Compositional Algebra of CONNECTors

Marco Autili, Antonia Bertolino, Chris Chilton, Antinisca Di Marco, Felicita Di Giandomenico, 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 D2.2

Compositional Algebra of CONNECTors

SEVENTH FRAMEWORK PROGRAMME

http://www.connect-forever.eu

INRIA · docomo DOCOMO Euro-Labs · LANCASTER UNIVERSITY · THALES · Universita' 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, Prototype

Deliverable Number	:	D2.2
Title of Deliverable	:	Compositional Algebra of CONNECTors
Nature of Deliverable	:	Report, Prototype
Dissemination Level	:	Public
Internal Version Number	:	1.0
Contractual Delivery Date	:	31 January 2011
Actual Delivery Date	:	18 February 2011
Contributing WPs	:	WP2
Editor(s)	:	Hongyang Qu, Chris Chilton
Author(s)	:	Marco Autili, Antonia Bertolino, Chris Chilton, Antinisca Di Marco, Felicita Di Giandomenico, Paola Inverardi, Bengt Jonsson, Marta Kwiatkowska, Marco Martinucci, Hongyang Qu, Antonino Sabetta, Massimo Tivoli
Reviewer(s)	:	Franck Chauvel, Valérie Issarny

Abstract

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. In the second year, we focused our attention on providing an underpinning for a framework capable of supporting the dimensions of interest to the project, and the generalisation of assume-guarantee properties beyond probabilistic safety properties, in addition to supporting the automated learning of assumptions. In this deliverable, we report on work carried out in two streams, compositional theory of connector behaviours and compositional assume-guarantee verification for probabilistic automata, which we are beginning to bring together. At the theory level, we first considered several probabilistic models, focusing on their interactive behaviour and asynchronous parallel composition. We then conducted a survey of the different formalisms of component models from the viewpoint of whether they supported a number of operators (e.g. parallel composition, conjunction and quotient) and a refinement relation. Based on this, we concluded that interface automata were one of the most amenable formalisms for basing an algebra of* CONNECT*ors upon. We have formulated a preliminary (non-quantitative) connector algebra for resolving protocol mismatches based on interface automata. A software tool IAAnalyzer was implemented to allow users to create interface automata models using the connector algebra. In order to address the need for a quantitative extension, we have proposed a specification theory based on a probabilistic extension of interface automata. In the second stream, we continued our work on compositional assume-guarantee verification for probabilistic automata, extending the framework to allow, as assumptions and guarantees, a broader class of* quantitative multi-objective properties *which include probabilistic liveness and expected rewards. We also proposed a novel learning technique based on the L* algorithm, which automatically generates probabilistic (safety) assumptions using the results of queries executed by a probabilistic model checker. These two techniques have been implemented for PRISM, a well-known probabilistic model checker. Finally, we have laid the foundations for a comprehensive framework capable of modelling the quantitative (in particular the probabilistic) behaviour of arbitrary components and associated compositional assume-guarantee proof rules. Connectors arise as certain types of components. This framework is intended to integrate the two streams of work.*

Keyword List

Software connectors, software components, quantitative verification, functional and non-functional requirements

Document History

Version	Type of Change	Author(s)
0.1	initial version	Hongyang
0.2	part of the introduction to Chapter 2, Section 2.3, part of Section 2.4	Massimo Tivoli - UNIVAQ
0.3	Chapter 1: Introduction (partial). Chapter 2: Introduction (partial), Preliminaries, Component Models, Quantitative Theory of Connectors. Chapter 5: Conclusion (partial).	Chris Chilton
0.4	Introduction, Algebra, Verification, and Conclusions	Marta, Hongyang and Chris - UOXF; Massimo Tivoli - UNIVAQ
0.5	The chapter for the meta model, its references in Chapter 1 and Conclusions	Antinisca Di Marco - UNIVAQ; Antonia Bertolino, Antonino Sabetta - ISTI-CNR
0.6	All chapters: complete version for the first interview	All
0.7	All chapters: incorporated comments from the first interview	All
0.8	Moved the chapter for the property meta model to D5.2	Antonia Bertolino
0.9	All chapters: incorporated comments from the second interview, and moved the chapter for the property meta model to D5.2	Bengt, Chris, Massimo, Hongyang
1.0	All chapters: revised for the final submission	Marta

Document Review

Date	Version	Reviewer	Comment
04 Feb 2011	0.8	Franck Chauvel	
12 Feb 2011	0.9	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 Ccontinuous-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

Work Package 2 (WP2) of 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 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 second 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 preliminary algebra of connectors, interpreted in terms of interface automata, to formalise their behaviour and capture characteristics of the interoperability of networked systems. To allow for the expression of non-functional properties, we have proposed a quantitative specification theory based on a probabilistic extension of interface automata, which includes a number of composition operators and compositional proof rules. 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 in the next year.

In this chapter, we first recall the role of WP2 in the project (Section 1.1), followed by a summary of the achievements in the first year (Section 1.2) and the review recommendations (Section 1.3). Next, we explain the research challenges we encountered in the second year and describe the work done.

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.

The work stream (i) is aimed at formulating a comprehensive algebra for modelling, specification, composition and reasoning about connector behaviours. The CONNECT consortium has selected interface automata as the underlying (stateful) model for interactive behaviours. For maximum flexibility, we are concentrating on modelling arbitrary component behaviours, where connectors correspond to particular subclasses. In future, we will seek to characterise the components that correspond to classes of connnectors observed in CONNECT scenarios. To capture non-functional properties, we must generalise interface automata to allow for quantitative behaviours. We also need to provide a range of composition operators and a reasoning framework. To this end, we intend to focus on formulating a quantitative specification theory in the first instance, and then integrating with an appropriate syntax for the algebra.

For work stream (ii), we settled on probabilistic model checking as a means to formally analyze non-functional properties. In the deliverable D5.2 [20], 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 and interface automata, combined with an enhancement of the specification theory with assume-guarantee reasoning proof rules.

1.2 Achievements in Year 1

In our first year deliverable D2.1 [17], we conducted a survey of existing connector modelling formalisms, covering not only classical connector algebra formalisms, but also their corresponding quantitative extensions when applicable. The formalisms were evaluated against eight dimensions of interest to the CONNECT project. To consider the practical suitability of these formalisms, we modelled a CONNECT scenario using each formalism, and also made use of the associated tool support, in order to demonstrate the formalism's capability in the context of CONNECT.

We also proposed an assume-guarantee framework for probabilistic automata [51], based on a reduction to the multi-objective techniques of [36]. In this framework, a system comprising multiple interacting components can be verified by analysing each component in isolation, rather than verifying the much larger model of the whole system. Indeed, it is the first fully-automated technique for compositional verification of systems exhibiting both probabilistic and nondeterministic behaviour, though limited to probabilistic safety properties.

1.3 Review recommendations

The first year review raised a number of useful questions that have offered us insight into the development of our work. We briefly address these issues below.

1. **Is a connector an arbitrary behaviour on a set of ports?** We have deliberately separated our work at two levels to address this concern. At the specification theory level, a connector is just an arbitrary component modelled by means of an interface automaton. A textual connector algebra can be laid on top of the specification theory in order to restrict the behaviours to those that are considered to characterise connectors. In our work, we proposed such an algebra based on patterns for resolving protocol mismatches. In the future, the algebra may need to be adapted, since at this stage we do not restrict the behaviours of connectors.

2. **Is it intended to include behaviour descriptions in this formalism?** We have adopted interface automata, which indeed permits the description of behaviour.

3. **What is exactly meant by compositional reasoning?** Given a system composed of multiple components and a property for the system, we first verify each component individually against certain properties it has under some assumptions on the environment to which the component can communicate. Then we make a conclusion on whether the desired property holds on the overall system using proof rules.

4. **The criterion for reusability should be made more precise.** We consider a connector to be reusable if (i) this new context is valid, i.e., its ports are compatible with the ports of the connector (implying that the parallel composition of the components and connector is well-defined), and (ii) the connector is a refinement of the specification of the interactions that should take place between the components.

5. **Tool support for the algebra should be made more precise.** There are two distinct classes of tool support required for the algebra. The first is a tool that supports the notation and nomenclature of our algebra. It must be capable of generating models of connectors in a suitable representation of the underlying specification theory, and support the construction of new models by means of composition operators defined on connectors/the specification theory, in addition to checking for refinement. We have started to implement such a tool, called IAAnalyzer, based on Eclipse and Ticc (a tool for interface automata).

   The second class of tool support concerns those systems that are used for performing (compositional) verification on connectors. There must be a way of translating models from our specialised algebra notation into the formalisms used by verification tools, specifically the probabilistic model checker PRISM [52, 40], so that the actual verification of properties can be performed.

6. **Quantitative properties should be made more precise and coordinated with WP5.** Quantitative properties studied in WP2 are expressed using probabilistic temporal logics (with probabilistic and reward operators), e.g., Continuous Stochastic Logic (CSL) for continuous time probabilistic models, and Probabilistic Computational Tree Logic (PCTL) and Linear Temporal Logic (LTL) for discrete time probabilistic models. All these logics are supported by the probabilistic model checker PRISM, with compositional assume-guarantee reasoning possible only for probabilistic LTL, and they represent a subclass of all the dependability properties considered in WP5. In Deliverable D5.2 [20], we have carried out a comparative study of the dependability and performance properties in the terrorist alert scenario. The properties included coverage and latency, described in Chapter 2 of Deliverable D5.2 [20] using the reward operator. In Chapter 4 of D5.2, these properties are specified as CSL formulae; in Chapter 5 of D5.2, one of the properties is specified in PCTL[1]. Currently, to develop compositional assume-guarantee reasoning in WP2, we focus on discrete time probabilistic models, and specifically Probabilistic Automata (PAs)[2], and therefore use PCTL/LTL to formulate quantitative properties.

7. **It is not clear whether roles will be used as in WRIGHT.** We have decided to build our specification theory on top of interface automata rather than WRIGHT. The concept of implementations of roles is superseded by refinement of interface automata in our context.

1.4 Challenges for Year 2

The main challenge of Year 2 has been to identify a suitable approach to provide the theoretical underpinning for a component/connector algebra that is capable of expressing non-functional properties and supporting compositional reasoning. This had to take account of the needs of the CONNECT project, and to this end we considered some typical CONNECT scenarios to inform our decisions, as well as some recent relevant developments in the area. Further technical challenges included the generalisation of assume-guarantee properties beyond probabilistic safety properties, in addition to supporting the automated learning of assumptions.

[1] This formula can also be seen as a probabilistic LTL formula, as it is a reachability property.
[2] Note that MDPs (Markov decision processes) and DTMCs (Discrete-Time Markov Chains) are included in PAs.

1.5 Overview of Year 2 achievements

In order to satisfy the requirements and objectives of WP2, we have focused our attentions on two streams of work: compositional CONNECTor algebra and compositional assume-guarantee verification. We are now in a position to integrate the two streams.

Concerning compositional CONNECTor algebra, we considered a number of existing models, as well as some recent relevant work, from the point of view of: interaction behaviour, notions of refinement and equivalence, parallel composition (and particularly any interference between probabilistic and non-deterministic behaviour), and further composition operators such as conjunction. This survey culminated in the conclusion by the CONNECT consortium that interface automata (IAs) were one of the most amenable formalisms for basing an algebra of connectors upon. The rationale for this choice includes an ability to specify interaction behaviour through (non-input-enabled) interfaces; feasibility a quantitative extension; and existence of tool support (Ticc). This consequently led to the adoption of IAs as the modelling formalism and the following two key results, respectively at a syntactic and semantic levels:

- The definition of a preliminary connector algebra for resolving protocol mismatches interpreted in terms of interface automata. The textual notation restricts the behaviour of the interface automata to those that correspond to the combination of a number of primitive connectors. We implemented a software tool IAAnalyzer, based on Ticc, to allow users to create IA models using the connector algebra. More details about IAAnalyzer have been reported in the Appendix.
- The laying of the foundations for a comprehensive framework capable of modelling the quantitative (in particular the probabilistic) behaviour of arbitrary components and the associated (assume-guarantee) proof rules. This framework will incorporate the verification techniques developed as part of the work-package, which we subsequently document. The overall aim is to define a textual quantitative connector algebra in terms of this framework.

Concerning the compositional verification techniques, we have extended the framework for assume-guarantee verification for probabilistic safety properties to allow *quantitative multi-objective properties* as assumptions and guarantees. This adds the ability to reason compositionally about, for example, probabilistic liveness or expected rewards. The latter is very useful for formalising dependability properties in CONNECT. Some reward property examples can be found in D5.2, e.g., $\mathcal{R}\{\text{“}Coverage\text{”}\}_{=?}[S]$ and $\mathcal{R}\{\text{“}Latency\text{”}\}_{=?}[Fp]$. To facilitate this, we also incorporate a notion of *fairness* into the framework. We have implemented the new framework in PRISM, a well-known probabilistic model checker, and illustrated its practical applicability by checking properties which cannot be handled in [51] for two large case studies [39]. More details about the tool and its implementation can be found in the Appendix.

We also proposed a novel learning technique based on the L* algorithm, which automatically generates probabilistic (safety) assumptions using the results of queries executed by a probabilistic model checker. Learnt assumptions either establish satisfaction of the verification problem or are used to generate a probabilistic counterexample that refutes it. In the case where an assumption cannot be generated, lower and upper bounds on the probability of satisfaction are produced. The effectiveness of our approach was demonstrated by using it to generate assumptions for a range of case studies.

