year 3, the key achievement in probabilistic compositional verification is a complete assume-guarantee technique, which can generate probabilistic assumptions automatically by a new L*-style learning methods. In this technique, probabilistic assumptions are specified as probabilistic finite automata (PFAs), and hence our achievement includes a semi-algorithm for checking language inclusion for PFAs. Currently, this technique can only be applied to probabilistic systems represented by Discrete-Time Markov Chains (DTMCs). In addition to the complete reasoning technique, we also extended the learning technique in D2.2 [22] to generate assumptions for the AG rule (ASYM-N), which allows a system to be decomposed into more than two components, and investigate an alternative learning method NL* for automated assumption construction.

1.4 Outline

We begin by introducing our compositional specification theory in Chapter 2. Linking the specification theory to the work of synthesis in WP3 is elaborated on in Chapter 3, where we also demonstrate our specification theory on CONNECT-related examples. Chapter 4 details an extension of the specification theory with data. Chapter 5 presents work on the second strand of the work package that encompasses

developing probabilistic assume-guarantee reasoning techniques for probabilistic systems. Finally, we state the key conclusions of our work in Chapter 6 and also state the immediate tasks to be completed during the remainder of the project.

2 Compositional specification theory for component behaviours

In this chapter we introduce a framework for modelling the behaviours of components. Based on our previous work, we choose not to distinguish CONNECTors from components. Chapter 3 shows how we can use our framework for synthesising CONNECTors that act as component mediators; thus that chapter will highlight the novel aspects of our work in terms of its application to CONNECTor synthesis. The work of this chapter is based predominantly on our recent paper accepted for publication at ESOP 2012 [18].

Our framework takes the form of a specification theory. We define an abstract (declarative) notation for describing components by means of trace sets, and provide an operational modelling notation in terms of transition systems. A refinement relation corresponding to substitutivity allows one to compare components, and unequivocally state when a component can be used safely in place of another. Operators of the specification theory allow for the construction of composite components out of smaller components.

Concerning the operators, we introduce *parallel composition* for examining how a number of components interact with one another. *Conjunction*, corresponding to shared refinement, allows for the combining of multiple component-specifications; it is essentially the greatest lower bound of refinement. *Disjunction* is defined as the least upper bound of refinement. Both conjunction and disjunction support independent development of components. *Hiding* can be used for abstraction, while *renaming* allows for the reuse of components in subtly different environments. The operator of *quotient* supports incremental development, by synthesising missing components given an overarching specification of the system.

2.1 Introduction

In our framework, components communicate by synchronising on input and output actions. A component comes equipped with an interface that specifies which actions it is willing to communicate on, as well as the type of the actions i.e., whether an action is an input or output in the component. Besides the interface, a component comes equipped with a behaviour that specifies the permitted temporal ordering of interactions between the component and its environment.

We do not insist that communication is handshaken. Therefore, we adopt synchronisation semantics in which outputs are non-blocking. If a component is willing to receive an input, this can only take place if the action is signalled by the environment. However, if a component wishes to emit an output, it can do so at its convenience, even if the environment is not willing to receive it (in which case an inconsistency would arise). This setting provides greater generality in modelling component-based systems, in comparison to the setting used by the CONNECT consortium in WPs 3-5. We note, however, that the set of allowable CONNECT-based systems is a sub-class of the connectors and components admitted by our framework.

An environment is said to be safe for a component, if any interaction between the component and environment does not permit the component to uncontrollably become inconsistent. In that respect, the environment must refuse to issue any input from which the component can encounter an inconsistency by a series of output actions. To support this principle in refinement, which corresponds to safe-substitutivity, we must allow a component to propagate its inconsistencies backwards over output actions. This is similar to the backward propagation of inconsistencies in parallel composition of interface automata, as defined by de Alfaro and Henzinger in [29]. Further detail on how this works in our setting will be elaborated in Section 2.2.1.

To highlight the novelty of our specification theory, we relate it to existing frameworks in the literature. Our models, in particular Logic IOLTSs, are inspired by the Logic LTS framework due to Lüttgen and Vogler [53], a compositional theory that admits as specifications LTSs without I/O distinctions. Their inconsistency predicate is induced from inequality of ready-sets, rather than communication mismatches as in our case. Refinement is based on ready-simulation; alphabetised parallel and conjunction are considered, but not quotient.

The operational component model in our framework has been greatly influenced by I/O automata [55] and interface automata [29]: both are based on I/O LTSs, with the proviso that I/O automata must be input-enabled, meaning that each state of the automaton is willing to accept any input. We differ from I/O automata by not imposing input-enabledness and from interface automata by working with an explicit

representation of inconsistencies. Another difference is refinement, which for interface automata is defined in terms of alternating simulation, rather than traces; the original definition in [29] is simplified in [30], but works only for input-deterministic interface automata. It should be noted that, unlike [29, 30], we use an associative variant of parallel composition, which combines an input and output into an output (as in [28]). Furthermore, we provide a definition of conjunction corresponding to shared refinement of interface automata, which substantially generalises that of [32] for synchronous components. Moreover, our quotienting operator on Logic IOLTSs generalises that in [9] defined only for deterministic components. Our work thus addresses a number of the shortcomings of interface automata, as detailed by Legay *et al.* in [65].

There are a number of process-algebraic frameworks that deal with asynchronous I/O interaction. We mention a characterisation of I/O automata by De Nicola and Segala [59], which is actually a generalisation (and also applicable to interface automata), since the inconsistent process Ω allows to distinguish between good and bad inputs. Similarly to our approach, refinement in [59] is given by trace containment, but does not extend to inconsistent trace containment. This is because we allow a Logic IOLTS to become inconsistent after emitting an output, whereas a process can only become inconsistent through receiving a bad input. Finally, we remark that [59] supports a number of operators of a specification theory, but does not deal with conjunction or quotient.

Our work is also related to the ioco theory in model based testing [69]. The ioco relation is similar to our refinement, but lacks compositionality of operators, so is not well-suited to a specification theory for components.

There have been several CSP-based frameworks that deal with asynchronous communication; of these, the receptive process theory (RPT) [49] utilises a model of concurrency similar to ours in that outputs are non-blocking. RPT also considers quotient (referred to as factorisation), but for the restricted class of delay-insensitive networks [50] that differ from our setting.

A further class of component-based modelling formalisms is based on may/ must modalities. A specification theory for components has been devised in [63] based on modalities [66, 64], but the definition of quotient is more restrictive than ours. Larsen *et al.* have made an effort in relating modal transition systems with interface automata [52]. The approach of modal I/O automata is based on a game-like definition of refinement, which we claim to be more complex than ours, see, e.g., the discussion of parallel composition in [64]. The framework in [64] can support reasoning about liveness properties which our framework does not (although they both support reasoning of safety properties). However, our framework can be easily extended by introducing quiescent states, and additionally considering containment of quiescent traces to reason about liveness.

CONNECTor synthesis in terms of quotient has been considered by Calvert and Lam in [14], however, their approach is more limited than ours. First of all, our quotient is defined on asynchronous systems without handshaking. Moreover, our quotient is the adjoint of parallel composition with respect to a substitutive refinement preorder that preserves non-inconsistency under all environmental contexts. Unlike the existing definition of quotient on interface automata [9], our definition does not require the components under consideration to be deterministic.

Relating the applications of quotient to CONNECT, Martinelli makes use of quotient-like techniques for reasoning about system security [57]. In his setting, a system can be thought of as a process with a hole, the latter can be used by an attacker to interact maliciously with the system. Ascertaining whether a security policy is upheld by the system is determined by considering all possible instantiations of the hole. Our use of quotient is different, as we are concerned with synthesising CONNECTors by means of quotient, rather than verifying properties of systems (although we can certainly infer properties of our synthesised CONNECTors). Chapter 3 explains in more detail how our quotient can be used for synthesising CONNECTors for components having different interaction primitives, by handling ontological constraints. The work is novel from a methodological perspective, although it shares similarities with existing techniques and methods.

2.2 Declarative (trace-based) modelling framework

In this section, we model components abstractly by means of declarative specifications. We introduce a substitutive refinement preorder together with three compositional operators on declarative specifications.

A declarative specification comes equipped with an interface, together with a set of behaviours over the interface. The interface is represented by a set of input actions and a set of output actions, which are necessarily disjoint, while the behaviour is characterised by traces.

Definition 2.1 (Declarative specification) *A* declarative specification $\mathcal{P}$ *is a tuple* $\langle \mathcal{A}^I_\mathcal{P}, \mathcal{A}^O_\mathcal{P}, T_\mathcal{P}, F_\mathcal{P} \rangle$ *in which* $\mathcal{A}^I_\mathcal{P}$ *and* $\mathcal{A}^O_\mathcal{P}$ *are disjoint sets referred to as inputs and outputs respectively (the union of which is denoted by* $\mathcal{A}_\mathcal{P}$*),* $T_\mathcal{P} \subseteq \mathcal{A}^*_\mathcal{P}$ *is a non-empty set of permissible traces, and* $F_\mathcal{P} \subseteq \mathcal{A}^*_\mathcal{P}$ *is a set of inconsistent traces. The trace sets must satisfy the constraints:*

1. $F_\mathcal{P} \subseteq T_\mathcal{P}$
2. *If* $t \in T_\mathcal{P}$ *and* $i \in \mathcal{A}^I_\mathcal{P}$*, then* $ti \in T_\mathcal{P}$
3. $T_\mathcal{P}$ *is prefix closed*
4. *If* $t \in F_\mathcal{P}$ *and* $t' \in \mathcal{A}^*_\mathcal{P}$*, then* $tt' \in F_\mathcal{P}$*.*

Outputs are under the control of the component, whereas inputs are issued by the environment. This means that, after any successful interaction between the component and the environment, the environment can issue any input i, even if it will be refused by the component. Naturally, if i is refused by the component after the trace t, we deem ti to be an inconsistent trace, since a communication mismatch has occurred. Given this treatment of inputs, we say that our theory is *not* input-enabled, even though $T_\mathcal{P}$ is closed under input-extensions.

Example A drinks machine dispenses either a tea or a coffee after a coin has been inserted. The drinks machine has sufficient water to produce only 2 drinks, after which a further coin insertion renders the machine inoperable. This behaviour can be encoded by the declarative specification $DM = \langle \{£\}, \{t, c\}, T, F_1 \cup F_2 \rangle$, where:

- $T = \{\epsilon, £, £(c+t), £(c+t)£, £(c+t)£(c+t)\} \cup F_1 \cup F_2$
- $F_1 = £(c+t)£(c+t)£(£+c+t)^*$ insertion of third coin after two dispensations
- $F_2 = (\epsilon + £(c+t))£££(£+c+t)^*$ insertion of second coin before dispensation.

As an issue of notation, for a trace t and set of actions $\mathcal{A}$, write $t \upharpoonright \mathcal{A}$ for the projection of t onto $\mathcal{A}$. From hereon let $\mathcal{P}$, $\mathcal{Q}$ and $\mathcal{R}$ be declarative specifications with signatures $\langle \mathcal{A}^I_\mathcal{P}, \mathcal{A}^O_\mathcal{P}, T_\mathcal{P}, F_\mathcal{P} \rangle$, $\langle \mathcal{A}^I_\mathcal{Q}, \mathcal{A}^O_\mathcal{Q}, T_\mathcal{Q}, F_\mathcal{Q} \rangle$ and $\langle \mathcal{A}^I_\mathcal{R}, \mathcal{A}^O_\mathcal{R}, T_\mathcal{R}, F_\mathcal{R} \rangle$ respectively.

2.2.1 Refinement

As refinement corresponds to safe substitutivity, for $\mathcal{Q}$ to be used in place of $\mathcal{P}$ we require that $\mathcal{Q}$ must exist safely in *any* environment that $\mathcal{P}$ can exist in safely. Whether an environment is safe for a specification depends on the sequences of message exchanges afforded by the component. If an environment can prevent a component from performing an inconsistent trace, then the environment is said to be safe.

We do not insist that a component $\mathcal{Q}$ must have the same interface as the component $\mathcal{P}$ to be refined. Instead $\mathcal{Q}$ must be accepting of at least all of $\mathcal{P}$'s inputs, while restricting to a subset of $\mathcal{P}$'s outputs. This can be formalised by the covariant relationship $\mathcal{A}^I_\mathcal{P} \subseteq \mathcal{A}^I_\mathcal{Q}$ on inputs and the contravariant constraint $\mathcal{A}^O_\mathcal{Q} \subseteq \mathcal{A}^O_\mathcal{P}$ on outputs.

In order to establish that refinement holds, we need to perform a weak form of alphabet equalisation on the inputs of the component to be refined. We refer to this operation as lifting. Informally, lifting extends the trace sets of $\mathcal{P}$ by explicitly refusing any input in $\mathcal{A}^I_\mathcal{Q} \setminus \mathcal{A}^I_\mathcal{P}$, after which it allows for arbitrary behaviour.

Definition 2.2 (Lifting) *Let* $\mathcal{P}$ *be a declarative specification, and let* $\mathcal{A}^I_\mathcal{Q}$ *be a set of input actions. The lifting of trace sets* $T_\mathcal{P}$ *and* $F_\mathcal{P}$ *to* $\mathcal{A}^I_\mathcal{Q}$*, written as* $T_\mathcal{P} \uparrow \mathcal{A}^I_\mathcal{Q}$ *and* $F_\mathcal{P} \uparrow \mathcal{A}^I_\mathcal{Q}$ *respectively, is defined as:*

- $T_\mathcal{P} \uparrow \mathcal{A}^I_\mathcal{Q} = T_\mathcal{P} \cup \{tit' : t \in T_\mathcal{P}, i \in \mathcal{A}^I_\mathcal{Q} \setminus \mathcal{A}^I_\mathcal{P}$ *and* $t' \in (\mathcal{A}^I_\mathcal{Q} \cup \mathcal{A}_\mathcal{P})^*\}$

- $F_{\mathcal{P}} \uparrow \mathcal{A}^I_{\mathcal{Q}} = F_{\mathcal{P}} \cup \{tit' : t \in T_{\mathcal{P}}, i \in \mathcal{A}^I_{\mathcal{Q}} \setminus \mathcal{A}^I_{\mathcal{P}}$ *and* $t' \in (\mathcal{A}^I_{\mathcal{Q}} \cup \mathcal{A}_{\mathcal{P}})^*\}$.

Recall that an environment is safe for a component if the environment can prevent the component from performing an inconsistent trace. As outputs are under the control of the component itself, a safe environment must refuse to issue an input on any trace from which there is a sequence of output actions after the input that allows the trace to become inconsistent.

Under such an arrangement, for each declarative specification $\mathcal{P}$ we can define the safe declarative specification $\mathcal{E}(\mathcal{P})$ containing all of $\mathcal{P}$'s permissible and inconsistent traces, but also satisfying the additional property: if $t \in T_{\mathcal{P}}$ and there exists $t' \in (\mathcal{A}^O_{\mathcal{P}})^*$ such that $tt' \in F_{\mathcal{P}}$, then $t \in F_{\mathcal{E}(\mathcal{P})}$. This has the effect of forcing all inconsistent traces to become inconsistent on the environment's issue of a bad input. If the environment respects this safe specification, by not issuing any input that results in an inconsistent trace, then the component can never encounter an inconsistent trace. Note that if $\epsilon \in F_{\mathcal{E}(\mathcal{P})}$ then there is no environment that can prevent $\mathcal{P}$ from performing an inconsistent trace. However, for uniformity we still refer to $\mathcal{E}(\mathcal{P})$ as the safe specification of $\mathcal{P}$.

Definition 2.3 (Safe specification) *Let $\mathcal{P}$ be a declarative specification. The most general safe specification for $\mathcal{P}$ is a declarative specification $\mathcal{E}(\mathcal{P}) = \langle \mathcal{A}^I_{\mathcal{P}}, \mathcal{A}^O_{\mathcal{P}}, T_{\mathcal{E}(\mathcal{P})}, F_{\mathcal{E}(\mathcal{P})} \rangle$, where $T_{\mathcal{E}(\mathcal{P})} = T_{\mathcal{P}} \cup F_{\mathcal{E}(\mathcal{P})}$ and $F_{\mathcal{E}(\mathcal{P})} = \{tt' \in \mathcal{A}^*_{\mathcal{P}} : t \in T_{\mathcal{P}}$ and $\exists t'' \in (\mathcal{A}^O_{\mathcal{P}})^* \cdot tt'' \in F_{\mathcal{P}}\}$.*

We can now define our substitutive refinement preorder. From the safe specification associated with an arbitrary declarative specification, it is easy to see whether a declarative specification can be substituted safely in place of another. Note that $F_{\mathcal{Q}} \subseteq F_{\mathcal{P}} \uparrow \mathcal{A}^I_{\mathcal{Q}}$ would be too strong to use for the last clause, as we are only interested in trace containment up to the point where an environment can issue a bad input.

Definition 2.4 (Refinement) *For declarative specifications $\mathcal{P}$ and $\mathcal{Q}$, $\mathcal{Q}$ is said to be a refinement of $\mathcal{P}$, written $\mathcal{Q} \sqsubseteq_{dec} \mathcal{P}$, iff:*

1. $\mathcal{A}^I_{\mathcal{P}} \subseteq \mathcal{A}^I_{\mathcal{Q}}$
2. $\mathcal{A}^O_{\mathcal{Q}} \subseteq \mathcal{A}^O_{\mathcal{P}}$
3. $T_{\mathcal{E}(\mathcal{Q})} \subseteq T_{\mathcal{E}(\mathcal{P})} \uparrow \mathcal{A}^I_{\mathcal{Q}}$
4. $F_{\mathcal{E}(\mathcal{Q})} \subseteq F_{\mathcal{E}(\mathcal{P})} \uparrow \mathcal{A}^I_{\mathcal{Q}}$.

Our trace-based definition of refinement respects the intuition behind de Alfaro and Henzinger's alternating refinement relation on interface automata, although theirs is simulation-based [29]. We believe that alternating simulation is too strong for checking of safe-substitutivity, as the environments compatible with a component should be independent of the component's resolution of non-determinism.

As our refinement is based on an extension of language inclusion, its complexity is in P, assuming regularity of the trace sets. Note that lifting maintains regularity.

Equivalence of declarative specifications in our framework is defined in terms of mutual refinement.

Definition 2.5 (Equivalence) *Let $\mathcal{P}$ and $\mathcal{Q}$ be declarative specifications. Then $\mathcal{P}$ and $\mathcal{Q}$ are said to be equivalent, written $\mathcal{P} \equiv_{dec} \mathcal{Q}$, iff $\mathcal{P} \sqsubseteq_{dec} \mathcal{Q}$ and $\mathcal{Q} \sqsubseteq_{dec} \mathcal{P}$.*

Lemma 2.6 (Preorder) *Refinement is both reflexive and transitive.*

2.2.2 Parallel composition

The parallel composition operator on declarative specifications yields a declarative specification representing the combined effect of its operands running asynchronously. We do not consider synchronous parallel composition, as this does not make sense when dealing with non-blocking output actions. To preserve the effect that a single output from a component can be received by multiple components in the environment, we must define the parallel composition to repeatedly broadcast an output: this means that an input $a?$ and output $a!$ combine to form an output $a!$ (as in certain variants of I/O automata), rather than a hidden action τ as is the case in Milner's CCS.

Not all declarative specifications can be composed with one another; we restrict to those that are said to be *composable*. $\mathcal{P}$ and $\mathcal{Q}$ are composable for parallel composition only if $\mathcal{A}^O_\mathcal{P} \cap \mathcal{A}^O_\mathcal{Q} = \emptyset$. This restriction is meaningful if we consider inputs on an interface as buttons and outputs as lights. Given two distinct components, it is not possible for them to share a common light, whereas it is possible to push their buttons at the same time. In practice, issues of composability can be avoided by employing renaming, if this is considered to be appropriate.

Definition 2.7 (Parallel composition) *Let $\mathcal{P}$ and $\mathcal{Q}$ be declarative specifications such that $\mathcal{A}^O_\mathcal{P}$ and $\mathcal{A}^O_\mathcal{Q}$ are disjoint. Then $\mathcal{P} \mid\mid \mathcal{Q}$ is the declarative specification $\langle \mathcal{A}^I_{\mathcal{P}\mid\mid\mathcal{Q}}, \mathcal{A}^O_{\mathcal{P}\mid\mid\mathcal{Q}}, T_{\mathcal{P}\mid\mid\mathcal{Q}}, F_{\mathcal{P}\mid\mid\mathcal{Q}} \rangle$, where:*

- $\mathcal{A}^I_{\mathcal{P}\mid\mid\mathcal{Q}} = (\mathcal{A}^I_\mathcal{P} \cup \mathcal{A}^I_\mathcal{Q}) \setminus (\mathcal{A}^O_\mathcal{P} \cup \mathcal{A}^O_\mathcal{Q})$
- $\mathcal{A}^O_{\mathcal{P}\mid\mid\mathcal{Q}} = \mathcal{A}^O_\mathcal{P} \cup \mathcal{A}^O_\mathcal{Q}$
- $T_{\mathcal{P}\mid\mid\mathcal{Q}} = \{t \in \mathcal{A}^*_{\mathcal{P}\mid\mid\mathcal{Q}} : t \upharpoonright \mathcal{A}_\mathcal{P} \in T_\mathcal{P}$ *and* $t \upharpoonright \mathcal{A}_\mathcal{Q} \in T_\mathcal{Q}\} \cup F_{\mathcal{P}\mid\mid\mathcal{Q}}$
- $F_{\mathcal{P}\mid\mid\mathcal{Q}} = \{tt' \in \mathcal{A}^*_{\mathcal{P}\mid\mid\mathcal{Q}} : t \upharpoonright \mathcal{A}_\mathcal{P} \in F_\mathcal{P}$ *and* $t \upharpoonright \mathcal{A}_\mathcal{Q} \in T_\mathcal{Q}$, *or* $t \upharpoonright \mathcal{A}_\mathcal{P} \in T_\mathcal{P}$ *and* $t \upharpoonright \mathcal{A}_\mathcal{Q} \in F_\mathcal{Q}\}$.

Informally, a trace is permissible in $\mathcal{P} \mid\mid \mathcal{Q}$ if its projection onto $\mathcal{A}_\mathcal{P}$ is a trace of $\mathcal{P}$ and its projection onto $\mathcal{A}_\mathcal{Q}$ is a trace of $\mathcal{Q}$. A trace is inconsistent if it has a prefix whose projection onto the alphabet of one of the components is inconsistent and the projection onto the alphabet of the other component is a permissible trace of that component.

We demonstrate the following result, a corollary of which is that mutual refinement is a congruence for parallel, subject to composability.

Theorem 2.8 (Compositionality of parallel) *Let $\mathcal{P}$, $\mathcal{Q}$ and $\mathcal{R}$ be declarative specifications such that $\mathcal{P}$ and $\mathcal{R}$ are composable for parallel composition, and $\mathcal{A}^I_\mathcal{Q} \cap \mathcal{A}^O_\mathcal{R} \subseteq \mathcal{A}^I_\mathcal{P} \cap \mathcal{A}^O_\mathcal{R}$. If $\mathcal{Q} \sqsubseteq_{dec} \mathcal{P}$, then $\mathcal{Q} \mid\mid \mathcal{R} \sqsubseteq_{dec} \mathcal{P} \mid\mid \mathcal{R}$.*

2.2.3 Conjunction

The conjunction operator on declarative specifications can be thought of as finding a common implementation for a number of properties, each of which are represented by declarative specifications. Naturally, any implementation of these properties should be a refinement of each of the properties to be implemented. The conjunction (or shared refinement) of two declarative specifications $\mathcal{P}$ and $\mathcal{Q}$ is the *coarsest* declarative specification that refines both $\mathcal{P}$ and $\mathcal{Q}$. Thus conjunction is the meet operator on the refinement preorder.

As remarked in Section 2.1, no definition of conjunction has been provided on interface automata, which are conceptually similar to the models of our framework. Henzinger *et al.* have previously provided a definition of conjunction with respect to alternating refinement [32], however this is only defined on a special type of synchronous component, rather than interface automata.

As for parallel composition, conjunction can only be performed on composable components. $\mathcal{P}$ and $\mathcal{Q}$ are composable for conjunction only if the sets $\mathcal{A}^I_\mathcal{P} \cup \mathcal{A}^I_\mathcal{Q}$ and $\mathcal{A}^O_\mathcal{P} \cup \mathcal{A}^O_\mathcal{Q}$ are disjoint.

Definition 2.9 (Conjunction) *Let $\mathcal{P}$ and $\mathcal{Q}$ be declarative specifications such that $\mathcal{A}^I_\mathcal{P} \cup \mathcal{A}^I_\mathcal{Q}$ and $\mathcal{A}^O_\mathcal{P} \cup \mathcal{A}^O_\mathcal{Q}$ are disjoint. Then $\mathcal{P} \wedge \mathcal{Q}$ is the declarative specification $\langle \mathcal{A}^I_{\mathcal{P}\wedge\mathcal{Q}}, \mathcal{A}^O_{\mathcal{P}\wedge\mathcal{Q}}, T_{\mathcal{P}\wedge\mathcal{Q}}, F_{\mathcal{P}\wedge\mathcal{Q}} \rangle$, where:*

- $\mathcal{A}^I_{\mathcal{P}\wedge\mathcal{Q}} = \mathcal{A}^I_\mathcal{P} \cup \mathcal{A}^I_\mathcal{Q}$
- $\mathcal{A}^O_{\mathcal{P}\wedge\mathcal{Q}} = \mathcal{A}^O_\mathcal{P} \cap \mathcal{A}^O_\mathcal{Q}$
- $T_{\mathcal{P}\wedge\mathcal{Q}} = T_\mathcal{P} \uparrow \mathcal{A}^I_\mathcal{Q} \cap T_\mathcal{Q} \uparrow \mathcal{A}^I_\mathcal{P}$
- $F_{\mathcal{P}\wedge\mathcal{Q}} = F_\mathcal{P} \uparrow \mathcal{A}^I_\mathcal{Q} \cap F_\mathcal{Q} \uparrow \mathcal{A}^I_\mathcal{P}$.

Conjunction has strong connections with the logical 'and' operator in Boolean algebra, as shown below. Mutual refinement is a congruence for conjunction, subject to composability.

Theorem 2.10 (Properties of conjunction)

- *Conjunction is the greatest lower bound operator for* $\sqsubseteq_{dec}$
- $\mathcal{R} \sqsubseteq_{dec} \mathcal{P}$ *and* $\mathcal{R} \sqsubseteq_{dec} \mathcal{Q}$ *iff* $\mathcal{R} \sqsubseteq_{dec} \mathcal{P} \wedge \mathcal{Q}$
- $\mathcal{P} \wedge \mathcal{Q} \equiv_{dec} \mathcal{Q}$ *iff* $\mathcal{Q} \sqsubseteq_{dec} \mathcal{P}$.

Theorem 2.11 (Compositionality of conjunction) *Let* $\mathcal{P}$, $\mathcal{Q}$ *and* $\mathcal{R}$ *be declarative specifications such that* $\mathcal{P}$ *is composable with* $\mathcal{R}$ *for conjunction. If* $\mathcal{Q} \sqsubseteq_{dec} \mathcal{P}$, *then* $\mathcal{Q} \wedge \mathcal{R} \sqsubseteq_{dec} \mathcal{P} \wedge \mathcal{R}$.

2.2.4 Quotient

The final operation that we consider on the specification theory is that of quotienting, which has strong connections to synthesis. Given a specification for a system $\mathcal{R}$, together with a sub-component $\mathcal{P}$ implementing[1] part of $\mathcal{R}$, the quotient yields the *coarsest* specification for the remaining part of $\mathcal{R}$ to be implemented. Thus, the parallel composition of the quotient with $\mathcal{P}$ should be a refinement of the system-wide specification $\mathcal{R}$. Therefore, quotient can be thought of as the adjoint of parallel composition.

A definition of quotient that acts on deterministic interface automata has been supplied by Bhaduri and Ramesh in [9]. As there is a close coupling between declarative specifications and deterministic interface automata, we should not be surprised that quotient is definable in our setting. However, contrasting their definition with ours reveals that the presentation is substantially different.

As $\mathcal{P}$ is a sub-component of $\mathcal{R}$, we make the reasonable assumption that $\mathcal{A}_\mathcal{P} \subseteq \mathcal{A}_\mathcal{R}$. Moreover, a necessary condition for the existence of the quotient is that $\mathcal{A}^O_\mathcal{P} \subseteq \mathcal{A}^O_\mathcal{R}$, otherwise refinement will fail on the alphabet containment checks.

Definition 2.12 (Quotient) *Let* $\mathcal{P}$ *and* $\mathcal{R}$ *be declarative specifications such that* $\mathcal{A}^O_\mathcal{P} \subseteq \mathcal{A}^O_\mathcal{R}$ *and* $\mathcal{A}_\mathcal{P} \subseteq \mathcal{A}_\mathcal{R}$. *The quotient of* $\mathcal{P}$ *from* $\mathcal{R}$ *is the specification* $\mathcal{R}/\mathcal{P}$ *with signature* $\langle \mathcal{A}^I_{\mathcal{R}/\mathcal{P}}, \mathcal{A}^O_{\mathcal{R}/\mathcal{P}}, T_{\mathcal{R}/\mathcal{P}}, F_{\mathcal{R}/\mathcal{P}} \rangle$, *where:*

- $\mathcal{A}^I_{\mathcal{R}/\mathcal{P}} = \mathcal{A}^O_\mathcal{P} \cup \mathcal{A}^I_\mathcal{R}$
- $\mathcal{A}^O_{\mathcal{R}/\mathcal{P}} = \mathcal{A}^O_\mathcal{R} \setminus \mathcal{A}^O_\mathcal{P}$
- $T_{\mathcal{R}/\mathcal{P}} = \{t \in \mathcal{A}^*_\mathcal{R} : \forall t'$ *a prefix of* $t \cdot L(t')$ *and* $\forall t'' \in {\mathcal{A}^I_{\mathcal{R}/\mathcal{P}}}^* \cdot L(tt'')\}$
- $F_{\mathcal{R}/\mathcal{P}} = \{t \in A^*_\mathcal{R} : (t \restriction \mathcal{A}_\mathcal{P} \in T_\mathcal{P} \implies t \in F_{\mathcal{E}(\mathcal{R})})$ *and* $\forall t'$ *a prefix of* $t \cdot L(t')\}$
- $L(t) = (t \restriction \mathcal{A}_\mathcal{P} \in F_\mathcal{P} \implies t \in F_{\mathcal{E}(\mathcal{R})})$ *and* $(t \restriction \mathcal{A}_\mathcal{P} \in T_\mathcal{P} \implies t \in T_\mathcal{R})$.

The alphabet of the quotient contains all of the actions from $\mathcal{A}_\mathcal{R}$ and $\mathcal{A}_\mathcal{P}$ so that $\mathcal{R}/\mathcal{P}$ can fully control $\mathcal{P}$ and emulate the behaviour of $\mathcal{R}$. Yet still, simple examples reveal that there may not exist a component $\mathcal{Q}$ over an interface consisting of inputs $\mathcal{A}^I_{\mathcal{R}/\mathcal{P}}$ and outputs $\mathcal{A}^O_{\mathcal{R}/\mathcal{P}}$ such that $\mathcal{P} \parallel \mathcal{Q} \sqsubseteq_{dec} \mathcal{R}$. Unfortunately, the existence of the quotient cannot be ascertained by a syntactic check on the alphabets of $\mathcal{P}$ and $\mathcal{R}$.

In Definition 2.12 we referred to $\mathcal{R}/\mathcal{P}$ as a specification, but not a declarative specification. If $T_{\mathcal{R}/\mathcal{P}}$ is non-empty, then $T_{\mathcal{R}/\mathcal{P}}$ is a declarative specification, otherwise it is not. As the following theorem shows, if $T_{\mathcal{R}/\mathcal{P}}$ is non-empty (a condition of being a declarative specification), then the quotient exists, and if it is empty, then the quotient does not exist.

Theorem 2.13 (Existence of quotient) *Let* $\mathcal{P}$ *and* $\mathcal{R}$ *be declarative specifications such that* $\mathcal{A}^O_\mathcal{P} \subseteq \mathcal{A}^O_\mathcal{R}$ *and* $\mathcal{A}_\mathcal{P} \subseteq \mathcal{A}_\mathcal{R}$. *Then there exists a declarative specification* $\mathcal{Q}$ *with input actions* $\mathcal{A}^I_{\mathcal{R}/\mathcal{P}}$ *and output actions* $\mathcal{A}^O_{\mathcal{R}/\mathcal{P}}$ *such that* $\mathcal{P} \parallel \mathcal{Q} \sqsubseteq_{dec} \mathcal{R}$ *iff* $T_{\mathcal{R}/\mathcal{P}} \neq \emptyset$.

The next two theorems show that $\mathcal{R}/\mathcal{P}$ satisfies the required properties of quotient when $T_{\mathcal{R}/\mathcal{P}}$ is non-empty, and that quotient is well-behaved with respect to refinement.

[1] We have not given a precise meaning to the term implementation nor sub-component; that is left up to the user of our specification theory. If the reader finds this unnerving, think of $\mathcal{P}$ as an arbitrary component.

Theorem 2.14 (Properties of quotient) *Let $\mathcal{P}$ and $\mathcal{R}$ be declarative specifications such that $\mathcal{A}^O_{\mathcal{P}} \subseteq \mathcal{A}^O_{\mathcal{R}}$ and $\mathcal{A}_{\mathcal{P}} \subseteq \mathcal{A}_{\mathcal{R}}$. If $T_{\mathcal{R}/\mathcal{P}} \neq \emptyset$, then $\mathcal{P} \parallel (\mathcal{R}/\mathcal{P}) \sqsubseteq_{dec} \mathcal{R}$ and for any declarative specification $\mathcal{Q}$ over inputs $\mathcal{A}^I_{\mathcal{R}/\mathcal{P}}$ and outputs $\mathcal{A}^O_{\mathcal{R}/\mathcal{P}}$ such that $\mathcal{P} \parallel \mathcal{Q} \sqsubseteq_{dec} \mathcal{R}$ it holds that $\mathcal{Q} \sqsubseteq_{dec} \mathcal{R}/\mathcal{P}$.*

Theorem 2.15 (Compositionality of quotient) *Let $\mathcal{P}$, $\mathcal{Q}$ and $\mathcal{R}$ be declarative specifications such that $\mathcal{Q} \sqsubseteq_{dec} \mathcal{P}$.*

- *If $\mathcal{Q}/\mathcal{R}$ and $\mathcal{P}/\mathcal{R}$ are defined, then $\mathcal{Q}/\mathcal{R} \sqsubseteq_{dec} \mathcal{P}/\mathcal{R}$.*
- *If $\mathcal{R}/\mathcal{Q}$ and $\mathcal{R}/\mathcal{P}$ are defined, and $(\mathcal{A}^I_{\mathcal{Q}} \cap \mathcal{A}^O_{\mathcal{R}}) \setminus \mathcal{A}_{\mathcal{P}} = \emptyset$, then $\mathcal{R}/\mathcal{Q} \sqsupseteq_{dec} \mathcal{R}/\mathcal{P}$.*

2.3 Operational modelling framework

In this section we take an operational view of components, by specifying their allowable interactions in terms of Logic IOLTSs, an I/O version of labelled transition systems augmented by an inconsistency predicate on states. We remain faithful to the trace-based substitutive preorder, and cast refinement at the operational level in terms of declarative refinement. For any operational model, we can derive an equivalent declarative specification, meaning that the observable safe interactions between the models and an arbitrary environment are indistinguishable.

To support a compositional theory of components, we define the operations of parallel composition, conjunction and quotient directly on our operational models. We further show that compositionality results for the operators on the declarative framework carry over to the operational framework as well.

An explicit definition of implementation is not provided for our models, although there are a number of candidates. One such suggestion for the characterisation of implementations would be the set of specifications in which no inconsistent states are reachable. We leave this for the user to decide.

We can now define the operational models formally. For a set $\mathcal{A}$, write $\mathcal{A}^\tau$ as shorthand for $\mathcal{A} \cup \{\tau\}$, where it is assumed that $\tau \notin \mathcal{A}$.

Definition 2.16 (IOLTS) *An* I/O labelled transition system (IOLTS) P *is a tuple $\langle S_{\mathsf{P}}, \mathcal{A}^I_{\mathsf{P}}, \mathcal{A}^O_{\mathsf{P}}, \longrightarrow_{\mathsf{P}} \rangle$, where S_{P} is a (possibly infinite) collection of processes (states), $\mathcal{A}^I_{\mathsf{P}}$ and $\mathcal{A}^O_{\mathsf{P}}$ are disjoint sets referred to as the inputs and outputs (the union of which we denote by $\mathcal{A}_{\mathsf{P}}$), and $\longrightarrow_{\mathsf{P}} \subseteq S_{\mathsf{P}} \times \mathcal{A}^\tau_{\mathsf{P}} \times S_{\mathsf{P}}$ is the transition relation.*

Note that since we do not insist on our components being fully input-enabled (unlike I/O automata [55]), meaning that at any stage a component can refuse to accept an input issued by the environment or another component, we must extend IOLTSs to reason about potential communication mismatches that occur during interactions. We accomplish this by augmenting IOLTSs with an inconsistency predicate for tracking mismatches. The resulting model, called a Logic IOLTS, takes its inspiration from the Logic LTSs of Lüttgen and Vogler [53, 54], although we have a different interpretation of inconsistency.