Finally, in order to merge the two streams of work, we have formulated a general framework that combines assume-guarantee reasoning with a specification theory for a probabilistic extension of interface automata. Thus, models in the specification theory may come decorated as an assume-guarantee triple, as a means of enhancing the types of compositional reasoning that can be performed. This will allow us to employ the compositional verification techniques developed for probabilistic automata at the level of the algebra.

1.6 Outline

In the rest of the deliverable, we first summarise the main characteristics of a number of component models in Chapter 2. In Chapter 3, we present our preliminary connector algebra for protocol mismatches and a proposal for a quantitative specification theory for interface automata, which is the subject of future work. Chapter 4 described recent results concerning quantitative compositional verification, including

the assume-guarantee reasoning approach for liveness and reward properties and automatic (probabilistic safety) assumption generation. We conclude the deliverable with a brief summary of future work in Chapter 5.

2 State of the Art in Component Modelling

In Deliverable D2.1 [17] we surveyed a collection of formalisms against a number of dimensions as determined by the CONNECT consortium. These dimensions included compositionality, incrementality, scalability, compositional reasoning, reusability, evolution, ability to express and reason about non-functional properties, and the existence of a specialised notation supported by automated tools for architectural analysis. The purpose of the survey was to determine how amenable existing formalisms are to capturing the functional and non-functional behaviour of connectors. As such, the formalisms chosen were predominantly designed with *connectors* in mind, and included WRIGHT, Reo, BIP and the Kell calculus. However, the CONNECT consortium concluded that none of the formalisms were suitable for modelling connectors in the context of CONNECT. This finding had an influence on our direction for the second year in WP2.

Following discussions, the CONNECT consortium has thus decided to draw on research into *component models*, with initial focus on automata-based formalisms to facilitate exchange with the other workpackages (e.g. WP3 and WP4) which currently use automata-like formalisms. Our rationale for this decision were as follows:

- Component models employ the notion of *interfaces*, which specify their interaction behaviour. A theory of connectors can be formulated based on general components, with a subclass of behaviours identified as corresponding to those that we consider to be characteristic of CONNECTors.
- *Specification theories* for component models have recently emerged as as an important concept to underpin component-based software engineering of heterogeneous systems. Such theories typically formalise the notions of refinement, substitutivity and contracts for components, but can be also equipped with composition operators, for example, conjunction of two independently developed components or quotient for incremental development. This is of particular importance to CONNECT, in view of our goal to composable connectors.
- In order to capture non-functional properties, we require a *quantitative* specification theory. Last year two such specification theories were proposed: one focusing on probabilistic contracts [73] and the other on constraint Markov chains [14]. These frameworks have demonstrated the feasibility of probabilistic extensions, and so are particularly relevant to CONNECT.
- A quantitative specification theory that supports a range of composition operators can serve as the basis for a syntactic *algebra* of arbitrary components. Connectors will correspond to certain subclasses of components. Composition operators at the syntactic level will have a corresponding interpretation in the specification theory, thus resulting in a compositional theory of connectors, the goal of CONNECT.

Outline. This chapter describes the review of state of the art of component models undertaken as part of our effort, which included some recent independent developments [73, 14] of relevance to CONNECT. This is to set the scene for Chapter 3, where we describe our proposed framework (including both the syntactic and specification theory strands) in more detail. We begin by recalling established probabilistic models in Section 2.1 and considering their compositionality properties (and specifically parallel asynchronous composition) and existing equivalence relations. These are key to formulating quantitative extensions of component models. Next, we survey several component modelling formalisms (I/O automata, interface automata, probabilistic I/O automata, constraint Markov chains and and specification theories, with the view to establish the type of refinement and range of composition operators of importance to CONNECT that they support.

Conclusion and contribution. The overall conclusion from the survey is that our theory of connectors should be based upon a probabilistic extension of interface automata. This was agreed in conjunction with the consortium at the Lancaster meeting in May 2010, and since then we have formulated a proposal for a comprehensive framework detailed in Chapter 3, which includes both a preliminary textual algebra with tool support, as well as a detailed technical proposal for a quantitative specification theory for components. In addition to reviewing existing work, Section 2.2.2 also contains a new contribution, namely, a

general construction for the conjunction of two interface automata (Definition 2.19), together with the proof correctness in Proposition 2.20.

2.1 Preliminaries

In this section we recall a number of established models that exhibit probabilistic behaviours, focusing on discrete-time Markov models: Markov chains, Markov decision processes and interactive Markov chains; as well as probabilistic automata. We examine these models from the perspectives of (i) issues of compositionality; and (ii) equivalences on models in terms of simulations and bisimulations. These underlying features are key to formulating a quantitative specification theory for components and connectors, and must be considered carefully.

2.1.1 Discrete-time Markov Models

Markov decision processes. Markov decision processes (MDPs), and also discrete-time Markov chains (DTMCs), which are a special case of MDPs, were documented in some detail in our previous deliverable D2.1 [17]. To recollect, DTMCs are fully probabilistic models, and therefore do not exhibit non-determinism. Yet, non-determinism is useful to model concurrency, underspecification and abstraction. Thus there is an overarching desire and need to provide explicit representations for non-deterministic behaviour in a model.

Recall that a DTMC can be viewed as a state transition graph, where each state has a probability distribution on successor states. A (discrete-time) Markov decision process (MDP) can be seen as a generalisation of a DTMC, by allowing each state to non-deterministically pick a probability distribution on the successors. A detailed treatment of MDPs is contained in [63].

Definition 2.1 *A* discrete-time Markov decision process *(MDP) is a tuple* $(S, s_0, \mathcal{A}, P, L)$*, where:*

- S *is a finite set referred to as the state space*
- $s_0 \in S$ *is the designated initial state*
- $\mathcal{A}$ *is a finite set of actions*
- $P : S \times \mathcal{A} \rightharpoonup Dist(S)$ *is the transition (partial) function*
- $L : S \to 2^{AP}$ *is a state labelling function.*

Given a state s*, we write* $\mathcal{A}(s)$ *for the maximal subset of* $\mathcal{A}$ *such that* $P(s, a)$ *is defined on each* $a \in \mathcal{A}(s)$*. Moreover, we require that for each* $s \in S$*,* $\mathcal{A}(s)$ *is non-empty.*

The set $\mathcal{A}$ is used to label the successor distributions of each state $s \in S$. The actions are uninterpreted and the labelling is independent of any external influence. Consequently, the non-deterministic choices of successor distributions can be uniquely labelled, which is why we employ a transition function rather than a relation.

The behaviour of an MDP can be observed as follows. The MDP starts in the initial state s_0, in which a non-deterministic choice is made on the actions available in $\mathcal{A}(s_0)$; let the choice be a. Subsequently, the successor state of s_0 is determined probabilistically according to the distribution $P(s_0, a)$. The MDP now continues to evolve over time in a similar manner. For each state s reachable from s_0 under $P(s_0, a)$, an action is non-deterministically picked in $\mathcal{A}(s)$ (let it be a_s), then the successor of s is determined by the distribution $P(s, a_s)$.

Typical properties considered for DTMCs, such as what is the probability of reaching a set of states T, can be generalised MDPs, except that we must take consideration of the non-determinism. This is done through the concept of an *adversary*, which given a history of an MDP decides which action should be used to determine the successor probability distribution. This is closely related to the run of an MDP, which we describe below.

Definition 2.2 *A finite run on an MDP $\mathcal{C}$ is an alternating sequence of states and actions $s_0 a_0 s_1 a_1 \ldots s_{n-1} a_{n-1} s_n$, where s_0 is the initial state of $\mathcal{C}$, $a_i \in \mathcal{A}(s_i)$ for $0 \leq i \leq n$ and $P(s_i, a_i)(s_{i+1}) > 0$ for $0 \leq i < n$. Let Run be the set of all runs on $\mathcal{A}$. Then an adversary for $\mathcal{C}$ is a function $adv : Run \to \mathcal{A}$ such that $adv(s_0 a_0 \ldots a_n s_n) \in \mathcal{A}(s_n)$.*

An adversary resolves the non-determinism in an MDP, which in turn induces a (possibly infinite) DTMC. For probabilistic model checking of MDPs against PCTL formulae [13], it is sufficient to consider the sub-class of history-independent adversaries, which resolves a non-deterministic choice based solely on the current state. In such a case, the induced DTMC is always finite.

MDPs do not make a distinction between internal and external non-determinism, which is problematic when we consider asynchronous composition of MDPs, typically based on process-algebraic synchronisation on common action labels. A desirable notion of compositionality is where synchronisation is based on action labels and interaction, but MDPs consider non-determinism inbuilt and uncontrollable, and certainly not interactive. To bridge this gap, we introduce discrete-time interactive Markov chains, which offer features to support model composition.

Discrete-time interactive Markov chains. We now treat non-determinism in an interactive way, so that we can observe how models interact with each other under asynchronous composition. Discrete-time interactive Markov chains (DTIMCs) [43] are such a model, as they can be seen as a cross between DTMCs and communicating processes from process algebra. Recall that, in an MDP, every state has a non-deterministic choice of successor probability distribution, each being labelled by a unique action. Under DTIMCs we now partition the state space into action (or interactive) states and probabilistic states. An action state evolves according to an interactively (possibly non-deterministically) picked action by the environment. A probabilistic state has a single probability distribution over successor states. Under both interactive and probabilistic transitions, the successor states may be action states, probabilistic states, or a mixture of both. This interpretation of DTIMCs is taken from [73], but originates from the concurrent Markov chains of Vardi [72].

Definition 2.3 *A* discrete-time interactive Markov chain *(DTIMC) is a tuple $(S^I, S^P, s_0, \mathcal{A}, \longrightarrow, \dashrightarrow, L)$, where:*

- *S^I is a finite set referred to as the action states (or interaction states)*
- *S^P is a finite set (disjoint from S^I) referred to as the probabilistic states*
- *$s_0 \in S^I \cup S^P$ is the designated initial state*
- *$\mathcal{A}$ is a finite set of actions*
- *$\longrightarrow \subseteq S^I \times \mathcal{A} \times (S^I \cup S^P)$ is the action transition relation*
- *$\dashrightarrow : S^P \to Dist(S^I \cup S^P)$ is the probabilistic transition function*
- *$L : (S^I \cup S^P) \to 2^{AP}$ is a state labelling function.*

For simplicity, we write $s \xrightarrow{a} s'$ for $(s, a, s') \in \longrightarrow$ and we write $s \overset{p}{\dashrightarrow} s'$ for $\dashrightarrow (s)(s') = p$.

The intuitive behaviour of a DTIMC $\mathcal{C}$ may be summarised as follows. As expected, $\mathcal{C}$ begins in the initial state s_0. If s_0 is a probabilistic state, the probability that the successor is state s is given by $\dashrightarrow (s_0)(s)$, which is independent of any environmental influence. From each of the successors s of s_0, $\mathcal{C}$ continues to evolve from s in the same manner as for the initial state. On the other hand, if s_0 is an action state, $\mathcal{C}$ waits until the environment signals one of the enabled outgoing actions of s_0, at which point the corresponding transition is committed. In the case that there are multiple transitions with the same label picked by the environment, the choice is made non-deterministically. $\mathcal{C}$ then continues execution from the successor of s_0.

It is natural to compose multiple DTIMCs together. Let $\mathcal{C}$ and $\mathcal{D}$ be two DTIMCs with signatures $(S^I_{\mathcal{C}}, S^P_{\mathcal{C}}, s^{\mathcal{C}}_0, \mathcal{A}_{\mathcal{C}}, \longrightarrow_{\mathcal{C}}, \dashrightarrow_{\mathcal{C}}, L_{\mathcal{C}})$ and $(S^I_{\mathcal{D}}, S^P_{\mathcal{D}}, s^{\mathcal{D}}_0, \mathcal{A}_{\mathcal{D}}, \longrightarrow_{\mathcal{D}}, \dashrightarrow_{\mathcal{D}}, L_{\mathcal{D}})$ respectively.

Definition 2.4 *The asynchronous parallel composition of $\mathcal{C}$ and $\mathcal{D}$ is a DTIMC $\mathcal{C} \parallel \mathcal{D}$ with signature $(S^I, S^P, s_0, \mathcal{A}, \longrightarrow, \dashrightarrow, L)$, where:*

- $S^I = S^I_{\mathcal{C}} \times S^I_{\mathcal{D}}$
- $S^P = (S^P_{\mathcal{C}} \times (S^I_{\mathcal{D}} \cup S^P_{\mathcal{D}})) \cup ((S^I_{\mathcal{C}} \cup S^P_{\mathcal{C}}) \times S^P_{\mathcal{D}})$
- $s_0 = (s_0^{\mathcal{C}}, s_0^{\mathcal{D}})$
- $\mathcal{A} = \mathcal{A}_{\mathcal{C}} \cup \mathcal{A}_{\mathcal{D}}$
- $\longrightarrow \subseteq S^I \times \mathcal{A} \times (S^I \cup S^P)$ *is the least relation satisfying:*

$$\textsc{Act-Left}\ \frac{s_c \xrightarrow{\alpha}_{\mathcal{C}} s'_c \qquad \alpha \in \mathcal{A}_{\mathcal{C}} \setminus \mathcal{A}_{\mathcal{D}} \qquad s_d \in S^I_{\mathcal{D}}}{(s_c, s_d) \xrightarrow{\alpha} (s'_c, s_d)}$$

$$\textsc{Act-Right}\ \frac{s_d \xrightarrow{\alpha}_{\mathcal{D}} s'_d \qquad \alpha \in \mathcal{A}_{\mathcal{D}} \setminus \mathcal{A}_{\mathcal{C}} \qquad s_c \in S^I_{\mathcal{C}}}{(s_c, s_d) \xrightarrow{\alpha} (s_c, s'_d)}$$

$$\textsc{Act-Sync}\ \frac{s_c \xrightarrow{\alpha}_{\mathcal{C}} s'_c \qquad s_d \xrightarrow{\alpha}_{\mathcal{D}} s'_d \qquad \alpha \in \mathcal{A}_{\mathcal{C}} \cap \mathcal{A}_{\mathcal{D}}}{(s_c, s_d) \xrightarrow{\alpha} (s'_c, s'_d)}$$

- $\dashrightarrow\colon S^P \to Dist(S^I \cup S^P)$ *is the function defined by:*
  - *For $s_c \in S^P_{\mathcal{C}}$ and $s_d \in S^I_{\mathcal{D}}$:*

$$\dashrightarrow (s_c, s_d)(s'_c, s'_d) = \begin{cases} \dashrightarrow_{\mathcal{C}} (s_c)(s'_c) & \text{if } s_d = s'_d \\ 0 & \text{otherwise} \end{cases}$$

  - *For $s_c \in S^I_{\mathcal{C}}$ and $s_d \in S^P_{\mathcal{D}}$:*

$$\dashrightarrow (s_c, s_d)(s'_c, s'_d) = \begin{cases} \dashrightarrow_{\mathcal{D}} (s_d)(s'_d) & \text{if } s_c = s'_c \\ 0 & \text{otherwise} \end{cases}$$

  - *For $s_c \in S^P_{\mathcal{C}}$ and $s_d \in S^P_{\mathcal{D}}$:*

$$\dashrightarrow (s_c, s_d)(s'_c, s'_d) = \dashrightarrow_{\mathcal{C}} (s_c)(s'_c) \cdot \dashrightarrow_{\mathcal{D}} (s_d)(s'_d).$$

- $L : (S^I \cup S^P) \to 2^{AP}$ *is defined by* $L(s_c, s_d) = L_{\mathcal{C}}(s_c) \cup L_{\mathcal{D}}(s_d)$.

The composition method just presented shows that probabilistic and interactive transitions are treated orthogonally to each other, with probabilistic transitions taking precedence. This design decision makes the composition straightforward, as actions and probability distributions do not need to be combined directly. Benefits of this include a simpler formulation of equivalence between DTIMCs in terms of bisimulations, in addition to simplifying a number of other compositional operators for component models, as we shall see in Section 2.2. We elaborate upon this design decision further in Section 2.1.2, when we consider probabilistic automata.

Quantitative properties of DTIMCs can be stated and analysed as for MDPs, that is, with respect to a scheduler that decides upon the interactive transitions to be taken.

2.1.2 Probabilistic Automata

We now consider probabilistic automata (PAs), a probabilistic model which underpins the compositional assume-guarantee verification being developed in Chapter 4. Here, we focus on composition operators for PAs.

Definition 2.5 *A probabilistic automaton is a tuple* $(S, s_0, \mathcal{A}, \longrightarrow)$*, where:*

- S *is a non-empty finite set of states*
- $s_0 \in S$ *is the designated initial state*
- $\mathcal{A}$ *is a finite set of action labels*
- $\longrightarrow \subseteq S \times \mathcal{A} \times Dist(S)$ *is the transition relation.*

Probabilistic automata come equipped with a full parallel composition operator $||$, as we reveal next. The definition is taken from [69], and is specific to the flavour of probabilistic automata that we defined.

Definition 2.6 *Let* $\mathcal{C}$ *and* $\mathcal{D}$ *be probabilistic automata with signatures* $(S_{\mathcal{C}}, s_0^{\mathcal{C}}, \mathcal{A}_{\mathcal{C}}, \longrightarrow_{\mathcal{C}})$ *and* $(S_{\mathcal{D}}, s_0^{\mathcal{D}}, \mathcal{A}_{\mathcal{D}}, \longrightarrow_{\mathcal{D}})$ *respectively. The parallel composition of* $\mathcal{C}$ *and* $\mathcal{D}$ *is always defined, and is taken to be the probabilistic automaton* $\mathcal{C} \,||\, \mathcal{D}$ *with signature* $(S, s_0, \mathcal{A}, \longrightarrow)$*, where:*

- $S = S_{\mathcal{C}} \times S_{\mathcal{D}}$
- $s_0 = (s_0^{\mathcal{C}}, s_0^{\mathcal{D}})$
- $\mathcal{A} = \mathcal{A}_{\mathcal{C}} \cup \mathcal{A}_{\mathcal{D}}$
- $\longrightarrow$ *is the least relation satisfying the following rules:*

$$\text{PA-ACT-LEFT}\ \frac{s_c \xrightarrow{\alpha}_{\mathcal{C}} \delta \qquad \alpha \in \mathcal{A}_{\mathcal{C}} \setminus \mathcal{A}_{\mathcal{D}}}{(s_c, s) \xrightarrow{\alpha} \delta \times \delta^s_{\mathcal{D}}}$$

$$\text{PA-ACT-RIGHT}\ \frac{s_d \xrightarrow{\alpha}_{\mathcal{D}} \delta \qquad \alpha \in \mathcal{A}_{\mathcal{D}} \setminus \mathcal{A}_{\mathcal{C}}}{(s, s_d) \xrightarrow{\alpha} \delta^s_{\mathcal{C}} \times \delta}$$

$$\text{PA-ACT-SYNC}\ \frac{s_c \xrightarrow{\alpha}_{\mathcal{C}} \delta \qquad s_d \xrightarrow{\alpha}_{\mathcal{D}} \delta' \qquad \alpha \in \mathcal{A}_{\mathcal{C}} \cap \mathcal{A}_{\mathcal{D}}}{(s_c, s_d) \xrightarrow{\alpha} \delta \times \delta'}$$

where:

$$\delta^s_{\mathcal{E}} = \{s \mapsto 1\} \cup \{s' \mapsto 0 : s' \in S_{\mathcal{E}} \wedge s' \neq s\}$$

i.e., the distribution that maps s *to* 1 *and everything else to* 0*, and:*

$$\delta \times \delta' : Dist(S_{\mathcal{C}} \times S_{\mathcal{D}}) \qquad\qquad \delta \times \delta'(s_c, s_d) = \delta(s_c) \cdot \delta'(s_d).$$

In [71], van Glabbeek, Smolka and Steffen categorise probabilistic processes as either reactive, generative or stratified. The categorisation can easily be applied to probabilistic automata in the following sense:

- A *reactive* probabilistic automaton has a probability distribution on states dependent on the action picked. Therefore, in any particular state, an enabled action is picked (there is at most one of each enabled action in a state), which determines a probability distribution on the successor states. Reactive probabilistic automata correspond to a deterministic (though not necessarily total) version of the probabilistic automata that we gave in Definition 2.5.

- In a *generative* probabilistic automaton, each state has a single probability distribution on successors. Action labels are pushed onto transitions, therefore action choice is now determined probabilistically, rather than non-deterministically. There is a simple way to convert any generative probabilistic automaton into a reactive one by re-normalising the probabilities. This variant of probabilistic automata was the original definition given by Segala in his PhD thesis [67], but it is not compositional - there is no natural way of composing multiple probabilistic automata of this form in an associative way.
- *Stratified* processes are very rare, so we do not consider them further here.

The discrete-time models that we presented in Section 2.1.1 can be expressed as variants of the probabilistic automata we defined in Definition 2.5. First and foremost, a DTMC can be seen as a total deterministic probabilistic automaton with a single (irrelevant) action. An MDP can be seen as a (not necessarily total) deterministic probabilistic automaton, i.e., a reactive probabilistic automaton. For DTIMCs, we can consider these as general probabilistic automata with a few restrictions. We introduce a unique action for labelling all DTIMC probabilistic transitions, while each DTIMC action transition is assigned probability 1. This induces a probabilistic automaton representing the DTIMC, but unfortunately the parallel composition on probabilistic automata does not correspond to parallel composition on DTIMCs, because in the latter probabilistic transitions take precedence over action transitions, whereas the former has no such constraints.

We also consider equivalence of probabilistic automata, in terms of bisimulation and simulation. The standard definitions of simulation and bisimulation for probabilistic automata [69] are provided below.

Definition 2.7 *Let $\mathcal{A}$ be a probabilistic automaton as defined in Definition 2.5. A relation R on S is a (strong) simulation if $s\ R\ s'$ implies that for each transition of the form $s \xrightarrow{a} \mu_s$, there exists a transition $s' \xrightarrow{a} \mu_{s'}$ such that $\mu_s\ \mathcal{L}(R)\ \mu_{s'}$. $\mathcal{L}(R)$ is a lifting of R as follows: $\mu_s\ \mathcal{L}(R)\ \mu_{s'}$ implies that there exists a weighting function $\delta : S \times S \to [0,1]$ such that:*

- $\delta(s_1, s_2) > 0$ *implies* $s_1\ R\ s_2$
- $\sum_{s_1} \delta(s_1, s_2) = \mu_{s'}(s_2)$
- $\sum_{s_2} \delta(s_1, s_2) = \mu_s(s_1)$.

R is said to be a (strong) bisimulation if it is a simulation and an equivalence relation.

Desharnais proposes an alternative definition of strong simulations and bisimulations in terms of upper closed sets, rather than weighting functions [28]. The two definitions coincide.

Besides strong simulations and bisimulations, we can consider the weak equivalents that allow for sequences of transitions containing only a single observable action. These issues will become apparent in Section 2.2 when we consider component models.

2.1.3 Summary

In this section we have recalled several key discrete-probabilistic models and examined the support they provide for interactivity and parallel composition. Our observations will shape the notion of probabilistic interface automata and the quantitative specification theory for component behaviours that we are developing.

2.2 Component Models

In this section we examine a range of component modelling and specification formalisms, the purpose being to identify which of these formalisms are amenable to a quantitative specification theory for connectors. We focus on composition operations, such as parallel composition and conjunction, together with relations that facilitate (incremental) compositional development via e.g. component substitutivity. In particular, we consider the following features as particularly relevant to CONNECT, in view of the heterogeneity of the CONNECTor framework, also raised in [66]:

- **Compatibility.** This relation is used to determine whether two or more models can coexist together in harmony. Therefore, compatibility is a check on whether the parallel composition of two components is well-defined. Compatibility is normally a simple check on the static type or interface of a model, although there are exceptions.

- **Parallel composition $||$.** This operation is used to build complex models out of simpler ones by modelling how they interact together. We will be interested in determining whether parallel composition is only a partial function (on compatible models), and also whether the operation is associative. Associativity is necessary for any compositional theory of components.

- **Refinement $\sqsubseteq$.** Refinement is a preorder on models that determines when a model can be used in place of another. Refinement allows one to determine whether a model is a specification or a potential implementation of another model.

  Some component models will have precise characterisations of what makes a model an implementation. We write $I \models S$ for I is an implementation of S. Normally we have that $S \sqsubseteq \lceil I \rceil$, where the ceiling operation is a lifting of I to a fully specified specification equivalent to I.

  Refinement can be used to provide a notion of seamless substitutivity for components (depending on the way it is defined). The following properties support this concept (NB. write $P \sqsubseteq Q$ for P is refined by Q):

  - **Independent implementability.** If $P \models S$ and $Q \models T$, then $P||Q \models S||T$.
  - **Compositionality.** If $P \sqsubseteq P'$ and $P||Q$ is defined, then $P'||Q$ is defined and $P||Q \sqsubseteq P'||Q$.

- **Hiding.** To support different levels of abstraction, we require hiding to make some actions internal to a component. This removes their visibility from the external environment, consequently affecting synchronisation of multiple components with respect to the different operators.

- **Conjunction $\wedge$.** The conjunction of two models P and Q yields the least specified model that refines both P and Q. Conjunction gives a way of combining multiple specifications, and indeed implementations, in order to build a specification/implementation satisfying multiple, independently determined sub-specifications and implementations.

- **Quotient $\setminus$.** Quotient (residuation) is an operation defined on models S and P, whereby $S \setminus P$ yields the least specified model Q such that $S \sqsubseteq P||Q$. In that sense, quotient is the adjoint of parallel composition. The operator arises often in model synthesis, for example, where we have a global specification and a partial implementation. The quotient of these models would yield a specification for the remaining part of the system to be implemented.

These operators and relations have direct relevance for CONNECT. For example, parallel composition can be used to compose multiple connectors to work with each other; refinement can be used to determine when a connector can be used in place of another; hiding can be used to build a hierarchy of connectors; conjunction can be used for combining multiple connector behaviours; and quotient can be used for synthesising connectors.

Parallel composition and refinement are frequently discussed in the literature on process algebra and component models, whereas conjunction and quotient are rarely mentioned in either, particularly the former. The argument for including these operators is as follows [66]:

- In the case of conjunction, one generally expects a specification for a model to detail all of the properties to hold. However, it makes more sense for the different concerns to be separated into disparate specifications and combined in some automated way. Under this arrangement, it is far easier to check the accuracy[1] of small single-purpose specifications, rather than large all encompassing ones.

[1] Both the accuracy of the specification itself, as well as any implementations of the specification.