Definition 2.17 (Logic IOLTS) *A* Logic IOLTS P *is a tuple $\langle S_{\mathsf{P}}, \mathcal{A}^I_{\mathsf{P}}, \mathcal{A}^O_{\mathsf{P}}, \longrightarrow_{\mathsf{P}}, F_{\mathsf{P}} \rangle$ in which $\langle S_{\mathsf{P}}, \mathcal{A}^I_{\mathsf{P}}, \mathcal{A}^O_{\mathsf{P}}, \longrightarrow_{\mathsf{P}} \rangle$ is an IOLTS, and $F_{\mathsf{P}} \subseteq S_{\mathsf{P}}$ is an inconsistency predicate on states satisfying the property: if $p \in S_{\mathsf{P}}$ can diverge (meaning there is an infinite sequence of τ-transitions emanating from p), then $p \in F_{\mathsf{P}}$.*

The inconsistency predicate annotates states that correspond to run-time errors such as communication mismatches, underspecification, or divergent behaviour. Regardless of why a state is inconsistent, we assume that on encountering an inconsistency, unspecified behaviour can ensue. Consequently, inconsistent states are resemblant of the process CHAOS from CSP [13].

Figure 2.1 shows a number of Logic IOLTSs represented pictorially. We adopt the convention of enclosing the transition system within a box corresponding to the interface of the component. Labelled arrows pointing at the interface correspond to inputs, whereas arrows emanating from the interface correspond to outputs. As a matter of clarity, we only represent the states that are reachable by a sequence of transitions from the process we are interested in. States annotated with an F are deemed to be inconsistent.

We introduce nomenclature for handling stability and hidden τ-transitions. A relation $\stackrel{\epsilon}{\Longrightarrow}_{\mathsf{P}} \subseteq S_{\mathsf{P}} \times S_{\mathsf{P}}$ is defined by $p \stackrel{\epsilon}{\Longrightarrow}_{\mathsf{P}} p'$ iff $p(\stackrel{\tau}{\longrightarrow}_{\mathsf{P}})^* p'$. Generalising $\stackrel{\epsilon}{\Longrightarrow}_{\mathsf{P}}$ for visible actions $a \in \mathcal{A}$, we obtain $p \stackrel{a}{\Longrightarrow}_{\mathsf{P}} p'$

iff there exist p_a and p'_a such that $p \stackrel{\epsilon}{\Longrightarrow}_{\mathsf{P}} p_a \stackrel{a}{\longrightarrow}_{\mathsf{P}} p'_a \stackrel{\epsilon}{\Longrightarrow}_{\mathsf{P}} p'$, and $p \stackrel{\neg a}{\Longrightarrow}_{\mathsf{P}} p'$ iff there exists p_a such that $p \stackrel{a}{\longrightarrow}_{\mathsf{P}} p_a \stackrel{\epsilon}{\Longrightarrow}_{\mathsf{P}} p'$. The extension to words $w = a_1 \dots a_n$ is defined in the natural way by $p \stackrel{w}{\Longrightarrow}_{\mathsf{P}} p'$ iff $p \stackrel{a_1}{\Longrightarrow}_{\mathsf{P}} \dots \stackrel{a_n}{\Longrightarrow}_{\mathsf{P}} p'$.

Furthermore, for a compositional operator $\oplus$, and sets A and B, we write $A \oplus B$ for the set $\{a \oplus b : a \in A \text{ and } b \in B\}$. This allows us to use a process-algebraic notation for states.

From hereon, let $\mathsf{P} = \langle S_{\mathsf{P}}, \mathcal{A}^I_{\mathsf{P}}, \mathcal{A}^O_{\mathsf{P}}, \longrightarrow_{\mathsf{P}}, F_{\mathsf{P}} \rangle$, $\mathsf{Q} = \langle S_{\mathsf{Q}}, \mathcal{A}^I_{\mathsf{Q}}, \mathcal{A}^O_{\mathsf{Q}}, \longrightarrow_{\mathsf{Q}}, F_{\mathsf{Q}} \rangle$ and $\mathsf{R} = \langle S_{\mathsf{R}}, \mathcal{A}^I_{\mathsf{R}}, \mathcal{A}^O_{\mathsf{R}}, \longrightarrow_{\mathsf{R}}, F_{\mathsf{R}} \rangle$ be three Logic IOLTSs, and let p_{P}, q_{Q} and r_{R} be processes in the Logic IOLTSs P, Q and R respectively.

2.3.1 Refinement

In keeping with the declarative framework, we wish refinement to correspond to safe-substitutivity. Hence, we cast refinement at the operational level in terms of refinement at the declarative level. To do this, we define a mapping $[\![\cdot]\!]^*$ from operational models to declarative models (Definition 2.20) that preserves the environments that the models can interact harmoniously with.

An essential feature of operational refinement is that the mapping from operational to declarative models preserves the safe traces of the component. For a declarative specification $\mathcal{P}$, a trace t is said to be immediately-safe iff t is permissible, but not inconsistent (i.e., t lies within $T_{\mathcal{P}} \setminus F_{\mathcal{P}}$). If t is contained within $T_{\mathcal{P}} \setminus F_{\mathcal{E}(\mathcal{P})}$, we say that t is safe. The calculation of the safe traces for a Logic IOLTS is slightly more involved, because it is necessary to deal with non-determinism and τ-transitions.

Definition 2.18 (Immediately-safe states) *The set of immediately-safe states that a process p_{P} can be in after following the trace t is given by $h_{p_{\mathsf{P}}}(t)$, where $h_{p_{\mathsf{P}}} : \mathcal{A}^*_{\mathsf{P}} \longrightarrow 2^{S_{\mathsf{P}}}$ is defined as:*

- $h_{p_{\mathsf{P}}}(\epsilon) = \begin{cases} \emptyset & \text{if } p_{\mathsf{P}} \stackrel{\epsilon}{\Longrightarrow}_{\mathsf{P}} p' \text{ with } p' \in F_{\mathsf{P}} \\ \{p' \in S_{\mathsf{P}} : p \stackrel{\epsilon}{\Longrightarrow}_{\mathsf{P}} p'\} & \text{otherwise} \end{cases}$
- $h_{p_{\mathsf{P}}}(to) = \begin{cases} \emptyset & \text{if } \exists p' \in h_{p_{\mathsf{P}}}(t) \text{ such that } p' \stackrel{\neg o}{\Longrightarrow}_{\mathsf{P}} p'' \text{ with } p'' \in F_{\mathsf{P}} \\ \{p'' \in S_{\mathsf{P}} : \exists p' \in h_{p_{\mathsf{P}}}(t) \cdot p' \stackrel{\neg o}{\Longrightarrow}_{\mathsf{P}} p''\} & \text{otherwise} \end{cases}$
  when $o \in \mathcal{A}^O_{\mathsf{P}}$
- $h_{p_{\mathsf{P}}}(ti) = \begin{cases} \emptyset & \text{if } \exists p' \in h_{p_{\mathsf{P}}}(t) \text{ such that } p' \stackrel{\neg i}{\Longrightarrow}_{\mathsf{P}} p'' \text{ with } p'' \in F_{\mathsf{P}}, \text{ or } p' \not\stackrel{i}{\longrightarrow}_{\mathsf{P}} \\ \{p'' \in S_{\mathsf{P}} : \exists p' \in h_{p_{\mathsf{P}}}(t) \cdot p' \stackrel{\neg i}{\Longrightarrow}_{\mathsf{P}} p''\} & \text{otherwise} \end{cases}$
  when $i \in \mathcal{A}^I_{\mathsf{P}}$.

Definition 2.19 (Safe traces) *A trace t of p_{P} is immediately-safe iff $h_{p_{\mathsf{P}}}(t) \neq \emptyset$ and is safe iff $h_{p_{\mathcal{E}(\mathsf{P})}}(t) \neq \emptyset$, where $\mathcal{E}$ propagates inconsistencies backwards over output and τ transitions. The set of immediately-safe traces of p_{P} is denoted $IST(p_{\mathsf{P}})$, while the set of safe traces is denoted $ST(p_{\mathsf{P}})$.*

An immediately-safe trace t of a process p characterises a permissible exchange between p and an arbitrary environment, such that t will never encounter an inconsistent state under any resolution of p's non-determinism. Relating this intuition to Definitions 2.18 and 2.19, suppose p and the environment can safely communicate on the trace t. If from some state that p is in after following t it can perform an output o, and every o it can output will never make the system inconsistent, then the environment must be willing to accept that output. Conversely, the environment can only safely issue an input i after t if i can be accepted from every state the process is in after following t, without making the system inconsistent. We must impose these restrictions to account for the fact that the process cannot be expected to know how to resolve its non-determinism prior to its communication with the environment.

Definition 2.20 (Model mapping) *The* model mapping *function $[\![\cdot]\!]^*$ from Logic IOLTSs to declarative specifications is defined by $[\![p_{\mathsf{P}}]\!]^* = \langle \mathcal{A}^I_{\mathsf{P}}, \mathcal{A}^O_{\mathsf{P}}, T_{[\![p_{\mathsf{P}}]\!]^*}, F_{[\![p_{\mathsf{P}}]\!]^*} \rangle$, where:*

- $T_{[\![p_{\mathsf{P}}]\!]^*} = \{t : p_{\mathsf{P}} \stackrel{t}{\Longrightarrow}_{\mathsf{P}}\} \cup F \cup FI$

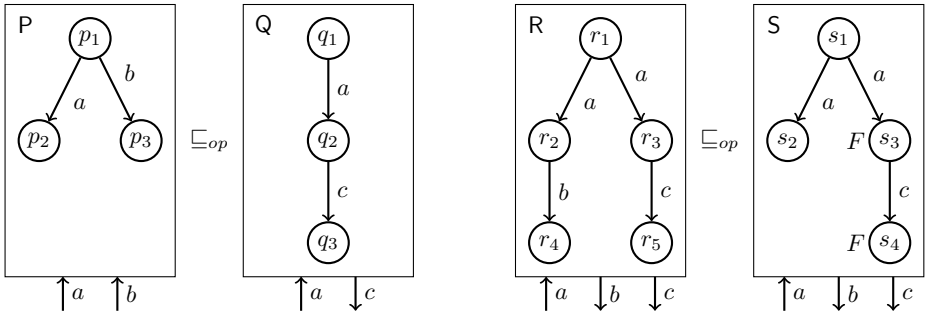

Figure 2.1: Refinement of Logic IOLTSs.

- $F_{[\![p_\mathsf{P}]\!]^*} = F \cup FI$
- $F = \{tt' : p_\mathsf{P} \stackrel{t}{\Longrightarrow}_\mathsf{P} p',\ p' \in F_\mathsf{P}$ *and* $t' \in \mathcal{A}^*_\mathsf{P}\}$
- $FI = \{tit' : p_\mathsf{P} \stackrel{t}{\Longrightarrow}_\mathsf{P} p',\ i \in \mathcal{A}^I_\mathsf{P},\ p' \stackrel{i}{\not\longrightarrow}_\mathsf{P}$ *and* $t' \in \mathcal{A}^*_\mathsf{P}\}$.

Theorem 2.21 (Model mapping preserves safe traces) *For an arbitrary process* p_P, $IST(p_\mathsf{P}) = T_{[\![p_\mathsf{P}]\!]^*} \setminus F_{[\![p_\mathsf{P}]\!]^*}$ *and* $ST(p_\mathsf{P}) = T_{[\![p_\mathsf{P}]\!]^*} \setminus F_{\mathcal{E}([\![p_\mathsf{P}]\!]^*)}$.

Having defined a mapping from operational to declarative models, we can now define operational refinement in the obvious way.

Definition 2.22 (Operational refinement) *Process* q_Q *is said to be a refinement of process* p_P*, written* $q_\mathsf{Q} \sqsubseteq_{op} p_\mathsf{P}$*, iff* $[\![q_\mathsf{Q}]\!]^* \sqsubseteq_{dec} [\![p_\mathsf{P}]\!]^*$.

Lemma 2.23 (Operational preorder) *Refinement is reflexive and transitive.*

Under the assumption of finiteness, we note that refinement checking is PSPACE-complete. This is similar to traces refinement in CSP, where the worst-case is rarely observed in practice.

Definition 2.24 (Operational equivalence) *Processes* p_P *and* q_Q *are said to be equivalent, written* $p_\mathsf{P} \equiv_{op} q_\mathsf{Q}$*, iff* $q_\mathsf{Q} \sqsubseteq_{op} p_\mathsf{P}$ *and* $p_\mathsf{P} \sqsubseteq_{op} q_\mathsf{Q}$.

Looking at the refinements in Figure 2.1, from q_1 the environment can safely issue a, after which it must be willing to accept c. Clearly a can be safely accepted by p_1, and as p_2 does not issue a c output the environment will be perfectly happy. Moreover, as the environment is not permitted to issue a b in q_1 there is no harm in p_1 being able to handle this behaviour. Hence $p_1 \sqsubseteq_{op} q_1$. Now, $r_1 \sqsubseteq_{op} s_1$ as r_1 is willing to accept the input a from the environment, which is not the case in s_1. This is because we cannot trust s_1 to resolve its non-determinism on a in an optimistic way by always moving to s_2.

Example To formally check $p_1 \sqsubseteq_{op} q_1$, it is necessary to resort to the definition of refinement on declarative specifications (Definition 2.4). It can easily be checked that all of the conditions of that definition hold by considering the sets below, obtained by computing the model mapping of the processes p_1 and q_1.

- $F_{[\![p_1]\!]^*} = F_{\mathcal{E}([\![p_1]\!]^*)} = (a+b)(a+b)^+$
- $F_{[\![q_1]\!]^*} = F_{\mathcal{E}([\![q_1]\!]^*)} = (a+ac)a(a+c)^*$
- $T_{[\![p_1]\!]^*} = (a+b)^*$
- $T_{[\![q_1]\!]^*} = \{\epsilon, a, ac\} \cup (a+ac)a(a+c)^*$
- $X \uparrow \mathcal{A}^I_\mathcal{P} = X \cup (\epsilon + a + ac + (aa + aca)(a+c)^*)b(a+b+c)^*$ for $X \in \{F_{[\![q_1]\!]^*}, T_{[\![q_1]\!]^*}\}$.

2.3.2 Error-completion

In order to simplify the definitions of the operators in our specification theory for the operational framework, we introduce the error-completion of a Logic IOLTS. This is a transformation that leaves the mapping from a Logic IOLTS to a declarative specification unchanged.

The error-completion of a Logic IOLTS provides an explicit operational representation for the inconsistent traces that would arise in mapping the Logic IOLTS to its corresponding declarative specification. Consequently, an error-completed Logic IOLTS is closed under input extensions. It is this property that simplifies the definitions of the operators in our framework. We do not say that an error-completed Logic IOLTS is input-enabled, however, as we can distinguish good inputs from bad inputs.

Definition 2.25 (Error-completion) *Let* P *be a Logic IOLTS, and assume* $f_{\mathsf{P}} \notin S_{\mathsf{P}}$*. The* error-completion *of* P *is a Logic IOLTS* $\mathsf{P}_\perp = \langle S_{\mathsf{P}_\perp}, \mathcal{A}^I_{\mathsf{P}}, \mathcal{A}^O_{\mathsf{P}}, \longrightarrow_{\mathsf{P}_\perp}, F_{\mathsf{P}_\perp}\rangle$*, where:*

- $S_{\mathsf{P}_\perp} = S_{\mathsf{P}} \cup \{f_{\mathsf{P}}\}$
- $\longrightarrow_{\mathsf{P}_\perp} = \longrightarrow_{\mathsf{P}} \cup \{(f, a, f) : f \in F_{\mathsf{P}_\perp}$ *and* $a \in \mathcal{A}_{\mathsf{P}}\} \cup \{(s, a, f_{\mathsf{P}}) : a \in \mathcal{A}^I_{\mathsf{P}}$ *and* $\nexists s' \cdot s \xrightarrow{a}_{\mathsf{P}} s'\}$
- $F_{\mathsf{P}_\perp} = F_{\mathsf{P}} \cup \{f_{\mathsf{P}}\}$.

As remarked, the error-completion of a Logic IOLTS preserves the mapping from Logic IOLTSs to declarative specifications, as the next lemma shows. Note that the corresponding declarative specifications are equal, rather than declaratively equivalent.

Lemma 2.26 (Error-completion respects mappings) *For any process* p_{P}*,* $[\![p_{\mathsf{P}}]\!]^* = [\![p_{\mathsf{P}_\perp}]\!]^*$.

Besides simplifying the definition of the compositional operators in our specification theory, error-completion of a Logic IOLTS also simplifies the definition of the model mapping function.

Lemma 2.27 (Simplified model mapping) *Let* p *be a process in Logic IOLTS* $\mathsf{P}_\perp$*. Then* $[\![p]\!]^* = \langle \mathcal{A}^I_{\mathsf{P}}, \mathcal{A}^O_{\mathsf{P}}, T_{[\![p]\!]^*}, F_{[\![p]\!]^*}\rangle$*, where:*

- $T_{[\![p]\!]^*} = \{t : p \stackrel{t}{\Longrightarrow}_{\mathsf{P}_\perp}\}$
- $F_{[\![p]\!]^*} = \{tt' : p \stackrel{t}{\Longrightarrow}_{\mathsf{P}_\perp} p' \stackrel{t'}{\Longrightarrow}_{\mathsf{P}_\perp}$ *and* $p' \in F_{\mathsf{P}_\perp}\}$.

2.3.3 Parallel composition

As for declarative specifications, the parallel composition of Logic IOLTSs yields a Logic IOLTS representing the combined effect of its operands running asynchronously. We insist that any given output should be under the control of one component only. Therefore Logic IOLTSs P and Q are composable for parallel composition only if $\mathcal{A}^O_{\mathsf{P}} \cap \mathcal{A}^O_{\mathsf{Q}} = \emptyset$.

Definition 2.28 (Parallel composition) *Let* P *and* Q *be Logic IOLTSs composable for parallel composition. Then the parallel composition of* P *and* Q *is a Logic IOLTS* $\mathsf{P} \mid\mid \mathsf{Q} = \langle S, \mathcal{A}^I, \mathcal{A}^O, \longrightarrow, F\rangle$*, where:*

- $S = S_{\mathsf{P}_\perp} \mid\mid S_{\mathsf{Q}_\perp}$
- $\mathcal{A}^I = (\mathcal{A}^I_{\mathsf{P}} \cup \mathcal{A}^I_{\mathsf{Q}}) \setminus (\mathcal{A}^O_{\mathsf{P}} \cup \mathcal{A}^O_{\mathsf{Q}})$
- $\mathcal{A}^O = \mathcal{A}^O_{\mathsf{P}} \cup \mathcal{A}^O_{\mathsf{Q}}$
- $\longrightarrow$ *is the smallest relation satisfying the following rules:*

  P1. If $p \xrightarrow{a}_{\mathsf{P}_\perp} p'$ *with* $a \in \mathcal{A}^\tau_{\mathsf{P}} \setminus \mathcal{A}_{\mathsf{Q}}$*, then* $p \mid\mid q \xrightarrow{a} p' \mid\mid q$

  P2. If $q \xrightarrow{a}_{\mathsf{Q}_\perp} q'$ *with* $a \in \mathcal{A}^\tau_{\mathsf{Q}} \setminus \mathcal{A}_{\mathsf{P}}$*, then* $p \mid\mid q \xrightarrow{a} p \mid\mid q'$

  P3. If $p \xrightarrow{a}_{\mathsf{P}_\perp} p'$ *and* $q \xrightarrow{a}_{\mathsf{Q}_\perp} q'$ *with* $a \in \mathcal{A}_{\mathsf{P}} \cap \mathcal{A}_{\mathsf{Q}}$*, then* $p \mid\mid q \xrightarrow{a} p' \mid\mid q'$.

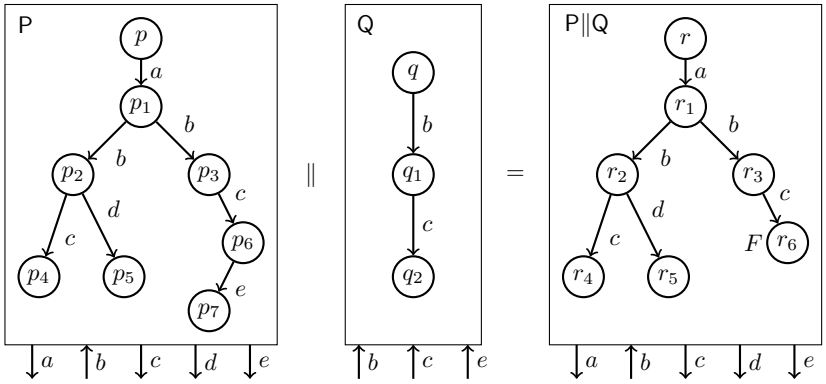

Figure 2.2: Example of parallel composition on Logic IOLTSs.

- $F = (S_{\mathsf{P}_\perp} \mid\mid F_{\mathsf{Q}_\perp}) \cup (F_{\mathsf{P}_\perp} \mid\mid S_{\mathsf{Q}_\perp})$.

Conditions P1 to P3 ensure that the parallel composition of Logic IOLTSs interleave on independent actions and synchronise on common actions. For P3, given the parallel composability constraint, synchronisation can take place between an output and an input, or two inputs. Figure 2.2 shows how the parallel composition operator works in practice, although we omit non-enabled input transitions to inconsistent states. In particular, the example demonstrates how inconsistencies can be introduced through non-input enabledness, as in state r_6 corresponding to $p_6 \mid\mid q_2$.

Reassuringly, parallel composition of Logic IOLTSs yields a Logic IOLTS. The following theorem shows the relationship between parallel composition on Logic IOLTSs and parallel composition on declarative specifications.

Theorem 2.29 (Parallel correspondences) *Let* P *and* Q *be Logic IOLTSs composable for parallel composition. For processes* p_{P} *and* q_{Q}*, it holds that* $[\![p_{\mathsf{P}} \mid\mid q_{\mathsf{Q}}]\!]^* = [\![p_{\mathsf{P}}]\!]^* \mid\mid [\![q_{\mathsf{Q}}]\!]^*$.

2.3.4 Conjunction

In keeping with conjunction of declarative specifications, Logic IOLTSs P and Q are composable for conjunction only if the sets $\mathcal{A}^I_{\mathsf{P}} \cup \mathcal{A}^I_{\mathsf{Q}}$ and $\mathcal{A}^O_{\mathsf{P}} \cup \mathcal{A}^O_{\mathsf{Q}}$ are disjoint.

Definition 2.30 (Conjunction) *Let* P *and* Q *be Logic IOLTSs composable for conjunction. Then the conjunction of* P *and* Q *is a Logic IOLTS* $\mathsf{P} \wedge \mathsf{Q} = \langle S, \mathcal{A}^I_{\mathsf{P}} \cup \mathcal{A}^I_{\mathsf{Q}}, \mathcal{A}^O_{\mathsf{P}} \cap \mathcal{A}^O_{\mathsf{Q}}, \longrightarrow, F \rangle$*, where:*

- $S = S_{\mathsf{P}_\perp} \wedge S_{\mathsf{Q}_\perp}$
- $\longrightarrow$ *is the smallest relation satisfying the following rules:*

  C1. If $a \in \mathcal{A}^O_{\mathsf{P}} \cap \mathcal{A}^O_{\mathsf{Q}}$, $p \xrightarrow{a}_{\mathsf{P}_\perp} p'$ *and* $q \xrightarrow{a}_{\mathsf{Q}_\perp} q'$*, then* $p \wedge q \xrightarrow{a} p' \wedge q'$

  C2. If $a \in \mathcal{A}^I_{\mathsf{P}} \cap \mathcal{A}^I_{\mathsf{Q}}$, $p \xrightarrow{a}_{\mathsf{P}_\perp} p'$ *and* $q \xrightarrow{a}_{\mathsf{Q}_\perp} q'$*, then* $p \wedge q \xrightarrow{a} p' \wedge q'$

  C3. If $a \in \mathcal{A}^I_{\mathsf{P}} \setminus \mathcal{A}^I_{\mathsf{Q}}$ *and* $p \xrightarrow{a}_{\mathsf{P}_\perp} p'$*, then* $p \wedge q \xrightarrow{a} p' \wedge f_{\mathsf{Q}}$

  C4. If $a \in \mathcal{A}^I_{\mathsf{Q}} \setminus \mathcal{A}^I_{\mathsf{P}}$ *and* $q \xrightarrow{a}_{\mathsf{Q}_\perp} q'$*, then* $p \wedge q \xrightarrow{a} f_{\mathsf{P}} \wedge q'$

  C5. If $p \xrightarrow{\tau}_{\mathsf{P}_\perp} p'$*, then* $p \wedge q \xrightarrow{\tau} p' \wedge q$

  C6. If $q \xrightarrow{\tau}_{\mathsf{Q}_\perp} q'$*, then* $p \wedge q \xrightarrow{\tau} p \wedge q'$

- $F = F_{\mathsf{P}_\perp} \wedge F_{\mathsf{Q}_\perp}$.

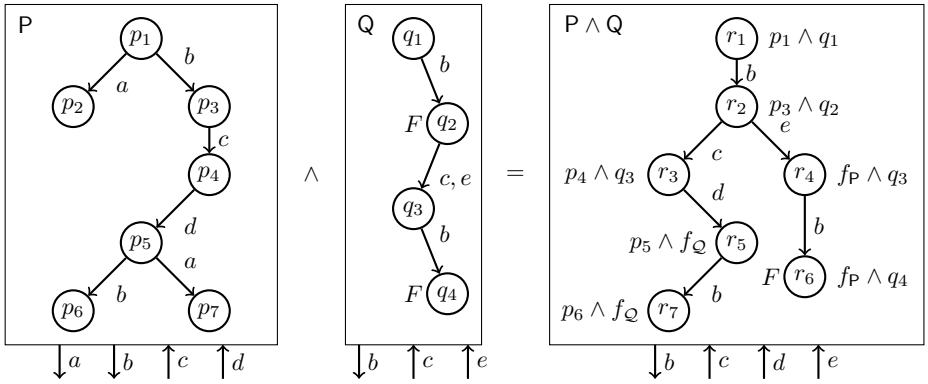

Figure 2.3: Example of conjunction on Logic IOLTSs.

The idea behind the definition of conjunction for $p \wedge q$ is that p and q must synchronise on common actions, interleave on τ-transitions, and on encountering independent input actions behave like the respective component to which the action belongs. On encountering a state $p \wedge q$ in which one of $p \in F_{\mathsf{P}}$ or $q \in F_{\mathsf{Q}}$ holds, let it be p, we know that whatever the behaviour of $p \wedge q$ it will always be a refinement of p. So the most general refinement of $p \wedge q$ will actually be q. This is supported by the fact that inconsistent states in the error-completed Logic IOLTS admit arbitrary behaviour.

Figure 2.3 shows the conjunction of processes p_1 and q_1 in the Logic IOLTSs P and Q (although for clarity we omit inputs leading to inconsistent states). In state r_1 corresponding to $p_1 \wedge q_1$, the b-output transitions of p_1 and q_1 synchronise. Independent output actions such as the a-transition in p_1 are not permitted to proceed, because it would not be the case that r_1 could be used safely in place of q_1 if this transition were to be permitted. State r_2 can evolve into r_3 by synchronising the c-inputs of p_3 and q_2, while it can also evolve into r_4 by proceeding on the independent input e of q_2. From this point, r_4 behaves like q_3, because e is an input-violation of p_3. Similar reasoning applies to r_3's evolution into r_5 by receiving the d-input.

As for parallel composition, there is a correspondence between conjunction at the operational and declarative levels.

Theorem 2.31 (Conjunction correspondences) *Let* P *and* Q *be Logic IOLTSs composable for conjunction. For processes p_{P} and q_{Q}, it holds that $[\![p_{\mathsf{P}} \wedge q_{\mathsf{Q}}]\!]^* = [\![p_{\mathsf{P}}]\!]^* \wedge [\![q_{\mathsf{Q}}]\!]^*$.*

2.3.5 Quotient

Non-determinism and τ-transitions arising in Logic IOLTSs make the definition of quotient more involved than the other operators we have considered on operational models. To ensure that the quotient is the coarsest specification, it is necessary to track the non-determinism of the system-wide specification and its partial implementation. This is because the non-determinism can affect the safe traces of a Logic IOLTS.

As for declarative specifications, we only compute the quotient of process p_{P} from r_{R} when $\mathcal{A}^O_{\mathsf{P}} \subseteq \mathcal{A}^O_{\mathsf{R}}$ and $\mathcal{A}_{\mathsf{P}} \subseteq \mathcal{A}_{\mathsf{R}}$. The quotient is the coarsest specification q over an interface consisting of inputs $\mathcal{A}^O_{\mathsf{P}} \cup \mathcal{A}^I_{\mathsf{R}}$ and outputs $\mathcal{A}^O_{\mathsf{R}} \setminus \mathcal{A}^O_{\mathsf{P}}$ such that $p_{\mathsf{P}} \mid\mid q \sqsubseteq_{op} r_{\mathsf{R}}$. If such a q exists, we denote it by $r_{\mathsf{R}}/p_{\mathsf{P}}$.

Our work in this section is more general than that of Bhaduri and Ramesh in [9], as we allow for our components to exhibit non-determinism and hidden transitions. Recall that in [9], the interface automata must be deterministic.

Before defining the quotient-construction, we introduce some functions and predicates that simplify the presentation.

Definition 2.32 *For Logic IOLTS* P*, set of states $S \subseteq S_{\mathsf{P}_\perp}$ and action $a \in \mathcal{A}_{\mathsf{P}}$, define:*

- $\mathsf{succ}^{\epsilon}_{\mathsf{P}}(S) = \{s' : s \stackrel{\epsilon}{\Longrightarrow}_{\mathsf{P}_\perp} s'$ *with* $s \in S\}$

- $\mathrm{succ}^a_{\mathsf{P}}(S) = \{s' : s \stackrel{\dashv a}{\Longrightarrow}_{\mathsf{P}_\perp} s'$ *with* $s \in S\}$.

Definition 2.33 (Quotient Logic IOLTS) *Let* P *and* R *be Logic IOLTSs such that* $\mathcal{A}^O_{\mathsf{P}} \subseteq \mathcal{A}^O_{\mathsf{R}}$ *and* $\mathcal{A}_{\mathsf{P}} \subseteq \mathcal{A}_{\mathsf{R}}$. *The* quotient *of* P *from* R *is the Logic IOLTS* $\mathsf{R/P} = \langle S_{\mathsf{R/P}}, \mathcal{A}^I_{\mathsf{R/P}}, \mathcal{A}^O_{\mathsf{R/P}}, \longrightarrow, F_{\mathsf{R/P}} \rangle$, *where:*

- $S_{\mathsf{R/P}} = \{R/P : R \subseteq S_{\mathsf{R}_\perp}$ *and* $P \subseteq S_{\mathsf{P}_\perp}\}$
- $\mathcal{A}^I_{\mathsf{R/P}} = \mathcal{A}^O_{\mathsf{P}} \cup \mathcal{A}^I_{\mathsf{R}}$
- $\mathcal{A}^O_{\mathsf{R/P}} = \mathcal{A}^O_{\mathsf{R}} \setminus \mathcal{A}^O_{\mathsf{P}}$
- $\longrightarrow$ *is the smallest relation satisfying the following rules:*

  Q1. $R'/P' \stackrel{a}{\longrightarrow} \mathrm{succ}^a_{\mathsf{R}}(R')/\mathrm{succ}^a_{\mathsf{P}}(P')$ *providing:*
  - *(a)* $a \in \mathcal{A}^I_{\mathsf{P}} \cap \mathcal{A}^I_{\mathsf{R}}$ *implies* $\mathrm{succ}^a_{\mathsf{R}}(R') \cap F_{\mathcal{E}(\mathsf{R})} = \emptyset$
  - *(b)* $a \in \mathcal{A}^O_{\mathsf{P}} \cap \mathcal{A}^O_{\mathsf{R}}$ *implies* $\mathrm{succ}^a_{\mathsf{R}}(R') \cap F_{\mathcal{E}(\mathsf{R})} = \emptyset$ *and* $\mathrm{succ}^a_{\mathsf{P}}(P') \neq \emptyset$
  - *(c)* $a \in \mathcal{A}^I_{\mathsf{P}} \cap \mathcal{A}^O_{\mathsf{R}}$ *implies* $\mathrm{succ}^a_{\mathsf{P}}(P') \cap F_{\mathsf{P}} = \emptyset$ *and* $\mathrm{succ}^a_{\mathsf{R}}(R') \neq \emptyset$

  Q2. $R'/P' \stackrel{a}{\longrightarrow} \mathrm{succ}^a_{\mathsf{R}}(R')/P'$ *providing:*
  - *(a)* $a \in \mathcal{A}^I_{\mathsf{R}} \setminus \mathcal{A}_{\mathsf{P}}$ *implies* $\mathrm{succ}^a_{\mathsf{R}}(R') \cap F_{\mathcal{E}(\mathsf{R})} = \emptyset$
  - *(b)* $a \in \mathcal{A}^O_{\mathsf{R}} \setminus \mathcal{A}_{\mathsf{P}}$ *implies* $\mathrm{succ}^a_{\mathsf{R}}(R') \neq \emptyset$.
- $R'/P' \in F_{\mathsf{R/P}}$ *iff at least one of the following rules holds:*

  F1. $R' = \emptyset$ *or* $P' = \emptyset$

  F2. $F_{\mathcal{E}(\mathsf{R}_\perp)} \cap R' \neq \emptyset$ *or* $F_{\mathsf{P}_\perp} \cap P' \neq \emptyset$

  F3. $R'/P' \stackrel{a}{\longrightarrow} R''/P''$ *with* $a \in \mathcal{A}^I_{\mathsf{R/P}}$ *and* $R''/P'' \in F_{\mathsf{R/P}}$.

Definition 2.34 (Quotient) *Let* P *and* R *be Logic IOLTSs such that* $\mathcal{A}^O_{\mathsf{P}} \subseteq \mathcal{A}^O_{\mathsf{R}}$ *and* $\mathcal{A}_{\mathsf{P}} \subseteq \mathcal{A}_{\mathsf{R}}$. *The* quotient *of process* p_{P} *from process* r_{R}, *written* $r_{\mathsf{R}}/p_{\mathsf{P}}$, *is the process* $\mathrm{succ}^\epsilon_{\mathsf{R}}(r_{\mathsf{R}})/\mathrm{succ}^\epsilon_{\mathsf{P}}(p_{\mathsf{P}})$ *in the Logic IOLTS* $\mathsf{R//P}$ *obtained from* $\mathsf{R/P}$ *by removing all transitions immediately leading to a state in* $F_{\mathsf{R/P}}$, *and removing all states* R/P *such that* $R/P \in F_{\mathsf{R/P}}$ *and* $R \notin F_{\mathcal{E}(\mathsf{R})}$. *If* $\mathrm{succ}^\epsilon_{\mathsf{R}}(r_{\mathsf{R}})/\mathrm{succ}^\epsilon_{\mathsf{P}}(p_{\mathsf{P}})$ *is not contained in* $\mathsf{R//P}$, *then the quotient is not defined.*

As for declarative specifications, the quotient of p_{P} from r_{R} may not exist. The following theorem shows that definedness of the quotient according to the previous definition coincides precisely with the existence of such a quotient.

Theorem 2.35 (Existence of quotient) *Let* P *and* R *be Logic IOLTSs such that* $\mathcal{A}^O_{\mathsf{P}} \subseteq \mathcal{A}^O_{\mathsf{R}}$ *and* $\mathcal{A}_{\mathsf{P}} \subseteq \mathcal{A}_{\mathsf{R}}$. *Then* $r_{\mathsf{R}}/p_{\mathsf{P}}$ *is defined (i.e.* $r_{\mathsf{R}} \in F_{\mathcal{E}(\mathsf{R})}$ *or* $r_{\mathsf{R}}/p_{\mathsf{P}} \notin F_{\mathsf{R/P}}$*) iff there exists a process* q *in a Logic IOLTS with inputs* $\mathcal{A}^I_{\mathsf{R/P}}$ *and outputs* $\mathcal{A}^O_{\mathsf{R/P}}$ *such that* $p_{\mathsf{P}} \parallel q \sqsubseteq_{op} r_{\mathsf{R}}$.

Consequently, the constraint $r_{\mathsf{R}} \notin F_{\mathcal{E}(\mathsf{R})}$ and $r_{\mathsf{R}}/p_{\mathsf{P}} \in F_{\mathsf{R/P}}$ gives a precise characterisation of whether the quotient exists or not. When the quotient does exist, it behaves in exactly the same way as for declarative specifications.

Theorem 2.36 (Quotient correspondences) *Let* P *and* R *be Logic IOLTSs such that* $\mathcal{A}^O_{\mathsf{P}} \subseteq \mathcal{A}^O_{\mathsf{R}}$ *and* $\mathcal{A}_{\mathsf{P}} \subseteq \mathcal{A}_{\mathsf{R}}$. *If* $r_{\mathsf{R}}/p_{\mathsf{P}} \notin F_{\mathsf{R/P}}$ *or* $r_{\mathsf{R}} \in F_{\mathcal{E}(\mathsf{R})}$, *then* $[\![r_{\mathsf{R}}/p_{\mathsf{P}}]\!]^* = [\![r_{\mathsf{R}}]\!]^*/[\![p_{\mathsf{P}}]\!]^*$.

Figure 2.4 provides an example in which processes p_1 and r_1 have no quotient. This tallies with Theorem 2.35, as we have $\{r_1\}/\{p_1\} \in F_{\mathsf{R/P}}$ when $r_1 \notin F_{\mathcal{E}(\mathsf{R})}$. On the other hand, quotients exist for the processes p_1 and r_1 of Figures 2.5 and 2.6. This is also supported by Theorem 2.35, as $\{r_1\}/\{p_1\} \notin F_{\mathsf{R/P}}$ for the processes in both figures.