- Concerning quotient, this operator has applications to reasoning about assume-guarantee contracts $(\mathcal{A}, \mathcal{G})$. For modal specifications, the following result shows that quotient can be used to eliminate implication occurring as part of a validity test for contracts: $\forall \mathcal{I} \cdot \mathcal{I} \models \mathcal{A} \Rightarrow \mathcal{I} \mid\mid \mathcal{I}' \models \mathcal{G}$ if and only if $\mathcal{I}' \models \mathcal{G} \setminus \mathcal{A}$.

Specification theories have historically been based on process algebras, such as CCS [60] and CSP [44], or variants of automata. In the context of CONNECT we will focus specifically on the automaton-based approach for two reasons:

1. Automata have been identified as a unifying framework for all work-packages in the CONNECT project.

2. A process-algebraic formulation for a specification theory can be endowed with operational semantics, which would induce a labelled transition system. Thus, the foundational work we will conduct at the automata-level could be formulated in terms of a process calculus at a later date.

Consequently, the remainder of this section focuses on analysing a number of automata-based formalisms against the aforementioned specification theory operators and relations. *In the case of interface automata, we contribute a general definition of conjunction and prove its correctness.* The end of the section concludes by comparing and contrasting the different formalisms.

2.2.1 I/O Automata

I/O automata [57] (which were also independently introduced in [47], an extended version of which is [48]) may be seen as finite state machines whose actions are partitioned into input, output and hidden sets. At the syntactic level this classification has no significance, but examination of the semantics reveals a notion of communication well suited to modelling connector interactions.

Definition 2.8 *An* I/O automaton $\mathcal{C}$ *is a tuple* $(S, s_0, \mathcal{A}_I, \mathcal{A}_O, \mathcal{A}_H, \longrightarrow)$*, where:*

- S *is a finite set of states*
- $s_0 \in S$ *is the designated initial state*
- $\mathcal{A} \triangleq \mathcal{A}_I \uplus \mathcal{A}_O \uplus \mathcal{A}_H$ *is the set of actions, specifically:*
  - $\mathcal{A}_I$ *is the set of input actions*
  - $\mathcal{A}_O$ *is the set of output actions*
  - $\mathcal{A}_H$ *is the set of internal/hidden actions.*
- $\longrightarrow \subseteq S \times \mathcal{A} \times S$ *is the transition relation.*

As the environment is allowed to behave in unpredictable ways, the requirement of full input enabledness is imposed on I/O automata, which asserts that each state must be willing to accept every input.

For brevity, we write $v \xrightarrow{a} v'$ iff $(v, a, v') \in \longrightarrow$. The automaton behaves in a manner similar to that for finite state machines; it starts in the initial state and evolves over time according to its transition relation. However, special consideration must be given to the transition types. A transition labelled by an input action may only be picked when the environment is offering that action. Output and hidden actions are non-blocking, so may be taken at any time if they are enabled in the current state. Since outputs are non-blocking, it follows that the environment must be willing to accept any action it is offered.

We now consider how the generic specification theory operators and relations map on to I/O automata. From hereon let $\mathcal{C} = (S_\mathcal{C}, s_0^\mathcal{C}, \mathcal{A}_I^\mathcal{C}, \mathcal{A}_O^\mathcal{C}, \mathcal{A}_H^\mathcal{C}, \longrightarrow_\mathcal{C})$ and $\mathcal{D} = (S_\mathcal{D}, s_0^\mathcal{D}, \mathcal{A}_I^\mathcal{D}, \mathcal{A}_O^\mathcal{D}, \mathcal{A}_H^\mathcal{D}, \longrightarrow_\mathcal{D})$ be two I/O automata.

Compatibility. $\mathcal{C}$ and $\mathcal{D}$ are said to be compatible if they can be composed together under parallel composition, which means that they can co-exist and potentially interact with each other in a sound manner. However, it so happens that not every pair of I/O automata can be composed.

Definition 2.9 *I/O automata $\mathcal{C}$ and $\mathcal{D}$ are said to be* composable*, written $\mathcal{C} \sim \mathcal{D}$, just if:*

- $\mathcal{A}_O^{\mathcal{C}} \cap \mathcal{A}_O^{\mathcal{D}} = \emptyset$*. Only a single automaton can instantiate a particular output action.*
- $\mathcal{A}_H^{\mathcal{C}} \cap \mathcal{A}^{\mathcal{D}} = \emptyset = \mathcal{A}^{\mathcal{C}} \cap \mathcal{A}_H^{\mathcal{D}}$*. Internal actions must be local to an automaton. Of course, because of the nature of internal actions we can always employ renaming to alleviate a conflict of this constraint.*

Although we do not do so in this report, it is possible to consider the composition of an infinite number of I/O automata. This requires a further constraint on the action sets, which stipulates that each action occurs within only finitely many of the automata to be composed.

Parallel composition. The asynchronous composition of two composable I/O automata is another I/O automaton realised by the standard product construction. Convention dictates how the action sets should be transformed.

Definition 2.10 *The asynchronous parallel composition of two composable I/O automata $\mathcal{C}$ and $\mathcal{D}$ is an I/O automaton $\mathcal{C} \times \mathcal{D} = (S_{\mathcal{C}} \times S_{\mathcal{D}}, (s_0^{\mathcal{C}}, s_0^{\mathcal{D}}), \mathcal{A}_I, \mathcal{A}_O, \mathcal{A}_H, \longrightarrow)$, where the action sets are defined as:*

- $\mathcal{A}_I = (\mathcal{A}_I^{\mathcal{C}} \cup \mathcal{A}_I^{\mathcal{D}}) \setminus (\mathcal{A}_O^{\mathcal{C}} \cup \mathcal{A}_O^{\mathcal{D}})$
- $\mathcal{A}_O = \mathcal{A}_O^{\mathcal{C}} \cup \mathcal{A}_O^{\mathcal{D}}$
- $\mathcal{A}_H = \mathcal{A}_H^{\mathcal{C}} \cup \mathcal{A}_H^{\mathcal{D}}$

and the transition relation is precisely characterised by:

$$(s,t) \xrightarrow{a} (s',t') \Leftrightarrow \begin{cases} s \xrightarrow{a}_{\mathcal{C}} s' \wedge t \xrightarrow{a}_{\mathcal{D}} t' & \text{if } a \in \mathcal{A}^{\mathcal{C}} \cap \mathcal{A}^{\mathcal{D}} \\ s \xrightarrow{a}_{\mathcal{C}} s' \wedge t = t' & \text{if } a \in \mathcal{A}^{\mathcal{C}} \setminus \mathcal{A}^{\mathcal{D}} \\ t \xrightarrow{a}_{\mathcal{D}} t' \wedge s = s' & \text{if } a \in \mathcal{A}^{\mathcal{D}} \setminus \mathcal{A}^{\mathcal{C}}. \end{cases}$$

The convention of combining common input and output actions into outputs has a considerable benefit for specification theories. Besides allowing us to mimic broadcasting, it makes parallel composition associative. This would not have been the case if input and output actions combine to form a hidden action.

Refinement. Refinement for I/O automata is classically given in terms of trace inclusion or simulation as described in [56]. However, neither of these solutions are adequate because an I/O automaton can refine another by constraining the behaviour of the environment – something that we should not permit. In that sense, this type of refinement does not correlate with substitutivity. The solution is to define refinement in terms of alternating simulation relations, as highlighted in [3] by Alur, Henzinger, Kupferman and Vardi. These alternating simulations were applied by de Alfaro and Henzinger to their interface automata, which are a generalisation of I/O automata. We therefore postpone our treatment of refinement until we consider interface automata. That definition will then carry over to here seamlessly. As a preview, the idea is to treat inputs and outputs in a contravariant fashion.

Hiding. To fully support abstraction, we need the notion of a hiding operator on I/O automata. Note that we do not allow hiding of input actions. This is because input actions are issued by the environment, so there is no way for a component to influence their visibility.

Definition 2.11 *Let $\mathcal{B} \subseteq \mathcal{A}_O^{\mathcal{C}} \cup \mathcal{A}_H^{\mathcal{C}}$ be a subset of actions of $\mathcal{C}$. The hiding of actions in $\mathcal{B}$ on $\mathcal{C}$ is given by the parameterised function $hide_{\mathcal{B}}$ defined as $hide_{\mathcal{B}}(\mathcal{C}) = (S_{\mathcal{C}}, s_0^{\mathcal{C}}, \mathcal{A}_I^{\mathcal{C}}, \mathcal{A}_O^{\mathcal{C}} \setminus \mathcal{B}, \mathcal{A}_H^{\mathcal{C}} \cup \mathcal{B}, \longrightarrow_{\mathcal{C}})$.*

Conjunction. The conjunction of two I/O automata $\mathcal{C}$ and $\mathcal{D}$ is the least specified I/O automaton that refines each of $\mathcal{C}$ and $\mathcal{D}$. All I/O automata are compatible for conjunction; therefore the operation is always defined.

Definition 2.12 *The conjunction of I/O automata $\mathcal{C}$ and $\mathcal{D}$ is itself an I/O automaton $\mathcal{C} \wedge \mathcal{D}$ with signature $(S, s_0, \mathcal{A}_I, \mathcal{A}_O, \mathcal{A}_H, \longrightarrow)$, where:*

- $S = S_{\mathcal{C}} \times S_{\mathcal{D}}$
- $s_0 = (s_0^{\mathcal{C}}, s_0^{\mathcal{D}})$
- $\mathcal{A}_I = \mathcal{A}_I^{\mathcal{C}} \cup \mathcal{A}_I^{\mathcal{D}}$
- $\mathcal{A}_O = \mathcal{A}_O^{\mathcal{C}} \cap \mathcal{A}_O^{\mathcal{D}}$
- $\mathcal{A}_H = \mathcal{A}_H^{\mathcal{C}} \cup \mathcal{A}_H^{\mathcal{D}}$
- $\longrightarrow$ *is the least relation generated by the following rules:*

$$\textsc{Act-In-Sync}\ \frac{s_c \xrightarrow{a}_{\mathcal{C}} s_c' \qquad s_d \xrightarrow{a}_{\mathcal{D}} s_d' \qquad a \in \mathcal{A}_I^{\mathcal{C}} \cap \mathcal{A}_I^{\mathcal{D}}}{(s_c, s_d) \xrightarrow{a} (s_c', s_d')}$$

$$\textsc{Act-In-Ind-C}\ \frac{s_c \xrightarrow{a}_{\mathcal{C}} s_c' \qquad a \in \mathcal{A}_I^{\mathcal{C}} \setminus \mathcal{A}_I^{\mathcal{D}}}{(s_c, s_d) \xrightarrow{a} (s_c', s_d)}$$

$$\textsc{Act-In-Ind-D}\ \frac{s_d \xrightarrow{a}_{\mathcal{D}} s_d' \qquad a \in \mathcal{A}_I^{\mathcal{D}} \setminus \mathcal{A}_I^{\mathcal{C}}}{(s_c, s_d) \xrightarrow{a} (s_c, s_d')}$$

$$\textsc{Act-Out-Sync}\ \frac{s_c \xrightarrow{a}_{\mathcal{C}} s_c' \qquad s_d \xrightarrow{a}_{\mathcal{D}} s_d' \qquad a \in \mathcal{A}_O}{(s_c, s_d) \xrightarrow{a} (s_c', s_d')}$$

$$\textsc{Act-Hide-C}\ \frac{s_c \xrightarrow{a}_{\mathcal{C}} s_c' \qquad a \in \mathcal{A}_H^{\mathcal{C}}}{(s_c, s_d) \xrightarrow{a} (s_c', s_d)}$$

$$\textsc{Act-Hide-D}\ \frac{s_d \xrightarrow{a}_{\mathcal{D}} s_d' \qquad a \in \mathcal{A}_H^{\mathcal{D}}}{(s_c, s_d) \xrightarrow{a} (s_c, s_d')}.$$

Quotient. There do not appear to be any attempts at defining quotient for I/O automata, although the corresponding construction for interface automata (see next section) could be carried over to here.

2.2.2 Interface Automata

Interface automata (IAs) [23] are an optimistic variant of I/O automata, which drop the constraint on states being fully input-enabled. The states of such an automaton define the set of legal inputs that can be issued by the environment in that particular state. Therefore, an interface automaton induces a set of legal environments that it can exist within (each of which may be modelled by an interface automaton itself).

Despite interface automata being a slight relaxation on the definition of I/O automata, the formulation of operators and relations is considerably more involved. In particular, the operation of conjunction has been treated in a particularly constrained setting with many restrictions [29], but in this section we provide our own generalised formulation in Definition 2.19 and prove its correctness in Proposition 2.20.

Definition 2.13 *An* interface automaton $\mathcal{C}$ *is a tuple* $(S, s_0, \mathcal{A}_I, \mathcal{A}_O, \mathcal{A}_H, \longrightarrow)$ *that can be seen as an I/O automaton not imposing full input-enabledness (cf Definition 2.9).*

Let $\mathcal{C} = (S_{\mathcal{C}}, s_0^{\mathcal{C}}, \mathcal{A}_I^{\mathcal{C}}, \mathcal{A}_O^{\mathcal{C}}, \mathcal{A}_H^{\mathcal{C}}, \longrightarrow_{\mathcal{C}})$ and $\mathcal{D} = (S_{\mathcal{D}}, s_0^{\mathcal{D}}, \mathcal{A}_I^{\mathcal{D}}, \mathcal{A}_O^{\mathcal{D}}, \mathcal{A}_H^{\mathcal{D}}, \longrightarrow_{\mathcal{D}})$ be two interface automata. We now consider the operators and relations.