For Figure 2.6, the quotient is the single consistent state $\{r_1\}/\{p_1\}$. This is because in going from $\mathsf{R/P}$ to $\mathsf{R//P}$ we remove the transition labelled by the output a between the processes $\{r_1\}/\{p_1\}$ and $\{r_2\}/\{p_2, p_3\}$, as the latter state is inconsistent. Maintaining this transition would yield an invalid quotient as $p_1 \parallel (r_1/p_1)$ would be inconsistent when r_1 is consistent. It is safe to discard this transition only because it is an output. Recalling the definition of $\sqsubseteq_{op}$, for safe-substitutivity it is perfectly safe to suppress outputs on the left that would have occurred on the right.

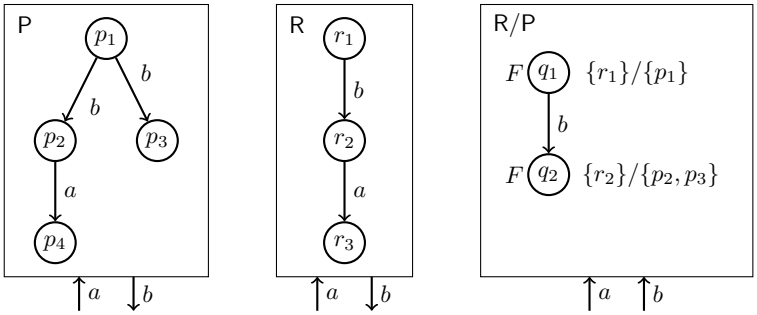

Figure 2.4: Example showing non-existence of quotient on Logic IOLTSs.

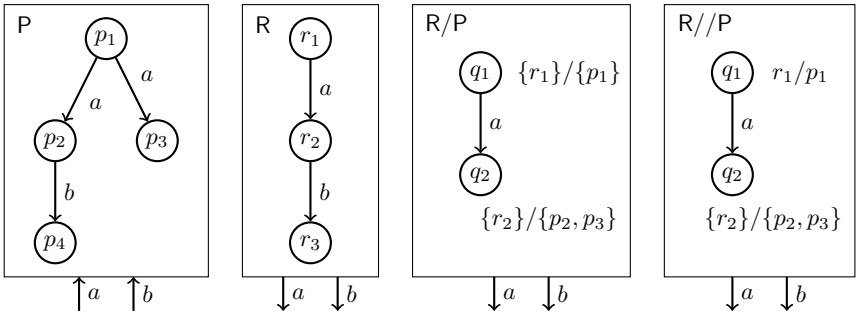

Figure 2.5: Example of quotient on Logic IOLTSs with no inconsistencies.

2.4 Summary and future work

We have developed a compositional specification theory for components that may be modelled operationally, closely mirroring actual implementations, or in an abstract manner by means of declarative specifications. Both frameworks admit a simple refinement relation, defined in terms of traces, which corresponds to safe-substitutivity. We define asynchronous parallel composition, conjunction and quotient, and prove that the induced equivalence is a congruence for these operations. It is straightforward to extend our framework with disjunction and hiding. CONNECT-centric examples demonstrating our specification theory are provided in Chapter 3.

The simplicity of our formalism facilitates reasoning about the temporal ordering of interactions needed for assume-guarantee inference. We are in the process of formulating an AG framework for reasoning compositionally about components. This will support the preservation of safety and liveness properties under the operations of the specification theory. As the original framework only deals with finite-executions, the form of liveness to be considered will be based on quiescence.

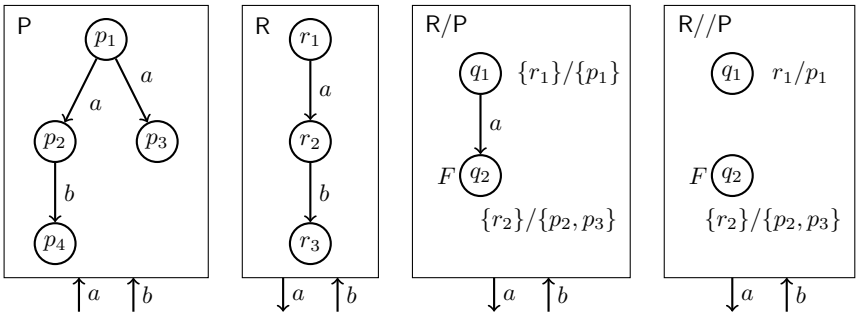

Figure 2.6: Example of quotient on Logic IOLTSs with inconsistencies.

3 Integration of the specification theory and synthesis

In this chapter we relate the specification theory introduced in Chapter 2 with the task of generating connectors. The latter must facilitate the successful interaction of multiple components that need to communicate with one another. The components are permitted to have incompatible behaviours and to utilise different interaction vocabularies. Therefore, to ensure that the connector generated is meaningful, we assume there is an ontology relating the actions of the components that need to communicate. Such an ontology can tell us about equivalence of actions, along with ordering constraints that the actions must satisfy.

The purpose of the connector is to resolve the communication mismatches inherent from the differing characteristics of the components. This essentially means that the connector functions as a mediator. The mediators generated by our theory are free from communication mismatches, but they also ensure that progress is made so that meaningful interaction takes place. Given a collection of components, our theory also allows one to deduce whether a mediator can exist or not.

3.1 Connector synthesis as quotient

As we are interested in using our theory developed in Chapter 2, we assume that the components needing to communicate are modelled as either declarative specifications or Logic IOLTSs (the choice is largely indifferent). The synthesis approach we document is applicable to any number of components needing to communicate, so for simplicity we assume that there are exactly two, which we denote by $\mathcal{P}$ and $\mathcal{Q}$.

A mediator $\mathcal{M}$ for the successful interaction of $\mathcal{P}$ and $\mathcal{Q}$ can be found as a solution to the equation $\mathcal{P} \parallel \mathcal{M} \parallel \mathcal{Q} \sqsubseteq \mathcal{G}$, where $\mathcal{G}$ is some specification stating what it means for $\mathcal{P}$ and $\mathcal{Q}$ to interact successfully, and $\sqsubseteq$ is the refinement relation defined in Chapter 2 (it is either $\sqsubseteq_{dec}$ or $\sqsubseteq_{op}$ depending on the model types). Thus, using the theory of Sections 2.2.4 and 2.3.5, it follows that the most general $\mathcal{M}$ can be found by quotient as $\mathcal{G}/(\mathcal{P} \parallel \mathcal{Q})$. Of course, as quotient is a partial operation, definedness of $\mathcal{M}$ is not guaranteed.

A problem with this formulation in CONNECT concerns the derivation of the goal $\mathcal{G}$. Conceivably, $\mathcal{G}$ would deal with application-specific knowledge, but in CONNECT this is not generally available, other than in the form of the ontology that specifies relationships between interaction primitives. We therefore consider the task of formulating an abstract goal that specifies the general principles of interaction that we are interested in. Specifically, the goal $\mathcal{G}$ must ensure that the synthesised mediator $\mathcal{M}$ satisfies the following properties:

G1. $\mathcal{P} \parallel \mathcal{M} \parallel \mathcal{Q}$ is not permitted to generate any communication inconsistencies

G2. $\mathcal{P} \parallel \mathcal{M} \parallel \mathcal{Q}$ is only permitted to deadlock in composite states where all three of $\mathcal{P}$, $\mathcal{M}$ and $\mathcal{Q}$ are permitted to deadlock

G3. $\mathcal{P} \parallel \mathcal{M} \parallel \mathcal{Q}$ must satisfy the constraints imposed by the ontology.

The following sections explain precisely how we can synthesise a mediator to satisfy these properties by means of quotient.

3.2 Capturing ontological constraints

Given components $\mathcal{P}$ and $\mathcal{Q}$ needing to communicate, we assume that there is an ontology that relates actions of $\mathcal{P}$ with actions of $\mathcal{Q}$ and vice versa. This assumption is widespread through CONNECT, otherwise it is not possible to generate a connector that respects the semantics of the components it interfaces with. In our work for WP2, we assume that the ontology can specify two kinds of relationship on the communication primitives of the components:

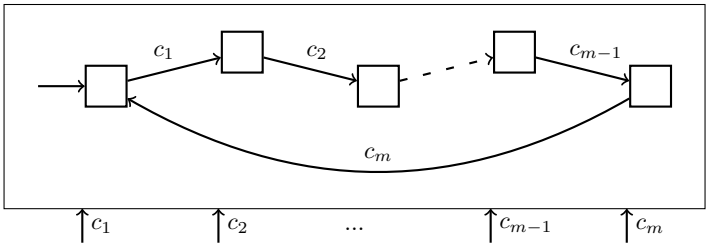

Figure 3.1: Component denoting $Seq(c_1 \dots c_m)$

1. **Equivalence of primitives.** The ontology tells us if a primitive in one component is semantically equivalent to a primitive in the other component; that is, equivalent actions are supposed to implement corresponding functionality. We extend these equivalences by allowing a primitive in one component to correspond to a sequence of primitives in the other; this supports fragmentation of messages. As an example, the ontology could specify that $a \equiv b$ and $c \equiv d_1 d_2 d_3$.
2. **Ordering of primitives.** The ontology further tells us about the ordering of primitives that the composition of the components and mediator must respect. This allows us to enforce constraints of causality (since we do not explicitly model data, causality can not be inferred by dataflow constraints or similar). For example, an ack in $\mathcal{Q}$ should only be issued after a $send$ has been emitted by $\mathcal{P}$.

For each relationship in the ontology, we construct an observation component that has as interface the collection of primitives that appear in the relationship. As we treat the component as an observer, all of the actions are considered to be inputs. The behaviour of the observer respects the intuition of the relationship (we explain how the behaviour is derived in the points below). If the mediator or a component violates a constraint, the corresponding observer will generate an inconsistency. The component $\mathcal{O}$ respecting the combined effect of all of the ontological constraints is then defined as the parallel composition of the representations of the individual relationships. Note that this is always defined.

- **Modelling equivalences.** Let $a \equiv b_1 \dots b_n$ be any equivalence relationship in the ontology (for $n \geq 1$), where it is assumed without loss of generality that a is a primitive of $\mathcal{P}$ and $b_1, \dots, b_n$ are primitives of $\mathcal{Q}$. We require that $b_1, \dots, b_n$ are either all inputs in $\mathcal{Q}$ or are all outputs in $\mathcal{Q}$. We further require that exactly one of a and b_1 is an output in the respective alphabets of $\mathcal{P}$ and $\mathcal{Q}$. Now, if $a \in \mathcal{A}^O_\mathcal{P}$ and $b_1 \in \mathcal{A}^I_\mathcal{Q}$, then $a \equiv b_1 \dots b_n$ is represented by $Seq(ab_1 \dots b_n)$. Otherwise, if $a \in \mathcal{A}^I_\mathcal{P}$ and $b_1 \in \mathcal{A}^O_\mathcal{Q}$, then $a \equiv b_1 \dots b_n$ is represented by $Seq(b_1 \dots b_n a)$.
- **Modelling orderings.** Let $a_1 \mathsf{precedes} a_2 \mathsf{precedes} \dots \mathsf{precedes} a_n$ be an ordering relationship in the ontology. Then the representation of this ordering is given by $Seq(a_1 a_2 \dots a_n)$.

We model an arbitrary observer $Seq(c_1 \dots c_m)$ as a component in our framework. Its behaviour is characterised by Figure 3.1. In practice, other constraints besides equivalence and ordering of primitives can be represented by observers. Although we do not consider them in this report, they can be handled by our theory in the same way.

3.3 Modelling deadlock

A trace t of a declarative specification $\mathcal{P}$ is said to be quiescent if the component cannot produce further output without receiving stimulation from the environment. Thus t is quiescent if $t \in T_\mathcal{P}$ and $o \in \mathcal{A}^O_\mathcal{P}$ implies $to \notin T_\mathcal{P}$. In the case of Logic IOLTSs, a state p_P is quiescent if $o \in \mathcal{A}^O_\mathsf{P}$ implies $p_\mathsf{P} \stackrel{o}{\not\Longrightarrow}_\mathsf{P}$. A trace t of p_P is quiescent if there exists some quiescent state p_P' such that $p_\mathsf{P} \stackrel{t}{\Longrightarrow}_\mathsf{P} p_\mathsf{P}'$. Thus quiescence on non-deterministic systems has may-like semantics.

We use the concept of quiescence to indicate when a component is deadlocked. Formally, a component can deadlock after a trace t if t is a quiescent trace of the component. Therefore, a deadlocked component is one that can become blocked waiting for input.

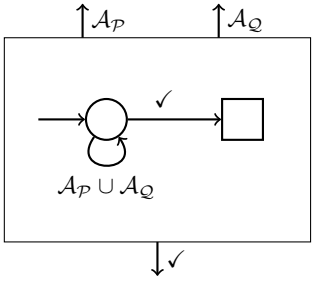

Figure 3.2: Logic IOLTS representing the goal $\mathcal{G}$

Yet, deadlock is not all bad news. For example, deadlock could signify successful termination of a component, or could indicate that the component requires interaction from the environment in order to proceed. We therefore classify deadlocked states as either good or bad, depending on whether we're happy for deadlock to occur or not.

Our synthesis framework ensures that the components and mediator will always make progress (i.e. will not deadlock), unless all of the components deadlock in a good way (desideratum G2). This means that we need a way of distinguishing good and bad deadlock in a component. To do this, we introduce a ✓ action that appears as an input in the interface of each component. Like in CSP, a ✓ action is used to signify successful termination. The behaviour of each component is extended so that a ✓ action can be observed at any point where the component is permitted to deadlock in a good state.

The mediator will have ✓ as an output action. This means that the mediator is only able to issue the ✓ in a composite state where all of the components are willing to receive it, otherwise an inconsistency will occur. The checking of deadlock follows implicitly from the synthesis procedure with quotient, after devising a suitable goal.

Note that this notion of progress is slightly weaker than that modelled by, e.g., liveness in the sense of linear temporal logic, since infinite computations are not required to achieve some particular goal. However, for components (or systems) that exhibit only finite computations, quiescence is a perfectly adequate way to model liveness.

3.4 Formulating the goal

We begin by assuming that $\mathcal{P} \mid\mid \mathcal{M} \mid\mid \mathcal{Q}$ is intended to be a closed system. This means that there are no inputs in the interface of $\mathcal{P} \mid\mid \mathcal{M} \mid\mid \mathcal{Q}$. In this setting, we take as the goal a general specification $\mathcal{G}$ that states "no inconsistency will arise" and "deadlock can only occur after a ✓ has been seen". The goal can thus only become quiescent after seeing a ✓ action.

This can be formulated as the declarative specification $\langle \mathcal{A}^I, \mathcal{A}^O, T, F, Q \rangle$, where:

- $\mathcal{A}^I = \emptyset$
- $\mathcal{A}^O = \mathcal{A}_\mathcal{P} \cup \mathcal{A}_\mathcal{Q} \cup \{\checkmark\}$
- $T = (\mathcal{A}^O \setminus \{\checkmark\})^*(\epsilon + \checkmark)$
- $F = \emptyset$
- $Q = (\mathcal{A}^O \setminus \{\checkmark\})^*\checkmark$.

A pictorial representation of the goal as a Logic IOLTS is provided in Figure 3.2.

In the case that $\mathcal{P} \mid\mid \mathcal{M} \mid\mid \mathcal{Q}$ is an open system, the goal would be slightly more involved, as it would have to interleave the interactions with the environment. However, If we have a model of the environment $\mathcal{E}$, we can use the same goal by considering $\mathcal{P} \mid\mid \mathcal{M} \mid\mid \mathcal{Q} \mid\mid \mathcal{E}$, where we assume that $\mathcal{E}$ is just another component.

If we do not have a model of the environment, it is not possible to guarantee that the mediator can avoid inconsistencies. This can only be done if the environment issues outputs only at the times when the components are willing to receive them. The most general safe environment will always issue an output, whenever the corresponding components are willing to receive it. If we wish to synthesise the mediator for the most general safest environment, this can be done by transforming the interfaces of $\mathcal{P}$ and $\mathcal{Q}$ such that open inputs become outputs. In this case, the parallel composition of the transformed components and the mediator would be a closed system, and so we can use the standard goal $\mathcal{G}$.

3.5 The synthesis procedure

We can now state how to synthesise the mediator. Explanation on how our synthesis procedure relates to the synthesis algorithm in WP3 [23] is provided in Section 3.8.

Given the goal $\mathcal{G}$ and ontological constraint $\mathcal{O}$, we formulate the mediator synthesis problem as finding a most general mediator $\mathcal{M}$ such that $\mathcal{P} \mid\mid \mathcal{M} \mid\mid \mathcal{O} \mid\mid \mathcal{Q} \sqsubseteq \mathcal{G}$. In our theory, $\mathcal{M}$ can be found as $\mathcal{G}/(\mathcal{P} \mid\mid \mathcal{Q} \mid\mid \mathcal{O})$, as per Definitions 2.12 and 2.33.

To check that the synthesised mediator $\mathcal{M}$ satisfies our desiderata, suppose that $\mathcal{P} \mid\mid \mathcal{M} \mid\mid \mathcal{Q}$ raises an inconsistency. Then certainly $\mathcal{P} \mid\mid \mathcal{M} \mid\mid \mathcal{Q} \mid\mid \mathcal{O}$ generates an inconsistency, but this means that $\mathcal{P} \mid\mid \mathcal{M} \mid\mid \mathcal{Q} \mid\mid \mathcal{O} \not\sqsubseteq \mathcal{G}$ on account of inconsistency containment as $\mathcal{G}$ has no inconsistencies. Therefore the quotient technique would not have generated $\mathcal{M}$.

Instead suppose that $\mathcal{P} \mid\mid \mathcal{M} \mid\mid \mathcal{Q}$ deadlocks at a point where we do not allow it to. Then $\mathcal{P} \mid\mid \mathcal{M} \mid\mid \mathcal{Q}$ is in a quiescent state and a ✓ action has not been seen. Then by the same reasoning as for inconsistencies, we know $\mathcal{P} \mid\mid \mathcal{M} \mid\mid \mathcal{O} \mid\mid \mathcal{Q}$ must deadlock in a quiescent state (as $\mathcal{O}$ is an observer so only deals with input actions) and a ✓ could not have been encountered as all components must synchronise on the ✓ action. But then $\mathcal{P} \mid\mid \mathcal{M} \mid\mid \mathcal{Q} \mid\mid \mathcal{O} \not\sqsubseteq \mathcal{G}$ because of quiescence containment, and so $\mathcal{M}$ would not have been synthesised by our quotienting technique.

Finally, suppose that $\mathcal{P} \mid\mid \mathcal{M} \mid\mid \mathcal{Q}$ does not respect one of the ontological constraints represented by $Seq(c_1 \dots c_m)$. If $\{c_1, \dots, c_m\} = A$, then there is a trace t of $\mathcal{P} \mid\mid \mathcal{M} \mid\mid \mathcal{Q}$ such that $t \upharpoonright A$ becomes inconsistent in $Seq(c_1 \dots c_m)$. Let $\mathcal{A}_\mathcal{O}$ be the set of actions arising in $\mathcal{O}$, then $t \upharpoonright \mathcal{A}_\mathcal{O}$ is inconsistent in $\mathcal{O}$. Hence t is inconsistent in $\mathcal{P} \mid\mid \mathcal{M} \mid\mid \mathcal{O} \mid\mid \mathcal{Q}$. But then it must be the case that $\mathcal{P} \mid\mid \mathcal{M} \mid\mid \mathcal{O} \mid\mid \mathcal{Q} \not\sqsubseteq \mathcal{G}$ because of inconsistency containment. Consequently, $\mathcal{M}$ could not have been synthesised by our quotienting procedure.

As a result, if there is a mediator $\mathcal{M}$ such that $\mathcal{P} \mid\mid \mathcal{M} \mid\mid \mathcal{Q}$ satisfies the desiderata we outlined in Section 3.1, then our algorithm based on quotient is guaranteed to find it. Moreover, by the Theorems in Sections 2.2.4 and 2.3.5, the mediator will always be the most general.

3.6 A simple messaging example

We begin by demonstrating our mediator synthesis procedure on a simple example, which is a variant of the instant messaging illustration we first considered in [4]. Our revised example is set within a humble train station. A *despatching system* holds details of the trains at the station, together with each train's destination list and departure platform. A number of dissemination systems interface with the despatcher to keep passengers informed about the status of each train. Such systems include electronic display boards and automated announcers. It is our intention to synthesise a connector for allowing a departure board to display information about a particular train.

Figure 3.3 models the behaviour of the despatcher, while Figure 3.4 models the behaviour of the departure board controller. Both models take the form of Logic IOLTSs, where square nodes are quiescent and circular nodes are non-quiescent. As the models have disjoint communication vocabularies, they are unable to interoperate with one another. Fortunately, however, an ontology tells us the relationships between the actions:

- $req \equiv hand$
- $req\_ack \equiv auth\_ok$

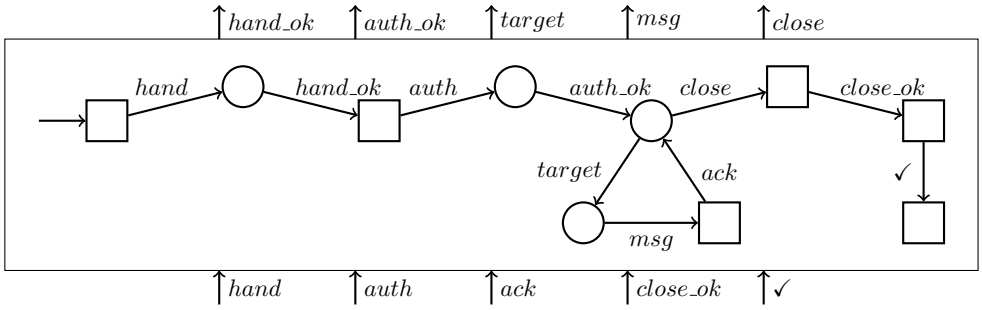

Figure 3.3: Model of the despatcher.

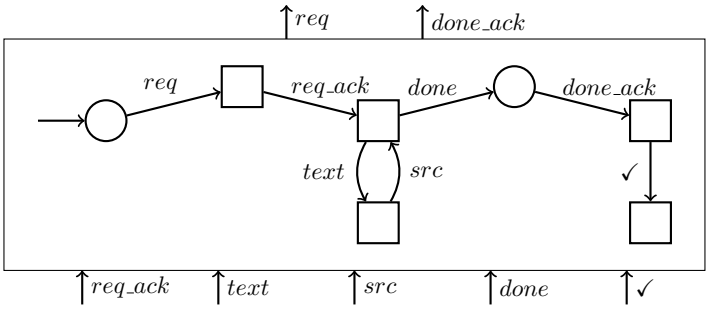

Figure 3.4: Model of the departure board controller.

- $done \equiv close$
- $done\_ack \equiv close\_ok$
- $text \equiv msg$
- $src \equiv target$.

Considering whether these actions are inputs or outputs in the respective models, we can derive a component $\mathcal{O}$ representing the ontological constraints as the parallel composition of the following Seq primitives:

- $Seq(req, hand)$
- $Seq(auth\_ok, req\_ack)$
- $Seq(close, done)$
- $Seq(done\_ack, close\_ok)$
- $Seq(msg, text)$
- $Seq(target, src)$.

Figure 3.5 shows the parallel composition of the despatcher, the display and the ontological constraints component $\mathcal{O}$. As $\mathcal{O}$ is an observer, meaning that all actions are inputs, the effect of taking $Despatcher \mid\mid Display \mid\mid \mathcal{O}$ is to make any path in $Despatcher \mid\mid Display$ inconsistent if it violates the ordering constraints of $\mathcal{O}$. Thus in Figure 3.5, inconsistencies arise whenever a second $target$ is observed before seeing a src action, as this violates the constraint $Seq(target, src)$.

Now by the quotienting procedure, we derive the mediator $\mathcal{M}$ as $\mathcal{G}/(Despatcher \mid\mid Display \mid\mid \mathcal{O})$, where $\mathcal{G}$ is given as defined in Section 3.4, taking $\mathcal{A}^O = \mathcal{A}_{Despatcher} \cup \mathcal{A}_{Display}$. The resulting $\mathcal{M}$ is shown in Figure 3.6.

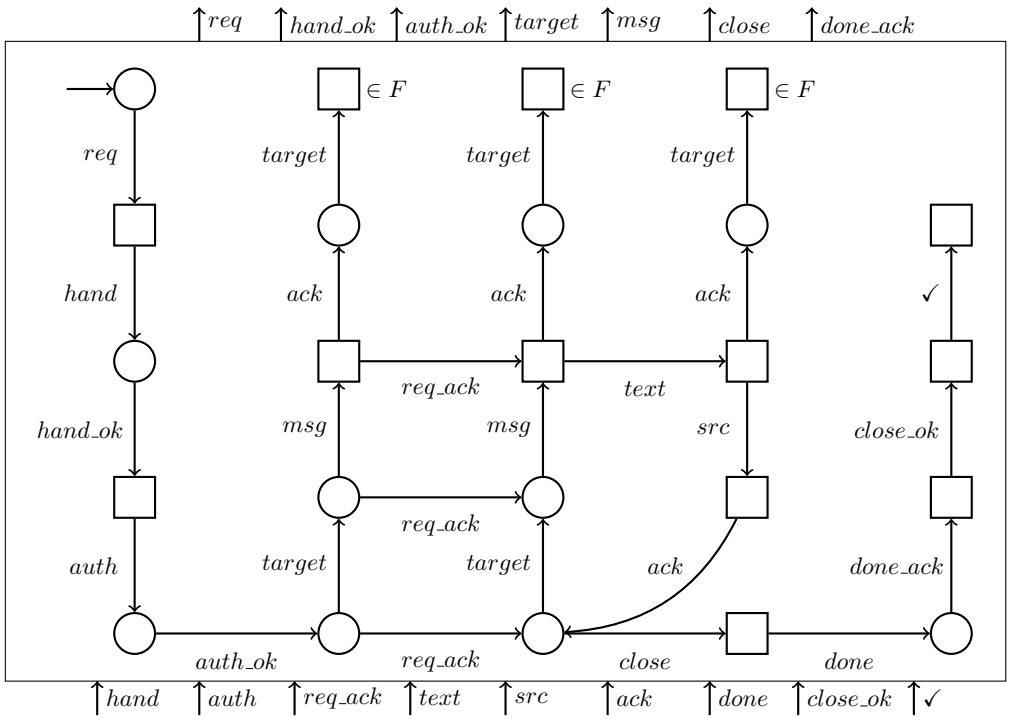

Figure 3.5: The parallel composition of the despatcher, the display and $\mathcal{O}$.

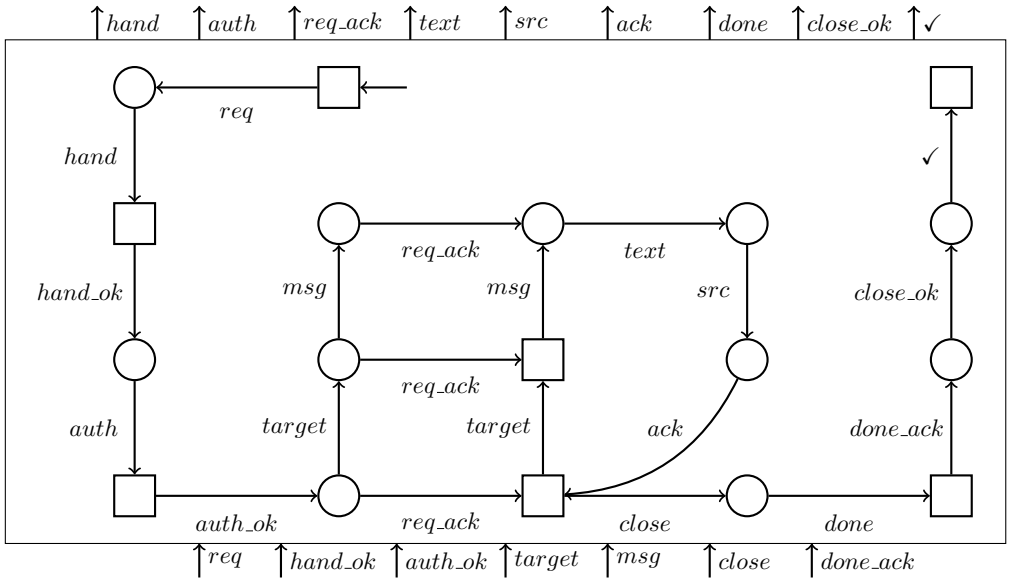

Figure 3.6: The quotient-synthesised connector.

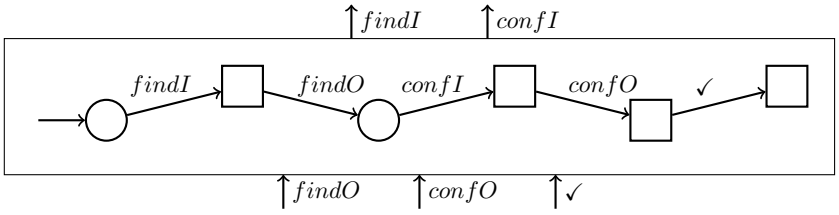

Figure 3.7: Model of the US client: US.

3.7 Example based on booking a packaged holiday

We now consider a slightly more involved example to demonstrate our synthesis procedure on that was first presented in the CONNECT publication [10]. The example involves two travel agencies: one based in the US and the other in the EU. The travel agencies have different booking systems, the technicalities being irrelevant for us in this section, although the full details can be consulted in [10]. We assume the existence of an ontology that relates the domain-specific concepts of the systems.

A client of the US system needs to interface with the EU booking system. It is thus our intention to synthesise a mediator that will allow the client to interact with the service. An added complication of this scenario is that the models of the clients and services deal with data. We show how we can encode data constraints within our notation so that the synthesised mediator respects the principles of data flow.

In keeping with the notation of WP3, we assume the transitions of models are labelled as:

- (op, In, Out), where op is a receptive operation, In is a list of data items that should be input and Out is a list of data items that should be output; or
- $(\overline{op}, In, Out)$, where $\overline{op}$ is an active operation, In is a list of data items that are to be output, and Out is a list of data items to be input.

Our setup in WP2 is unable to cope with these structured transitions atomically, so we give a transformation that respects the semantics. Each transition labelled with (op, In, Out) is represented by two consecutive transitions labelled $opI?In$ and $opO!Out$. On the other hand, each transition labelled $(\overline{op}, In, Out)$ is represented by two consecutive transitions labelled by $opI!In$ and $opO?Out$. Note that the $?$ and ! symbols indicate the I/O type of the actions opI and opO.

Modelling the components. We model the components (i.e. the US client and EU service) by Logic IOLTSs. Figure 3.7 shows the US client, which first *finds* a complete holiday and then *confirms* it. The model for the EU service is too cumbersome to draw, so we give a characterisation in terms of the following LTSA description:

```
EUF = (selFlightI -> selFlightO -> makeResI -> makeResO -> END).
EUH = (selHotelI -> selHotelO -> makeResI -> makeResO -> END).
EUC = (selCarI -> selCarO -> makeResI -> makeResO -> END).
||EU = (EUF || EUH || EUC).
```

The EU service requires the user to select a flight, hotel and car in any order, before being permitted to make a reservation. Actions ending in an I are treated as inputs, while actions ending in an O are treated as outputs.

Ontological constraints. The ontology states that a find trip request $findI$ must take place before the constituent parts of the trip can be selected ($selFlightI$, $selCarI$ and $selHotelI$). Moreover, the results of finding a trip $findO$ can only be returned once the flight, car and hotel have been selected ($selFlightO$, $selCarO$ and $selHotelO$ have been seen). Confirming a trip $confI$ on the US client corresponds to making a reservation $makeResI$ on the EU booking service, and the reservation acknowledgement $makeResO$ of

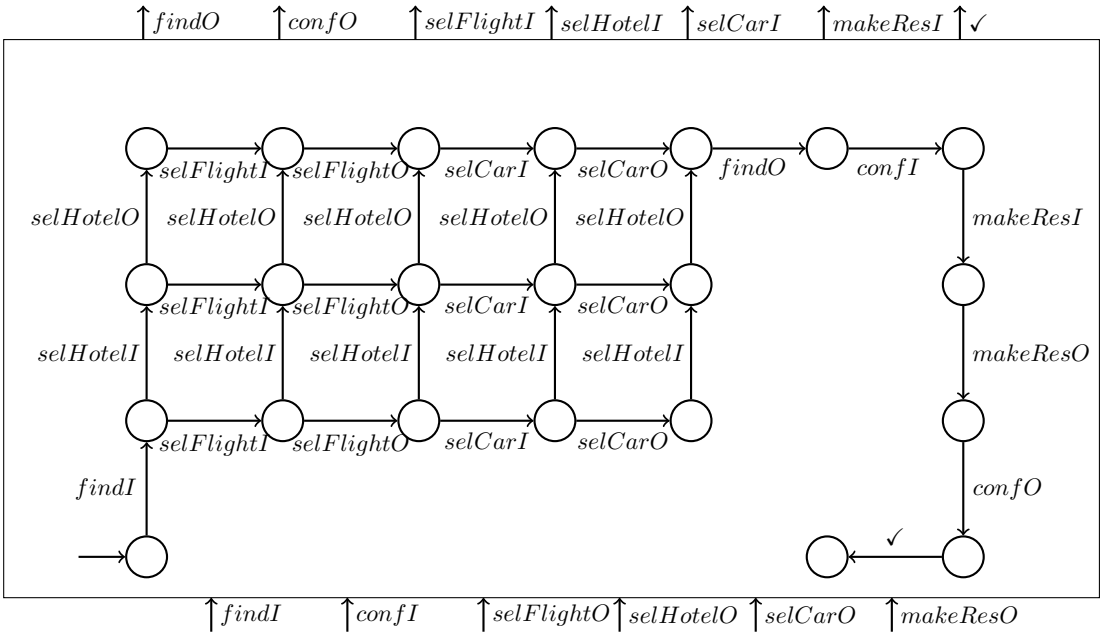

Figure 3.8: The mediator for the holiday booking example.

the EU service corresponds to the US trip confirmation $confO$. A further constraint states that a car can only be hired from the airport that the flight goes to. Thus, the flight result $selFlightO$, is required before the car can be selected $selCarI$.

The following points formally state the ontological constraints as according to the preceding text, together with the component representation:

1. $findI$ precedes $selFlightI$, $Seq(findI, selFlightI)$
2. $findI$ precedes $selHotelI$, $Seq(findI, selHotelI)$
3. $findI$ precedes $selCarI$, $Seq(findI, selCarI)$
4. $selFlightO$ precedes $findO$, $Seq(selFlightO, findO)$
5. $selHotelO$ precedes $findO$, $Seq(selHotelO, findO)$
6. $selCarO$ precedes $findO$, $Seq(selCarO, findO)$
7. $makeResI \equiv confI$, $Seq(confI, makeResI)$
8. $makeResO \equiv confO$, $Seq(makeResO, confO)$
9. $selFlightO$ precedes $selCarI$, $Seq(selFlightO, selCarI)$.

The component $\mathcal{O}$ representing the ontological constraints is obtained as the parallel composition of the Seq primitives listed above. Note that all of the actions are treated as inputs, as $\mathcal{O}$ is an observing component. Therefore, all states are quiescent and by input-enabledness, any trace over these actions that does not satisfy the ontological constraints is an inconsistent trace of the component.

Synthesis. The mediator $\mathcal{M}$ is obtained as the quotient $\mathcal{G}/(US \parallel \mathcal{O} \parallel EU)$. As usual, $\mathcal{G}$ is the Logic IOLTS shown in Section 3.4 taking the output set as the union of all actions in the components EU and US. The Logic IOLTS of $\mathcal{M}$ generated using our quotienting procedure is shown in Figure 3.8.

3.8 Mapping the WP3 mediator theory into the WP2 specification theory

In the previous sections, we have shown how the specification theory defined in Chapter 2 can be used to formulate the *mediator synthesis problem* as a suitable quotienting problem, in which ontological constraints and an abstract goal specification are taken into account. Since the mediator synthesis problem has been deeply investigated within the work of WP3, we show the relationships between the specification theory of WP2 and the mediator theory of WP3. Demonstrating integration of the work packages is achieved by giving a mapping from the mediator theory of WP3 to our specification theory. This allows the work of WP3 to inherit the interesting results of the specification theory. More verbosely, it allows one to state that a mediator synthesized *à la* WP3 is *correct*. That is, the mediator does not introduce communication mismatches or deadlocks, and is most general.

In our work, WP2 has opted for a model of concurrency that does not rely on handshaking between components, to better support distributed systems with asynchronous communication. This has culminated in an I/O model of concurrency where outputs are under the control of the component to which they belong, while inputs are controlled by the environment. WP3, on the other hand, has assumed that input and output actions must always handshake between components, much like in CSP and CCS. The WP2 model of concurrency is more general than that used by WP3, as handshaking can be explicitly encoded by synchronisation of common input actions. We utilise this property in giving our mapping.

The discussion hereafter requires a basic knowledge of the work described in Chapter 2 of Deliverable D3.3 [23]. In the following, we try to keep the discussion self-contained, and where that is not possible we provide the reader with references to the specific content of [23].

Aligning the action sets. WP3 utilises extended LTSs (cf. Definition 2, Chapter 2 of [23]) for modelling components, which are standard LTSs augmented with final states. Transitions can either be hidden, in which case they are labelled with τ, or are visible and support the communication of data. Let l range over action labels, $id_1, \ldots, id_m, od_1, \ldots, od_n$ range over data values, and $?$ and ! be the label prefixes used to denote input actions and output actions respectively. A visible transition of an extended LTS has one of the following forms:

- **expectation of action with data and production of return data** - all actions of the form $?l(id_1 \ldots id_m,od_1 \ldots od_n)$ where both l and $id_1, \ldots, id_m$ are expected as input, and $od_1, \ldots, od_n$ are produced as output;
- **production of action with data and expectation of return data** - all actions of the form $!l(id_1 \ldots id_m,od_1 \ldots od_n)$ where both l and $id_1, \ldots, id_m$ are produced as output, and $od_1, \ldots, od_n$ are expected as input.

Since Logic IOLTSs (as in Chapter 2) do not deal with data parameters (although an extension has been developed in Chapter 4), we give an explicit representation of each instantiation of the data variables as separate transitions. Thus, analogously to what is done in Section 3.7, we give a transformation that respects the action semantics:

- **expectation of action with data and production of return data** - each transition labeled with $?l(id_1 \ldots id_m,od_1 \ldots od_n)$ is represented by two consecutive transitions labeled with $lI\_id_1 \ldots id_m$ of type input and $lO\_od_1 \ldots od_n$ of type output;
- **production of action with data and expectation of return data** - each transition labeled with $!l(id_1 \ldots id_m,od_1 \ldots od_n)$ is represented by two consecutive transitions labeled with $lI\_id_1 \ldots id_m$ of type output and $lO\_od_1 \ldots od_n$ of type input.

From extended LTSs to Logic IOLTSs. As our quotient procedure works on Logic IOLTSs, we need to give a semantic preserving mapping from extended LTSs to Logic IOLTSs. Besides converting actions as mentioned previously, we also need to ensure that parallel composition of extended LTSs corresponds to parallel composition on the semantically equivalent Logic IOLTSs. We can achieve this by forcing the

Logic IOLTSs to handshake on each communication. Handshaking can be enforced by requiring each action in a component to be preceded by a special handshaking action that is always of type input.

Definition 3.1 *Let $\mathcal{P} = \langle S_\mathcal{P}, L_\mathcal{P}, \longrightarrow_\mathcal{P}, F_\mathcal{P}, s_{0_\mathcal{P}} \rangle$ be an extended LTS. The mapping $\mathcal{L}$ from extended LTSs to Logic IOLTSs is defined by $\mathcal{L}(\mathcal{P}) = \langle S_{\mathcal{L}(\mathcal{P})}, \mathcal{A}^I_{\mathcal{L}(\mathcal{P})}, \mathcal{A}^O_{\mathcal{L}(\mathcal{P})}, \longrightarrow_{\mathcal{L}(\mathcal{P})}, \emptyset \rangle$, where:*

- $S_{\mathcal{L}(\mathcal{P})} = S_\mathcal{P} \cup \{s'_a, s''_a : s \in S_\mathcal{P},$ *and* $?a(\dots) \in L_\mathcal{P}$ *or* $!a(\dots) \in L_\mathcal{P}\} \cup \{s_\checkmark\}$, *assuming* $s'_a, s''_a, s_\checkmark \notin S_\mathcal{P}$
- $\mathcal{A}^I_{\mathcal{L}(\mathcal{P})}$ *is the smallest set containing:*
  - $\{lI\_id_1 \dots id_m : ?l(id_1 \dots id_m, od_1 \dots od_n) \in L_\mathcal{P}\}$
  - $\{lO\_od_1 \dots od_n : !l(id_1 \dots id_m, od_1 \dots od_n) \in L_\mathcal{P}\}$
  - $\{hand\_a : ?a(\dots) \in L_\mathcal{P}$ *or* $!a(\dots) \in L_\mathcal{P}\}$
  - $\{\checkmark\}$
- $\mathcal{A}^O_{\mathcal{L}(\mathcal{P})}$ *is the smallest set containing:*
  - $\{lO\_od_1 \dots od_n : ?l(id_1 \dots id_m, od_1 \dots od_n) \in L_\mathcal{P}\}$
  - $\{lI\_id_1 \dots id_m : !l(id_1 \dots id_m, od_1 \dots od_n) \in L_\mathcal{P}\}$
- $\longrightarrow_{\mathcal{L}(\mathcal{P})}$ *is the smallest set containing:*
  - $\{(s, hand\_a, t'_a) : (s, ?a(\dots), t) \in \longrightarrow_\mathcal{P}$ *or* $(s, !a(\dots), t) \in \longrightarrow_\mathcal{P}\}$
  - $\{(t'_a, aI\_id_1 \dots id_m, t''_a), (t''_a, aO\_od_1 \dots od_n, t) : (s, ?a(id_1 \dots id_m, od_1 \dots od_n), t) \in \longrightarrow_\mathcal{P}\}$
  - $\{(t'_a, aI\_id_1 \dots id_m, t''_a), (t''_a, aO\_od_1 \dots od_n, t) : (s, !a(id_1 \dots id_m, od_1 \dots od_n), t) \in \longrightarrow_\mathcal{P}\}$
  - $\{(s, \tau, t) : (s, \tau, t) \in \longrightarrow_\mathcal{P}\}$.
  - $\{(s, \checkmark, s_\checkmark) : s \in F_\mathcal{P}\}$.

The mapping $\mathcal{L}$ ensures that each transition of the original extended LTS is split into three parts. The first part corresponds to handshaking so that synchronisation only takes place if all relevant parties agree. After handshaking, the first collection of data items $id_1 \dots id_m$ are communicated. Once this has taken place, the data items $od_1 \dots od_n$ are transmitted. We indicate that a component is safe to deadlock in a final state by allowing it to perform the $\checkmark$-transition. Note that none of the states are inconsistent.

Relating this mapping to WP3, Figure 3.9 shows the transformed extended LTS of the Blue component from Figure 3.6 of D3.3 [23]. Due to the extensive interface of the component, we indicate the I/O type of actions by the ? and ! symbols after the action name. Note that towards the bottom of the figure, we introduce actions prefixed PlaceOrder2, while in the original model of D3.3 the operation of the corresponding action with data is PlaceOrder. A discrepancy arises between our formalism and that of WP3, in that we assume an action with data has a particular data type. However, in WP3 this is not the case, and moreover is unnecessary as an action with data is treated as an atomic action for synchronisation purposes. As PlaceOrder arises twice with different data types in the original Blue model, we must distinguish the two instances in our formalism, which is why we use PlaceOrder2 for the second such occurrence. Discussion with WP3 is underway to see if we can standardise our approaches.

Formulating the ontological constraints. The mediator synthesis problem in WP2 takes as input the components that need to communicate, together with a component representing the constraints imposed by the ontology. In WP3, however, the ontological constraints are deduced as part of the synthesis algorithm. Thus, in order to map the WP3 mediator synthesis problem to that in WP2, the derived ontological constraints from WP3 need to be fed into WP2. Although we omit the detail here, it suffices to know that the synthesis algorithm in WP3 can be amended to generate these ontological constraints for us.

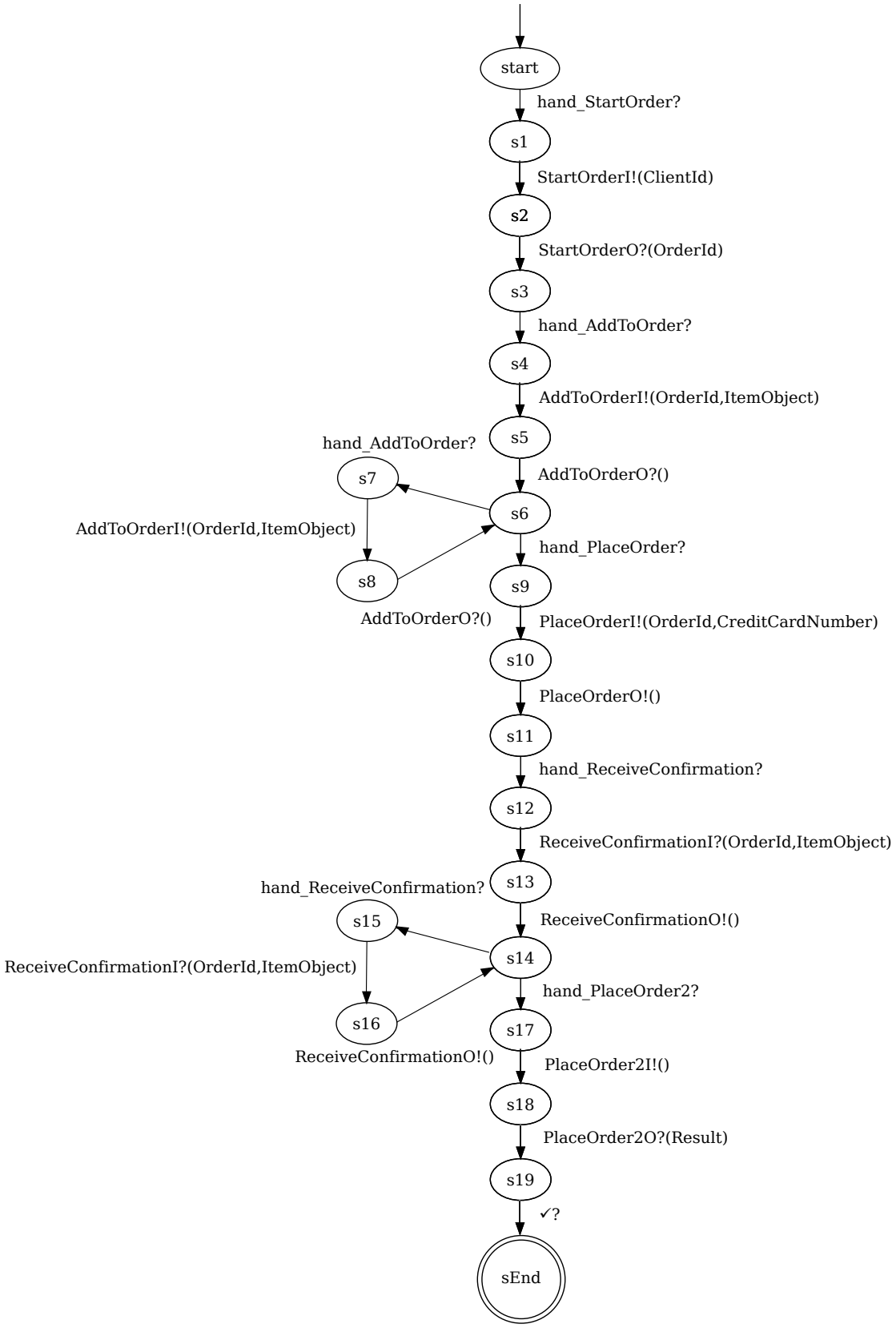

Figure 3.9: Logic IOLTS obtained by transforming the Blue extended LTS.

Obtaining the mediator. Equipped with the ontological constraints represented by a component $\mathcal{O}$, together with the components $\mathcal{P}$ and $\mathcal{Q}$ represented by Logic IOLTSs $\mathcal{L}(\mathcal{P})$ and $\mathcal{L}(\mathcal{Q})$, we can obtain the Logic IOLTS mediator $\mathcal{M}$ as the unique solution of $\mathcal{G}/(\mathcal{L}(\mathcal{P}) \mid\mid \mathcal{O} \mid\mid \mathcal{L}(\mathcal{Q}))$, if a solution exists, otherwise there is no mediator. As in Section 3.4, the goal has as the output set $\mathcal{A}_{\mathcal{L}(\mathcal{P})} \cup \mathcal{A}_{\mathcal{L}(\mathcal{Q})}$, and the input set is empty. A trace can only become quiescent in $\mathcal{G}$ after a ✓ has been witnessed, and there are no inconsistent traces. This formulation of the goal $\mathcal{G}$ correlates with the notion of *coordination policies* (i.e., goal) provided by the WP3 mediator theory [23].

If the mediator $\mathcal{M}$ does not exist for $\mathcal{L}(\mathcal{P})$ and $\mathcal{L}(\mathcal{Q})$, then there is no mediator allowing $\mathcal{P}$ and $\mathcal{Q}$ to successfully communicate with one another. If $\mathcal{M}$ does exist, then the most general mediator allowing $\mathcal{P}$ and $\mathcal{Q}$ to communicate is the extended LTS $\mathcal{M}'$, obtained from $\mathcal{M}$ by collapsing actions back down into the form used by WP3, and stripping out all handshaking and ✓-labelled transitions. Note that in the mediator, the split-form of transitions may not be consecutive, so cannot be collapsed back into a single transition. This fits with the WP3 theory, whereby each transition $?l(id_1 \ldots id_m, od_1 \ldots od_n)$ or $!l(id_1 \ldots id_m, od_1 \ldots od_n)$ is permitted to be represented by split transitions of the form $?l(id_1 \ldots id_m, -)$ and $?l(-, od_1 \ldots od_n)$, or $!l(id_1 \ldots id_m, -)$ and $!l(-, od_1 \ldots od_n)$ respectively.

3.9 Summary

In this chapter, we have formalised the mediator synthesis problem as an application of finding the quotient of a collection of components to communicate from a goal specification that guarantees absence of communication mismatches and premature deadlock. We have executed our methodology on two examples, one involving simple messaging, and the other based on the CONNECT-oriented travel agency example of [10]. We have linked the work of WPs 2 and 3 by giving a mapping from the extended LTSs of WP3 to the Logic IOLTSs of WP2, which allows us to compare the mediators synthesised by the two approaches. From this we can tell whether the mediator synthesised by WP3 is the most general, and whether it is free of communication errors.

4 Data extension of the specification theory

In this chapter, we extend the formalism for specifying components, presented in Chapter 2, by introducing the ability to model and reason about data. Data can occur as parameters of actions that are used for communication, and can also be stored in variables. The result is a model that covers the register automata developed in WP4 (see Chapter 2 of D4.3 and [15]). This model can be used for learning (as we show in Chapter 3 of D 4.3 and [45]) and for synthesis of mediators (see D3.3 and this chapter). Our extension is more general than the register automata of WP4 in that it need not impose restrictions (determinism, canonicity) where this is not necessary. The treatment of data in our extension can also be seen as inspired by the treatment of data in I/O-automata [55, 47], but the semantical notions (refinement, consistency, etc.) will remain aligned with the theory of logical I/O transition systems presented in Chapter 2. We will present operations for parallel composition and conjunction: these are rather straight-forward adaptations of the corresponding operations in the finitary case. Refinement between two specifications is undecidable in the general case, but sufficient techniques for the problem can be adapted from simulation relations between infinite-state transition systems (e.g., [55, 47]).

Continuing the analogy with Chapter 2, we also present an operation of quotient, which can be seen as an inverse of the parallel composition operation. We provide a technique for generating quotients, which provides an algorithm under assumptions that are met in many practical situations. In our data extension, the quotient is not computable for general nondeterministic specifications. However, for the deterministic case, a quotient can be calculated by a fixpoint calculation adapted from controller synthesis (e.g., [68]), based on computations of pre- and postconditions.

In Section 4.4, we illustrate how this operation can be used to formulate (and solve) the problem of generating a most general mediator on the basis of register automata obtained, e.g., by learning. The main purpose of including an application to mediator synthesis is to demonstrate how the algebra developed in this deliverable provides unifying links with the developments of WP3 and WP4. From a CONNECT perspective, the contribution of Section 4.4 (an also of Chapter 3) is to show that the specification theory developed in WP2 can be used to formulate the mediator synthesis problem. We also provide a framework for synthesizing a most general mediator (in a certain technical sense). Note, however, that this procedure may be of unnecessary high complexity and produce unnecessarily general mediators. More specific and efficient approaches to the synthesis problem are presented in Deliverable D3.3. Moreover, the approach in Section 4.4 assumes that suitable information about the relationship between communication primitives and data of two networked components have been extracted, e.g., from an ontology; we do not here consider the problem of how to perform this extraction. In contrast, the approaches presented in Deliverable D3.3 are able to work directly on the information provided by some ontology.

Note on Related Work Relationships between our specification theory (without data) and previous literature are discussed in Chapter 2. There has been no published work which systematically extends interface automata with data. On the other hand, our data extension follows lines similar to definitions of I/O-automata (e.g., [55, 47] and follow-up works). These works do not consider the quotient operation or something analogous.

Concerning our application of the quotient operation for mediator synthesis, there is a related line of work on web service composition, represented by works such as [8, 42, 6, 46]. Here, components that represent web services are specified in some formalism as automata that exchange data values. Specifications can be given in different forms, typically in terms of control states that may and must be reached, and in som cases also including dependencies of data values. In contrast to the work presented in this chapter, that line of work is not a part of an overall coherent specification framework. Also, the work does not aim to specify a “best” or “most general” mediator, but merely search for some mediator that fulfills the specified purpose.

A close work is [8], which presents a the synthesis algorithm based on backward search, which is an adaptation of classical algorithms for solving games over finite graphs (games on finite graphs have been extensively studied, see e.g. [43, 41, 34, 71, 68, 17]). A specified property is a combination of safety and reachability, given as a combination of target states together with unsafe states. The algorithm produces loop-free controllers that guarantee that the composed system reaches a target state while avoiding unsafe states. In contrast, our specification is a combination of a safety condition and a deadlock freeness

condition; these two kinds of specifications are of incomparable expressive power. One advantage of our type of specification is that synthesized controllers may contain loops. Another difference is that [8] underapproximates infinite data domains by finite ones, arguing that to achieve enough precision, it is in practical cases enough to use finite domains of very small sizes. We combine the game solving with a knowledge level reasoning similar to [61] where the precise values of variables are abstracted into a set of important relations between them, here equalities and inequalities, called knowledge. Since specifications relate values of variables only in terms of equalities and inequalities and since the data are not manipulated in other means than testing equality and sending/receiving, this abstraction is precise. We believe that this kind of abstraction also results in smaller state spaces. For instance, a state of the quotient where the knowledge contains the equality $x = y$ would correspond to a set of states containing a state with $x = val$ and $y = val$ for every value val from the domain of x and y. In knowledge [61], it is only discussed how to compute postconditions and perform symbolic *forward* exploration of state space on the level of knowledge. Our contribution is that we show how to compute preconditions and perform symbolic *backward* exploration needed for the computation of bad states.

As another remark, we note that Chapter 3 and Section 4.4 use two different techniques for achieving proper synchronization between components, both of which can be regarded as extractable from ontology information. The approach in Chapter 3 extracts explicit ordering between actions, whereas the approach in Section 4.4 extracts ordering which is implicitly defined by data constraints. A related contrasting between such approaches has been made in [56]. According to [56], none of the two approaches clearly wins: explicit approaches tend to scale better and are easier to write, implicit approaches offer higher degree of reuse and abstraction and clearer separation of components from the composition goal.

4.1 Processes: Operational Specifications with Data

In this section, we define our operational model of Logic IOLTS, which also includes data. We will refer to such objects as *processes w. data* or data-processes.

We assume a finite collection of unbounded domains of *data values* – each domain is called a *sort* – and a set of *actions*. Each action has a certain *arity*, which is a tuple of sorts that determines which combinations of parameters it may take. A *symbol* is a term of the form $\alpha(d_1, \ldots, d_n)$, where α is an action, and $d_1, \ldots, d_n$ are data values in the appropriate sorts. For a set I of actions, let Σ^I be the set of data symbols with an action in I.

We assume a set of (sorted) *variables*, ranged over by x, y, z, etc., and a set of *formal parameters* ranged over by p. A *parameterized symbol* is of form $\alpha(p_1, \ldots, p_n)$, where α is an action, and $p_1, \ldots, p_n$ are distinct formal parameters. We assume that each action always uses the same fixed tuple of formal parameters when forming parameterized symbols. A *guard* is a conjunction of equalities and negated equalities over variables and formal parameters.

Remark: Our theory can with modest effort be extended to cover richer sets of functions and predicates, rather than just equalities and negated equalities. In this version, we restrict to equalities.

Let us define our notion of data-processes (i.e., Logic IOLTS with data), which is actually very similar to the register automata defined in Chapter 2 of D4.3 (see also [15]).

Definition 4.1 A *data-process* $\mathcal{P}$ is a tuple $\mathcal{P} = (I_\mathcal{P}, O_\mathcal{P}, L_\mathcal{P}, \ell^0_\mathcal{P}, X_\mathcal{P}, \delta_\mathcal{P}, \varphi^F_\mathcal{P})$, where

- $I_\mathcal{P}$ is a finite set of *input actions*,
- $O_\mathcal{P}$ is a finite set of *output actions*,
- $L_\mathcal{P}$ is a finite set of *locations*,
- $\ell^0_\mathcal{P} \in L_\mathcal{P}$ is the *initial location*
- $X_\mathcal{P}$ is a set of (sorted) *variables*, each of which has a defined initial value.
- $\delta_\mathcal{P}$ is a finite set of *transitions*, each of which is of form $\langle \ell, \mathit{stmt}, \ell' \rangle$ where *stmt* is a statement of form

$$g \ ; \ \alpha(p_1, \ldots, p_n); y_1, \ldots, y_k := z_1, \ldots, z_k \ ,$$

  where

– g is a guard over $X_{\mathcal{P}}$ and $p_1, \ldots, p_n$,

– $\alpha(p_1, \ldots, p_n)$ is a parameterized symbol with $\alpha \in (I_{\mathcal{P}} \cup O_{\mathcal{P}})$,

– $y_1, \ldots, y_k$ are distinct variables in $X_{\mathcal{P}}$, and

– $z_1, \ldots, z_k$ is a tuple, consisting of variables in $X_{\mathcal{P}}$ and formal parameters in $p_1, \ldots, p_n$.

- $\varphi^F_{\mathcal{P}}$, the *inconsistency predicate*, maps each location to a boolean combination of equalities and negated equalities over variables in $X_{\mathcal{P}}$. □

We will write $\bar{p}$ for $p_1, \ldots, p_n$, $\bar{d}$ for $d_1, \ldots, d_n$, $\bar{y}$ for $y_1, \ldots, y_k$, and $\bar{z}$ for $z_1, \ldots, z_k$.

Intuitively, a data-process is at any point in time in a state, given by a location and an assignment to its variables. The data-process can perform a statement of form $g; \alpha(\bar{p}); \bar{y} := \bar{z}$ provided that the guard $g[\bar{d}/\bar{p}]$ evaluates to true in the current state; it then synchronizes via the symbol $\alpha(\bar{d})$, binds the formal parameters $\bar{p}$ to data values $\bar{d}$, and simultaneously assigns new values to the variables $\bar{y}$ according to the multiple assignment statement. A statement is performed atomically.

We will often use the terms input (output) transition and input (output) statement, when the involved action is an input (output) action. For convenience, input and output statements have the same basic form, even though the data values that are bound to the formal parameters are chosen by the environment in input statements, and by the data-process itself in output statements. In input statements, the guard is often *true* to reflect that the environment chooses the parameter values. However, a guard can also be used to reflect constraints on data values that the data-process is prepared to receive. In output statements, the parameters $\bar{p}$ are often taken directly from the variables of the data-process. We therefore introduce the notation $g; !\alpha(x_1, \ldots, x_n); \bar{y} := \bar{z}$ as a shorthand for $(g \wedge p_1 = x_1 \wedge \cdots p_n = x_n; !\alpha(p_1, \ldots, p_n); \bar{y} := \bar{z}$.

The semantics of a data-process is defined by showing how a data-process can be mapped to an (infinite-state) Logic IOLTS, as defined in Definition 2.16. For a set X of variables, an X-valuation is an assignment of data values in the appropriate domains to the variables in X, respecting sorts. Valuations are extended to expressions in the natural way. For a valuation σ and a boolean combination of equalities over variables and data values g, we write $\sigma \models g$ to denote that $\sigma(g)$ evaluates to true.

A *state* of $\mathcal{P}$ is a pair $\langle \ell, \sigma \rangle$ where $\ell \in L_{\mathcal{P}}$ and σ is a $X_{\mathcal{P}}$-valuation. The *initial state* of $\mathcal{P}$ is $\langle \ell^0_{\mathcal{P}}, \sigma_0 \rangle$, where $\ell^0_{\mathcal{P}}$ is the initial location and σ_0 is the $X_{\mathcal{P}}$-valuation that maps all variables to their initial values. A *step* of $\mathcal{P}$, denoted by $\langle \ell, \sigma \rangle \xrightarrow{\alpha(\bar{d})} \langle \ell', \sigma' \rangle$, transfers $\mathcal{P}$ from $\langle \ell, \sigma \rangle$ to $\langle \ell', \sigma' \rangle$ while performing the (input or output) symbol $\alpha(\bar{d})$. It is derived from a transition $\langle \ell, \mathit{stmt}, \ell' \rangle \in \delta_{\mathcal{P}}$, with *stmt* of the form $g; \alpha(p_1, \ldots, p_n); y_1, \ldots, y_k := z_1, \ldots, z_k$, such that

- $\sigma \models g[d_1, \ldots, d_n / p_1, \ldots, p_n]$,
- $\sigma'(y_i) = \sigma(z_i)$ when z_i is a variable, for $i = 1, \ldots, k$,
- $\sigma'(y_i) = d_j$ when z_i is the parameter p_j, for $i = 1, \ldots, k$, and
- $\sigma'(x) = \sigma(x)$ when x does not occur among $y_1, \ldots, y_k$.

A symbol $\alpha(\bar{d})$ is *enabled* in a state $\langle \ell, \sigma \rangle$ if there is a step of $\mathcal{P}$ of form $\langle \ell, \sigma \rangle \xrightarrow{\alpha(\bar{d})} \langle \ell', \sigma' \rangle$ for some state $\langle \ell', \sigma' \rangle$.

We can now define the semantics of data-processes by mapping them to logic IOLTSs:

Definition 4.2 *Definition the mapping $[\![\cdot]\!]^{Op}$ from data-processes to logic IOLTSs as follows. For $\mathcal{P} = (I_{\mathcal{P}}, O_{\mathcal{P}}, L_{\mathcal{P}}, \ell^0_{\mathcal{P}}, X_{\mathcal{P}}, \delta_{\mathcal{P}}, \varphi^F_{\mathcal{P}})$, we define $[\![\mathcal{P}]\!]^{Op}$ as the logic IOLTS $[\![\mathcal{P}]\!]^{Op} = \langle \mathcal{S}_{\mathcal{P}}, \Sigma^{I_{\mathcal{P}}}, \Sigma^{O_{\mathcal{P}}}, \longrightarrow_{\mathcal{P}}, F_{\mathcal{P}} \rangle$, where*

- *$\mathcal{S}_{\mathcal{P}}$ is the set of states of $\mathcal{P}$,*
- *$\longrightarrow_{\mathcal{P}}$ is defined by: $\langle \langle \ell, \sigma \rangle, \alpha(\bar{d}), \langle \ell', \sigma' \rangle \rangle \in \longrightarrow_{\mathcal{P}}$ iff $\langle \ell, \sigma \rangle \xrightarrow{\alpha(\bar{d})} \langle \ell', \sigma' \rangle$, and*
- *$F_{\mathcal{P}}$ is the set of states $\langle \ell, \sigma \rangle$ such that $\sigma \models \varphi^F_{\mathcal{P}}(\ell)$.* □

We can now reuse the definitions in Section 2.3, and define a mapping $[\![\cdot]\!]^D$ from data-processes to declarative specifications (see Definition 2.1) by $[\![\mathcal{P}]\!]^D = [\![[\![\mathcal{P}]\!]^{Op}]\!]^*$. In the following, we will often use the set of safe traces $ST_\mathcal{P}$, defined in Definition 2.19, as a characterization of the behavior of a process.

For future development, we will sometimes use an alternative, but equivalent, definition of refinement, which is phrased in terms of the sets of safe traces of $\mathcal{Q}$ and $\mathcal{P}$, by adapting the definition of alternating simulation [29].

Proposition 4.3 *Assume that $\mathcal{P}$ and $\mathcal{Q}$ are processes with nonempty sets of safe traces. Then $\mathcal{Q} \sqsubseteq_{dec} \mathcal{P}$ iff: $I_\mathcal{P} \subseteq I_\mathcal{Q}$, and $O_\mathcal{Q} \subseteq O_\mathcal{P}$, and*

- *whenever t is a safe trace of both $\mathcal{P}$ and $\mathcal{Q}$ (i.e., $t \in ST_P \cap ST_Q$), then*
  - *for any input symbol i, if $ti \in ST_P$ then $ti \in ST_Q$,*
  - *for any output symbol o, if $to \in ST_Q$ then $to \in ST_P$.* □

Thus, the extensional semantics can be characterized by the set of safe traces.

There are different (incomplete) techniques for establishing $\mathcal{Q} \sqsubseteq_{dec} \mathcal{P}$ when $\mathcal{Q}$ and $\mathcal{P}$ are data-processes, including simulation [47]. We note that in general, the problem of checking $\mathcal{Q} \sqsubseteq_{dec} \mathcal{P}$ is undecidable. This follows from the undecidability of checking universality for register automata [58].

4.2 Parallel Composition

Let us now adapt the definition of parallel composition (Definition 2.28) to data-processes. Intuitively, the parallel composition operator yields the combined effect of its operands running asynchronously. We use a model of communication in which an output from a component can be received by multiple components in the environment: this means that we use a broadcast model of communication. An input $?a(\bar{d})$ and output $!a(\bar{d})$ combine to form an output $!a(\bar{d})$ (as in certain variants of I/O automata), rather than a hidden action τ as is the case in Milner's CCS. Note that the attributes $?$ and $!$ on actions (as in $!a()$) are not part of the actions, they serve only to remind that the action in question is an input or output in the considered context.

Two data-processes $\mathcal{P} = (I_\mathcal{P}, O_\mathcal{P}, L_\mathcal{P}, \ell^0_\mathcal{P}, X_\mathcal{P}, \delta_\mathcal{P}, \varphi^F_\mathcal{P})$ and $\mathcal{Q} = (I_\mathcal{Q}, O_\mathcal{Q}, L_\mathcal{Q}, \ell^0_\mathcal{Q}, X_\mathcal{Q}, \delta_\mathcal{Q}, \varphi^F_\mathcal{Q})$ are *composable* if $O_\mathcal{P} \cap O_\mathcal{Q} = \emptyset$ and $X_\mathcal{P} \cap X_\mathcal{Q} = \emptyset$.

As a building block for the definition of parallel composition on data-processes, let us define the parallel composition of two statements $\mathit{stmt}_1 = g_1; \alpha(\overline{p}); \overline{y}_1 := \overline{z}_1$ and $\mathit{stmt}_2 = g_2; \alpha(\overline{p}); \overline{y}_2 := \overline{z}_2$ with the same action α, in two processes with disjoint sets of variables, as

$$\mathit{stmt}_1 \| \mathit{stmt}_2 = g_1 \wedge g_2; \alpha(\overline{p}); \overline{y}_1, \overline{y}_2 := \overline{z}_1, \overline{z}_2 \ .$$

In the special case where stmt_1 is an output statement of form $g_1; \alpha(\overline{x}); \overline{y}_1 := \overline{z}_1$ stmt_2 is an input statement of form $\alpha(\overline{p}); \overline{y}_1 := \overline{p}_1$, then $\mathit{stmt}_1 \| \mathit{stmt}_2$ can be written as $g_1; \alpha(\overline{x}); \overline{y}_1, \overline{y}_2 := \overline{z}_1, \overline{p}_1[\overline{x}/\overline{p}]$. For an action α and a process $\mathcal{P}$, let $g^\alpha_{\mathcal{P},\ell_\mathcal{P}}$ denote the disjunction of the guards of the outgoing α-statements from $\ell_\mathcal{P}$: if there is no such statement, then $g^\alpha_{\mathcal{P},\ell_\mathcal{P}}$ is defined as *false*.

We are now ready for the definition of parallel composition. But first, given processes $\mathcal{P}$ and $\mathcal{Q}$ let us define a mapping, denoted $Incompat_{\mathcal{P}||\mathcal{Q}}$ from pairs of locations of $\mathcal{P}$ and $\mathcal{Q}$ to boolean combinations of equalities and inequalities over the state variables of $\mathcal{P}$ and $\mathcal{Q}$, such that $Incompat_{\mathcal{P}||\mathcal{Q}}(\langle \ell_\mathcal{P}, \ell_\mathcal{Q} \rangle)$ is true if $\mathcal{P}$ in location $\ell_\mathcal{P}$ can produce an output symbol for which $\mathcal{Q}$ does not have a matching input step, or vice versa. Formally, we can represent $Incompat_{\mathcal{P}||\mathcal{Q}}(\langle \ell_\mathcal{P}, \ell_\mathcal{Q} \rangle)$ as a disjunction, which for each synchronizing action $\alpha \in O_\mathcal{P} \cap I_\mathcal{Q}$ contains the disjunct $g^\alpha_{\mathcal{P},\ell_\mathcal{P}} \wedge \neg g^\alpha_{\mathcal{Q},\ell_\mathcal{Q}}$, and symmetrically for each synchronizing action $\alpha \in O_\mathcal{Q} \cap I_\mathcal{P}$ contains the disjunct $g^\alpha_{\mathcal{Q},\ell_\mathcal{Q}} \wedge \neg g^\alpha_{\mathcal{P},\ell_\mathcal{P}}$.

Definition 4.4 The parallel composition of two composable data-processes $\mathcal{P} = (I_\mathcal{P}, O_\mathcal{P}, L_\mathcal{P}, \ell^0_\mathcal{P}, X_\mathcal{P}, \delta_\mathcal{P}, \varphi^F_\mathcal{P})$ and $\mathcal{Q} = (I_\mathcal{Q}, O_\mathcal{Q}, L_\mathcal{Q}, \ell^0_\mathcal{Q}, X_\mathcal{Q}, \delta_\mathcal{Q}, \varphi^F_\mathcal{Q})$, denoted $\mathcal{P} \| \mathcal{Q}$, is obtained as $\mathcal{P} \| \mathcal{Q} = (I_{\mathcal{P}||\mathcal{Q}}, O_{\mathcal{P}||\mathcal{Q}}, L_{\mathcal{P}||\mathcal{Q}}, \ell^0_{\mathcal{P}||\mathcal{Q}}, X_{\mathcal{P}||\mathcal{Q}}, \delta_{\mathcal{P}||\mathcal{Q}}, \varphi^F_{\mathcal{P}||\mathcal{Q}})$, where

- $I_{\mathcal{P}||\mathcal{Q}} = (I_\mathcal{P} \cup I_\mathcal{Q}) \setminus O_{\mathcal{P}||\mathcal{Q}}$,

- $O_{\mathcal{P}||\mathcal{Q}} = O_\mathcal{P} \cup O_\mathcal{Q}$,
- $L_{\mathcal{P}||\mathcal{Q}} = L_\mathcal{P} \times L_\mathcal{Q}$,
- $\ell^0_{\mathcal{P}||\mathcal{Q}} = \langle \ell^0_\mathcal{P}, \ell^0_\mathcal{Q} \rangle$,
- $X_{\mathcal{P}||\mathcal{Q}} = X_\mathcal{P} \cup X_\mathcal{Q}$,
- $\delta_{\mathcal{P}||\mathcal{Q}}$ is obtained from $\delta_\mathcal{P}$ and $\delta_\mathcal{Q}$ by two forms of transitions.
  - If $\langle \ell_\mathcal{P}, \mathit{stmt}, \ell'_\mathcal{P} \rangle \in \delta_\mathcal{P}$ has an action which is not an action of $\mathcal{Q}$, (i.e., it is non-synchronizing), then $\langle \langle \ell_\mathcal{P}, \ell_\mathcal{Q} \rangle, \mathit{stmt}, \langle \ell'_\mathcal{P}, \ell_\mathcal{Q} \rangle \rangle \in \delta_{\mathcal{P}||\mathcal{Q}}$ for any location $\ell_\mathcal{Q} \in L_\mathcal{Q}$.
  - Symmetrically, if $\langle \ell_\mathcal{Q}, \mathit{stmt}, \ell'_\mathcal{Q} \rangle \in \delta$ has an action which is not an action of $\mathcal{P}$, then $\langle \langle \ell_\mathcal{P}, \ell_\mathcal{Q} \rangle, \mathit{stmt}, \langle \ell_\mathcal{P}, \ell'_\mathcal{Q} \rangle \rangle \in \delta_{\mathcal{P}||\mathcal{Q}}$ for any location $\ell_\mathcal{P} \in L_\mathcal{P}$.
  - If $\langle \ell_\mathcal{P}, \mathit{stmt}_\mathcal{P}, \ell'_\mathcal{P} \rangle \in \delta_\mathcal{P}$ and $\langle \ell_\mathcal{Q}, \mathit{stmt}_\mathcal{Q}, \ell'_\mathcal{Q} \rangle \in \delta_\mathcal{Q}$ have the same action, i.e., they synchronize, then $\langle \langle \ell_\mathcal{P}, \ell_\mathcal{Q} \rangle, \mathit{stmt}_\mathcal{P} \| \mathit{stmt}_\mathcal{Q}, \langle \ell'_\mathcal{P}, \ell'_\mathcal{Q} \rangle \rangle \in \longrightarrow$.
- $\varphi^F_{\mathcal{P}||\mathcal{Q}}$ is defined by $\varphi^F_{\mathcal{P}||\mathcal{Q}}(\langle \ell_\mathcal{P}, \ell_\mathcal{Q} \rangle) = \varphi^F_\mathcal{P}(\ell_\mathcal{P}) \vee \varphi^F_\mathcal{Q}(\ell_\mathcal{Q}) \vee \mathit{Incompat}_{\mathcal{P}||\mathcal{Q}}(\langle \ell_\mathcal{P}, \ell_\mathcal{Q} \rangle)$. □

We remark that this form of parallel composition is slightly different from the form of Definition 2.28. In particular, Definition 2.28 does not include a term that correspond to the "incompatibility" predicate. This is because Definition 2.28 assumes that the processes to be composed are error-completed, which implies that incompatibility of a composed state is simply the disjunction of the inconsistency predicates on the two components.