Compatibility. Compatibility of interface automata is considerably more involved than on I/O automata. For interface automata to be compatible, it is necessary for them to be compatible/composable in the sense of I/O automata, but it is also necessary to construct the parallel composition and perform a check, as we'll see.

Parallel composition. The parallel composition of two composable interface automata $\mathcal{C}$ and $\mathcal{D}$ is an interface automaton $\mathcal{C} \mid\mid \mathcal{D}$ realised by a pruning of the parallel composition of $\mathcal{C}$ and $\mathcal{D}$ interpreted as I/O automata.

Let $\mathcal{C} \times \mathcal{D}$ be the parallel composition of $\mathcal{C}$ and $\mathcal{D}$ interpreted as I/O automata. $\mathcal{C} \times \mathcal{D}$ can introduce a number of illegal states, whereby one of the automata is willing to offer an output action in the common alphabet, but the second is not able to accept the corresponding input action. This is a consequence of not requiring IA to be fully input-enabled.

We must identify the global set of illegal states in the product $\mathcal{C} \times \mathcal{D}$, denoted by $Illegal(\mathcal{C}, \mathcal{D})$. The kernel of the illegal states is taken to be the set of states (s_c, s_d) for which there is some $a \in (\mathcal{A}_I^{\mathcal{C}} \cap \mathcal{A}_O^{\mathcal{D}}) \cup (\mathcal{A}_O^{\mathcal{C}} \cap \mathcal{A}_I^{\mathcal{D}})$ such that one of s_c and s_d can make an a-labelled output transition, but the other cannot match it with the corresponding input transition. The illegal set is then taken to be all those states in the kernel, plus those that can reach a state in the kernel by a sequence of transitions labelled by hidden and output actions.

Definition 2.14 *The parallel composition of two composable interface automata $\mathcal{C}$ and $\mathcal{D}$ is defined to be an interface automaton $\mathcal{C} \mid\mid \mathcal{D}$ obtained from $\mathcal{C} \times \mathcal{D}$ by removing all states in $Illegal(\mathcal{C}, \mathcal{D})$, providing the initial state $(s_0^{\mathcal{C}}, s_0^{\mathcal{D}})$ is not removed. Otherwise, the parallel composition is undefined.*

Definition 2.15 *If the parallel composition of two composable interface automata $\mathcal{C}$ and $\mathcal{D}$ is defined, we say that $\mathcal{C}$ and $\mathcal{D}$ are compatible, written as $\mathcal{C} \sim \mathcal{D}$. Otherwise they are not compatible.*

Refinement. For an interface automaton $\mathcal{D}$ to refine another interface automaton $\mathcal{C}$ (which we write as $\mathcal{C} \sqsubseteq \mathcal{D}$), it is necessary for the inputs to be related covariantly, but the outputs contravariantly: $\mathcal{A}_I^{\mathcal{C}} \subseteq \mathcal{A}_I^{\mathcal{D}}$ and $\mathcal{A}_O^{\mathcal{D}} \subseteq \mathcal{A}_O^{\mathcal{C}}$. Thus $\mathcal{D}$ may allow more inputs, but its outputs must be contained within those of $\mathcal{C}$.

The previous condition is a type check for refinement, but it does not actually determine whether $\mathcal{C} \sqsubseteq \mathcal{D}$. To verify this assertion, it is necessary to check the relationship between the behaviour of the respective automata. This is done by means of an alternating simulation as described by Alur, Henzinger, Kupferman and Vardi in [3].

Definition 2.16 *A relation $\leqslant \subseteq S_{\mathcal{C}} \times S_{\mathcal{D}}$ is an alternating simulation on $\mathcal{C}$ and $\mathcal{D}$ if whenever $s_c \leqslant s_d$ holds the following points all hold:*

- *If $s_c \xrightarrow{a}_{\mathcal{C}} s'_c$ with $a \in \mathcal{A}_I^{\mathcal{C}}$, then there exists a transition $s_d \xrightarrow{a}_{\mathcal{D}} s'_d$ such that $s'_c \leqslant s'_d$.*
- *If $s_d \xrightarrow{a}_{\mathcal{D}} s'_d$ with $a \in \mathcal{A}_O^{\mathcal{D}}$, then there exists a state s'_c reachable from s_c by zero or more hidden actions such that $s'_c \xrightarrow{a}_{\mathcal{C}} s''_c$ and $s''_c \leqslant s'_d$.*
- *If $s_d \xrightarrow{a}_{\mathcal{D}} s'_d$ with $a \in \mathcal{A}_H^{\mathcal{D}}$, then there exists a state s'_c reachable from s_c by zero or more hidden actions such that $s'_c \leqslant s'_d$.*

Definition 2.17 *An interface automaton $\mathcal{D}$ refines an interface automaton $\mathcal{C}$, written $\mathcal{C} \sqsubseteq \mathcal{D}$, if and only if:*

- *$\mathcal{A}_I^{\mathcal{C}} \subseteq \mathcal{A}_I^{\mathcal{D}}$ and $\mathcal{A}_O^{\mathcal{D}} \subseteq \mathcal{A}_O^{\mathcal{C}}$; and*
- *$s_0^{\mathcal{C}} \leqslant s_0^{\mathcal{D}}$ for some alternating simulation $\leqslant \subseteq S_{\mathcal{C}} \times S_{\mathcal{D}}$.*

Thus refinement on interface automata can be seen as an alternating game between two players: an antagonist and a protagonist. To check whether $\mathcal{C} \sqsubseteq \mathcal{D}$, the game proceeds from the current state as follows: the antagonist tries to pick an input of $\mathcal{C}$ or an output / hidden action of $\mathcal{D}$ that the protagonist cannot match. If the protagonist cannot match the move, they lose and $\mathcal{C} \not\sqsubseteq \mathcal{D}$, otherwise the game

recommences from the resultant state. If the antagonist is forced to a state where it can't make any move, they lose and $\mathcal{C} \sqsubseteq \mathcal{D}$. In the case of infinite play, $\mathcal{C} \sqsubseteq \mathcal{D}$.

In [24], de Alfaro and Henzinger show that refinement on interface automata is a pre-order, and also that interface automata are compositional with respect to parallel composition.

Lemma 2.18 *If $\mathcal{C} \sim \mathcal{D}$, $\mathcal{C} \sqsubseteq \mathcal{C}'$ and $\mathcal{A}^{\mathcal{C}'} \cap \mathcal{A}^{\mathcal{D}} \subseteq \mathcal{A}^{\mathcal{C}} \cap \mathcal{A}^{\mathcal{D}}$, then $\mathcal{C}' \sim \mathcal{D}$ and $\mathcal{C} \,||\, \mathcal{D} \sqsubseteq \mathcal{C}' \,||\, \mathcal{D}$.*

For interface automata, an implementation for $\mathcal{C}$ can be seen as any interface automata $\mathcal{D}$ such that $\mathcal{C} \sqsubseteq \mathcal{D}$. Thus, interface automata trivially support independent implementability.

Hiding. The definition for hiding on interface automata is exactly the same as for on I/O automata, because it is an operation that acts upon the signature and not on the transition system.

Conjunction. The shared refinement (or conjunction) of two interface automata $\mathcal{C}$ and $\mathcal{D}$, written $\mathcal{C} \wedge \mathcal{D}$, is only defined if the action sets of the automata do not conflict with each other. $\mathcal{C}$ and $\mathcal{D}$ are compatible for conjunction if $\mathcal{A}_I^{\mathcal{C}} \cup \mathcal{A}_I^{\mathcal{D}}$, $\mathcal{A}_O^{\mathcal{C}} \cup \mathcal{A}_O^{\mathcal{D}}$ and $\mathcal{A}_H^{\mathcal{C}} \cup \mathcal{A}_H^{\mathcal{D}}$ are pairwise disjoint.

Informally, $\mathcal{C} \wedge \mathcal{D}$ is the least specified interface automaton that refines each of $\mathcal{C}$ and $\mathcal{D}$. More precisely, $\mathcal{C} \sqsubseteq \mathcal{C} \wedge \mathcal{D}$, $\mathcal{D} \sqsubseteq \mathcal{C} \wedge \mathcal{D}$ and for all interface automata $\mathcal{E}$ such that $\mathcal{C} \sqsubseteq \mathcal{E}$ and $\mathcal{D} \sqsubseteq \mathcal{E}$, it is the case that $\mathcal{C} \wedge \mathcal{D} \sqsubseteq \mathcal{E}$. Recall that refinement on interface automata is a pre-order, and not a partial order. Therefore the conjunction is not necessarily uniquely defined, because $\sqsubseteq$ is not anti-symmetric in general.

We now present our own construction for the conjunction of compatible interface automata, as we have yet to find a satisfactory description in the literature. One such attempt is provided in [29], but this only deals with deterministic synchronous interface automata without hidden actions.

Definition 2.19 *The conjunction of two interface automata $\mathcal{C}$ and $\mathcal{D}$ is the interface automaton $\mathcal{C} \wedge \mathcal{D}$ with signature $(S, (s_0^{\mathcal{C}}, s_0^{\mathcal{D}}), \mathcal{A}_I^{\mathcal{C}} \cup \mathcal{A}_I^{\mathcal{D}}, \mathcal{A}_O^{\mathcal{C}} \cup \mathcal{A}_O^{\mathcal{D}}, \mathcal{A}_H^{\mathcal{C}} \cup \mathcal{A}_H^{\mathcal{D}}, \longrightarrow)$, where $S = (S_{\mathcal{C}} \times S_{\mathcal{D}}) \cup S_{\mathcal{C}} \cup S_{\mathcal{D}}$ and $\longrightarrow$ is the smallest relation satisfying:*

1. *For $a \in \mathcal{A}_O^{\mathcal{C}} \cap \mathcal{A}_O^{\mathcal{D}}$:*
   (a) *If $s_c \xrightarrow{a}_{\mathcal{C}} s_c'$ and $s_d \xrightarrow{a}_{\mathcal{D}} s_d'$, then $(s_c, s_d) \xrightarrow{a} (s_c', s_d')$*
2. *For $a \in \mathcal{A}_I^{\mathcal{C}} \cup \mathcal{A}_I^{\mathcal{D}}$:*
   (a) *If $s_c \xrightarrow{a}_{\mathcal{C}} s_c'$ and $s_d \not\xrightarrow{a}_{\mathcal{D}}$, then $(s_c, s_d) \xrightarrow{a} s_c'$*
   (b) *If $s_c \not\xrightarrow{a}_{\mathcal{C}}$ and $s_d \xrightarrow{a}_{\mathcal{D}} s_d'$, then $(s_c, s_d) \xrightarrow{a} s_d'$*
   (c) *If $s_c \xrightarrow{a}_{\mathcal{C}} s_c'$ and $s_d \xrightarrow{a}_{\mathcal{D}} s_d'$, then $(s_c, s_d) \xrightarrow{a} (s_c', s_d')$*
3. *For $a \in \mathcal{A}_H^{\mathcal{C}} \cup \mathcal{A}_H^{\mathcal{D}}$:*
   (a) *If $s_c \xrightarrow{a}_{\mathcal{C}} s_c'$, then $(s_c, s_d) \xrightarrow{a} (s_c', s_d)$*
   (b) *If $s_d \xrightarrow{a}_{\mathcal{D}} s_d'$, then $(s_c, s_d) \xrightarrow{a} (s_c, s_d')$*
4. *For $a \in \mathcal{A}^{\mathcal{C}} \cup \mathcal{A}^{\mathcal{D}}$:*
   (a) *If $a \in \mathcal{A}^{\mathcal{C}}$ and $s_c \xrightarrow{a}_{\mathcal{C}} s_c'$, then $s_c \xrightarrow{a} s_c'$*
   (b) *If $a \in \mathcal{A}^{\mathcal{D}}$ and $s_d \xrightarrow{a}_{\mathcal{D}} s_d'$, then $s_d \xrightarrow{a} s_d'$.*

Proposition 2.20 *Definition 2.19 correctly defines conjunction on interface automata.*
Proof: Let $\mathcal{C}$ and $\mathcal{D}$ be two interface automata compatible for conjunction. We show that (i) $\mathcal{C} \sqsubseteq \mathcal{C} \wedge \mathcal{D}$ and (ii) if $\mathcal{C} \sqsubseteq \mathcal{E}$ and $\mathcal{D} \sqsubseteq \mathcal{E}$, then $\mathcal{C} \wedge \mathcal{D} \sqsubseteq \mathcal{E}$.