Definition of Parallel Composition in Terms of Safe Traces As a preparation for the definition of Quotient in Section 4.4, we will show how parallel composition can be defined on the basis of safe traces of participating data-processes.

Extend the definition of $\mathit{Incompat}_{\mathcal{P}||\mathcal{Q}}$ to traces of $\mathcal{P} \,||\, \mathcal{Q}$ by saying that $\mathit{Incompat}_{\mathcal{P}||\mathcal{Q}}(t)$ is true if there is a state $\langle \langle \ell_\mathcal{P}, \ell_\mathcal{Q} \rangle, \sigma \rangle$ of $\mathcal{P} \,||\, \mathcal{Q}$ such that $\langle \langle \ell_{0,\mathcal{P}}, \ell_{0,\mathcal{Q}} \rangle \ell_0, \sigma_0 \rangle \stackrel{t}{\Longrightarrow}_\mathcal{P} \langle \langle \ell_\mathcal{P}, \ell_\mathcal{Q} \rangle, \sigma \rangle$, where $\langle \langle \ell_{0,\mathcal{P}}, \ell_{0,\mathcal{Q}} \rangle, \sigma_0 \rangle$ is the initial state of $\mathcal{P} \,||\, \mathcal{Q}$, and such that $\sigma \models \mathit{Incompat}_{\mathcal{P}||\mathcal{Q}}(\langle \ell_\mathcal{P}, \ell_\mathcal{Q} \rangle)$.

For a trace t and a set I of actions, define $t \restriction I$ as the subsequence of t consisting of symbols in Σ^I. Define $ST_\mathcal{P} \,||\, ST_\mathcal{Q}$ as the set of traces $t \in (\Sigma^{I_{\mathcal{P}||\mathcal{Q}}} \cup \Sigma^{O_{\mathcal{P}||\mathcal{Q}}})^*$ with $t \restriction (I_\mathcal{P} \cup O_\mathcal{P}) \in ST_\mathcal{P}$ and $t \restriction (I_\mathcal{Q} \cup O_\mathcal{Q}) \in ST_\mathcal{Q}$. Then we have the following proposition

Proposition 4.5 $ST_{\mathcal{P}||\mathcal{Q}}$ is the maximal prefix-closed subset of $ST_\mathcal{P} \,||\, ST_\mathcal{Q}$ such that

- whenever $t \in ST_{\mathcal{P}||\mathcal{Q}}$, then $\mathit{Incompat}_{\mathcal{P}||\mathcal{Q}}(t)$ is false, and
- whenever $t \in ST_{\mathcal{P}||\mathcal{Q}}$ and $to \in (ST_\mathcal{P} \,||\, ST_\mathcal{Q})$ where $o \in \Sigma^{(O_\mathcal{P} \cup O_\mathcal{Q})}$, then $to \in ST_{\mathcal{P}||\mathcal{Q}}$. □

4.3 Conjunction

In this section we define a conjunctive operator on data-processes that corresponds to the meet operation on the refinement preorder. Although our formulation shares similarities with Definition 2.30, we do not consider the error-completion of components in this chapter, so we must explicitly handle non-enabled inputs.

Let $\mathcal{P} = (I_\mathcal{P}, O_\mathcal{P}, L_\mathcal{P}, \ell^0_\mathcal{P}, X_\mathcal{P}, \delta_\mathcal{P}, \varphi^F_\mathcal{P})$ and $\mathcal{Q} = (I_\mathcal{Q}, O_\mathcal{Q}, L_\mathcal{Q}, \ell^0_\mathcal{Q}, X_\mathcal{Q}, \delta_\mathcal{Q}, \varphi^F_\mathcal{Q})$ be two data-processes, and let $\mathit{stmt}_\mathcal{P} = g_\mathcal{P}; \alpha_\mathcal{P}(\overline{p}_\mathcal{P}); \overline{y}_\mathcal{P} := \overline{z}_\mathcal{P}$ and $\mathit{stmt}_\mathcal{Q} = g_\mathcal{Q}; \alpha_\mathcal{Q}(\overline{p}_\mathcal{Q}); \overline{y}_\mathcal{Q} := \overline{z}_\mathcal{Q}$ be statements of $\mathcal{P}$ and $\mathcal{Q}$ respectively. We say that $\mathcal{P}$ and $\mathcal{Q}$ are composable for conjunction only if $I_\mathcal{P} \cup I_\mathcal{Q}$ is disjoint from $O_\mathcal{P} \cup O_\mathcal{Q}$, and $X_\mathcal{P} \cap X_\mathcal{Q} = \emptyset$.

Definition 4.6 *The conjunction of two composable data-processes $\mathcal{P}$ and $\mathcal{Q}$, denoted $\mathcal{P} \wedge \mathcal{Q}$, is the data-process $(I_{\mathcal{P}\wedge\mathcal{Q}}, O_{\mathcal{P}\wedge\mathcal{Q}}, L_{\mathcal{P}\wedge\mathcal{Q}}, \ell^0_{\mathcal{P}\wedge\mathcal{Q}}, X_{\mathcal{P}\wedge\mathcal{Q}}, \delta_{\mathcal{P}\wedge\mathcal{Q}}, \varphi^F_{\mathcal{P}\wedge\mathcal{Q}})$, where:*

- $I_{\mathcal{P}\wedge\mathcal{Q}} = I_\mathcal{P} \cup I_\mathcal{Q}$

- $O_{\mathcal{P}\wedge\mathcal{Q}} = O_\mathcal{P} \cap O_\mathcal{Q}$
- $L_{\mathcal{P}\wedge\mathcal{Q}} = (L_\mathcal{P} \times L_\mathcal{Q}) \cup L_\mathcal{P} \cup L_\mathcal{Q}$
- $\ell^0_{\mathcal{P}\wedge\mathcal{Q}} = \ell^0_\mathcal{P}$ *if* $\neg\varphi^F_\mathcal{P}(\ell^0_\mathcal{P}) \wedge \varphi^F_\mathcal{Q}(\ell^0_\mathcal{Q})$, $\ell^0_\mathcal{Q}$ *if* $\varphi^F_\mathcal{P}(\ell^0_\mathcal{P}) \wedge \neg\varphi^F_\mathcal{Q}(\ell^0_\mathcal{Q})$, *and* $\langle \ell^0_\mathcal{P}, \ell^0_\mathcal{Q}\rangle$ *otherwise*
- $X_{\mathcal{P}\wedge\mathcal{Q}} = X_\mathcal{P} \cup X_\mathcal{Q}$
- $\delta_{\mathcal{P}\wedge\mathcal{Q}}$ *is the smallest set satisfying:*
  - *If* $\langle \ell_\mathcal{P}, \mathit{stmt}_\mathcal{P}, \ell'_\mathcal{P}\rangle \in \delta_\mathcal{P}$, $\langle \ell_\mathcal{Q}, \mathit{stmt}_\mathcal{Q}, \ell'_\mathcal{Q}\rangle \in \delta_\mathcal{Q}$, $\alpha_\mathcal{P} = \alpha_\mathcal{Q}$, $\neg\varphi^F_\mathcal{P}(\ell_\mathcal{P})$ *and* $\neg\varphi^F_\mathcal{Q}(\ell_\mathcal{Q})$, *then* $\langle\langle \ell_\mathcal{P}, \ell_\mathcal{Q}\rangle, \mathit{stmt}_\mathcal{P} \parallel \mathit{stmt}\mathcal{Q}, \langle \ell'_\mathcal{P}, \ell'_\mathcal{Q}\rangle\rangle \in \delta_{\mathcal{P}||\mathcal{Q}}$
  - *If* $\langle \ell_\mathcal{P}, \mathit{stmt}_\mathcal{P}, \ell'_\mathcal{P}\rangle \in \delta_\mathcal{P}$, $\alpha_\mathcal{P} \in I_\mathcal{P} \cap I_\mathcal{Q}$ *then* $\langle\langle \ell_\mathcal{P}, \ell_\mathcal{Q}\rangle, (g_\mathcal{P} \wedge \neg g^{\alpha_\mathcal{P}}_{\mathcal{Q},\ell_\mathcal{Q}})\,; \alpha_\mathcal{P}(\overline{p}_\mathcal{P}); \overline{y}_\mathcal{P} := \overline{z}_\mathcal{P}, \ell'_\mathcal{P}\rangle \in \delta_{\mathcal{P}||\mathcal{Q}}$
  - *If* $\langle \ell_\mathcal{Q}, \mathit{stmt}_\mathcal{Q}, \ell'_\mathcal{Q}\rangle \in \delta_\mathcal{Q}$, $\alpha_\mathcal{Q} \in I_\mathcal{P} \cap I_\mathcal{Q}$ *then* $\langle\langle \ell_\mathcal{P}, \ell_\mathcal{Q}\rangle, (g'_\mathcal{P} \wedge \neg g^{\alpha_\mathcal{Q}}_{\mathcal{P},\ell_\mathcal{P}}); \alpha_\mathcal{Q}(\overline{p}_\mathcal{Q}); \overline{y}_\mathcal{Q} := \overline{z}_\mathcal{Q}, \ell'_\mathcal{Q}\rangle \in \delta_{\mathcal{P}||\mathcal{Q}}$
  - *If* $\langle \ell_\mathcal{P}, \mathit{stmt}_\mathcal{P}, \ell'_\mathcal{P}\rangle \in \delta_\mathcal{P}$, *and* $\alpha_\mathcal{P} \in I_\mathcal{P} \setminus I_\mathcal{Q}$ *or* $\varphi^F_\mathcal{Q}(\ell_\mathcal{Q})$, *then* $\langle\langle \ell_\mathcal{P}, \ell_\mathcal{Q}\rangle, \mathit{stmt}_\mathcal{P}, \ell'_\mathcal{P}\rangle \in \delta_{\mathcal{P}||\mathcal{Q}}$
  - *If* $\langle \ell_\mathcal{Q}, \mathit{stmt}_\mathcal{Q}, \ell'_\mathcal{Q}\rangle \in \delta_\mathcal{Q}$, *and* $\alpha_\mathcal{Q} \in I_\mathcal{Q} \setminus I_\mathcal{P}$ *or* $\varphi^F_\mathcal{P}(\ell_\mathcal{P})$, *then* $\langle\langle \ell_\mathcal{P}, \ell_\mathcal{Q}\rangle, \mathit{stmt}_\mathcal{Q}, \ell'_\mathcal{Q}\rangle \in \delta_{\mathcal{P}||\mathcal{Q}}$
  - *If* $\langle \ell_\mathcal{P}, \mathit{stmt}_\mathcal{P}, \ell'_\mathcal{P}\rangle \in \delta_\mathcal{P}$ *and* $\alpha_\mathcal{P} \in I_{\mathcal{P}\wedge\mathcal{Q}} \cup O_{\mathcal{P}\wedge\mathcal{Q}}$, *then* $\langle \ell_\mathcal{P}, \mathit{stmt}_\mathcal{P}, \ell'_\mathcal{P}\rangle \in \delta_{\mathcal{P}||\mathcal{Q}}$
  - *If* $\langle \ell_\mathcal{Q}, \mathit{stmt}_\mathcal{Q}, \ell'_\mathcal{Q}\rangle \in \delta_\mathcal{Q}$ *and* $\alpha_\mathcal{Q} \in I_{\mathcal{P}\wedge\mathcal{Q}} \cup O_{\mathcal{P}\wedge\mathcal{Q}}$, *then* $\langle \ell_\mathcal{Q}, \mathit{stmt}_\mathcal{Q}, \ell'_\mathcal{Q}\rangle \in \delta_{\mathcal{P}||\mathcal{Q}}$.
- $\varphi^F_{\mathcal{P}||\mathcal{Q}}$ *is defined by:*
  - $\varphi^F_{\mathcal{P}||\mathcal{Q}}(\langle \ell_\mathcal{P}, \ell_\mathcal{Q}\rangle) \iff \varphi^F_\mathcal{P}(\ell_\mathcal{P}) \wedge \varphi^F_\mathcal{Q}(\ell_\mathcal{Q})$
  - $\varphi^F_{\mathcal{P}||\mathcal{Q}}(\ell_\mathcal{P}) \iff \varphi^F_\mathcal{P}(\ell_\mathcal{P})$
  - $\varphi^F_{\mathcal{P}||\mathcal{Q}}(\ell_\mathcal{Q}) \iff \varphi^F_\mathcal{Q}(\ell_\mathcal{Q})$. □

4.4 Quotient

In analogy with the development in Section 2.2.4, we consider the quotient operation, which can be seen as an "inverse" of parallel composition and therefore has a connection to synthesis. Given a specification for a system $\mathcal{R}$, together with a component $\mathcal{P}$ implementing part of $\mathcal{R}$, the quotient, denoted $\mathcal{R} \setminus \mathcal{P}$, yields the *least refined* (in the sense of $\sqsubseteq$) process for the remaining part of $\mathcal{R}$, i.e., such that $\mathcal{P} \parallel (\mathcal{R} \setminus \mathcal{P}) \sqsubseteq \mathcal{R}$. Therefore, quotient can be thought of as an adjoint of parallel composition.

Looking at the treatment of sets of actions in Definition 4.4, we see that a necessary requirement for the existence of a quotient is that $O_\mathcal{P} \subseteq O_\mathcal{R}$. Then, the set of output actions of the quotient must be $O_\mathcal{R} \setminus O_\mathcal{P}$. However, there is some freedom for the set $I_{\mathcal{R}\setminus\mathcal{P}}$ of input actions of the quotient. From Definition 4.4, we take as a natural choice $I_{\mathcal{R}\setminus\mathcal{P}}$ to be $(O_\mathcal{P} \cup I_\mathcal{R})$, since a quotient with these actions will always exist if there is one with a smaller set, and since the subsequently presented technique to produce a quotient will not have to consider the difficulties that come with actions that are not visible by the quotient.

Say that a symbol $\alpha(\bar{d})$ *may be disabled* after a trace t if t is the sequence of symbols in a computation, which ends in a state where $\alpha(\bar{d})$ is not enabled. I.e., if there is a state $\langle \ell, \sigma\rangle$, such that $\langle \ell_0, \sigma_0\rangle \stackrel{t}{\Longrightarrow}_\mathcal{P} \langle \ell, \sigma\rangle$, where $\langle \ell_0, \sigma_0\rangle$ is the initial state, and such that $\alpha(\bar{d})$ is not enabled in $\langle \ell, \sigma\rangle$.

We can now construct the safe traces of the quotient $\mathcal{R} \setminus \mathcal{P}$ by first defining the situations that should be avoided when $\mathcal{P}$ is composed with $\mathcal{R} \setminus \mathcal{P}$. As follows from Proposition 4.3, these are that after some safe trace t of $\mathcal{R}$,

1. for an input symbol i of $\mathcal{R}$, the symbol i may be disabled after $t \restriction (I_\mathcal{P} \cup O_\mathcal{P})$ in $\mathcal{P}$, and
2. for an output symbol o of $\mathcal{P}$, the trace $to \restriction (I_\mathcal{P} \cup O_\mathcal{P})$ is a safe trace of $\mathcal{P}$ but the trace to is not a safe trace of $\mathcal{R}$.

Let us define the set $F_{\mathcal{P}\triangleright\mathcal{R}}$ as the set of safe traces of $\mathcal{R}$ for which one of the above situations occur.

The safe traces $ST_{\mathcal{R}\setminus\mathcal{P}}$ of the quotient can now be obtained as the maximal prefix-closed subset of $ST_\mathcal{R}$ such that if $t \in ST_{\mathcal{R}\setminus\mathcal{P}}$, then

- $t \notin F_{\mathcal{P} \triangleright \mathcal{R}}$, and
- if $to \in ST_{\mathcal{R}}$ for some output action $o \in \Sigma^{(O_{\mathcal{R}} \setminus O_{\mathcal{P}})}$, then $to \in ST_{\mathcal{R} \setminus \mathcal{P}}$.

This construction is similar in spirit to the one defined for deterministic interface automata without data by Bhaduri and Ramesh [9].

The alphabet of the quotient contains all of the actions of $\mathcal{R}$ and $\mathcal{P}$ so that $\mathcal{R} \setminus \mathcal{P}$ can control as much as possible $\mathcal{P}$ and emulate the behaviour of $\mathcal{R}$. Yet still, simple examples reveal that there may not exist a component $\mathcal{Q}$ over an interface consisting of input actions $I_{\mathcal{R} \setminus \mathcal{P}}$ and output actions $O_{\mathcal{R} \setminus \mathcal{P}}$ such that $\mathcal{P} \,||\, \mathcal{Q} \sqsubseteq_{dec} \mathcal{R}$. Unfortunately, the existence of the quotient cannot be ascertained by a syntactic check on the alphabets of $\mathcal{P}$ and $\mathcal{R}$.

In fact, for data-processes the question whether there exists a quotient is undecidable. This follows from the result that checking refinement between data-processes is undecidable (which follows from [58]).

4.4.1 Reformulating the Quotient Problem as a Synthesis Problem

Let us in this section provide a more constructive construction of the quotient, as a process of the generic form proposed in Definition 4.1. In the treatment in this subsection, we often use $\mathcal{Q}$ to refer to the quotient $\mathcal{R} \setminus \mathcal{P}$.

Due to the undecidability result referred to at the end of the previous section, our construction of qoutient will assume that processes $\mathcal{P}$ and $\mathcal{R}$ are deterministic (this concept will soon be defined). In this case, we will construct the quotient $\mathcal{Q}$ as the least refined deterministic process, whose set of safe traces satisfies the condition stated at the end of the previous subsection. To make these conditions more concrete, we will first rephrase the problem of finding a qoutient $\mathcal{Q}$ into a problem of finding a most general process $\mathcal{Q}$ whose parallel composition with a given process is guaranteed not to reach an inconsistent state. This "given process", which we denote $\mathcal{P} \triangleright \mathcal{R}$, is constructed from $\mathcal{P}$ and $\mathcal{R}$ by synchronizing their actions, in a way similar to, but not the same as, the parallel composition of $\mathcal{P}$ and $\mathcal{R}$. The resulting synthesis problem has a simpler structure, and can be addressed by adapting techniques from controller synthesis. We therefore think that this reformulation is a way to make our approach clearer.

Let us now perform these things.

Definition 4.7 We define a process $\mathcal{P}$ to be *deterministic* if for every location, guards of different transititons starting from the location with the same action are mutually exclusive. Formally, for any two transitions

$$\langle \ell \,,\, g_1; \alpha(\bar{p}); \bar{y}_1 := \bar{z}_1 \,,\, \ell_1' \rangle, \langle \ell \,,\, g_2; \alpha(\bar{p}); \bar{y}_2 := \bar{z}_2 \,,\, \ell_2' \rangle \in \delta\mathcal{P}$$

and for any valuation σ of variables of $X_{\mathcal{P}}$ and parameters of $\bar{p}$, it holds that

$$\sigma \models \neg g_1 \vee \neg g_2 \,.$$

□

Let us remark that the restriction to deterministic processes is not as severe as one might first think. For instance, it is possible to model processes that nondeterministically generate identifiers, sequence numbers, etc. by suitable modeling techniques. Consider for instance, a process which nondeterministically selects an identifier, assigns it to a variable id and then transmits it in an action *showid*(p), where p is id. We can represent this behavior by a single transition which is labeled by

$$\textit{true} \,;\, \textit{showid}(p) \,;\, id := p \;\;.$$

Note that the transition has no guard (or more precisely, the guard is *true*). Thus the parameter of *showid* can be an arbitrary identifier. We have just slightly remodeled the process so that id is assigned in connection with the transition, rather than before the transition; the external behavior remains the same. We use this modeling idiom in several of the examples of Section 4.5.

We will from now on consider the quotient construction $\mathcal{R} \setminus \mathcal{P}$ for the case that both $\mathcal{R}$ and $\mathcal{P}$ are deterministic.

The structure of our construction is the following. We first construct a new process, denoted $\mathcal{P} \triangleright \mathcal{R}$, which "summarizes" the requirements on $\mathcal{Q}$ that are imposed by $\mathcal{P}$ and $\mathcal{R}$. The process $\mathcal{P} \triangleright \mathcal{R}$ has the

property that the problem of finding a process $\mathcal{Q}$ such that $\mathcal{P} \mid\mid \mathcal{Q} \sqsubseteq \mathcal{R}$ is transformed into the (conceptually simpler) problem of finding a process $\mathcal{Q}$ which interacts with $\mathcal{P} \rhd R$ without causing an inconsistency. We thereafter show how to find such a $\mathcal{Q}$ by adaptation of controller synthesis techniques.

We construct the process $\mathcal{P} \rhd \mathcal{R}$ as the process obtained by running $\mathcal{P}$ and $\mathcal{R}$ in parallel, synchronizing on common symbols.

As a first step, given processes $\mathcal{P}$ and $\mathcal{R}$, let us define a mapping $NotRefine_{\mathcal{P}\rhd\mathcal{R}}$ from pairs of locations of $\mathcal{P}$ and $\mathcal{R}$ to predicates such that $NotRefine_{\mathcal{P}\rhd\mathcal{R}}(\langle \ell_\mathcal{P}, \ell_\mathcal{R} \rangle)$ is true in the situations that should be avoided, according to Criterion 1 or 2 above. We can represent $NotRefine_{\mathcal{P}\rhd\mathcal{R}}(\langle \ell_\mathcal{P}, \ell_\mathcal{R} \rangle)$ as a disjunction, which for input action $\alpha \in I_\mathcal{R}$ of $\mathcal{R}$ with $\alpha \in I_\mathcal{P}$ contains the disjunct $g^\alpha_{\mathcal{R},\ell_\mathcal{R}} \wedge \neg g^\alpha_{\mathcal{P},\ell_\mathcal{P}}$, and which for each output action $\alpha \in O_\mathcal{P}$ of $\mathcal{P}$ contains the disjunct $g^\alpha_{\mathcal{P},\ell_\mathcal{P}} \wedge \neg g^\alpha_{\mathcal{R},\ell_\mathcal{R}}$.

Definition 4.8 The synchronization $\mathcal{P} \rhd \mathcal{R}$ of two processes $\mathcal{P} = (I_\mathcal{P}, O_\mathcal{P}, L_\mathcal{P}, \ell^0_\mathcal{P}, X_\mathcal{P}, \delta_\mathcal{P}, \varphi^F_\mathcal{P})$ and $\mathcal{R} = (I_\mathcal{R}, O_\mathcal{R}, L_\mathcal{R}, \ell^0_\mathcal{R}, X_\mathcal{R}, \delta_\mathcal{R}, \varphi^F_\mathcal{R})$, is obtained as $\mathcal{P} \rhd \mathcal{R} = (I_{\mathcal{P}\rhd\mathcal{R}}, O_{\mathcal{P}\rhd\mathcal{R}}, L_{\mathcal{P}\rhd\mathcal{R}}, \ell^0_{\mathcal{P}\rhd\mathcal{R}}, X_{\mathcal{P}\rhd\mathcal{R}}, \delta_{\mathcal{P}\rhd\mathcal{R}}, \varphi^F_{\mathcal{P}\rhd\mathcal{R}})$, where

- $I_{\mathcal{P}\rhd\mathcal{R}} = O_\mathcal{R} \setminus O_\mathcal{P}$,
- $O_{\mathcal{P}\rhd\mathcal{R}} = O_\mathcal{P} \cup I_\mathcal{R}$,
- $L_{\mathcal{P}\rhd\mathcal{R}} = L_\mathcal{P} \times L_\mathcal{R}$,
- $\ell^0_{\mathcal{P}\rhd\mathcal{R}} = \langle \ell^0_\mathcal{P}, \ell^0_\mathcal{R} \rangle$,
- $X_{\mathcal{P}\rhd\mathcal{R}} = X_\mathcal{P} \cup X_\mathcal{R}$,
- $\delta_{\mathcal{P}\rhd\mathcal{R}}$ is obtained from $\delta_\mathcal{P}$ and $\delta_\mathcal{R}$ by two forms of transitions.
  - If $\langle \ell_\mathcal{R}, \mathit{stmt}, \ell'_\mathcal{R} \rangle \in \delta_\mathcal{R}$ has an action which is not an action of $\mathcal{P}$, then $\langle \langle \ell_\mathcal{P}, \ell_\mathcal{R} \rangle, \mathit{stmt}, \langle \ell_\mathcal{P}, \ell'_\mathcal{R} \rangle \rangle \in \delta_{\mathcal{P}\rhd\mathcal{R}}$ for any location $\ell_\mathcal{P} \in L_\mathcal{P}$.
  - If $\langle \ell_\mathcal{P}, \mathit{stmt}_\mathcal{P}, \ell'_\mathcal{P} \rangle \in \delta_\mathcal{P}$ and $\langle \ell_\mathcal{R}, \mathit{stmt}_\mathcal{R}, \ell'_\mathcal{R} \rangle \in \delta_\mathcal{R}$ have the same action, i.e., they synchronize, then $\langle \langle \ell_\mathcal{P}, \ell_\mathcal{R} \rangle, \mathit{stmt}_\mathcal{P} \| \mathit{stmt}_\mathcal{R}, \langle \ell'_\mathcal{P}, \ell'_\mathcal{R} \rangle \rangle \in \delta_{\mathcal{P}\rhd\mathcal{R}}$.
- $\varphi^F_{\mathcal{P}\rhd\mathcal{R}}$ is defined as

$$\varphi^F_{\mathcal{P}\rhd\mathcal{R}}(\langle \ell_\mathcal{P}, \ell_\mathcal{R} \rangle) = \left[\varphi^F_\mathcal{P}(\langle \ell_\mathcal{P} \rangle) \wedge \neg \varphi^F_\mathcal{R}(\langle \ell_\mathcal{R} \rangle)\right] \vee NotRefine_{\mathcal{P}\rhd\mathcal{R}}(\langle \ell_\mathcal{P}, \ell_\mathcal{R} \rangle) \; .$$

□

Having defined the auxiliary process $\mathcal{P} \rhd \mathcal{R}$, we can now reformulate the quotient problem into a more standard synthesis problem

Definition 4.9 *We say that processes $\mathcal{P}$ and $\mathcal{Q}$* collaborate harmoniously *if $I_\mathcal{Q} = O_\mathcal{P}$ and $O_\mathcal{Q} = I_\mathcal{P}$, and $\mathcal{P} \mid\mid \mathcal{Q}$ has a nonempty set of safe traces.*

Intuitively, $\mathcal{P}$ and $\mathcal{Q}$ collaborate harmoniously if neither $\mathcal{P}$ nor $\mathcal{Q}$ will produce an output symbol which the other process is not prepared to receive, or which causes some process to reach an inconsistency. We now get

Proposition 4.10 *Let $\mathcal{P}$ and $\mathcal{R}$ be as assumed above, such that $(I_\mathcal{P} \cup O_\mathcal{P}) \subseteq (I_\mathcal{R} \cup O_\mathcal{R})$ and $O_\mathcal{P} \subseteq O_\mathcal{R}$. Let $\mathcal{Q}$ be a process with $I_\mathcal{Q} = O_\mathcal{P} \cup I_\mathcal{R}$ and $O_\mathcal{Q} = O_\mathcal{R} \setminus O_\mathcal{P}$. Then $\mathcal{P} \mid\mid \mathcal{Q} \sqsubseteq \mathcal{R}$ iff $(\mathcal{P} \rhd R)$ and $\mathcal{Q}$ collaborate harmoniously.* □

4.4.2 Solving the Synthesis Problem

If $\mathcal{P}$ and $\mathcal{R}$ are as above, then it follows that $(\mathcal{P} \triangleright R)$ is deterministic. Due to Proposition 4.10, we will from now on consider the following synthesis problem:

> Given a deterministic process $\mathcal{P}$, find the least refined process $\mathcal{Q}$ such that $\mathcal{P}$ and $\mathcal{Q}$ collaborate harmoniously.

To be able to collaborate harmoniously, $\mathcal{Q}$ should maintain as much relevant information as possible about the "current states" of $\mathcal{P}$. More precisely, from the past sequence of exchanged symbols, $\mathcal{Q}$ can infer the current location of $\mathcal{P}$ and values of its variables. A most general $\mathcal{Q}$ should keep track of this information, since any piece of it may become important when performing an output action. A process $\mathcal{Q}$ which has remembered more of this information can do "better" choices than a process which has remembered less, and so can be at least as general. The restriction to deterministic processes guarantees that $\mathcal{Q}$ can maintain *complete* information about the location and values of variables of $\mathcal{P}$.

Our solution to the synthesis problem is essentially obtained using two constructions. We first define a *control skeleton* $\mathcal{S}$ for $\mathcal{Q}$. It stores the location of $\mathcal{P}$ in its control location (i.e., its locations are locations of $\mathcal{P}$), its input and output actions are $O_\mathcal{P}$ and $I_\mathcal{P}$, it contains a primed variable x' for each variable x of $\mathcal{P}$, and in every location $\ell_\mathcal{P}$, it may perform transitions which synchronise with actions that $\mathcal{P}$ can perform in $\ell_\mathcal{P}$. It copies the guards of $\mathcal{P}$ and the assignments performed by $\mathcal{P}$ (but replaces occurrences of variables of $\mathcal{P}$ in the guards by their primed counterpart). This means that when $\mathcal{S}$ synchronises with an input statement of $\mathcal{P}$, it transmits an arbitrary value (it does not guard the formal parameters by any additional guard). When $\mathcal{S}$ synchronises with an output statement of $\mathcal{P}$, it records newly generated values created and sent by $\mathcal{P}$. For instance, in the case the statement of $\mathcal{P}$ is of the form $!\alpha(p); x := x$, $\mathcal{S}$ stores p to the primed variable x'. The control skeleton is formally defined as the process $\mathcal{S} = (I_\mathcal{S}, O_\mathcal{S}, L_\mathcal{S}, \ell^0_\mathcal{S}, X_\mathcal{S}, \delta_\mathcal{S}, \varphi^F_\mathcal{S})$, where

- $I_\mathcal{S} = O_\mathcal{P}$,
- $O_\mathcal{S} = I_\mathcal{P}$,
- $L_\mathcal{S} = L_\mathcal{P}$,
- $\ell^0_\mathcal{S} = \ell^0_\mathcal{P}$,
- $X_\mathcal{S} = \{x' \mid x \in X_\mathcal{P}\}$,
- $\varphi^F_\mathcal{S}$ assigns *false* to every location of $\mathcal{S}$,
- $\delta_\mathcal{P}$ contains a transition $\langle \ell,\, g'; \alpha(p_1, \ldots, p_n),\, \ell' \rangle; \bar{x}' := \bar{z}'$ for every transition $\langle \ell,\, g; \alpha(p_1, \ldots, p_n); \bar{x} := \bar{z},\, \ell' \rangle$ of $\mathcal{P}$ where g', x', and z' are obtained from g, x, and z by replacing occurrences of variables of $X_\mathcal{P}$ by their primed versions.

$\mathcal{Q}$ will be later obtained from $\mathcal{S}$ by refining its transitions by additional guards that ensure that $\mathcal{P} \,||\, \mathcal{Q}$ avoids inconsistent traces. To achieve this, the guards on transitions of $\mathcal{Q}$ should prevent computations of $\mathcal{P} \,||\, \mathcal{Q}$ to reach a *bad state*, which is either an inconsistent state or a state from where $\mathcal{Q}$ cannot prevent $\mathcal{P} \,||\, \mathcal{Q}$ to reach a bad state. Since we are interested in the least refined process $\mathcal{Q}$, we want to add as weak guards as possible.

We will show how to compute a characterisation of bad states in a form of a function Bad which assigns to every state $\ell_\mathcal{P}$ of $\mathcal{P}$ a boolean formula $Bad(\ell_\mathcal{P})$ over equalities and inequalities over $X_\mathcal{P}$. A state $\langle\langle \ell_\mathcal{P}, \ell_\mathcal{S} \rangle, \sigma\rangle$ of $\mathcal{P} \,||\, \mathcal{S}$ is bad iff σ restricted to $X_\mathcal{P}$ satisfies $Bad(\ell_\mathcal{P})$.

Computing Bad States

Calculation of Bad is the key step in the synthesis process. It corresponds to computation of a controller invariant for $\mathcal{Q}$. It is done by starting from the set of inconsistent states given by the mapping $\varphi^F_\mathcal{P}$.

From the set of inconsistent states, we derive the set of bad states, which is done by taking predecessors of inconsistent and also bad states under statements that are not output statements of $\mathcal{S}$. Predecessor computation is performed as follows. Let φ be a knowledge predicate. Then

$$\begin{array}{lcl} pre(\overline{x} := \overline{y}\ ;\ \varphi) & = & \varphi[\overline{y}/\overline{x}] \\ pre(g;\ \varphi) & = & \varphi \wedge g \end{array}$$

We can put the above rules together to derive the precondition of a statement as follows.

$$pre(g; \alpha(\overline{p}); \overline{y} := \overline{z}\ ;\ \varphi)\ =\ \exists \overline{p}\ .\ (g \wedge \varphi[\overline{z}/\overline{y}])$$

Application of an existential quantifier here involves two steps, saturation and elimination. Assume that φ is a formula in disjunctive normal form (DNF) over equalities and inequalities over a set of variables. Formula $\exists(x_1, \ldots, x_n)\ .\ \varphi$ is obtained by transforming every clause of φ in the following way:

1. Saturation: (a) saturate equalities, i.e. compute the transitive and reflexive closure of the equality relation $=$; (b) saturate inequalities, i.e. include an inequality $x \neq y$ whenever there is some z with $x \neq z \wedge z = y$.

2. Elimination: remove all predicates involving some of $x_1, \ldots, x_n$.

We can now give an algorithm for computing Bad, presented as Algorithm 1.

Algorithm 1: Computing Bad

```
1 Bad := φ_P^F;
2 repeat
3     Bad_prev := Bad;
4     forall output transitions ⟨ℓ_P, stmt, ℓ'_P⟩ of P do
5         Bad(ℓ_P) := Bad(ℓ_P) ∨ pre(stmt; Bad(ℓ'_P))
6 until Bad = Bad_prev ;
```

Computing $\mathcal{Q}$

Having computed Bad, we compute $\mathcal{Q}$ from $\mathcal{S}$. We first obtain safe as the complement of Bad. Then $\mathcal{Q}$ is the process $\mathcal{Q} = (I_\mathcal{Q}, O_\mathcal{Q}, L_\mathcal{Q}, \ell^0_\mathcal{Q}, X_\mathcal{Q}, \delta_\mathcal{Q}, \varphi^F_\mathcal{Q})$ where

- $I_\mathcal{Q} = O_\mathcal{S}$,
- $O_\mathcal{Q} = I_\mathcal{S}$,
- $L_\mathcal{Q} = L_\mathcal{S}$,
- $\ell^0_\mathcal{Q} = \ell^0_\mathcal{S}$,
- $X_\mathcal{Q} = X_\mathcal{S}$, and
- $\varphi^F_\mathcal{Q} = \varphi^F_\mathcal{S}$.

The input transitions of $\mathcal{Q}$ are the same as the input transitions of $\mathcal{S}$. Output transitions of $\mathcal{Q}$ will be obtained from output transitions of $\mathcal{S}$ by equipping them with additional guards which keep computations of $\mathcal{P}\ ||\ \mathcal{Q}$ within safe. For every output transition $\langle \ell_\mathcal{P}, \textit{stmt}, \ell'_\mathcal{P}\rangle$ of $\mathcal{S}$, there is a unique corresponding input transition $\langle \ell_\mathcal{P}, \textit{stmt}', \ell'_\mathcal{P}\rangle$ of $\mathcal{P}$. Let *stmt* $||$ *stmt*′ be of form $g; \alpha(\overline{p}); \overline{y} := \overline{z}$. We calculate the precondition wrt. $\mathsf{safe}(\ell'_\mathcal{P})$ as $\psi \triangleq \exists X_\mathcal{P}.\ g \wedge \mathsf{safe}(\ell'_\mathcal{P})[\overline{z}/\overline{y}]$. Thereafter, the transition $\langle \ell_\mathcal{P}, \psi \wedge \textit{stmt}', \ell'_\mathcal{P}\rangle$ is added to $\mathcal{Q}$.

Pruning $\mathcal{Q}$

The $\mathcal{Q}$ obtained by the above method satisfies the requirement for being a least refined process such that $\mathcal{P}$ and $\mathcal{Q}$ collaborate harmoniously. However, it may contain a significant amount of redundancy. Some of its states can never be reached, some transitions can never be taken, and some parts of conditions within guards are true for every computation reaching the source location of the transition.

To obtain a more optimal solution, we will prune the redundant parts of $\mathcal{Q}$. We compute the function $Reach$ that for every state $\langle \ell_\mathcal{P}, \ell_\mathcal{Q}\rangle$ of $\mathcal{P}\ ||\ \mathcal{Q}$ returns a formula $Reach(\langle \ell_\mathcal{P}, \ell_\mathcal{Q}\rangle)$ such that there exists

a computation of $\mathcal{P} \mid\mid \mathcal{Q}$ reaching a state $\langle\langle \ell_{\mathcal{P}}, \ell_{\mathcal{Q}}\rangle, \sigma\rangle$ iff σ satisfies $Reach(\langle \ell_{\mathcal{P}}, \ell_{\mathcal{Q}}\rangle)$. The algorithm for computing $Reach$ is given below. Then, for every state $\ell_{\mathcal{Q}}$ of $\mathcal{Q}$, we will compute

$$Reach(\ell_{\mathcal{Q}}) = \exists X_{\mathcal{P}}. \bigvee_{\ell_{\mathcal{P}} \in L_{\mathcal{P}}} Reach(\langle \ell_{\mathcal{P}}, \ell_{\mathcal{Q}}\rangle)$$

The formula $Reach(\ell_{\mathcal{Q}})$ characterises all possible valuations of variables of $\mathcal{Q}$ that can appear in a computation of $\mathcal{P} \mid\mid \mathcal{Q}$ together with $\ell_{\mathcal{Q}}$. We then prune $\mathcal{Q}$ as follows:

- remove states $\ell_{\mathcal{Q}}$ such that $Reach(\ell_{\mathcal{Q}}) =$ *false*
- remove transitions $\langle \ell_{\mathcal{Q}},\ g; \alpha(\overline{p}); \overline{y} := \overline{z},\ \ell'_{\mathcal{Q}}\rangle$ such that $Reach(\ell_{\mathcal{Q}}) \Rightarrow \neg g$
- For each transition $\langle \ell_{\mathcal{Q}},\ g; \alpha(\overline{p}); \overline{y} := \overline{z},\ \ell'_{\mathcal{Q}}\rangle$, we replace the guard g by $Reach(\ell_{\mathcal{Q}}) \Rightarrow g$ (and possibly transform this formula to DNF and remove redundant literals and clauses).