For (i), define a relation $R \subseteq S_{\mathcal{C}} \times S$ such that $s \; R \; t$ iff $s = t$ or $t = (s, t')$. We show that R is an alternating simulation. So let $s_c \in S_{\mathcal{C}}$ and $s_d \in S_{\mathcal{D}}$ be arbitrary states, and note that $s_c \; R \; (s_c, s_d)$ and $s_c \; R \; s_c$. For the case of $s_c \; R \; s_c$, the identity relation is always an alternating simulation, so there is nothing to prove. Instead, just consider the case of $s_c \; R \; (s_c, s_d)$:

- If $s_c \xrightarrow{a}_{\mathcal{C}} s'_c$ with $a \in \mathcal{A}_I^{\mathcal{C}}$, then by our definition of conjunction:
  - If $s_d \xrightarrow{a}_{\mathcal{D}} s'_d$ then $(s_c, s_d) \xrightarrow{a} (s'_c, s'_d)$ with $s'_c \; R \; (s'_c, s'_d)$
  - If $s_d \not\xrightarrow{a}_{\mathcal{D}} s'_d$ then $(s_c, s_d) \xrightarrow{a} s'_c$ with $s'_c \; R \; s'_c$.
- If $(s_c, s_d) \xrightarrow{a} (s'_c, s'_d)$ with $a \in \mathcal{A}_O$, then by our definition of conjunction it must be the case that $a \in \mathcal{A}_O^{\mathcal{C}}$ and $s_c \xrightarrow{a}_{\mathcal{C}} s'_c$. Hence $s'_c \; R \; (s'_c, s'_d)$ as required.
- Finally, if $(s_c, s_d) \xrightarrow{a} (s'_c, s'_d)$ with $a \in \mathcal{A}_H$, then either $s_c \xrightarrow{a}_{\mathcal{C}} s'_c$ with $s_d = s'_d$, or $s_d \xrightarrow{a}_{\mathcal{D}} s'_d$ with $s_c = s'_c$. So in both cases we obtain $s'_c \; R \; (s'_c, s'_d)$.

Thus R is an alternating simulation as required, and so $\mathcal{C} \sqsubseteq \mathcal{C} \wedge \mathcal{D}$ because $s_0^{\mathcal{C}} \; R \; (s_0^{\mathcal{C}}, s_0^{\mathcal{D}})$. Similarly, $\mathcal{D} \sqsubseteq \mathcal{C} \wedge \mathcal{D}$.

For (ii), suppose $\mathcal{C} \sqsubseteq \mathcal{E}$ and $\mathcal{D} \sqsubseteq \mathcal{E}$, then there must exist alternating simulations $R_{\mathcal{C}}$ and $R_{\mathcal{D}}$ such that $s_0^{\mathcal{C}} \; R_{\mathcal{C}} \; s_0^{\mathcal{E}}$ and $s_0^{\mathcal{D}} \; R_{\mathcal{D}} \; s_0^{\mathcal{E}}$. Now define a relation $R \subseteq S \times S_{\mathcal{E}}$ by $R \triangleq \{((s,t),u) : s \; R_{\mathcal{C}} \; u \wedge t \; R_{\mathcal{D}} \; u\} \cup \{(v,u) : v \; R_{\mathcal{C}} \; u \wedge v \in S_{\mathcal{C}}$ or $v \; R_{\mathcal{D}} \; u \wedge v \in S_{\mathcal{D}}\}$. Note that $(s_0^{\mathcal{C}}, s_0^{\mathcal{D}}) \; R \; s_0^{\mathcal{E}}$, so show that R is an alternating simulation because then $\mathcal{C} \wedge \mathcal{D} \sqsubseteq \mathcal{E}$. So for arbitrary states $s_c \in S_{\mathcal{C}}$, $s_d \in S_{\mathcal{D}}$ and $s_e \in S_{\mathcal{E}}$, assume $(s_c, s_d) \; R \; s_e$.

- If $(s_c, s_d) \xrightarrow{a} s'$ with $a \in \mathcal{A}_I$, then there are three ways that this transition could have arisen:
  1. $s_c \xrightarrow{a}_{\mathcal{C}} s'_c$, $s' = s'_c$ and $s_d \not\xrightarrow{a}_{\mathcal{D}}$. Since $s_c \; R_{\mathcal{C}} \; s_e$ and $s_c \xrightarrow{a}_{\mathcal{C}} s'_c$, it follows that $s_e \xrightarrow{a}_{\mathcal{E}} s'_e$ with $s'_c \; R_C \; s'_e$. Thus $s'_c \; R \; s'_e$ as required.
  2. $s_d \xrightarrow{a}_{\mathcal{D}} s'_d$, $s' = s'_d$ and $s_c \not\xrightarrow{a}_{\mathcal{C}}$. Since $s_d \; R_{\mathcal{D}} \; s_e$ and $s_d \xrightarrow{a}_{\mathcal{D}} s'_d$, it follows that $s_e \xrightarrow{a}_{\mathcal{E}} s'_e$ with $s'_d \; R_D \; s'_e$. Thus $s'_d \; R \; s'_e$ as required.
  3. $s_c \xrightarrow{a}_{\mathcal{C}} s'_c$, $s_d \xrightarrow{a}_{\mathcal{D}} s'_d$ and $s' = (s'_c, s'_d)$. Since $s_c \; R_{\mathcal{C}} \; s_e$ and $s_c \xrightarrow{a}_{\mathcal{C}} s'_c$, it follows that $s_e \xrightarrow{a}_{\mathcal{E}} s'_e$ with $s'_c \; R_{\mathcal{C}} \; s'_e$. Since $s_d \; R_{\mathcal{D}} \; s_e$ and $s_d \xrightarrow{a}_{\mathcal{D}} s'_d$, it follows that $s_e \xrightarrow{a}_{\mathcal{E}} s''_e$ with $s'_d \; R_{\mathcal{D}} \; s''_e$. Under the assumption of input-determinism, we have $s'_e = s''_e$ and so $(s'_c, s'_d) \; R \; s'_e$.
- If $s_e \xrightarrow{a}_{\mathcal{E}} s'_e$ with $a \in \mathcal{A}_O^{\mathcal{E}}$, then because $s_c \; R_{\mathcal{C}} \; s_e$ and $s_d \; R_{\mathcal{D}} \; s_e$, it follows that there exist transitions $s_c \xrightarrow{a}_{\mathcal{C}} s'_c$ and $s_d \xrightarrow{a}_{\mathcal{D}} s'_d$ such that $s'_c \; R_{\mathcal{C}} \; s'_e$ and $s'_d \; R_{\mathcal{D}} \; s'_e$. By our definition of conjunction, $(s_c, s_d) \xrightarrow{a} (s'_c, s'_d)$ and so $(s'_c, s'_d) \; R \; s'_e$.
- If $s_e \xrightarrow{a}_{\mathcal{E}} s'_e$ with $a \in \mathcal{A}_H^{\mathcal{E}}$, then because $s_c \; R_{\mathcal{C}} \; s_e$ and $s_d \; R_{\mathcal{D}} \; s_e$, it follows that there exists a state s'_c reachable by hidden transitions from s_c, and there is a state s'_d reachable by hidden transitions from s_d, such that $s'_c \; R_{\mathcal{C}} \; s'_e$ and $s'_d \; R_{\mathcal{D}} \; s'_e$. Looking at our definition of conjunction, we can conclude that (s'_c, s'_d) is reachable by hidden transitions from (s_c, s_d), and so $(s'_c, s'_d) \; R \; s'_e$ as required.

The cases of single states on the left hand side of R are trivial, thus R is an alternating simulation. Hence $\mathcal{C} \sqsubseteq \mathcal{E}$ and $\mathcal{D} \sqsubseteq \mathcal{E}$ imply $\mathcal{C} \wedge \mathcal{D} \sqsubseteq \mathcal{E}$. We have therefore demonstrated that our definition of conjunction defines the least specified interface automaton that refines both of its arguments. □

Quotient. The operation of quotient has been defined for the sub-class of deterministic interface automata without hidden actions. Under such a restriction, Bhaduri and Ramesh show in [11] that the quotient of two interface automata $\mathcal{C}$ and $\mathcal{D}$, written as $\mathcal{C} \setminus \mathcal{D}$, is an interface automaton $(\mathcal{C}^{\perp} \,||\, \mathcal{D})^{\perp}$, where the $^{\perp}$ operator inverts the input and output sets.

2.2.3 Interactive Markov Chains

In [73], Xu, Gössler and Girault propose a probabilistic contract framework based on interactive Markov chains (IMCs). In this setting, an interactive Markov chain can be seen as an implementation for a system, while a contract is a specification for a possibly infinite number of interactive Markov chains.

The definition of a (discrete-time) interactive Markov chain was provided in Definition 2.3. We therefore move straight on to the concept of a contract as a finite representation for an infinite number of IMCs.

Definition 2.21 *A probabilistic contract is a tuple* $(S^I, S^P, s_0, \mathcal{A}, \longrightarrow, \sigma)$*, where:*

- S^I *is a finite set referred to as the action states (or interaction states)*
- S^P *is a finite set (disjoint from* S^I*) referred to as the probabilistic states*
- $S \triangleq S^I \uplus S^P \uplus \{\top\}$ *is the set of states*
- $s_0 \in S$ *is the designated initial state*
- $\mathcal{A}$ *is a finite set of actions*
- $\longrightarrow \subseteq S^I \times \mathcal{A} \times S$ *is the action transition relation*
- $\sigma : S^P \to (S \to 2^{[0,1]})$ *is the probabilistic transition function, where for each* $s, s' \in S$ *the value of* $\sigma(s)(s')$ *is an interval on* $[0,1]$.

The $\top$ state is related to assumptions and guarantees on action state transitions. A transition of the form $s \xrightarrow{a} s'$ with $s' \neq \top$ means that state s is guaranteed to accept a if it is offered, whereas $s \not\xrightarrow{a}$ asserts that a is guaranteed not to be offered in s. A transition of the form $s \xrightarrow{a} \top$ implies that the issue of a in state s violates the assumption, so unpredictable behaviour is permitted from the $\top$ state.

The probabilistic behaviour of contracts is a generalisation of that for IMCs. For states s and s' (with s probabilistic), the probability of moving from s to s' is contained in the interval $\sigma(s)(s')$.

From hereon let $\mathcal{C}$ and $\mathcal{D}$ be contracts with signatures $(S^I_{\mathcal{C}}, S^P_{\mathcal{C}}, s^{\mathcal{C}}_0, \mathcal{A}_{\mathcal{C}}, \longrightarrow_{\mathcal{C}}, \sigma_{\mathcal{C}})$ and $(S^I_{\mathcal{D}}, S^P_{\mathcal{D}}, s^{\mathcal{D}}_0, \mathcal{A}_{\mathcal{D}}, \longrightarrow_{\mathcal{D}}, \sigma_{\mathcal{D}})$ respectively. Moreover, let $\mathcal{I}$ be an IMC with signature $(S^I, S^P, s_0, \mathcal{A}, \longrightarrow, \dashrightarrow)$.

Compatibility. All probabilistic contracts are compatible with each other.

Parallel composition. In [73], the authors take the BIP interaction model for parameterising the parallel composition operator with the sets of interactions that should be synchronised upon. Instead, we treat the synchronisation set to be the intersection of the respective alphabets.

Definition 2.22 *The parallel composition of contracts* $\mathcal{C}$ *and* $\mathcal{D}$ *is a contract* $\mathcal{C} \parallel \mathcal{D}$*, with signature* $(S^I, S^P, s_0, \mathcal{A}, \longrightarrow, \sigma)$ *defined in the same way as in Definition 2.4, except that:*

- $\{\top\} = (S_{\mathcal{C}} \times \{\top_{\mathcal{D}}\}) \cup (\{\top_{\mathcal{C}}\} \times S_{\mathcal{D}})$. *An assumption violation occurs whenever any of the constituents' assumptions are violated.*
- $\sigma : S^P \to (S \to 2^{[0,1]})$ *is defined as:*
  - *For* $s_c \in S^P_{\mathcal{C}}$ *and* $s_d \in S^I_{\mathcal{D}}$*:*

$$\sigma(s_c, s_d)(s'_c, s'_d) = \begin{cases} \sigma_{\mathcal{C}}(s_c)(s'_c) & \text{if } s_d = s'_d \\ [0,0] & \text{otherwise} \end{cases}$$

  - *For* $s_c \in S^I_{\mathcal{C}}$ *and* $s_d \in S^P_{\mathcal{D}}$*:*

$$\sigma(s_c, s_d)(s'_c, s'_d) = \begin{cases} \sigma_{\mathcal{D}}(s_d)(s'_d) & \text{if } s_c = s'_c \\ [0,0] & \text{otherwise} \end{cases}$$

  - *For* $s_c \in S^P_{\mathcal{C}}$ *and* $s_d \in S^P_{\mathcal{D}}$*:*

$$\sigma(s_c, s_d)(s'_c, s'_d) = [\inf \sigma_{\mathcal{C}}(s_c)(s'_c) * \inf \sigma_{\mathcal{D}}(s_d)(s'_d),\ \sup \sigma_{\mathcal{C}}(s_c)(s'_c) * \sup \sigma_{\mathcal{D}}(s_d)(s'_d)].$$

Refinement. We now elaborate on refinement for contracts, and determine whether an IMC is an implementation of a contract. Following the methodology employed in [73], we only consider the case in which $\mathcal{A}_\mathcal{C} = \mathcal{A}_\mathcal{D} = \mathcal{A}$.