We will compute values of $Reach$ for states of $\mathcal{P} \mid\mid \mathcal{Q}$ by abstract postcondition computation on $\mathcal{P} \mid\mid \mathcal{Q}$ starting from the strongest condition that holds in its initial state. To do this, we will need to compute post-conditions for predicates with respect to the different forms of synchronization statements in processes. Let φ be a formula over equalities and inequalities over $X_{\mathcal{P}||\mathcal{Q}}$. Then

$$\begin{array}{lcl} post(\overline{x} := \overline{y}\ ;\ \varphi) & = & (\exists \overline{x}.\varphi) \wedge \overline{x} = \overline{y} \\ post(g;\ \varphi) & = & \varphi \wedge g \end{array}$$

We can put the above rules together to derive the postcondition of a statement as follows.

$$post(g; \alpha(\overline{p}); \overline{y} := \overline{z}\ ;\ \varphi)\ =\ \exists \overline{p}\ .\ (\exists \overline{y}\ .\ (g \wedge \varphi) \wedge \overline{y} = \overline{z})$$

Let φ_0 be the strongest predicate satisfied by the initial valuation of $\mathcal{P} \mid\mid \mathcal{Q}$. The computation of $Reach$ is then carried out by Algorithm 2.

Algorithm 2: Computing $Reach$

```
1 Initialise Reach as φ0 for ⟨ℓ0_P, ℓ0_Q⟩ and false otherwise;
2 repeat
3     Reach_prev := Reach;
4     forall transitions ⟨⟨ℓ_P, ℓ_Q⟩, stmt, ⟨ℓ'_P, ℓ'_Q⟩⟩ of P || Q do
5         Reach(⟨ℓ'_P, ℓ'_Q⟩) := Reach(⟨ℓ'_P, ℓ'_Q⟩) ∨ post(stmt; Reach(⟨ℓ_P, ℓ_Q⟩))
6 until Reach = Reach_prev ;
```

Modelling Deadlock

In analogy with the treatment in Chapter 3, we also extend our framework so that it allows specification of deadlock freeness properties and subsequently synthesis of quotients that guarantee the deadlock freeness properties. We say that a state $\langle \ell, \sigma\rangle$ of a computation of a process $\mathcal{P}$ is *deadlock* if no output transition can be taken from ℓ in the current valuation σ without reaching a bad state. Formally, we specify deadlock as the predicate $Deadlock_{\mathcal{P}}^{Bad}$ parametrised by Bad as follows

$$Deadlock_{\mathcal{P}}^{Bad}(\ell)\ \triangleq\ \bigwedge\nolimits_{\substack{\langle \ell\ ,\ g; \alpha(\overline{p}); \overline{x} := \overline{y}\ ,\ \ell' \rangle \in \delta_{\mathcal{P}} \\ \alpha \in I_{\mathcal{P}}}} \forall \overline{p}\ .\ g\ \Longrightarrow\ Bad(\ell')[\overline{y}/\overline{x}]$$

($Deadlock_{\mathcal{P}}^{Bad}(\ell)$ is automatically *true* if ℓ has no output transitions). We specify in which situations a process is allowed to deadlock by providing a specification of the so called *quiescent* states. Similarly as $\varphi_{\mathcal{P}}^{F}$, the map $Quiescent_{\mathcal{P}}$ maps each location to a boolean combination of equalities and negated equalities over variables in $X_{\mathcal{P}}$. A consistent computation is not allowed to reach a deadlock state which is not quiescent. That means that a computation is now inconsistent if it contains a state $\langle \ell, \sigma\rangle$ such that

$$\sigma\ \models\ \varphi_{\mathcal{P}}^{F}(\ell)\ \vee\ \Big(Deadlock_{\mathcal{P}}^{Bad}(\ell)\ \wedge\ \neg Quiescent_{\mathcal{P}}(\ell)\Big)\ .$$

Algorithm 3: Computing Bad when absence of deadlock is required

$Bad := \varphi^F_{\mathcal{P}}$;
repeat
  $Bad_{prev} := Bad$;
  forall *output transitions* $\langle \ell_{\mathcal{P}}, \mathit{stmt}, \ell'_{\mathcal{P}} \rangle$ *of* $\mathcal{P}$ **do**
    $Bad(\ell_{\mathcal{P}}) := Bad(\ell_{\mathcal{P}}) \vee pre(\mathit{stmt}; Bad(\ell'_{\mathcal{P}}))$
  forall $\ell_{\mathcal{P}} \in L_{\mathcal{P}}$ **do**
    $Bad(\ell_{\mathcal{P}}) := Bad(\ell_{\mathcal{P}}) \vee (Deadlock^{Bad}_{\mathcal{P}}(\ell_{\mathcal{P}}) \wedge \neg Quiescent_{\mathcal{P}}(\ell_{\mathcal{P}}))$
until $Bad = Bad_{prev}$;

This is reflected in Algorithm 3 for computing the set of bad states under a deadlock freeness requirement.

The treatment of parallel composition and quotient stays the same as before with the difference that we define:

- $Quiescent_{\mathcal{P}||\mathcal{Q}}(\langle \ell_{\mathcal{P}}, \ell_{\mathcal{Q}} \rangle) = Quiescent_{\mathcal{P}}(\ell_{\mathcal{P}}) \wedge Quiescent_{\mathcal{Q}}(\ell_{\mathcal{Q}})$, and
- $Quiescent_{\mathcal{P} \rhd \mathcal{R}}(\langle \ell_{\mathcal{P}}, \ell_{\mathcal{R}} \rangle) = Quiescent_{\mathcal{P}}(\ell_{\mathcal{P}}) \Rightarrow Quiescent_{\mathcal{R}}(\ell_{\mathcal{R}})$.

4.5 Applications to the Synthesis of Mediators

To demonstrate our formulation in practice, we choose three examples. First, the presented concepts are demonstrated in detail on a simple example with messaging protocols. Then we present an example with protocols for booking seats in a theatre and a slightly more complicated example with an e-commerce service.

In all the examples, we assume the input processes to be preprocessed so that the inner structure of communicated values is not important. For example, suppose that our task is to synchronise a mediator for a client and a server in an e-commerce application. The client orders an item using an output action $!order(item\_object)$ where $item\_object$ is an object consisting of two values: value $item$ specifying the ordered item and value $quantity$ specifying how many of items the client orders. On the other hand, the service receives an order in a sequence of two actions, $?OrderItem(p); Item := p$ and $?SetQuantity(p); Quantity := p$. The mediator should guarantee that the service gets the clients orders with right values of items and quantities. In this case, the preprocessing phase replaces the action $!order(item\_object)$ by $!order(item, quantity)$ in order to make the components of $item\_object$ visible for the synthesis algorithm.

4.5.1 Messaging Protocol

The scenario consists of two mismatching messaging protocols, $CFRing$ and $XMPP$, both of which need to communicate with one another through a mediator that we must construct.

The safe traces of the messaging clients are represented by the two processes shown in Figure 4.1. In the description, we omit guards of transitions that are *true*. All variables of every sort are initialised by a special value $\perp$. There is a value $\perp$ for every sort of variables.

- The process $CFRing$ has input actions $start$, output actions $msgout$ and id, variables $msg1$ and $id1$, and locations $\{l_0, l_1, l_2, l_3, l_4\}$.
- The process $XMPP$ has input actions $msgin$ and who, variables $msg2$ and $id2$, and locations $\{m_0, m_1, m_2, m_3\}$.

We specify the composed system by a process $\mathcal{R}$ shown in Figure 4.2 which initially performs the action $start$, thereafter can perform any sequence of actions $msgout$, $msgin$, id, and who, and which thereafter outputs the action $\checkmark$, which will synchronise with the $\checkmark$ actions of the two processes only if they properly agree on the values of the variables. To specify that the communication should stop only when both

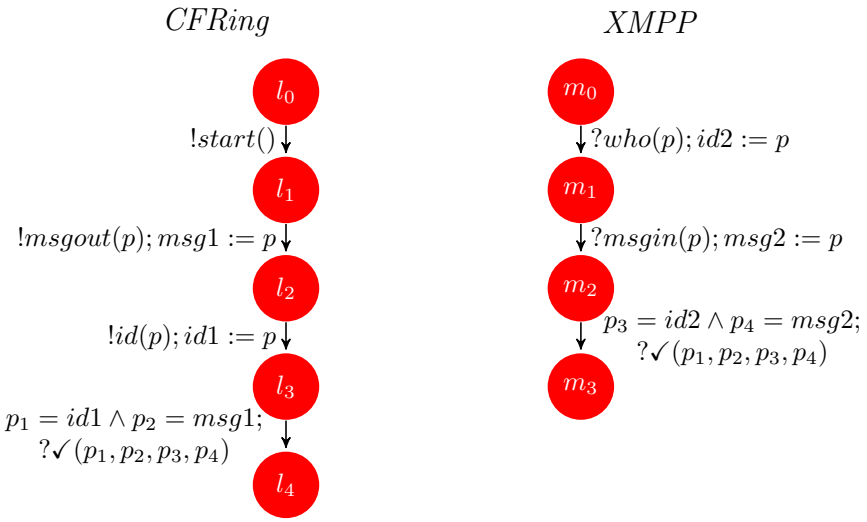

Figure 4.1: Mediator synthesis problem for $CFRing$ and $XMPP$

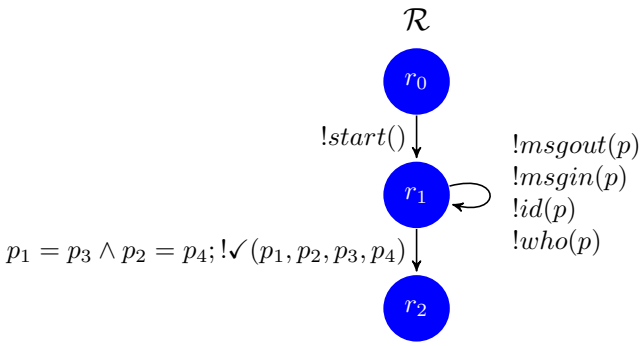

Figure 4.2: Specification of the synthesis problem with messaging in the form of the process $\mathcal{R}$.

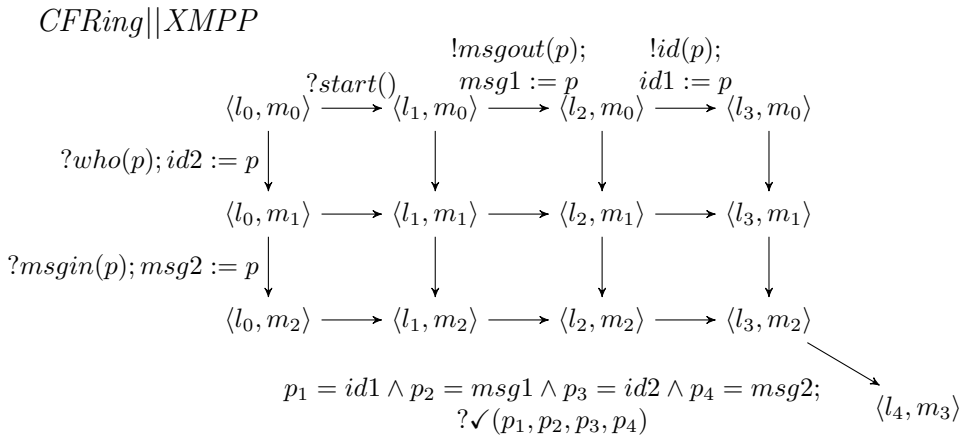

Figure 4.3: The process $CFRing \mid\mid XMPP$. The statements of the transitions marked by vertical arrows are defined by the labels on the left margin of the figure. The statements of the transitions marked by horizontal arrows are defined by the labels on the top margin of the figure.

processes reach their terminal states, we define $Quiescent_{\mathcal{R}}$ to be *false* for r_0, r_1 and *true* for r_2. We follow the scheme of the previous sections, and compute first the process $CFRing \mid\mid XMPP$ showed in Figure 4.3 which inherits all the transitions of $CFRing$ and $XMPP$ unsynchronized. We are looking for the least refined process $\mathcal{Q}$ such that $(CFRing \mid\mid XMPP) \mid\mid \mathcal{Q} \sqsubseteq \mathcal{R}$. The next step is the construction of the process $\mathcal{P} = (CFRing \mid\mid XMPP) \rhd \mathcal{R}$, which is shown in Figure 4.4. Notice that its only quiescent state is $\langle\langle l_4, m_3\rangle, r_2\rangle$. This is, $Quiescent_{\mathcal{P}}(\langle\langle l_4, m_3\rangle, r_2\rangle) =$ *true* and it evaluates to *false* for all the other locations of $\mathcal{P}$. $\mathcal{Q}$ will now be obtained as the least refined process such that $\mathcal{Q} \mid\mid \mathcal{P}$ cannot reach an inconsistent state. The inconsistent states here are only those which are deadlock and not quiescent. Since $Bad(\ell_{\mathcal{P}})$ is initially *false* for all locations of $\mathcal{P}$, the computation starts by evaluating $Deadlock_{\mathcal{P}}^{Bad}$. Let us pick the location $\ell = \langle\langle l_3, m_2\rangle, r_1\rangle$. $Deadlock_{\mathcal{P}}^{Bad}(\ell)$ evaluates initially to the disjunction $\varphi = id1 \neq id2 \vee msg1 \neq msg2$ which falsifies the guard of the only output transition of ℓ (leading to $\langle\langle l_4, m_4\rangle, r_2\rangle$). We set $Bad(\ell)$ to φ and we then propagate $Bad(\ell) = \varphi$ backwards via output transitions of $\mathcal{P}$. We compute the precondition of φ with respect to the transition from $\ell' = \langle\langle l_2, m_2\rangle, r_1\rangle$ to ℓ. We see that it becomes *true* (particularly, the disjunct $id2 \neq id1$ is first transformed into $id2 = p$ which is then turned into *true* by the existential quantification over p). We set $Bad(\ell')$ to *true* which means that $\mathcal{Q}$ must avoid visiting ℓ' in all situations. Notice that it is indeed the case that if $\mathcal{Q}$ allows $XMPP$ to reach state m_2 (where the value of $msg2$ is already fixed) before $msg1$ is generated by $CFRing$, then $\mathcal{Q}$ cannot guarantee $msg1 = msg2$ at the end of the communication which may lead to an undesired deadlock. Let us now look at the precondition of φ wrt. the transition from $\langle\langle l_3, m_1\rangle, r_1\rangle$. Since $Bad(\ell)$ is now φ, $Deadlock_{\mathcal{P}}^{Bad}(\langle\langle l_3, m_1\rangle, r_1\rangle)$ evaluates to $id2 \neq id1$ (particularly, the disjunct $msg1 \neq msg2$ turns into $msg1 \neq p$ which after the universal quantification over p becomes *false*). Intuitively, this reflects the fact that when $CFRing$ is in state l_3 and $XMPP$ in state m_1, the values of $id1$ and $id2$ should be already equal since they cannot become equal otherwise. The rest of the symbolic backward computation of Bad is carried out analogously.

The construction of $\mathcal{Q}$ starts by assembling its control skeleton $\mathcal{S}$ depicted in Figure 4.5. Guards are then added to its transitions to guarantee that any computation stays in safe. Finally, we prune unreachable states, useless transitions, and redundant guard conditions as described in Section 4.4.2. The resulting mediator $\mathcal{Q}$ is in Figure 4.6.

4.5.2 Booking Seats

The next example is given as an incompatible client $Client$ and a server $Server$ shown in Figure 4.7 which are meant to implement a service for booking seats in a theatre. Our task is again to synthesize a mediator $\mathcal{Q}$. The goal is to achieve that both processes arrive to the end of the transaction and when this happens, they agree on all important values. This is specified by process $\mathcal{R}$ in Figure 4.8. $Quiescent_{\mathcal{R}}$ is defined

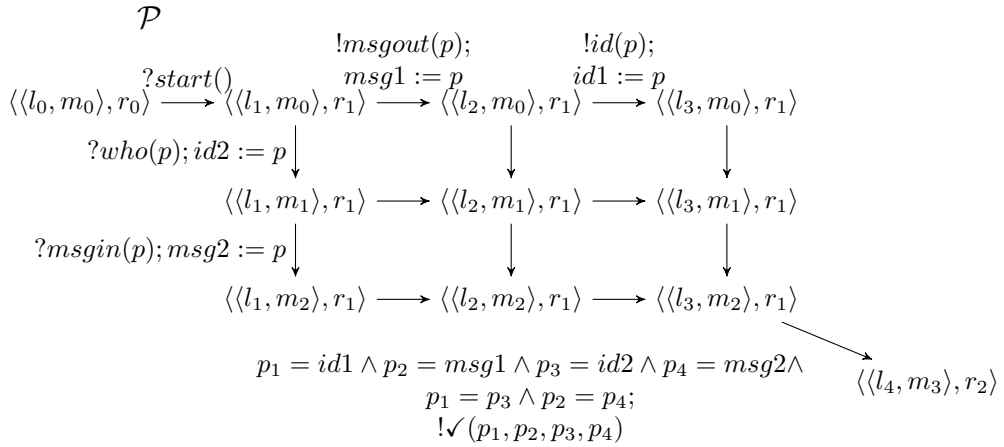

Figure 4.4: The process $\mathcal{P} = (CFRing \mid\mid XMPP) \rhd \mathcal{R}$. The statements of the transitions marked by vertical and horizontal arrows are defined by the labels on the top and the left margin of the figure (as in Figure 4.3).

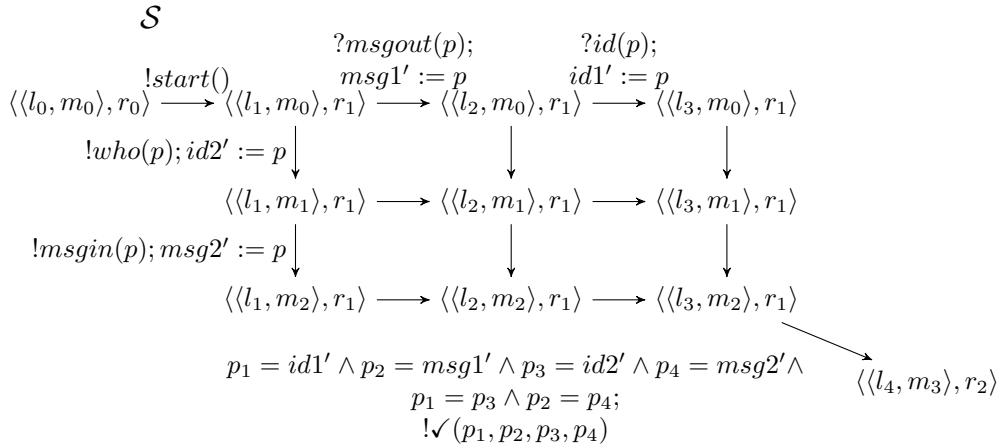

Figure 4.5: The skeleton $\mathcal{S}$ of $\mathcal{Q}$.

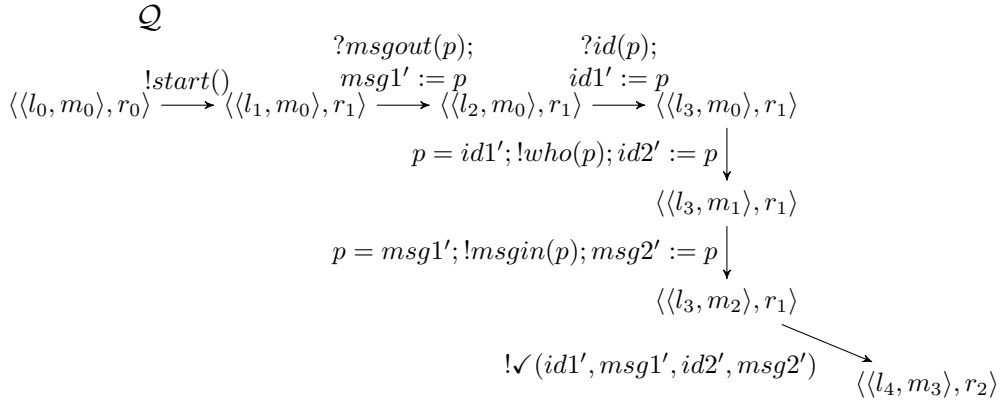

Figure 4.6: The resulting mediator $\mathcal{Q}$ for the application with messaging.

as *true* for r_2 and *false* otherwise. As before, we take $\mathcal{P} = (Client \mid\mid Server) \rhd \mathcal{R}$ and construct $\mathcal{Q}$ as the least refined process such that $(\mathcal{P} \rhd \mathcal{R}) \mid\mid \mathcal{Q}$ cannot reach an inconsistent state. The mediator $\mathcal{Q}$ obtained by our synthesis algorithm from Section 4.4.2 is in Figure 4.9.

4.5.3 E-Commerce

The following example is slightly more complex than previous ones. We are given a client and a customer service applications in Figure 4.10 that together are supposed to implement an e-commerce service, but are incompatible. The client $Blue$ first starts by sending its id and expects to receive an id of a new order. Then it orders a number of items in some quantities using the $AddOrder$ action, provides its payment informations using $PlaceOrder$ action, and expects all items together with its quantities to be confirmed by the customer service. It blocks in the case that it does not receive the right confirmation. The client then announces that he is ready to quit the transaction and expects to receive the result of the payment transaction $Result$.

The customer service $Moon$ expects to receive the clients id, then it sends a confirmation and sends an id of a new order, together with a client verification. It is then prepared to repeat a loop in which it 1) receives an order of an item, 2) receives a quantity in which the item is ordered, and 3) confirms that the item in the given quantity is ordered. After that, it receives payment information, it arranges the payment via a third party service (which is invisible to the client and not modelled here), and sends the result of the payment transaction.

Ideally, we would like to model the scenario where the client can order any number of items. However, our modelling mechanism allows only processes with a finite number of variables. This scenario would require processes with unbounded number of variables both for specification of $Blue$ and $Moon$ services and for specification, as well as for mediator. We must restrict ourselves to the case where the number of ordered items is bounded by a constant n, which becomes a parameter of the synthesis problem.

We assume that variables of the same sort are initialised with the same initial values $\perp$. For instance, all variables $item_1, item_2, Item_1, Item_2$ have the same initial value $\perp$.

The specification (parameterised by the maximum number of ordered items n) is given by the process $\mathcal{R}(n)$ in Figure 4.11. $Quiescent_{\mathcal{R}}$ is defined as *true* for r_2 and *false* otherwise. Similarly as in the previous examples, $\mathcal{R}$ captures that both sides finish the transaction and that at the end of the transaction, both sides agree on all the important values.

The synthesised mediator for the case when $n = 2$ is shown in Figure 4.12. It was obtained as the least refined process $\mathcal{Q}(2)$ such that $((Blue(2) \mid\mid Moon(2)) \rhd \mathcal{R}(2)) \mid\mid \mathcal{Q}(2)$ does not generate an inconsistent trace and subsequently pruned as described in Section 4.4.2. The figure displays only the component of states of $\mathcal{Q}(2)$ which is a states of $Blue(2) \mid\mid Moon(2)$.

The mediator first brings the system to the point when $Blue$ has ordered its first item (the state $\langle b_2^1, m_{3a}^0 \rangle$). It can now decide to start forwarding the first order to $Moon$ (vertical transitions). At the same time, it has to be ready to receive either the second order or the credit card details from $Blue$ (the horizontal transitions). If $Blue$ ordered also the second item, the mediator will forward the second order to $Moon$ after it has forwarded the first one. The mediator will send the payment credit card details to $Moon$ only a after it has received it from $Blue$ and after it has forwarded all orders of $Blue$ (this is taken care of by the guard of the statement $stmt_d^*$). It waits for $Blue$ to confirm all orders. Sending confirmations to $Blue$ is independent from receiving confirmations from $Moon$ since to send the right confirmations, the mediator only needs to know what was ordered by $Blue$. The communication with $Blue$ and $Moon$ can be interleaved in many ways.

4.6 Summary and Future Work

We have extended the compositional specification theory in Chapter 2 with data, using mechanisms that have been naturally adapted from similar other works in the literature. The resulting theory naturally bridges the models developed and used in WP4 (see Chapter 2 of D4.3), and can also be used to formulate the synthesis problem addressed in WP3, as we illustrated on some examples. A major contribution is the integration of these elements in a single specification framework.

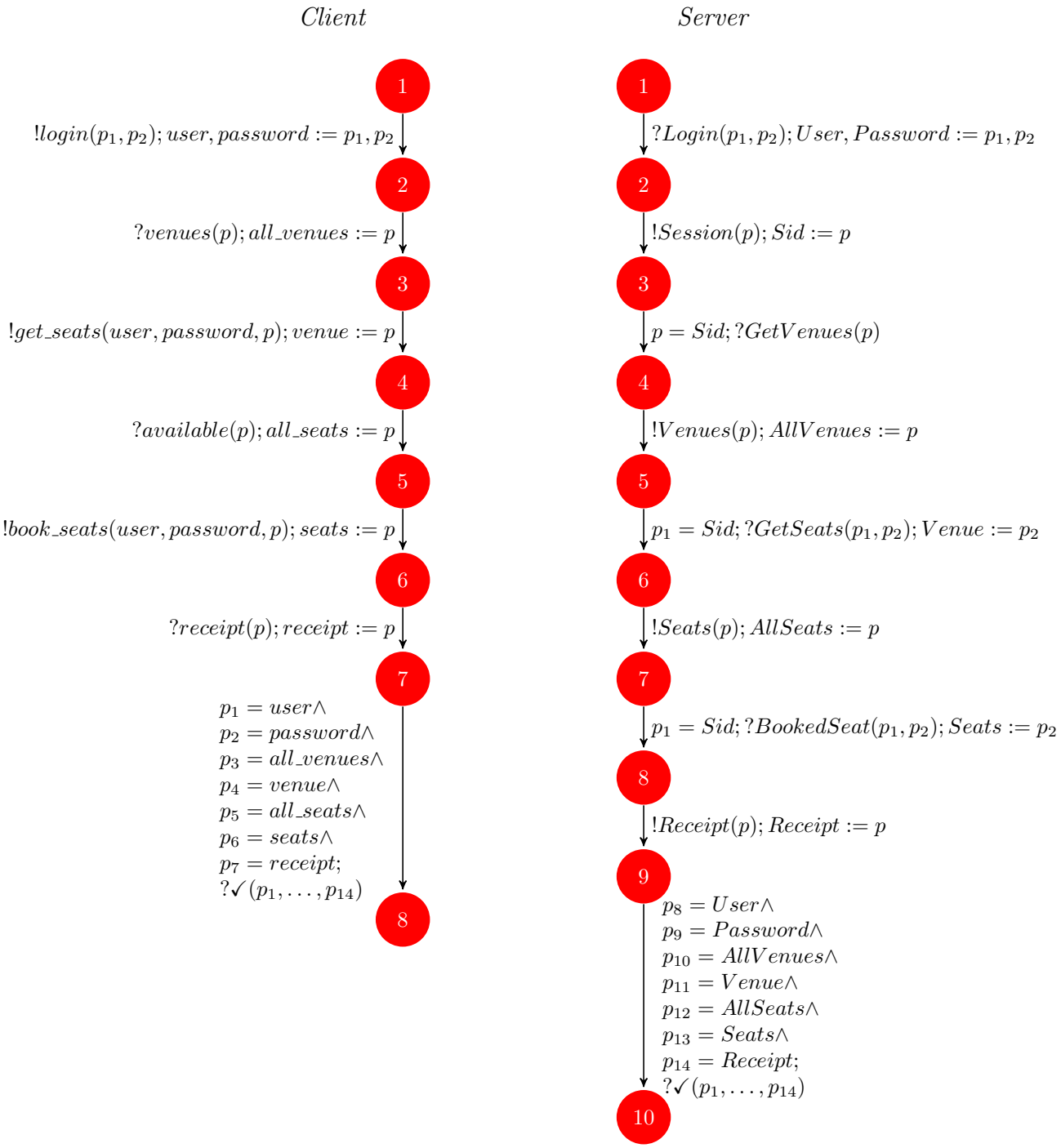

Figure 4.7: Incompatible client *Client* and server *Server* of an application for booking seats.

.

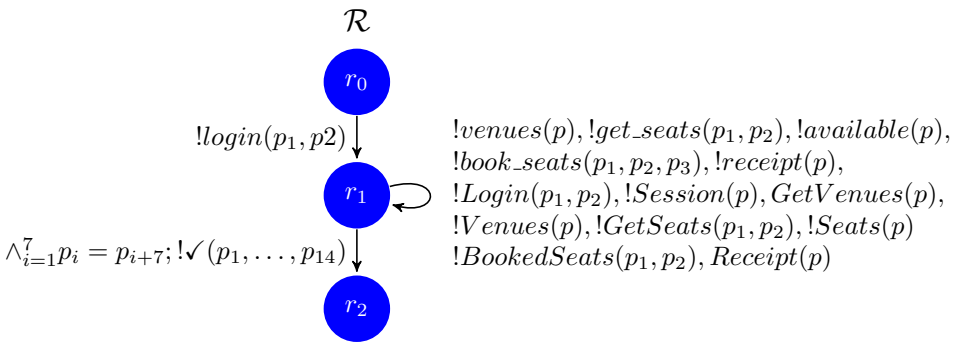

Figure 4.8: Specification of the synthesis problem with booking seats in the form of the process $\mathcal{R}$.

As future work, it remains to implement and evaluate the most important mechanisms of the theory, as well as to extend the framework for solving of quotients to more general specifications. The extension can be done along two dimensions. One is to handle nondeterministic processes, which brings the problem of uncertainty – the connector cannot be completely sure about the state of the controlled system. Another dimension is to extend the speficiation framework to cover liveness properties. Such extensions have been considered for the finite-state case in the literature (see, e.g., [17]), but their adaption to handle data appears nontrivial.

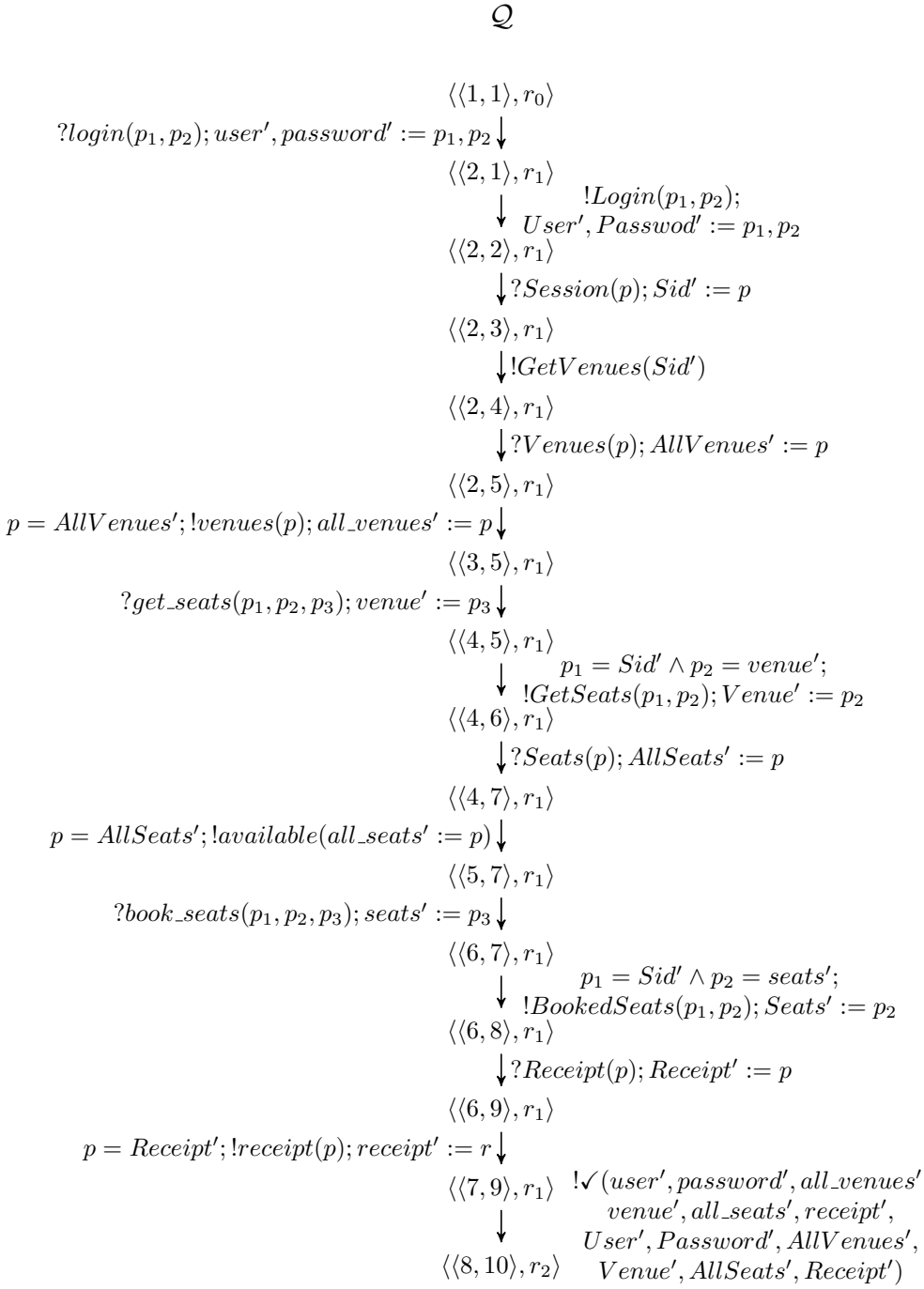

Figure 4.9: Synthesized mediator $\mathcal{Q}$ for the application with booking seats.

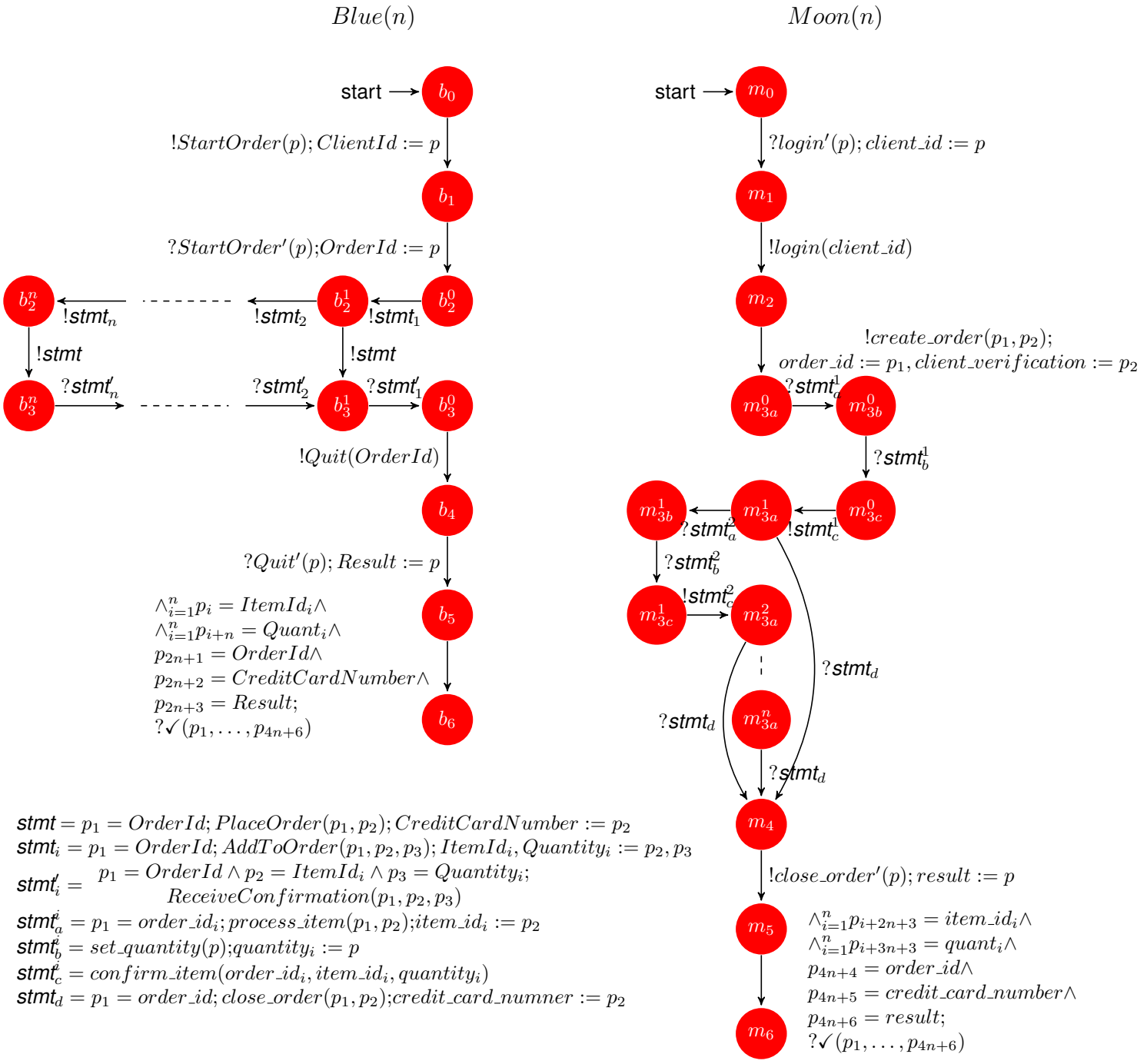

Figure 4.10: Client $Blue$ and customer service $Moon$ in an e-commerce service, parameterized by the maximum number of ordered items n.

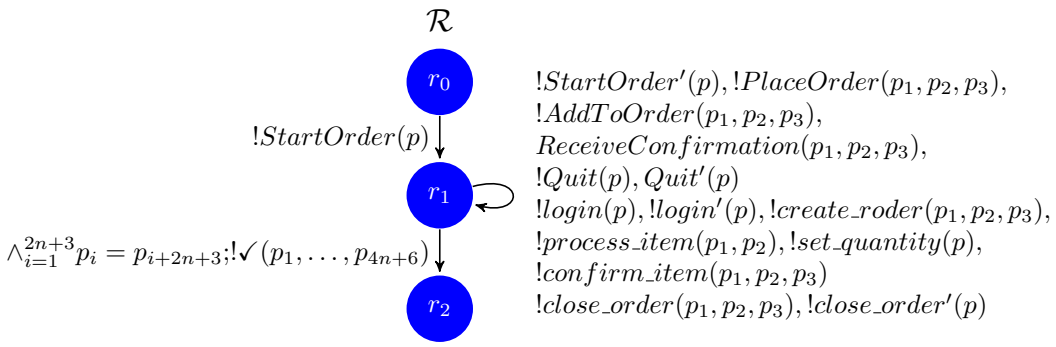

Figure 4.11: Specification of the synthesis problem where the number of ordered items is bounded by n is given by the process $\mathcal{R}(n)$.

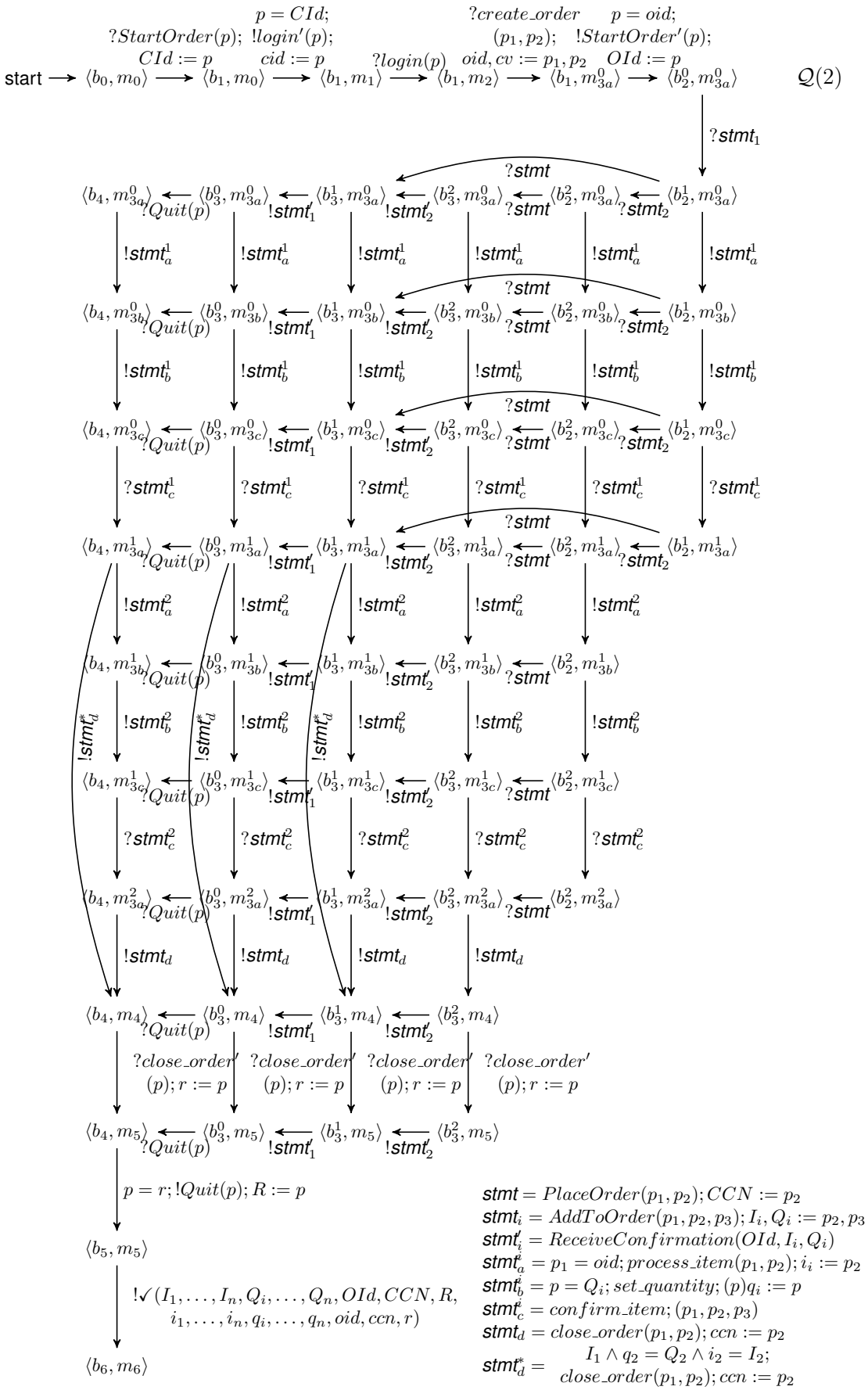

Figure 4.12: Resulting mediator $\mathcal{Q}$ for the e-commerce service with $n = 2$.

5 Progress on quantitative assume-guarantee reasoning

As mentioned in the Introduction, the work of this work package is organised into two streams: (i) compositional theory of connector behaviours, and (ii) compositional assume-guarantee verification for probabilistic automata, with the view to integrate these together. This chapter provides an overview of the progress in the work stream (ii).

Probabilistic verification techniques have been applied to the formal modelling and analysis of a wide range of systems, from communication protocols such as Bluetooth, to nanoscale computing devices, to biological cellular processes. In order to tackle the inherent challenge of scalability and independent development, compositional approaches to verification are sorely needed. An example is assume-guarantee reasoning, where each component of a system is analysed independently, using assumptions about the other components that it interacts with. In the previous deliverables D2.1 [21] and D2.2 [22], we have already proposed an assume-guarantee framework for probabilistic automata and its extensions. In D2.1, assumptions are *probabilistic safety properties* (e.g. "event A always occurs before event B with probability at least 0.98") and D2.2 generalises this to boolean combinations of ω-regular and reward properties. In both cases, this yields efficiently checkable assumptions and the approaches were successfully implemented and applied to some large case studies, including CONNECT-relevant examples.

As mentioned in Section 2.4, we are in the process of formulating an assume-guarantee framework for the specification theory presented in Chapter 2. We shall also develop a quantitative extension of the specification theory based on discrete-probabilities. Once this has been completed, we shall marry the quantitative assume-guarantee techniques presented in this chapter with the those we are working on in the non-quantitative case, and apply them to the probabilistic specification theory. This will be a key priority of WP2 in the remaining period of the project.

In this chapter, we present recent developments in work stream (ii) concerning compositional assume-guarantee verification techniques for probabilistic automata. In particular, we describe in Section 5.1 extensions to techniques for automatically generating probabilistic assumptions that can be used as the basis for compositional reasoning. In Section 5.2, our focus is on using a more expressive class of assumptions. We present novel techniques [37] for compositional probabilistic verification of synchronous systems in which assumptions are *probabilistic finite automata* (PFAs) [62]. Probabilistic safety property assumptions in D2.1 can only capture a limited amount of information about a component, restricting the cases where assume-guarantee reasoning can be applied. Therefore, the framework in D2.1 is *incomplete* in the sense that, if the property being verified is true, there does not necessarily exist an assumption that can be used to verify it compositionally. In contrast, our new approach *is* complete. Furthermore, as in D2.2, we use a novel L*-inspired learning algorithm to automatically generate assumptions. The assume-guarantee method has been implemented and applied to a number of examples, including communication protocols of relevance to CONNECT, but it relies on semi-algorithms, with no guarantee of termination, and its performance is not competitive compared to the method reported in D2.2; see [37] for more detail.

We emphasise that the use of learning in WP2 differs from that in WP4; we are using learning to fully automate compositional reasoning, by learning assumptions that suffice to derive the required conclusion.

5.1 Extensions to the learning approach

5.1.1 Preliminaries

In the deliverable D2.1 [21], we reported a probabilistic assume-guarantee framework, in which probabilistic safety properties can be verified in a compositional way. An assume-guarantee triple now takes the form $\langle A \rangle_{\geq p_A} M_i \langle G \rangle_{\geq p_G}$ where M_i is a PA and $\langle A \rangle_{\geq p_A}, \langle G \rangle_{\geq p_G}$ are two probabilistic safety properties; the former is a *probabilistic assumption*, the latter a property to be checked. This is interpreted as follows: $\langle A \rangle_{\geq p_A} M_i \langle G \rangle_{\geq p_G}$ is true if, for all adversaries σ of M_i such that $Pr^{\sigma}_{M_i}(A) \geq p_A$ holds, $Pr^{\sigma}_{M_i}(G) \geq p_G$ also holds. Crucially, verifying whether this is true reduces to a multi-objective model checking problem [35, 21], which can be carried out efficiently by solving an LP problem. The proof rule used earlier now

becomes:

$$\frac{M_1 \models \langle A \rangle_{\geq p_A} \qquad \langle A \rangle_{\geq p_A}\, M_2\, \langle G \rangle_{\geq p_G}}{M_1 \parallel M_2 \models \langle G \rangle_{\geq p_G}} \qquad \textit{(ASYM)}$$

thus reducing the problem of checking property $\langle G \rangle_{\geq p_G}$ on $M_1 \parallel M_2$ to two smaller sub-problems: (i) verifying a probabilistic safety property on M_1; and (ii) checking an assume-guarantee triple for M_2.

In [21], this proof rule, along with several others, are proved and then used to perform compositional verification on a set of large case studies. This includes cases where non-compositional verification is either slower or infeasible. In the deliverable D2.2 [22], this framework is extended to permit the use of more expressive assumptions (and properties): *quantitative multi-objective properties*, essentially Boolean combinations of probabilistic safety, ω-regular and expected total reward properties. An example of an expressible assumption is "with probability 1, component M_i always eventually sends a message and the expected time to do so is at most 10 time-units." Again, the framework is implemented through multi-objective model checking and applied to several case studies.

In this section, we focus on the use of probabilistic safety properties, as in [21]. Although these have limited expressivity, an advantage is that the generation of such assumptions can be automated by adapting the L*-based techniques developed for non-probabilistic compositional verification [19, 60].

The approach was reported in D2.2 [22] and has been applied to several large case studies. In D2.2, we consider the assume-guarantee proof rule (ASYM), which is used to check that the parallel composition of two PAs M_1 and M_2 satisfies a property $\langle G \rangle_{\geq p_G}$. To do this, we need an assumption $\langle A \rangle_{\geq p_A}$, which will be generated automatically.

The first key point to make is that, although we are required to learn a *probabilistic assumption*, i.e. a probabilistic safety property $\langle A \rangle_{\geq p_A}$, we can essentially reduce this task to the problem of learning a *non-probabilistic* assumption, i.e. the corresponding safety property A. The reasoning behind this is as follows.

We need the probabilistic assumption $\langle A \rangle_{\geq p_A}$ to be such that both premises of the proof rule (ASYM) hold: (i) $M_1 \models \langle A \rangle_{\geq p_A}$; and (ii) $\langle A \rangle_{\geq p_A}\, M_2\, \langle G \rangle_{\geq p_G}$. However, if (i) holds for a particular value of p_A, then it also holds for any lower value of p_A. Conversely, if (ii) holds for some p_A, it then must hold for any higher value. Thus, given a safety property A, we can determine an appropriate probability bound p_A (if one exists) by finding the lowest value of p_A (if any) such that (ii) holds and checking it against (i). Alternatively, we can find the highest value of p_A such that (i) holds (this is just $Pr^{\min}_{M_1}(A)$ in fact) and seeing if this suffices for (ii). A benefit of the latter is that, even if $M_1 \parallel M_2 \models \langle G \rangle_{\geq p_G}$ cannot be shown to be true with this particular assumption, we still obtain a *lower bound* on $Pr^{\min}_{M_1 \parallel M_2}(G)$. Furthermore, with an additional simple check, an *upper bound* can also be generated.

In D2.2, we described an approach for adapting the L*-based approach [19, 60], which generates non-probabilistic assumptions, to our setting. The underlying idea beind the use of L* in [19, 60] is the notion of *weakest assumption*: this will always exist and, if the property G being verified is true, will permit a compositional verification. It is used as the target language for L* and forms the basis of the membership and equivalence queries. In practice, however, this language is often not the one that is finally generated since intermediate conjectured assumptions may suffice, either to show that the property is true or that it is false.

An important difference in the probabilistic assume-guarantee framework of [21] is that it is *incomplete*, meaning that, even if the property $\langle G \rangle_{\geq p_G}$ holds, there may be no probabilistic assumption $\langle A \rangle_{\geq p_A}$ for which the rule (ASYM) can be used to prove the property correct. So, there can be no equivalent notion of weakest assumption to be used as a target language. However, we can adopt a similar approach whereby we use L* to generate a sequence of conjectured assumptions A and, for each one, potentially show that $\langle G \rangle_{\geq p_G}$ is either true or false. Furthermore, as described above, each assumption A yields a lower and an upper bound on $Pr^{\min}_{M_1 \parallel M_2}(G)$. We can retain these values as the learning algorithm progresses, keeping the highest lower bound and lowest upper bound discovered to report back to the user.[1] This means that, even if the algorithm terminates early (without concluding that $\langle G \rangle_{\geq p_G}$ is true or false), it produces useful *quantitative* information about the property of interest.

[1] Note that the sequence of assumptions generated is *not* monotonic, e.g. it does *not* yield a sequence of increasing lower bounds on $Pr^{\min}_{M_1 \parallel M_2}(G)$.

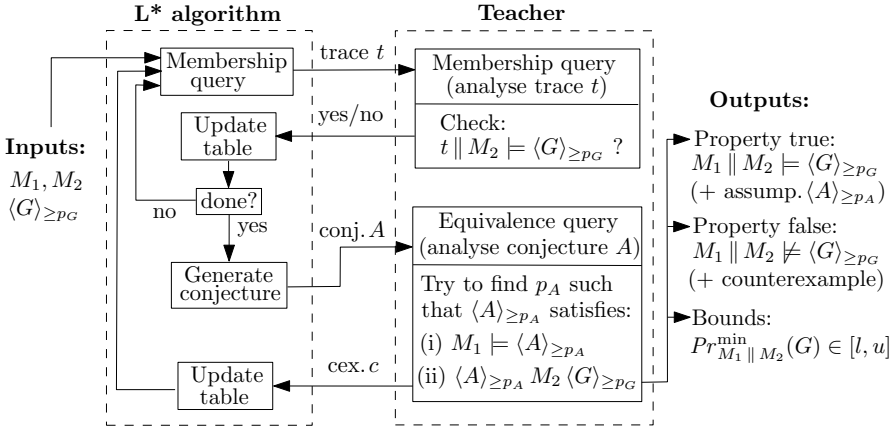

Figure 5.1: Overview of probabilistic assumption generation [39], using an adaption of L*: generates assumption $\langle A \rangle_{\geq p_A}$ for verification of $M_1 \parallel M_2 \models \langle G \rangle_{\geq p_G}$ using rule (ASYM).

In order to produce conjectures, L* needs answers to membership queries about whether certain traces t should be in the language being generated. For this, we use the following check: $t \parallel M_2 \models \langle G \rangle_{\geq p_G}$, which can be seen as an analogue of the corresponding one for the non-probabilistic case. Intuitively, the idea is that if, under a single possible behaviour t of M_1, M_2 satisfies the property, then t should be included in the assumption A. There may be situations where this scheme leads to an assumption that cannot be used to verify the property. This is because it is possible that there are multiple traces which do not violate property $\langle G \rangle_{\geq p_G}$ individually but, when combined, cause $\langle G \rangle_{\geq p_G}$ to be false. In practice, though, this approach seems to work well in most cases.

A final aspect of L* that needs discussion is *counterexamples*. In [19, 60], when the response to an equivalence query is "no", a counterexample in the form of a trace is returned to L*. In our case, there are two differences in this respect. Firstly, a counterexample may constitute multiple traces; this is because the results of the model checking queries executed by the teacher yield *probabilistic counterexamples* [44], which comprise multiple paths (again, see [39] for precise details). Secondly, situations may arise where no such counterexample can be generated (recall that there is no guarantee that an assumption can eventually be created). In this case, since no trace can be returned to L*, we terminate the learning algorithm, returning the current tightest bounds on $Pr^{\min}_{M_1 \| M_2}(G)$ that have been computed so far.

We summarise in Figure 5.1 the overall structure of the L*-based algorithm for generating probabilistic assumptions. The left-hand side shows the basic L* algorithm. This interacts, through queries, with the teacher, shown on the right-hand side. The teacher responds to queries as described above. Notice that the equivalence query, which analyses a particular conjectured assumption A has four possible outcomes: the first two are when A can be used to show either that $M_1 \parallel M_2 \models \langle G \rangle_{\geq p_G}$ or $M_1 \parallel M_2 \not\models \langle G \rangle_{\geq p_G}$; the third is when a counterexample (comprising one or more traces) is passed back to L*; and the fourth is when no counterexample can be generated so the algorithm returns the tightest bounds computed so far for $Pr^{\min}_{M_1 \| M_2}(G)$.

The above approach was successfully used in [39] to automatically generate probabilistic assumptions for several large case studies. Furthermore, these assumptions were much smaller than the corresponding components that they represented, leading to gains in performance. In this section, we present some recent extensions to that work.

5.1.2 An Alternative Learning Method: NL*

We investigate the use of an alternative learning algorithm to generate probabilistic assumptions. Whereas L* learns a minimal DFA for a regular language, the algorithm NL* [12] learns a minimal *residual finite-state automaton* (RFSA). RFSAs are a subclass of nondeterministic finite automata. For the same regular language $\mathcal{L}$, the minimal RFSA that accepts $\mathcal{L}$ can be exponentially more succinct than the corresponding minimal DFA. In fact, for the purposes of probabilistic model checking, the RFSA needs to be determinised anyway [27]. However, the hope is that the smaller size of the RFSA may lead to a faster

learning procedure. NL* works in a similar fashion to L*, making it straightforward to substitute into the learning loop shown in Figure 5.1.

According to the experimental results in [40], the NL* algorithms successfully generated a correct (and small) assumption in all cases with the exception of one model. The results shows that the L*-based method is faster than NL* in most cases. However, on several of the larger models, NL* has better performance due to a smaller number of equivalence queries. NL* needs more membership queries, but these are less costly.

5.1.3 Learning Multiple Assumptions: Rule (ASYM-N)

In this section, we consider another extension of the probabilistic assumption generation scheme of Section 5.1.1, adapting it to the proof rule (ASYM-N) of [21]. This is motivated by the observation that, for several of the case studies in the previous section, scalability is limited because one of the two components comprises several sub-components. As the number of sub-components increases, model checking becomes infeasible due to the size of the state space. The (ASYM-N) proof rule allows decomposition of the system into more than 2 components:

$$\frac{\begin{array}{c}\langle \mathit{true}\rangle\ M_1\ \langle A_1\rangle_{\geq p_1} \\ \langle A_1\rangle_{\geq p_1}\ M_2\ \langle A_2\rangle_{\geq p_2} \\ \ldots \\ \langle A_{n-1}\rangle_{\geq p_{n-1}}\ M_n\ \langle G\rangle_{\geq p_G}\end{array}}{\langle \mathit{true}\rangle\ M_1 \parallel \ldots \parallel M_n\ \langle G\rangle_{\geq p_G}} \quad \text{(ASYM-N)}$$

Adapting our probabilistic assumption generation process to (ASYM-N) works by learning assumptions for the rule in a recursive fashion, with each step requiring a separate instantiation of the learning algorithm for (ASYM), as is done for the non-probabilistic version of a similar rule in [60]. The experimental results shown in [40] demonstrate that, using the rule (ASYM-N), we can successfully learn small assumptions and perform compositional verification in several cases where the rule (ASYM) runs out of memory. Furthermore, in two instances, (ASYM-N) permits verification of models which cannot be checked in a non-compositional fashion.

5.2 Learning-based Compositional Verification for Synchronous Probabilistic Systems

In this section, we present novel techniques for automated compositional verification of synchronous probabilistic systems. First, we give an assume-guarantee framework for verifying probabilistic safety properties of systems modelled as discrete-time Markov chains (DTMC). Assumptions about system components are represented as probabilistic finite automata (PFAs) and the relationship between components and assumptions is captured by weak language inclusion. More concretely, PFAs represent weighted languages, mapping finite words to probabilities. An assumption about a system component M is represented by a PFA that gives upper bounds on the probabilities of traces being observed in M. This is an inherently *linear-time* relation, which is well-known to be difficult to adapt to compositional techniques for systems that exhibit both probabilistic and nondeterministic behaviour [67]. So, in the present work, we restrict our attention to *fully probabilistic* systems. To do so, we model components as *probabilistic I/O systems* (PIOSs), which, when combined through *synchronous* parallel composition, result in a (fully probabilistic) discrete-time Markov chain. The relation between a PIOS M and a PFA A representing an assumption about M is captured by *weak language inclusion*. Based on this, we give an asymmetric proof rule for verifying probabilistic safety properties on a DTMC composed of two PIOSs.

In order to implement our approach, we give an algorithm to check weak language inclusion, reducing it to the existing notion of (strong) language inclusion for PFAs. Although checking PFA language *equivalence* (that each word maps to the same probability) is decidable in polynomial time [70, 33], checking language *inclusion* is undecidable [11]. We propose a semi-algorithm, inspired by [70], to check language inclusion; in the case where the check fails, a minimal counterexample is produced. We also develop a

novel technique for *learning* PFAs, which we use to automatically generate assumptions. Our algorithm, like L*, is based on *active learning*, posing queries in an interactive fashion about the PFA to be generated.

In the rest of the section, we first briefly describe *probabilistic finite automata* and *discrete-time Markov chains* in Section 5.2.1, and define an assume-guarantee framework for synchronous probabilistic systems in Section 5.2.2. Next, we develop a semi-algorithm to check language inclusion for PFAs in Section 5.2.3 and a learning algorithm, similar to the L* algorithm, for constructing PFAs as assumptions in Section 5.2.4. In Section 5.2.5, we integrate the techniques in Sections 5.2.3 and 5.2.4 into the framework described in Section 5.2.2.

5.2.1 Preliminaries

We use $SDist(S)$ to denote the set of probability *sub-distributions* over set S, η_s for the point distribution on $s \in S$, and $\mu_1 \times \mu_2$ for the product distribution of μ_1 and μ_2.

Definition 5.1 (PFA) *A* probabilistic finite automaton *(PFA) is a tuple $A = (S, \overline{s}, \alpha, \mathcal{P})$, where S is a finite set of states, $\overline{s} \in S$ is an initial state, α is an alphabet and $\mathcal{P} : \alpha \to (S \times S \to [0,1])$ is a function mapping actions to transition probability matrices. For each $a \in \alpha$ and $s \in S$, $\sum_{s' \in S} \mathcal{P}(a)[s, s'] \in [0,1]$.*

A PFA A defines a mapping $Pr^A : \alpha^* \to [0,1]$ giving the probability of accepting each finite word $w \in \alpha^*$. Intuitively, the probability $Pr^A(w)$ for a word $w = a_1 \cdots a_n$ is determined by tracing paths through A that correspond to w, with $\mathcal{P}(a)[s, s']$ giving the probability to move from s to s' on reading a. More precisely, we let $\vec{\iota}$ be an S-indexed 0-1 row vector with $\vec{\iota}[s] = 1$ if and only if $s = \overline{s}$, $\vec{\kappa}$ be an S-indexed column vector of 1s and $\mathcal{P}(w) = \mathcal{P}(a_1) \cdots \mathcal{P}(a_n)$. Then, we define $Pr^A(w) = \vec{\iota}\mathcal{P}(w)\vec{\kappa}$.

Definition 5.2 (Language inclusion/equivalence) *Given two PFAs A_1 and A_2 with the same alphabet α, we say A_1 and A_2 are related by (strong)* language inclusion *(resp.* language equivalence*), denoted $A_1 \sqsubseteq A_2$ (resp. $A_1 \equiv A_2$), if for every word $w \in \alpha^*$, $Pr^{A_1}(w) \leqslant Pr^{A_2}(w)$ (resp. $Pr^{A_1}(w) = Pr^{A_2}(w)$).*

Definition 5.3 (DTMC) *A* discrete-time Markov chain *(DTMC) is a tuple $D = (S, \overline{s}, \alpha, \delta)$, where S is a finite set of states, $\overline{s} \in S$ is an initial state, α is an alphabet of* action labels *and $\delta : S \times (\alpha \cup \{\tau\}) \to SDist(S)$ is a (partial)* probabilistic transition function*, such that, for any s, $\delta(s, a)$ is defined for at most one $a \in \alpha \cup \{\tau\}$.*

If $\delta(s, a) = \mu$, the DTMC can make a transition, labelled with action a, and move to state s' with probability $\mu(s')$. We denote such transitions by $s \xrightarrow{a} \mu$ (or $s \xrightarrow{a} s'$). The DTMC deadlocks when $\delta(s, a)$ is not defined for any a, which we denote by $s \not\to$. We use action label τ to denote a "silent" (or "internal") transition. A (finite or infinite) path through D is a sequence of transitions $\theta = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \cdots$ with $s_0 = \overline{s}$.

In this deliverable, we consider *probabilistic safety properties* $\langle G \rangle_{\geqslant p}$, where G is a *regular safety property* [5], defining a set of "good" executions, and $p \in [0,1]$ is a probability bound. Model checking $\langle G \rangle_{\geqslant p}$ reduces to solving a linear equation system [5].

5.2.2 Assume-Guarantee for Synchronous Probabilistic Systems

We now define a compositional verification framework for fully probabilistic systems. Components are modelled by *probabilistic I/O systems* (PIOSs). These exhibit (input) nondeterminism but, when composed *synchronously* in parallel, result in a DTMC.

Definition 5.4 (PIOS) *A* probabilistic I/O system *(PIOS) is a tuple $M = (S, \overline{s}, \alpha, \delta)$, where S and $\overline{s}$ are as for DTMCs, and the alphabet α and transition function $\delta : S \times (\alpha \cup \{\tau\}) \to SDist(S)$ satisfy the following two conditions: (i) α is partitioned into three disjoint sets of* input, output *and* hidden *actions, which we denote α^I, α^O and α^H, respectively; input actions α^I are further partitioned into m disjoint* bundles *$\alpha^{I,i}$ ($1 \leqslant i \leqslant m$) for some m; (ii) the set $enab(s) \subseteq \alpha \cup \{\tau\}$ of* enabled *actions for each state s (i.e. the actions a for which $\delta(s, a)$ is defined) satisfies either $|enab(s)| = 1$ if $enab(s) \subseteq \alpha^O \cup \alpha^H \cup \{\tau\}$ or $enab(s) = \alpha^{I,i}$ for some input action bundle $\alpha^{I,i}$.*

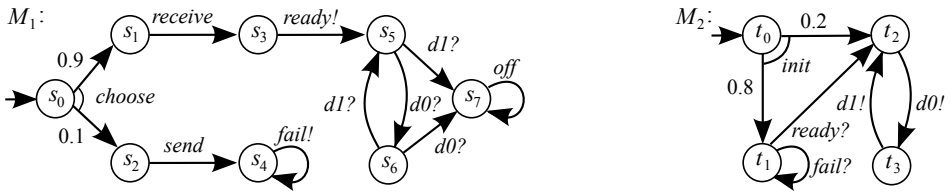

Figure 5.2: Running example: two PIOSs M_1 and M_2.

From any state s of a PIOS M, there is either a single transition with an output, hidden or τ action, or k transitions, each with one action from a particular bundle $\alpha^{I,i}$ comprising k input actions. Transitions and paths in PIOSs are defined as for DTMCs. The probability of a finite path $\theta = s_0 \xrightarrow{a_0} s_1 \cdots \xrightarrow{a_{n-1}} s_n$ in M is given by $Pr^M(\theta) = \prod_{i=0}^{n-1} \delta(s_i, a_i)(s_{i+1})$. Since PIOSs only have nondeterminism on input actions, the probability for a word $w \in (\alpha \cup \{\tau\})^*$ is well defined: letting $wd(\theta)$ denote the word $a_0 \dots a_{n-1}$ of actions from path θ, we have $Pr^M(w) = \sum_{wd(\theta)=w} Pr^M(\theta)$. Then, letting $st : (\alpha \cup \{\tau\})^*\alpha \to \alpha^*$ be the function that removes all τs, we define the probability $Pr^M_\tau(w')$ for a τ-free word $w' \in \alpha^*$ as $Pr^M_\tau(w) = \sum_{w=st(w')} Pr^M(w')$.

Example 1. Figure 5.2 depicts two PIOSs M_1 and M_2. M_1 is a data communicator which chooses (probabilistically) to either send or receive data. This simple example only models receiving; choosing to send results in a failure. M_1 tells M_2, a data generator, that it is ready to receive using action $ready$. M_2 should then send a sequence of packets, modelled by the alternating actions $d0$ and $d1$. If M_1 has failed, it sends a message $fail$. M_2 also has an initialisation step ($init$), which can fail. With probability 0.8, it is ready to receive signals from M_1; otherwise, it just tries to send packets anyway. Input/output actions for M_1, M_2 are labelled with ?/! in the figure; all other actions are hidden. Each PIOS has a single input action bundle: $\alpha_1^{I,1} = \{d0, d1\}$, $\alpha_2^{I,1} = \{ready, fail\}$.

Given PIOSs M_1, M_2 with alphabets α_1, α_2, we say M_1 and M_2 are *composable* if $\alpha_1^I{=}\alpha_2^O$, $\alpha_1^O{=}\alpha_2^I$ and $\alpha_1^H \cap \alpha_2^H{=}\emptyset$ and define their parallel composition as follows.

Definition 5.5 (Parallel composition) *The* parallel composition *of composable PIOSs* $M_i = (S_i, \overline{s}_i, \alpha_i, \delta_i)$ *for* $i{=}1,2$ *is given by the PIOS* $M_1||M_2 = (S_1{\times}S_2, (\overline{s}_1, \overline{s}_2), \alpha, \delta)$*, where* $\alpha = \alpha^H = \alpha_1^I \cup \alpha_1^O \cup \big((\alpha_1^H \cup \{\bot\}) * (\alpha_2^H \cup \{\bot\})\big)$ *and, for* $b_i \in \alpha_i^H \cup \tau$ *and* $a \in \alpha_1^I \cup \alpha_1^O$*,* δ *is defined such that* $(s_1, s_2) \xrightarrow{\gamma} \mu_1{\times}\mu_2$ *iff one of the following holds: (i)* $s_1 \xrightarrow{a} \mu_1$*,* $s_2 \xrightarrow{a} \mu_2$*,* $\gamma = a$*; (ii)* $s_1 \xrightarrow{b_1} \mu_1$*,* $s_2 \xrightarrow{b_2} \mu_2$*,* $\gamma = b_1{*}b_2$*; (iii)* $s_1 \xrightarrow{b_1} \mu_1$*,* $s_2 \xrightarrow{a}$ *(or* $s_2 \not\to$*),* $\mu_2 = \eta_{s_2}$*,* $\gamma = b_1 * \bot$*; (iv)* $s_1 \xrightarrow{a}$ *(or* $s_1 \not\to$*),* $s_2 \xrightarrow{b_2} \mu_2$*,* $\mu_1 = \eta_{s_1}$*,* $\gamma = \bot * b_2$*.*

Notice PIOS $M_1 \| M_2$ has only τ or hidden actions and can thus be considered a DTMC.

We next introduce our notion of *assumptions* about PIOSs, for which we use a specific class of PFAs and *weak language inclusion*, which relaxes the definition of language inclusion for PFAs introduced earlier by ignoring τ actions.

Definition 5.6 (Assumption) *Let* M *be a PIOS with alphabet* $\alpha = \alpha^I \uplus \alpha^O \uplus \alpha^H$ *and input action bundles* $\alpha^I = \biguplus_{i=1}^m \alpha^{I,i}$*. An* assumption A *about* M *is a PFA* $A = (S, \overline{s}, \alpha, \mathcal{P})$ *satisfying, for each state* $s \in S$*: (i) either all or none of the actions in a bundle* $\alpha^{I,i}$ *(*$1 \leqslant i \leqslant m$*) are enabled in* s*; (ii)* $p^{\max}(s) \in [0,1]$*, where:*

$$p^{\max}(s) \stackrel{\text{def}}{=} \sum_{a \in \alpha^O \cup \alpha^H} \sum_{s' \in S} \mathcal{P}(a)[s, s'] + \sum_{i=1}^{m} p_i^{\max}(s) \;\text{ and }\; p_i^{\max}(s) \stackrel{\text{def}}{=} \max_{a \in \alpha^{I,i}} \sum_{s' \in S} \mathcal{P}(a)[s, s']$$

Definition 5.7 (Weak language inclusion/equivalence) *For PIOS* M *with alphabet* α *and an assumption* A *about* M*, we say that* M *and* A *are related by* weak language inclusion (resp. equivalence)*, denoted* $M \sqsubseteq_w A$ *(resp.* $M \equiv_w A$*), if for every word* $w \in \alpha^*$*,* $Pr^M_\tau(w) \leqslant Pr^A(w)$ *(resp.* $Pr^M_\tau(w) = Pr^A(w)$*).*

A *valid* assumption A for M is one that satisfies $M \sqsubseteq_w A$. We can reduce the problem of checking whether this is true to the problem of checking (strong) language inclusion between two PFAs (see Section 5.2.3) by the following proposition.

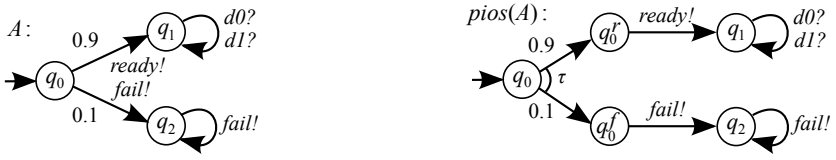

Figure 5.3: Assumption A and its PIOS conversion $pios(A)$.

Proposition 5.8 *Let $M = (S, \overline{s}, \alpha, \delta)$ be a PIOS and A be an assumption about M. $pfa(M) = (S, \overline{s}, \alpha \cup \{\tau\}, \mathcal{P})$ is the translation of M to a PFA, where $\mathcal{P}(a)[s, s'] = \delta(s, a)(s')$ for $a \in \alpha \cup \{\tau\}$. Letting A^τ be the PFA derived from A by adding τ to its alphabet and a probability 1 τ-loop to every state, then: $M \sqsubseteq_w A \Leftrightarrow pfa(M) \sqsubseteq A^\tau$.*

We will also need to perform a conversion in the opposite direction, translating an assumption PFA A into a (weak language) equivalent PIOS, which we denote $pios(A)$.

Definition 5.9 (Assumption-to-PIOS conversion) *Given assumption $A = (S, \overline{s}, \alpha, \mathcal{P})$, and action partition $\alpha = (\biguplus_{i=1}^{m} \alpha^{I,i}) \uplus \alpha^O \uplus \alpha^H$, its conversion to a PIOS is defined as $pios(A) = (S', \overline{s}, \alpha, \delta)$, where $S' = S \uplus \{s^a | s \in S, a \in \alpha^H \cup \alpha^O\} \uplus \{s^i | s \in S, 1 \leqslant i \leqslant m\}$ and δ is constructed as follows. For any transition $s \xrightarrow{a} s'$, let p denote $\mathcal{P}(a)[s, s']$ and $p^{\max}(s)$ and $p_i^{\max}(s)$ be as defined in Definition 5.6. Then:*

- *if $a \in \alpha^O \cup \alpha^H$, then $\delta(s, \tau)(s^a) = \frac{p}{p^{\max}(s)}$ and $\delta(s^a, a)(s') = p^{\max}(s)$;*
- *if $a \in \alpha^{I,i}$ (for $1 \leqslant i \leqslant m$), then $\delta(s, \tau)(s^i) = \frac{p_i^{\max}(s)}{p^{\max}(s)}$ and $\delta(s^i, a)(s') = p \cdot \frac{p^{\max}(s)}{p_i^{\max}(s)}$.*

Example 2. Consider PIOS M_1 from Example 1. Figure 5.3 shows a valid assumption A for M_1 (i.e. $M_1 \sqsubseteq_w A$) and the corresponding PIOS $pios(A)$. In A, state q_0 has two output actions leading to respective sub-distributions. Thus A is not a PIOS. In $pios(A)$, a τ transition and the states q_0^{ready} and q_0^{fail} (abbreviated to q_0^r and q_0^f) are added.