Definition 2.23 *A contract $\mathcal{C}$ is refined by contract $\mathcal{D}$, written $\mathcal{C} \sqsubseteq \mathcal{D}$, if and only if $s_0^\mathcal{C} \leqslant s_0^\mathcal{D}$, where $\leqslant \subseteq S_\mathcal{C} \times S_\mathcal{D}$ is the greatest relation satisfying for all $s_c \leqslant s_d$:*

- *Case:* $(s_c, s_d) \in S_\mathcal{C}^I \times S_\mathcal{D}^I$
  - *If* $s_d = \top$, *then* $s_c = \top$
  - *If* $s_d \xrightarrow{a}_\mathcal{D} s'_d$, *then* $s_c = \top$ *or* $\exists s'_c \cdot s_c \xrightarrow{a}_\mathcal{C} s'_c \wedge s'_c \leqslant s'_d$
  - *If* $s_c \xrightarrow{a}_\mathcal{C} s'_c \neq \top$, *then* $\exists s'_d \cdot s_d \xrightarrow{a}_\mathcal{D} s'_d \wedge s'_c \leqslant s'_d$.
- *Case:* $(s_c, s_d) \in S_\mathcal{C}^P \times S_\mathcal{D}^P$

  There exists a weighting function $\delta : S_\mathcal{C} \times S_\mathcal{D} \to [0,1]$ *such that for each probability distribution* $f \in Dist(S_\mathcal{D})$, *if* $f \in \sigma_\mathcal{D}(s_d)$, *then:*
  - $\delta(s'_c, s'_d) > 0$ *implies* $s'_c \leqslant s'_d$
  - $f(s'_d) \neq 0$ *implies* $\delta(\_, s'_d) \in Dist(S_\mathcal{C})$
  - $\sum_{s'_d} f(s'_d) * \delta(s'_c, s'_d) \in \sigma_\mathcal{C}(s_c)(s'_c)$.
- *Case:* $(s_c, s_d) \in S_\mathcal{C}^I \times S_\mathcal{D}^P$
  - $\exists s'_d \in S_\mathcal{D}^I$ *reachable from* s_d *by a sequence of non-zero probabilistic transitions such that* $s_c \leqslant s'_d$
  - $\forall s'_d \in S_\mathcal{D}$, *if* $\sigma_\mathcal{D}(s_d)(s'_d) \neq [0,0]$, *then* $s_c \leqslant s'_d$.
- *Case:* $(s_c, s_d) \in S_\mathcal{C}^P \times S_\mathcal{D}^I$
  - $\exists s'_c \in S_\mathcal{C}^I$ *reachable from* s_c *by a sequence of non-zero probabilistic transitions such that* $s'_c \leqslant s_d$
  - $\forall s'_c \in S_\mathcal{C}$, *if* $\sigma_\mathcal{C}(s_c)(s'_c) \neq [0,0]$, *then* $s'_c \leqslant s_d$.

It can easily be shown that refinement on contracts is a pre-order.

An IMC $\mathcal{I}$ is an implementation of a contract $\mathcal{C}$, written as $\mathcal{I} \models \mathcal{C}$, just if $\mathcal{C} \sqsubseteq \lceil \mathcal{I} \rceil$, where $\lceil \mathcal{I} \rceil$ is a lifting of $\mathcal{I}$ from an IMC to a contract by replacing each probabilistic transition p by an interval $[p,p]$. We have that, if $\mathcal{I} \models \mathcal{D}$ and $\mathcal{C} \sqsubseteq \mathcal{D}$, then $\mathcal{I} \models \mathcal{C}$. Moreover, satisfaction and refinement support implementation inclusion: $\mathcal{C} \sqsubseteq \mathcal{D}$ implies $\{\mathcal{I} : \mathcal{I} \models \mathcal{C}\} \supseteq \{\mathcal{I} : \mathcal{I} \models \mathcal{D}\}$.

Other key results are independent implementability: $\mathcal{I} \models \mathcal{C}$ and $\mathcal{J} \models \mathcal{D}$ implies that $\mathcal{I} \,||\, \mathcal{J} \models \mathcal{C} \,||\, \mathcal{D}$; together with congruence of refinement: for contracts $\mathcal{C}$, $\mathcal{D}$ and $\mathcal{E}$, if $\mathcal{C} \sqsubseteq \mathcal{D}$, then $\mathcal{C} \,||\, \mathcal{E} \sqsubseteq \mathcal{D} \,||\, \mathcal{E}$.

Hiding. The form of probabilistic contracts that we have considered in this report do not represent internal actions explicitly. To hide $\mathcal{B} \subseteq \mathcal{A}_\mathcal{C}$ in a contract $\mathcal{C}$, we simply convert all $\mathcal{B}$-actions in $\longrightarrow_\mathcal{C}$ to ϵ-transitions, then remove the ϵ-transitions by determinising the relation, to obtain a new relation $\longrightarrow'_\mathcal{C}$. The result is a probabilistic contract with signature $(S_\mathcal{C}^I, S_\mathcal{C}^P, s_0^\mathcal{C}, \mathcal{A}_\mathcal{C} \setminus \mathcal{B}, \longrightarrow'_\mathcal{C}, \sigma_\mathcal{C})$.

Conjunction. Unfortunately, conjunction is a partial operation on contracts, specifically those that are defined to be delimited, and whose normalisations are unambiguous. A contract is said to be *delimited* if for each probabilistic state s and each probability $p \in \sigma(s)(s')$ for each state s', there exist probabilities $p_i \in \sigma(s)(s_i)$ for all other states $s_i \neq s'$ such that $p + \sum_i p_i = 1$.

The normalisation of a contract $\mathcal{C}$ is itself a contract $\bar{\mathcal{C}}$ obtained from bisimulation minimisation of $\mathcal{C}$. For the class of delimited contracts, it can be shown that $\mathcal{C} \sqsubseteq \bar{\mathcal{C}}$ and $\bar{\mathcal{C}} \sqsubseteq \mathcal{C}$. This constraint is a necessity for conjunction to be sound, which is why conjunction is only defined on delimited contracts.

Definition 2.24 *A bisimulation on $\mathcal{C}$ is a relation $R \subseteq S_\mathcal{C} \times S_\mathcal{C}$ such that if $s \; R \; t$, then:*

- *Case:* $(s,t) \in S_\mathcal{C}^I \times S_\mathcal{C}^I$

- *If* $s \xrightarrow{a}_{\mathcal{C}} s'$*, then* $\exists t' \cdot t \xrightarrow{a}_{\mathcal{C}} t'$ *and* $s' \; R \; t'$
- *If* $t \xrightarrow{a}_{\mathcal{C}} t'$*, then* $\exists s' \cdot s \xrightarrow{a}_{\mathcal{C}} s'$ *and* $s' \; R \; t'$*.*

- *Case:* $(s,t) \in S^P_{\mathcal{C}} \times S^P_{\mathcal{C}}$
  - *There exists a weighting function* $\delta : S_{\mathcal{C}} \times S_{\mathcal{C}} \to [0,1]$ *such that:*
    - $\delta(s',t') > 0$ *implies* $s' \; R \; t'$
    - *for each* $s' \in S_{\mathcal{C}}$*,* $\delta(s') \in Dist(S_{\mathcal{C}})$ *and every* $f \in \sigma_{\mathcal{C}}(s)$ *we require:*

$$\sum_{s'} f(s') * \delta(s',t') \in \sigma_{\mathcal{C}}(t)(t').$$

  - *There exists a weighting function* $\varepsilon : S_{\mathcal{C}} \times S_{\mathcal{C}} \to [0,1]$ *such that:*
    - $\varepsilon(s',t') > 0$ *implies* $s' \; R \; t'$
    - *for each* $t' \in S_{\mathcal{C}}$*,* $\varepsilon(t') \in Dist(S_{\mathcal{C}})$ *and every* $f \in \sigma_{\mathcal{C}}(t)$ *we require:*

$$\sum_{t'} f(t') * \varepsilon(t',s') \in \sigma_{\mathcal{C}}(s)(s').$$

- *Case:* $(s,t) \in S^I_{\mathcal{C}} \times S^P_{\mathcal{C}}$
  - $\exists t' \in S^I_{\mathcal{C}}$ *reachable from* t *by a sequence of non-zero probabilistic transitions such that* $s \; R \; t'$
  - $\forall t' \in S_{\mathcal{C}}$*, if* $\sigma_{\mathcal{C}}(t)(t') \neq [0,0]$*, then* $s \; R \; t'$*.*

- *Case:* $(s,t) \in S^P_{\mathcal{C}} \times S^I_{\mathcal{C}}$
  - $\exists s' \in S^I_{\mathcal{C}}$ *reachable from* s *by a sequence of non-zero probabilistic transitions such that* $s' \; R \; t$
  - $\forall s' \in S_{\mathcal{C}}$*, if* $\sigma_{\mathcal{C}}(s)(s') \neq [0,0]$*, then* $s' \; R \; t$*.*

Bisimulations are closed under union. Consequently, the largest bisimulation, which we denote by $\simeq$ and call the bisimilarity relation, is equal to the union of all bisimulations. Given a contract $\mathcal{C}$, we will use $\simeq$ to derive a normalised contract $\bar{\mathcal{C}}$, which is equivalent to $\mathcal{C}$ modulo $\simeq$-equivalence.

Definition 2.25 *The bisimulation minimisation of a contract* $\mathcal{C}$ *is a contract* $\bar{\mathcal{C}}$ *with signature* $(S^I, S^P, s_0, \mathcal{A}, \longrightarrow, \sigma)$*, where:*

- $S^I = \{\{s' \in S_{\mathcal{C}} : s \simeq s'\} : s \in S^I_{\mathcal{C}}\}$
- $S^P = \{\{s' \in S_{\mathcal{C}} : s \simeq s'\} : s \in S^P_{\mathcal{C}}\}$
- $s_0 = \{s' \in S_{\mathcal{C}} : s_0 \simeq s'\}$
- $\mathcal{A} = \mathcal{A}_{\mathcal{C}}$
- $S' \xrightarrow{a} S''$ *if, and only if, there exists* $s' \in S'$ *and* $s'' \in S''$ *such that* $s' \xrightarrow{a}_{\mathcal{C}} s''$
- $\sigma(S')(S'') = \sum_{s'' \in S''} \sigma_{\mathcal{C}}(s')(s'')$ *for any* $s' \in S'$ *(where the summation of a collection of intervals* A *is the interval* $X \triangleq [\sum_{a \in A} \inf a, \sum_{a \in A} \sup a] \cap [0,1]$ *providing* $X \neq \emptyset$*, otherwise is the interval* $[0,0]$*).*

Thus given a contract $\mathcal{C}$, we can construct a normalised form for it, denoted by $\bar{\mathcal{C}}$. As remarked earlier, the normalisation of a delimited contract is related to the original delimited contract by mutual refinement.

In order to determine whether two contracts have a common refinement, it is necessary to check whether each can simulate the other on the common alphabet. We thus present the definition of mutual simulation.

Definition 2.26 *Let* $\mathcal{C}$ *and* $\mathcal{D}$ *be normalised contracts. Mutual simulation of* $\mathcal{C}$ *and* $\mathcal{D}$ *is the largest relation* $\sim \subseteq S_{\mathcal{C}} \times S_{\mathcal{D}}$ *such that if* $s_c \sim s_d$*, then* $s_c = \top$*,* $s_d = \top$ *or:*

- *Case:* $(s_c, s_d) \in S^I_{\mathcal{C}} \times S^I_{\mathcal{D}}$

- *If* $s_c \xrightarrow{a}_{\mathcal{C}} s'_c$*, then either* $\exists s'_d \cdot s_d \xrightarrow{a}_{\mathcal{D}} s'_d$ *and* $s'_c \sim s'_d$*, or* $a \notin \mathcal{A}_{\mathcal{D}}$ *and* $s'_c \sim s_d$
- *If* $s_d \xrightarrow{a}_{\mathcal{D}} s'_d$*, then either* $\exists s'_c \cdot s_c \xrightarrow{a}_{\mathcal{C}} s'_c$ *and* $s'_c \sim s'_d$*, or* $a \notin \mathcal{A}_{\mathcal{C}}$ *and* $s_c \sim s'_d$*.*

- *Case:* $(s_c, s_d) \in S^P_{\mathcal{C}} \times S^P_{\mathcal{D}}$
  - *If* $\sigma_{\mathcal{C}}(s_c)(s'_c) = P_{\mathcal{C}}$*, then* $\exists s'_d \cdot \sigma_{\mathcal{D}}(s_d)(s'_d) = P_{\mathcal{D}}$ *with* $P_{\mathcal{C}} \cap P_{\mathcal{D}} \neq \emptyset$ *such that* $s'_c \sim s'_d$ *or* $0 \in P_{\mathcal{C}} \cap P_{\mathcal{D}}$
  - *If* $\sigma_{\mathcal{D}}(s_d)(s'_d) = P_{\mathcal{D}}$*, then* $\exists s'_c \cdot \sigma_{\mathcal{C}}(s_c)(s'_c) = P_{\mathcal{C}}$ *with* $P_{\mathcal{C}} \cap P_{\mathcal{D}} \neq \emptyset$ *such that* $s'_c \sim s'_d$ *or* $0 \in P_{\mathcal{C}} \cap P_{\mathcal{D}}$*.*
- *Case:* $(s_c, s_d) \in S^I_{\mathcal{C}} \times S^P_{\mathcal{D}}$
  - *If* $\sigma_{\mathcal{D}}(s_d)(s'_d) = P_{\mathcal{D}}$*, then* $s_c \sim s'_d$ *or* $0 \in P_{\mathcal{D}}$*.*
- *Case:* $(s_c, s_d) \in S^P_{\mathcal{C}} \times S^I_{\mathcal{D}}$
  - *If* $\sigma_{\mathcal{C}}(s_c)(s'_c) = P_{\mathcal{C}}$*, then* $s'_c \sim s_d$ *or* $0 \in P_{\mathcal{C}}$*.*

We are now close to defining conjunction of delimited normalised contracts, however we need to impose yet another constraint. The requirement that contracts are unambiguous is necessary, otherwise problems occur in the construction. A contract $\mathcal{C}$ is said to be unambiguous, if $\sigma_{\mathcal{C}}(s)(s') = P'$ and $\sigma_{\mathcal{C}}(s)(s'') = P''$ with $s' \sim s''$ and $P' \cap P'' \neq \emptyset$ implies that $s' = s''$. Conjunction of ambiguous contracts can lead to a contract that is unsatisfiable.