Now, we describe how to perform compositional verification using our framework. We focus on verifying $\langle G \rangle_{\geqslant p}$ on a DTMC $M_1 \| M_2$ where M_i are PIOSs. For simplicity, we will assume that the property refers only to input/output actions of M_1 and M_2 and assume that all hidden actions of M_1 and M_2 have been renamed as τ actions, which affects neither the parallel composition $M_1 \| M_2$ nor the probability of satisfying G.

An *assume-guarantee triple* $\langle A \rangle\ M\ \langle G \rangle_{\geqslant p}$ means "whenever component M is part of a system satisfying the assumption A, the system is guaranteed to satisfy $\langle G \rangle_{\geqslant p}$".

Definition 5.10 (Assume-guarantee triple) *If M is a PIOS with alphabet α, A is an assumption about M and $\langle G \rangle_{\geqslant p}$ is a probabilistic safety property, then $\langle A \rangle\ M\ \langle G \rangle_{\geqslant p}$ is an* assume-guarantee triple*, with the following meaning:*

$$\langle A \rangle\ M\ \langle G \rangle_{\geqslant p} \Leftrightarrow \forall M'.\ (M' \sqsubseteq_w A \implies M' \| M \models \langle G \rangle_{\geqslant p}).$$

Using the translation $pios(A)$ from PFA to PIOS described above, checking whether a triple is true reduces to standard probabilistic model checking (see Section 5.2.1).

Proposition 5.11 *For A, M and $\langle G \rangle_{\geqslant p}$ as given in Definition 5.10, the assume-guarantee triple $\langle A \rangle\ M\ \langle G \rangle_{\geqslant p}$ holds if and only if $pios(A) \| M \models \langle G \rangle_{\geqslant p}$.*

Finally, we give an asymmetric assume-guarantee proof rule (in the style of those from [60, 51]) for verifying a system $M_1 \| M_2$ compositionally.

Theorem 5.12 *Let M_1, M_2 be PIOSs, A an assumption for M_1 and $\langle G \rangle_{\geqslant p}$ a probabilistic safety property for $M_1 \| M_2$. Then the following proof rule holds:*

$$\frac{M_1 \sqsubseteq_w A \text{ and } \langle A \rangle\ M_2\ \langle G \rangle_{\geqslant p}}{M_1 \| M_2 \models \langle G \rangle_{\geqslant p}} \qquad \text{(ASYM-PIOS)}$$

Thus, given an appropriate assumption A about M_1, we can decompose the verification of $M_1 \| M_2$ into two sub-problems: checking weak language inclusion between M_1 and A; and checking that $\langle A \rangle\, M_2\, \langle G \rangle_{\geqslant p}$. The former, as shown in Proposition 5.8, reduces to (strong) language inclusion on PFAs, which we discuss in the next section. The latter, as shown in Proposition 5.11, requires construction of the DTMC $pios(A) \| M_2$ and then application of standard probabilistic model checking techniques.

Example 3. Consider probabilistic safety property $\langle G \rangle_{\geqslant 0.9}$, where G means "$fail$ never occurs". We can check this on running example $M_1 \| M_2$ using assumption A from Example 2. Since $M_1 \sqsubseteq_w A$, we just need to check that $pios(A) || M_2 \models \langle G \rangle_{\geqslant 0.9}$. As $pios(A) || M_2$ has a single path $(q_0 t_0) \xrightarrow{\tau * init, 0.08} (q_2 t_1) \xrightarrow{fail, 1} (q_4 t_1) \cdots$ containing $fail$ with probability 0.08, $\langle G \rangle_{\geqslant 0.9}$ is satisfied (since $1 - 0.08 \geqslant 0.9$) and we are done.

Completeness. Our framework is *complete* in the sense that, if $M_1 \| M_2 \models \langle G \rangle_{\geqslant p}$, we can always find an assumption A to apply Theorem 5.12 by converting M_1 to a PFA.

5.2.3 Deciding Language Inclusion for PFAs

As discussed above, verifying whether a component satisfies an assumption in our framework reduces to checking language inclusion between PFAs, i.e. deciding whether two PFAs A_1 and A_2 over the same alphabet α satisfy $A_1 \sqsubseteq A_2$. In this section, we propose a *semi*-algorithm for performing this check. If $A_1 \sqsubseteq A_2$ does *not* hold, then the algorithm is guaranteed to terminate and return a lexicographically minimal word as a counterexample; but if $A_1 \sqsubseteq A_2$ *does* hold, then the algorithm may not terminate. The latter case is unavoidable since the problem is undecidable (see [38]).

Require: PFAs A_1 and A_2 over the same alphabet α.
Ensure: **true** if $A_1 \sqsubseteq A_2$; or **false** and a cex $w' \in \alpha^*$.
1: *queue* $:= \{(\vec{\iota}_1, \vec{\iota}_2, \varepsilon)\}$, $V := \{(\vec{\iota}_1, \vec{\iota}_2, \varepsilon)\}$
2: **while** *queue* $\neq \emptyset$ **do**
3: remove $(\vec{v}_1, \vec{v}_2, w)$ from the head of *queue*
4: **for all** $a \in \alpha$ **do**
5: $\vec{v}_1' := \vec{v}_1 \mathcal{P}_1(a)$; $\vec{v}_2' := \vec{v}_2 \mathcal{P}_2(a)$; $w' := wa$
6: **if** $\vec{v}_1' \vec{\kappa}_1 > \vec{v}_2' \vec{\kappa}_2$ **then return false** and cex w'
7: **else if** $(\vec{v}_1', \vec{v}_2', w')$ does not satisfy (C1), (C2) **then**
8: add $(\vec{v}_1', \vec{v}_2', w')$ to the tail of *queue*
9: $V := V \cup \{(\vec{v}_1', \vec{v}_2', w')\}$
10: **return true**

Figure 5.4: Semi-algorithm for deciding PFA language inclusion

Figure 5.4 shows the semi-algorithm to decide if $A_1 \sqsubseteq A_2$, where $A_i = (S_i, \bar{s}_i, \alpha, \mathcal{P}_i)$ for $i = 1, 2$. We also define $\vec{\iota}_i$ and $\vec{\kappa}_i$ as in Section 5.2.1. Inspired by the language equivalence decision algorithm in [70], our method proceeds by expanding a tree. Each node of the tree is of the form $(\vec{v}_1, \vec{v}_2, w)$, where w is a word and $\vec{v}_i = \vec{\iota}_i \mathcal{P}_i(w)$ (for $i = 1, 2$) is the vector of probabilities of reaching each state via word w in A_i. Note that $\vec{v}_i \vec{\kappa}_i$ is the probability of PFA A_i accepting the word w. The root of the tree is $(\vec{\iota}_1, \vec{\iota}_2, \varepsilon)$, where ε is the empty word. As shown in Figure 5.4, we use a *queue* of tree nodes, which expands the tree in breadth-first order. In addition, we maintain a set V of non-leaf nodes, which initially only contains the root. The main difference between our method and [70] is that we adopt different criteria to decide when to add a node to the non-leaf set V. In [70], the set V is maintained by calculating the span of vector space. However, for the language inclusion check, we cannot simply use the same criteria.

In each iteration, we remove a node $(\vec{v}_1, \vec{v}_2, w)$ from the head of *queue*. We then expand the tree by appending a set of its child nodes $(\vec{v}_1', \vec{v}_2', w')$, where $\vec{v}_1' := \vec{v}_1 \mathcal{P}_1(a)$, $\vec{v}_2' := \vec{v}_2 \mathcal{P}_2(a)$ and $w' := wa$ for all actions $a \in \alpha$. If there is a node $(\vec{v}_1', \vec{v}_2', w')$ such that $Pr_1(w') = \vec{v}_1' \vec{\kappa}_1 > \vec{v}_2' \vec{\kappa}_2 = Pr_2(w')$, then the algorithm terminates and returns w' as a counterexample for $A_1 \sqsubseteq A_2$. Otherwise, we check if we can *prune* each child node $(\vec{v}_1', \vec{v}_2', w')$ (i.e. make it a leaf node) by seeing if it satisfies either of the following two criteria: (C1) $\vec{v}_1' \vec{\kappa}_1 = 0$; (C2) There exist $|V|$ non-negative rational numbers ρ^i such that, for all $(\vec{v}_1^i, \vec{v}_2^i, w^i) \in V$,

$\vec{v}'_1 \leq \sum_{0 \leq i < |V|} \rho^i \vec{v}^i_1$ and $\vec{v}'_2 \geq \sum_{0 \leq i < |V|} \rho^i \vec{v}^i_2$, where $\leq$ and $\geq$ denote pointwise comparisons between vectors.

Criterion (C1) is included because it is never possible to find a counterexample word with accepting probability less than $\vec{v}'_1 \vec{\kappa}_1 = 0$. Criterion (C2) is included because any node satisfying it would guarantee $\vec{v}'_1 \vec{\kappa}_1 \leq \vec{v}'_2 \vec{\kappa}_2$; moreover, if the algorithm terminates and a node satisfies (C2), all of its descendants also satisfy (C2). We can thus make it a leaf node. In practice, (C2) can easily be checked using an SMT solver. If a node cannot be pruned, we add it to the tail of *queue* and to the non-leaf set *V*. The algorithm terminates if *queue* becomes empty, concluding that $A_1 \sqsubseteq A_2$.

Correctness and termination. The correctness of the semi-algorithm in Figure 5.4 is shown formally in [38]. A guarantee of termination, on the other hand, cannot be expected due to the undecidability of the underlying problem.

5.2.4 L*-Style Learning for PFAs

In this section, we propose a novel method to learn a PFA for a target weighted language generated by an unknown PFA. It works in a similar style to the well-known L* algorithm [3] for learning regular languages: it constructs an *observation table* (of acceptance probabilities for each word) based on two types of *queries* posed to a *teacher*. *Membership queries* ask the probability of accepting a particular word in the target PFA; *equivalence queries* ask whether a hypothesised PFA yields exactly the target language.

Figure 5.5 shows the learning algorithm. It builds an observation table (P, E, T), where P is a finite, non-empty, prefix-closed set of words, E is a finite, non-empty, suffix-closed set of words and $T : ((P \cup P \cdot \alpha) \cdot E) \to [0, 1]$ maps each word to its accepting probability in the target language ($\cdot$ denotes concatenation over sets). The rows of table (P, E, T) are labelled by elements in the prefix set $P \cup P \cdot \alpha$ and the columns are labelled by elements in the suffix set E. The value $T(u \cdot e)$ of the entry at row u and column e is the acceptance probability of the word $u \cdot e$. We use $row(u)$ to represent the $|E|$-dimensional row vector in the table labelled by the prefix $u \in (P \cup P \cdot \alpha)$.

Inspired by [7], which gives an L*-style algorithm for learning multiplicity automata, we define the notions of *closed* and *consistent* observation tables by establishing linear dependencies between row vectors. Observation table (P, E, T) is *closed* if, for all $u \in P$ and $a \in \alpha$, there exist non-negative rational coefficients ϕ_i such that $row(u \cdot a) = \sum_{u_i \in P} \phi_i row(u_i)$ and *consistent* if, for any rational coefficients ψ_i, $\forall e \in E. \sum_{u_i \in P} \psi_i T(u_i \cdot e) = 0$ implies $\forall a \in \alpha, e \in E. \sum_{u_i \in P} \psi_i T(u_i \cdot a \cdot e) = 0$. The need for coefficients to be non-negative (for *closed*) is a stronger condition than in [7].

As shown in Figure 5.5, the observation table is filled with the results of membership queries until it is both closed and consistent. At each step, if (P, E, T) is not closed (resp. consistent), then the algorithm finds $u \in P, a \in \alpha$ (resp. $a \in \alpha, e \in E$) that make it not closed (resp. consistent), according to the definitions above, and adds $u.a$ (resp. $a.e$) to the table. When (P, E, T) is closed and consistent, the learning algorithm builds a hypothesis PFA A (see below) and poses an equivalence query. If the teacher answers "no" (that A does not yield the target language), a counterexample $c \in \alpha^*$ is given, for which $Pr^A(c)$ is incorrect. The algorithm adds c and all its prefixes to P, updates the observation table and continues to check if the table is closed and consistent. If the teacher answers "yes", the algorithm terminates and returns A.

Construction of a hypothesis PFA $A = (S, \overline{s}, \alpha, \mathcal{P})$, from a closed and consistent table (P, E, T), proceeds as follows. First, we find a subset of P, denoted $con(P)$, such that every element of $\{row(u) | u \in P\}$ can be represented as a *conical combination* of elements in $\{row(v) | v \in con(P)\}$, i.e. there are non-negative rational coefficients λ_i such that, for all $u \in P$, $row(u) = \sum_{v_i \in con(P)} \lambda_i row(v_i)$. The set of states in the PFA is then $S = \{s_0, \ldots, s_{n-1}\}$, where each state s_i corresponds to a row vector in $\{row(v) | v \in con(P)\}$ and the initial state $\overline{s}$ corresponds to $row(\varepsilon)$. To obtain $\mathcal{P}(a)$ for each $a \in \alpha$, we compute, for $s_i \in S$, rational coefficients γ_j such that $row(s_i \cdot a) = \sum_{s_j \in S} \gamma_j row(s_j)$ and then define $\mathcal{P}(a)[s_i, s_j] := \gamma_j \cdot (T(s_j \cdot \varepsilon) / T(s_i \cdot \varepsilon))$.

Correctness and termination. When the learning algorithm terminates, it returns a correct PFA, as guaranteed by the equivalence query check. Unfortunately, we cannot prove the termination of our method. For L*, the corresponding proof uses the existence of a unique minimal DFA for a regular language. However, an analogous property does not exist for weighted languages and PFAs. According to [7], the smallest multiplicity automaton *can* be learnt given a weighted language. However, as shown in [31], converting

Require: The alphabet α of a target weighted language generated by an unknown PFA.
Ensure: A PFA accepting the target language.
1: initialise the observation table (P, E, T), letting $P = E = \{\varepsilon\}$, where ε is the empty word
2: fill T by asking membership queries for ε and each action $a \in \alpha$
3: **while** (P, E, T) is not *closed* or not *consistent* **do**
4: **if** (P, E, T) is not *closed* **then** find $u \in P, a \in \alpha$ that make (P, E, T) not closed
5: add $u \cdot a$ to P, and extend T to $(P \cup P \cdot \alpha) \cdot E$ using membership queries
6: **if** (P, E, T) is not *consistent* **then** find $a \in \alpha, e \in E$ that make (P, E, T) not consistent
7: add $a \cdot e$ to E, and extend T to $(P \cup P \cdot \alpha) \cdot E$ using membership queries
8: construct a hypothesised PFA A and ask an equivalence query
9: **if** answer = no, with a counterexample c **then** add c and all its prefixes to P
10: extend T to $(P \cup P \cdot \alpha) \cdot E$ using membership queries, **goto** Line 4
11: **else return** PFA A

Figure 5.5: L*-style learning algorithm for PFAs

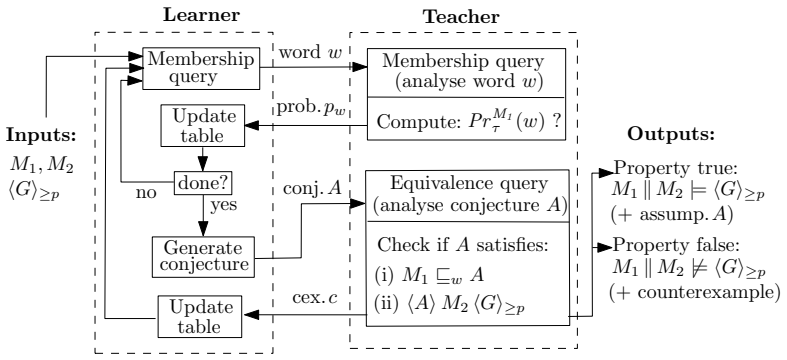

Figure 5.6: L*-style PFA learning loop for probabilistic assumption generation.

a multiplicity automaton to a PFA (even for the subclass that define stochastic languages) is not always possible.

5.2.5 Learning Assumptions for Compositional Verification

Finally, we build upon the techniques introduced in Sections 5.2.3 and 5.2.4 to produce a fully-automated implementation of the assume-guarantee framework proposed in Section 5.2.2. In particular, we use PFA learning to automatically generate assumptions to perform compositional verification. Figure 5.6 summarises the overall structure of our approach, which aims to verify (or refute) $M_1 \parallel M_2 \models \langle G \rangle_{\geqslant p}$ for two PIOSs M_1, M_2 and a probabilistic safety property $\langle G \rangle_{\geqslant p}$. This is done using proof rule (ASYM-PIOS) from Section 5.2.2, with the required assumption PFA A about component M_1 being generated through learning. The left-hand side of the figure shows the learning algorithm of Section 5.2.4, which drives the whole process; the right-hand side shows the teacher.

The teacher answers *membership queries* (about word w) by computing the probability $Pr_\tau^{M_1}(w)$ of word w in M_1. It answers *equivalence queries* (about conjectured PFA A) by checking if A satisfies both premises of rule (ASYM-PIOS): (i) $M_1 \sqsubseteq_w A$, and (ii) $\langle A \rangle\ M_2\ \langle G \rangle_{\geqslant p}$. The first is done using Proposition 5.8 and the algorithm in Section 5.2.3. The second is done using Proposition 5.11, which reduces to probabilistic model checking of the DTMC $pios(A) \parallel M_2$.

If both premises are true, we can conclude that $M_1 \parallel M_2 \models \langle G \rangle_{\geqslant p}$ holds. Otherwise, the teacher needs to provide a counterexample c for the learning algorithm to update the observation table and proceed. If premise (i) failed, then c is taken as the word showing the violation of (weak) language inclusion. If premise (ii) failed, we try to extract c from the results of model checking. We extract a *probabilistic counterexample* [44] C: a set of paths showing $pios(A)||M_2 \not\models \langle G \rangle_{\geqslant p}$. Following the same approach as [39], we transform C into a (small) fragment of M_1 (denoted M_1^C) and check whether $M_1^C||M_2 \not\models \langle G \rangle_{\geqslant p}$. If so,

we stop the learning loop, concluding that $M_1||M_2 \not\models \langle G \rangle_{\geqslant p}$. If, on the other hand, C is a *spurious counterexample*, we can always extract, from C a counterexample (word) c such that the learning algorithm can update its observation table. Full details can be found in [38].

From the arguments above, we can show that, when the learning loop terminates, it always yields a correct result. It should be pointed out, though, that since the loop is driven by the learning algorithm of Section 5.2.4, whose termination we cannot prove, we are also unable to guarantee that the loop finishes. Furthermore, weak language inclusion checks use the semi-algorithm of Section 5.2.3, which is not guaranteed to terminate.

5.3 Conclusions and future work

In the first section, we have described some recent improvements and extensions to the learning framework for probabilistic assume-guarantee verification. There are a variety of possible directions for future research in this area. One is to extend our techniques for learning probabilistic assumptions to the assume-guarantee framework in [22], which additionally includes ω-regular and expected reward properties. Here, the ω-regular language learning algorithms of [36, 16] may provide a useful starting point. There are also possibilities to enhance the underlying compositional verification framework. This includes developing efficient techniques to work with richer classes of probabilistic assumptions and extending the approach to handle more expressive types of probabilistic models, such as those that incorporate continuous-time behaviour.

In the second section, we have presented a novel assume-guarantee framework where multi-component systems are modelled by discrete-time Markov chains, components are probabilistic I/O systems and assumptions are probabilistic finite automata. Based on new techniques for checking PFA language inclusion and active learning of PFAs, we have built a fully-automated implementation of the framework that generates assumptions and then performs compositional verification. Unfortunately, the method relies on semi-algorithms, with no guarantee of termination, and its performance is not competitive compared to the method reported in D2.2; see [37] for more detail. Future work in this area will include investigating more alternative representations for assumptions, exploring ways to extend our techniques to models with nondeterminism and investigating termination conditions for the language inclusion check and PFA learning.

6 Conclusion and future work

A key contribution of WP2 in the third year is the formulation of a comprehensive specification theory for modelling components. This is a bedrock for the work package in that, come the end of the project, the reasoning frameworks introduced in WP2 should all be based on this specification theory. The specification theory itself introduces formal notions such as refinement for comparing components and determining when they can be substituted for one another. The operation of parallel composition allows users to examine the structure of composed component-based systems, while conjunction supports independent development, a necessary feature for supporting compositionality in CONNECT. The operation of quotient supports the principles of synthesis, and can be used for proving technical results about the properties of mediators synthesised *à la* WP3, such as deadlock and error freedom.

We have successfully demonstrated through Chapter 3 that the specification theory introduced is not an abstract notation disconnected from CONNECT, but can be used as a genuine modelling framework of the project. We have given a simple semantic preserving mapping from the extended LTSs of WP3 to the Logic IOLTSs of our specification theory. This allows the other work packages to exploit the compositionality results we have presented in Chapter 2, with respect to the refinement preorder.

To better support modelling across the project, we have introduced in Chapter 4 an extension of the specification theory with data, which should assist greatly with the modelling of components in both WPs 3 and 4. To broach the work of WP5, it is our intention to commence work on a quantitative specification theory based on discrete probabilistic behaviours, with reward structures for capturing non-functional properties. Such models will be easily adoptable by WP5 so that quantitative analysis techniques can be applied to examine the performance aspects of systems.

Besides a unified modelling approach across the project, we also intend to develop frameworks for reasoning compositionally about components, primarily in the form of assume-guarantee specifications. We have made a start on such a framework in the non-quantitative case, although due to its infancy we do not document it in this report. So far we have devised a reasoning framework for safety properties, and we have proposed a framework for liveness based on quiescence. Concerning the safety framework, we have also formulated AG rules for the operations of parallel composition and conjunction, and have produced the associated soundness proofs. Formulation of a rule for quotient, which will be necessary for showing strong integration and linking with WP3, is still in progress. We will resume work on this framework over the remainder of the project.

Work on compositional reasoning for probabilistic systems has already commenced independently (see Chapter 5), with formulation of a fully-automated assume-guarantee technique for DTMCs, including automatic generation of probabilistic assumptions represented as PFAs, a semi-algorithm for checking language inclusions on PFAs, and a new L*-style learning methods for PFAs. The learning framework for assumption generation in D2.2 has been extended to support the (ASYM-N) rule and included another learning algorithm. Over the course of the final year, it is our intention to link our compositional reasoning techniques to the probabilistic extension of the specification theory.

We also plan to investigate an approach that enhances the connector synthesis by considering non-functional aspects. The approach, by leveraging non-functional estimation and (multi-objective) optimization techniques, selects the best connector with respect to specified non-functional requirements for the system to be connected. The synthesized connector is selected among the possible ones that are correct w.r.t. functional aspects (i.e., a refinement of the most general one) but different w.r.t non-functional perspectives. In this case the approach estimates the non-functional aspects of the possible alternative connectors and then decides which to provide. In case more than one non-functional aspect is considered, a trade-off analysis (based on multi-objective function optimization) needs to be executed. The devised approach could also enhance our specification theory by defining a specific operator for this aim. More specifically, the proposed specification theory could be enhanced to allow the specification of non-functional characteristics of actions (inside a connector) and connectors as whole. Moreover, several compositional operators considering in the connector composition the non-functional aspects could be defined. By considering the approach for synthesis characterized in this deliverable, we envisage two different operator types, one able to calculate the non-functional characteristics of the possible (functional equivalent) connectors and another that is able to select a connector functionally equivalent to the most general one but that optimizes a specific (multi-objective) function.

Bibliography

[1] M. Abadi and L. Lamport. Composing specifications. *ACM Transactions on Programming Languages and Systems*, 15(1):73–132, January 1993.

[2] M. Abadi and L. Lamport. Conjoining specifications. *ACM Transactions on Programming Languages and Systems*, 17(3):507–534, May 1995.

[3] D. Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75(2):87–106, 1987.

[4] M. Autili, C. Chilton, P. Inverardi, M. Kwiatkowska, and M. Tivoli. Towards a connector algebra. In T. Margaria and B. Steffen, editors, *Leveraging Applications of Formal Methods, Verification, and Validation*, volume 6416 of *Lecture Notes in Computer Science*, pages 278–292. Springer, 2010.

[5] C. Baier and J.-P. Katoen. *Principles of Model Checking*. MIT Press, 2008.

[6] D. Berardi, D. Calvanese, G. De Giacomo, R. Hull, and M. Mecella. Automatic composition of transition-based semantic web services with messaging. In *Proceedings of the 31st international conference on Very large data bases*, VLDB '05. VLDB Endowment, 2005.

[7] F. Bergadano and S. Varricchio. Learning behaviors of automata from multiplicity and equivalence queries. *SIAM J. Comput.*, 25(6):1268–1280, 1996.

[8] P. Bertoli, M. Pistore, and P. Traverso. Automated composition of web services via planning in asynchronous domains. *Artificial Intelligence*, 174(34):316 – 361, 2010.

[9] P. Bhaduri and S. Ramesh. Interface synthesis and protocol conversion. *Form. Asp. Comput.*, 20(2):205–224, March 2008.

[10] G. S. Blair, A. Bennaceur, N. Georgantas, P. Grace, V. Issarny, V. Nundloll, and M. Paolucci. The role of ontologies in emergent middleware: Supporting interoperability in complex distributed systems. In F. Kon and A.-M. Kermarrec, editors, *Middleware*, volume 7049 of *Lecture Notes in Computer Science*, pages 410–430. Springer, 2011.

[11] V. Blondel and V. Canterini. Undecidable problems for probabilistic automata of fixed dimension. *Theory of Computing Systems*, 36:231–245, 2001.

[12] B. Bollig, P. Habermehl, C. Kern, and M. Leucker. Angluin-style learning of NFA. In *Proc. IJCAI'09*, pages 1004–1009. Morgan Kaufmann Publishers Inc., 2009.

[13] S. D. Brookes, C. A. R. Hoare, and A. W. Roscoe. A theory of communicating sequential processes. *J. ACM*, 31(3):560–599, 1984.

[14] K. L. Calvert and S. S. Lam. Deriving a protocol converter: a top-down method. *SIGCOMM Comput. Commun. Rev.*, 19:247–258, August 1989.

[15] S. Cassel, F. Howar, B. Jonsson, M. Merten, and B. Steffen. A succinct canonical register automaton model. In *ATVA*, volume 6996 of *Lecture Notes in Computer Science*, pages 366–380. Springer Verlag, 2011.

[16] S. Chaki and A. Gurfinkel. Automated assume-guarantee reasoning for omega-regular systems and specifications. In *Proc. NFM'10*, pages 57–66, 2010.

[17] K. Chatterjee, L. Doyen, T. A. Henzinger, and J.-F. Raskin. Algorithms for omega-regular games with imperfect information. *CoRR*, abs/0706.2619, 2007.

[18] T. Chen, C. Chilton, B. Jonsson, and M. Kwiatkowska. A Compositional Specification Theory for Component Behaviours. In H. Seidl, editor, *Proc. 21st European Symposium on Programming (ESOP'12)*, volume 7211 of *Lecture Notes in Computer Science*. Springer-Verlag, 2012. To appear.

[19] J. Cobleigh, D. Giannakopoulou, and C. Pasareanu. Learning assumptions for compositional verification. In *Proc. TACAS'03*, volume 2619 of *LNCS*. Springer, 2003.

[20] P. Collette. Application of the composition principle to Unity-like specifications. In *TAPSOFT '93: Theory and Practice of Software Development, LNCS 668*, pages 230–242. Springer-Verlag, 1993.

[21] CONNECT consortium. CONNECT Deliverable D2.1: Capturing functional and non-functional connector behaviours. FET IP CONNECT EU project, FP7 grant agreement number 231167, http://connect-forever.eu/.

[22] CONNECT consortium. CONNECT Deliverable D2.2: Compositional algebra of connectors. FET IP CONNECT EU project, FP7 grant agreement number 231167, http://connect-forever.eu/.

[23] CONNECT consortium. CONNECT Deliverable D3.3: Dynamic connector synthesis: revised prototype implementation. FET IP CONNECT EU project, FP7 grant agreement number 231167, http://connect-forever.eu/.

[24] CONNECT consortium. CONNECT Deliverable D4.3: Deployment of learning techniques. FET IP CONNECT EU project, FP7 grant agreement number 231167, http://connect-forever.eu/.

[25] CONNECT consortium. CONNECT Deliverable D5.2: Design of Approaches for dependability and initial prototypes. FET IP CONNECT EU project, FP7 grant agreement number 231167, http://connect-forever.eu/.

[26] CONNECT consortium. CONNECT Deliverable D5.3: Consolidated dependability framework. FET IP CONNECT EU project, FP7 grant agreement number 231167, http://connect-forever.eu/.

[27] L. de Alfaro. *Formal Verification of Probabilistic Systems*. PhD thesis, Stanford University, 1997.

[28] L. de Alfaro. Game models for open systems. In N. Dershowitz, editor, *Verification: Theory and Practice*, volume 2772 of *LNCS*, pages 192–193. Springer-Verlag, 2004.

[29] L. de Alfaro and T. A. Henzinger. Interface automata. *SIGSOFT Softw. Eng. Notes*, 26(5):109–120, September 2001.

[30] L. de Alfaro and T. A. Henzinger. Interface-based design. In M. Broy, J. Grünbauer, D. Harel, and T. Hoare, editors, *Engineering Theories of Software Intensive Systems*, volume 195 of *NATO Science Series II: Mathematics, Physics and Chemistry*, pages 83–104. Springer-Verlag, 2005.

[31] F. Denis and Y. Esposito. Learning classes of probabilistic automata. In *COLT*, 2004.

[32] L. Doyen, T. A. Henzinger, B. Jobstmann, and T. Petrov. Interface theories with component reuse. In *Proc. 8th ACM international conference on Embedded software*, EMSOFT '08, pages 79–88. ACM, 2008.

[33] L. Doyen, T. A. Henzinger, and J.-F. Raskin. Equivalence of labeled Markov chains. *Int. J. Found. Comput. Sci.*, 19(3):549–563, 2008.

[34] E. A. Emerson and C. S. Jutla. Tree automata, mu-calculus and determinacy. In *32nd Annual Symposium on Foundations of Computer Science – FOCS'91, San Juan, Puerto Rico, 1-4 October 1991*. IEEE Computer Society, 1991.

[35] K. Etessami, M. Kwiatkowska, M. Vardi, and M. Yannakakis. Multi-objective model checking of Markov decision processes. *LMCS*, 4(4):1–21, 2008.

[36] A. Farzan, Y.-F. Chen, E. Clarke, Y.-K. Tsay, and B.-Y. Wang. Extending automated compositional verification to the full class of omega-regular languages. In *Proc. TACAS'08*, volume 4963 of *LNCS*, pages 2–17. Springer, 2008.

[37] L. Feng, T. Han, M. Kwiatkowska, and D. Parker. Learning-based compositional verification for synchronous probabilistic systems. In *Proc. 9th International Symposium on Automated Technology for Verification and Analysis (ATVA'11)*, volume 6996 of *LNCS*, pages 511–521. Springer, 2011.

[38] L. Feng, T. Han, M. Kwiatkowska, and D. Parker. Learning-based compositional verification for synchronous probabilistic systems. Technical Report RR-11-05, Department of Computer Science, University of Oxford, 2011.

[39] L. Feng, M. Kwiatkowska, and D. Parker. Compositional verification of probabilistic systems using learning. In *Proc. QEST'10*, pages 133–142. IEEE CS Press, 2010.

[40] L. Feng, M. Kwiatkowska, and D. Parker. Automated learning of probabilistic assumptions for compositional reasoning. In D. Giannakopoulou and F. Orejas, editors, *Proc. 14th International Conference on Fundamental Approaches to Software Engineering (FASE'11)*, volume 6603 of *LNCS*, pages 2–17. Springer, 2011.

[41] E. Grädel, W. Thomas, and T. Wilke, editors. *Automata, Logics, and Infinite Games: A Guide to Current Research [outcome of a Dagstuhl seminar, February 2001]*, volume 2500 of *Lecture Notes in Computer Science*. Springer, 2002.

[42] N. Guermouche, O. Perrin, and C. Ringeissen. A mediator based approach for services composition. In *Proceedings of the 2008 Sixth International Conference on Software Engineering Research, Management and Applications*, Washington, DC, USA, 2008. IEEE Computer Society.

[43] Y. Gurevich and L. Harrington. Trees, automata, and games. In *Proc. 14th ACM Symp. on Theory of Computing*, pages 60–65, 1982.

[44] T. Han, J.-P. Katoen, and B. Damman. Counterexample generation in probabilistic model checking. *IEEE Trans. Software Eng.*, 35(2):241–257, 2009.

[45] F. Howar, B. Steffen, B. Jonsson, and S. Cassel. Inferring Canonical Register Automata. In *Proc. VMCAI 2012*, volume 7148 of *Lecture Notes in Computer Science*, pages 251–266. Springer Verlag, 2012.

[46] R. Hull, M. Benedikt, V. Christophides, and J. Su. E-services: a look behind the curtain. In *Proceedings of the Twenty-Second ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 9-12, 2003, San Diego, CA, USA*. ACM, 2003.

[47] B. Jonsson. Compositional specification and verification of distributed systems. *ACM Trans. on Programming Languages and Systems*, 16(2):259–303, 1994.

[48] B. Jonsson and Y.-K. Tsay. Assumption/guarantee specifications in linear-time temporal logic. *Theoretical Computer Science*, 167:47–72, October 1996. An extended abstract appeared earlier in TAPSOFT '95, LNCS 915.

[49] M. B. Josephs. Receptive process theory. *Acta Inf.*, 29(1):17–31, February 1992.

[50] M. B. Josephs and H. K. Kapoor. Controllable delay-insensitive processes. *Fundam. Inf.*, 78(1):101–130, January 2007.

[51] M. Kwiatkowska, G. Norman, D. Parker, and H. Qu. Assume-guarantee verification for probabilistic systems. In *Proc. TACAS'10*, 2010.

[52] K. G. Larsen, U. Nyman, and A. Wasowski. Modal I/O automata for interface and product line theories. In R. D. Nicola, editor, *ESOP*, volume 4421 of *LNCS*, pages 64–79. Springer, 2007.

[53] G. Lüttgen and W. Vogler. Conjunction on processes: Full abstraction via ready-tree semantics. *Theor. Comput. Sci.*, 373(1-2):19–40, 2007.

[54] G. Lüttgen and W. Vogler. Ready simulation for concurrency: It's logical! *Inf. Comput.*, 208(7):845–867, 2010.

[55] N. A. Lynch and M. R. Tuttle. An introduction to input/output automata. *CWI Quarterly*, 2(3):219–246, September 1989.

[56] A. Marconi, M. Pistore, and P. Traverso. Implicit vs. explicit data-flow requirements in web service composition goals. In A. Dan and W. Lamersdorf, editors, *Service-Oriented Computing ICSOC 2006*, volume 4294 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2006.

[57] F. Martinelli. Analysis of security protocols as open systems. *Theor. Comput. Sci.*, 290:1057–1106, January 2003.

[58] F. Neven, T. Schwentick, and V. Vianu. Finite state machines for strings over infinite alphabets. *ACM Trans. Comput. Log.*, 5(3):403–435, 2004.

[59] R. D. Nicola and R. Segala. A process algebraic view of input/output automata. *Theor. Comput. Sci.*, 138(2):391–423, 1995.

[60] C. Pasareanu, D. Giannakopoulou, M. Bobaru, J. Cobleigh, and H. Barringer. Learning to divide and conquer: Applying the L* algorithm to automate assume-guarantee reasoning. *Formal Methods in System Design*, 32(3):175–205, 2008.

[61] M. Pistore, A. Marconi, P. Bertoli, and P. Traverso. Automated composition of web services by planning at the knowledge level. In *Proceedings of IJCAI'05, the Nineteenth International Joint Conference on Artificial Intelligence, Edinburgh, Scotland, UK*, 2005.

[62] M. Rabin. Probabilistic automata. *Information and Control*, 6:230–245, 1963.

[63] J.-B. Raclet. Residual for component specifications. *Electr. Notes Theor. Comput. Sci.*, 215:93–110, 2008.

[64] J.-B. Raclet, E. Badouel, A. Benveniste, B. Caillaud, A. Legay, and R. Passerone. Modal Interfaces: Unifying Interface Automata and Modal Specifications. In *Proc. 7th International Conference on Embedded Software*, EMSOFT '09, pages 87–96. ACM, 2009.

[65] J.-B. Raclet, E. Badouel, A. Benveniste, B. Caillaud, A. Legay, and R. Passerone. A modal interface theory for component-based design. *Fundam. Inform*, 108(1):119–149, 2011.

[66] J.-B. Raclet, E. Badouel, A. Benveniste, B. Caillaud, and R. Passerone. Why are modalities good for Interface Theories? In *Proc. 9th International Conference on Application of Concurrency to System Design*, ACSD '09, pages 119–127. IEEE Computer Society, 2009.

[67] R. Segala. *Modelling and Verification of Randomized Distributed Real Time Systems*. PhD thesis, Massachusetts Institute of Technology, 1995.

[68] W. Thomas. *Languages, automata, and logic*. Springer-Verlag New York, Inc., New York, NY, USA, 1997.

[69] J. Tretmans. Model-based testing and some steps towards test-based modelling. In M. Bernardo and V. Issarny, editors, *SFM*, volume 6659 of *LNCS*, pages 297–326. Springer, 2011.

[70] W.-G. Tzeng. A polynomial-time algorithm for the equivalence of probabilistic automata. *SIAM J. Comput.*, 21(2):216–227, 1992.

[71] W. Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *Theor. Comput. Sci.*, 200, 1998.