Definition 2.27 *Let* $\mathcal{C}$ *and* $\mathcal{D}$ *be two delimited contracts. If* $\bar{\mathcal{C}}$ *and* $\bar{\mathcal{D}}$ *are unambiguous,* $\mathcal{C} \wedge \mathcal{D}$ *is defined to be* $\bar{\mathcal{C}} \mathbin{\widetilde{\wedge}} \bar{\mathcal{D}}$ *providing* $\bar{\mathcal{C}} \sim \bar{\mathcal{D}}$*, otherwise* $\mathcal{C} \wedge \mathcal{D}$ *is defined to be* $\mathcal{C}_\perp$*, the least specified unrealisable contract.*

$\mathcal{C} \mathbin{\widetilde{\wedge}} \mathcal{D}$ *is a probabilistic contract with signature* $(S^I, S^P, s_0, \mathcal{A}, \longrightarrow, \sigma)$*, where:*

- $S^I = S^I_{\mathcal{C}} \times S^I_{\mathcal{C}}$
- $S^P = S^P_{\mathcal{C}} \times S_{\mathcal{D}} \cup S_{\mathcal{C}} \times S^P_{\mathcal{D}}$
- $s_0 = (s_0^{\mathcal{C}}, s_0^{\mathcal{D}})$
- $\mathcal{A} = \mathcal{A}_{\mathcal{C}} \cup \mathcal{A}_{\mathcal{D}}$
- $\longrightarrow$ *is the least relation satisfying the following rules:*

$$\textsc{Act-Sync}\ \frac{s_c \xrightarrow{a}_{\mathcal{C}} s'_c \qquad s_d \xrightarrow{a}_{\mathcal{D}} s'_d}{(s_c, s_d) \xrightarrow{a} (s'_c, s'_d)}$$

$$\textsc{Act-Ind-Left}\ \frac{s_c \xrightarrow{a}_{\mathcal{C}} s'_c \qquad a \notin \mathcal{A}_{\mathcal{D}} \qquad s_d \in S^I_{\mathcal{D}}}{(s_c, s_d) \xrightarrow{a} (s'_c, s_d)}$$

$$\textsc{Act-Ind-Right}\ \frac{s_d \xrightarrow{a}_{\mathcal{D}} s'_d \qquad a \notin \mathcal{A}_{\mathcal{C}} \qquad s_c \in S^I_{\mathcal{C}}}{(s_c, s_d) \xrightarrow{a} (s_c, s'_d)}$$

$$\textsc{Act-Viol-Right}\ \frac{s_c \xrightarrow{a}_{\mathcal{C}} s'_c}{(s_c, \top_{\mathcal{D}}) \xrightarrow{a} (s'_c, \top_{\mathcal{D}})}$$

$$\textsc{Act-Viol-Left}\ \frac{s_d \xrightarrow{a}_{\mathcal{D}} s'_d}{(\top_{\mathcal{C}}, s_d) \xrightarrow{a} (\top_{\mathcal{C}}, s'_d)}$$

- σ *is the function defined by:*

$$\textsc{Prob-Sync}\ \frac{\sigma_{\mathcal{C}}(s_c)(s'_c) = P_{\mathcal{C}} \qquad \sigma_{\mathcal{D}}(s_d)(s'_d) = P_{\mathcal{D}} \qquad s'_c \sim s'_d}{\sigma(s_c, s_d)(s'_c, s'_d) = P_{\mathcal{C}} \cap P_{\mathcal{D}}}$$

$$\textsc{Prob-Ind-Left}\ \frac{\sigma_{\mathcal{C}}(s_c)(s'_c) = P_{\mathcal{C}} \qquad s_d \in S^I_{\mathcal{D}} \cup \{\top_{\mathcal{D}}\}}{\sigma(s_c, s_d)(s'_c, s_d) = P_{\mathcal{C}}}$$

$$\textsc{Prob-Ind-Right}\ \frac{\sigma_{\mathcal{D}}(s_d)(s'_d) = P_{\mathcal{D}} \qquad s_c \in S^I_{\mathcal{C}} \cup \{\top_{\mathcal{C}}\}}{\sigma(s_c, s_d)(s_c, s'_d) = P_{\mathcal{D}}}.$$

Thus, the shared refinement of multiple contracts is definable, albeit under a number of restrictions.

Quotient. There does not, as of yet, appear to be a successful attempt at defining quotient for probabilistic contracts.

2.2.4 Constraint Markov Chains

This section considers a probabilistic specification theory based on constraint Markov chains [14]. A constraint Markov chain (CMC) can be seen as a generalisation of the probabilistic contracts presented in the previous section, as probabilistic transitions from a state must satisfy a collection of linear constraints, rather than lie in a certain interval. Intervals can naturally be represented by constraints.

However, CMCs are not a complete generalisation of probabilistic contracts, because CMCs do not have interactive transitions. Therefore, a CMC can be seen as a finite representation for a (possibly) infinite number of DTMCs, whereas probabilistic contracts are a finite representation for a (possibly) infinite number of DTIMCs.

Definition 2.28 *A* constraint Markov chain (CMC) *is a tuple* $(S, s_0, \mathcal{A}, L, \varphi)$*, where:*

- S *is a finite collection of states*
- $s_0 \in S$ *is the initial state*
- $\mathcal{A}$ *is a finite set of atomic propositions*
- $L : S \to 2^{2^{\mathcal{A}}}$ *is a state valuation function*
- $\varphi : S \to 2^{Dist(S)}$ *is a constraint function, such that* $\varphi(s)$ *yields a set of successor distributions for* s*.*

From hereon let $\mathcal{C} = (S_{\mathcal{C}}, s^{\mathcal{C}}_0, \mathcal{A}_{\mathcal{C}}, L_{\mathcal{C}}, \varphi_{\mathcal{C}})$ and $\mathcal{D} = (S_{\mathcal{D}}, s^{\mathcal{D}}_0, \mathcal{A}_{\mathcal{D}}, L_{\mathcal{D}}, \varphi_{\mathcal{D}})$ be CMCs, and $\mathcal{E} = (S_{\mathcal{E}}, s^{\mathcal{E}}_0, P_{\mathcal{E}}, L_{\mathcal{E}})$ a DTMC. We now consider the different operators and relations on constraint Markov chains.

Compatibility. All CMCs are compatible for parallel composition. A more interesting concept is that of consistency, which is elaborated upon next.

Consistency. The term consistency is used to stipulate whether a CMC has an implementation or not. Consistency can be determined by iteratively applying a pruning operator β to a CMC until a fixed-point is reached. If the initial state is removed, then the CMC is inconsistent. Otherwise, the CMC is consistent.

The pruning operator β removes inconsistent states from a CMC. A state s is said to be inconsistent if $L(s) = \emptyset$ or there is no distribution d with $d \in \varphi(s)$. The transformation of a CMC under β is given below.

Definition 2.29 *Let* $\mathcal{I}$ *be the set of inconsistent states occurring in a CMC* $\mathcal{C}$*. The pruning of* $\mathcal{C}$ *under* β *is the CMC* $\beta(\mathcal{C})$ *with signature* $(S_{\mathcal{C}} \setminus \mathcal{I}, s^{\mathcal{C}}_0, \mathcal{A}_{\mathcal{C}}, L_{\mathcal{C}} \restriction S_{\mathcal{C}} \setminus \mathcal{I}, \varphi)$*, where:*

$$d \in \varphi(s) \iff d \cup \{s' \mapsto 0 : s' \in \mathcal{I}\} \in \varphi_{\mathcal{C}}(s).$$

Refinement. In terms of refinement of CMCs, the authors of [14] propose three variants. The strongest, which is referred to as strong refinement, is based on the definition provided in [49]. This version is also a generalisation of the definition given for probabilistic contracts in Definition 2.23. The remaining forms of refinement are weak refinement and model inclusion.

For contrast with Definition 2.23, I have modified the definition of strong refinement on CMCs to talk about weighting functions rather than correspondence matrices. The resulting refinement is the same, but the added benefit of this form is that it is clear to see that the refinement is nearly identical to that on probabilistic contracts, except that intervals are replaced with constraints.

Definition 2.30 *A CMC $\mathcal{C}$ is strongly refined by a CMC $\mathcal{D}$, written as $\mathcal{C} \sqsubseteq \mathcal{D}$, iff $\mathcal{A}_\mathcal{C} \subseteq \mathcal{A}_\mathcal{D}$ and there exists a relation $R \subseteq S_\mathcal{C} \times S_\mathcal{D}$ containing $(s_0^\mathcal{C}, s_0^\mathcal{D})$ such that R satisfies the following property: if $s_c \; R \; s_d$, then*

- $L_\mathcal{D}(s_d) \upharpoonright \mathcal{A}_\mathcal{C} \subseteq L_\mathcal{C}(s_c)$
- *there exists a weighting function $\delta : S_\mathcal{C} \times S_\mathcal{D} \to [0,1]$ such that for each probability distribution $f \in Dist(S_\mathcal{D})$, if $f \in \varphi_\mathcal{D}(s_d)$, then*
  - $\delta(s'_c, s'_d) \neq 0$ *implies* $s'_c \; R \; s'_d$
  - $f(s'_d) \neq 0$ *implies* $\delta(\_, s'_d) \in Dist(S_\mathcal{C})$
  - $\exists d \in \varphi_\mathcal{C}(s_c)$ *such that* $\sum_{s'_d} f(s'_d) * \delta(s'_c, s'_d) = d(s'_c)$.

Definition 2.31 *The definition of weak refinement of CMCs is the same as for strong refinement, except that the weighting function δ is picked after settling upon the probability distribution f. We write $\mathcal{C} \preceq \mathcal{D}$ for $\mathcal{C}$ is weakly refined by $\mathcal{D}$.*

The notion of satisfiability is used to describe when a DTMC is a model for/implementation of a CMC.

Definition 2.32 *A DTMC $\mathcal{E}$ is a model for CMC $\mathcal{C}$, written $\mathcal{E} \models \mathcal{C}$ iff $\mathcal{C} \sqsubseteq \lceil \mathcal{E} \rceil$, where $\lceil \mathcal{E} \rceil$ is a lifting of $\mathcal{E}$ to a CMC $(S_\mathcal{E}, s_0^\mathcal{E}, \mathcal{A}_\mathcal{E}, \{L_\mathcal{E}\}, \varphi_\mathcal{E})$, where $\varphi_\mathcal{E}(s) = \{P(s)\}$ and $\mathcal{A}_\mathcal{E}$ is the set of atomic propositions used for $L_\mathcal{E}$.*

Definition 2.33 *A CMC $\mathcal{C}$ is refined by a CMC $\mathcal{D}$ with respect to models iff $\{\mathcal{I} : \mathcal{I} \models \mathcal{D}\} \subseteq \{\mathcal{I} : \mathcal{I} \models \mathcal{C}\}$.*

Hiding. Abstraction of CMCs is straightforward, as it is a simple operation on the signature. For a CMC $\mathcal{C}$, the hiding of $\mathcal{B} \subseteq \mathcal{A}_\mathcal{C}$ on $\mathcal{C}$ is a CMC $hide_\mathcal{B}(\mathcal{C})$ with signature $(S_\mathcal{C}, s_0^\mathcal{C}, \mathcal{A}_\mathcal{C} \setminus \mathcal{B}, L_\mathcal{C} \upharpoonright \mathcal{A}_\mathcal{C} \setminus \mathcal{B}, \varphi_\mathcal{C})$. Naturally, hiding preserves consistency.

Conjunction. Thankfully, conjunction on CMCs is far simpler than on probabilistic contracts, as constraints are closed under conjunction. Nevertheless, pruning is required after the construction of the conjunction in order to eliminate any inconsistent states.

Definition 2.34 *The pseudo-conjunction of two CMCs $\mathcal{C}$ and $\mathcal{D}$ is a CMC $\mathcal{C} \mathbin{\tilde{\wedge}} \mathcal{D}$ with signature $(S, s_0, \mathcal{A}, L, \varphi)$, where:*

- $S = S_\mathcal{C} \times S_\mathcal{D}$
- $s_0 = (s_0^\mathcal{C}, s_0^\mathcal{D})$
- $\mathcal{A} = \mathcal{A}_\mathcal{C} \cup \mathcal{A}_\mathcal{D}$
- $L(s_c, s_d) = L_\mathcal{C}(s_c) \upharpoonright \mathcal{A} \cap L_\mathcal{D}(s_d) \upharpoonright \mathcal{A}$[2]
- $f_{c \otimes d} \in \varphi(s_c, s_d) \iff f_c \in \varphi_\mathcal{C}(s_c)$ *and* $f_d \in \varphi_\mathcal{D}(s_d)$, *where $f_{c\otimes d}$ is any distribution on $S_\mathcal{C} \times S_\mathcal{D}$ satisfying*

$$f_c(s'_c) = \sum_{s'_d} f_{c\otimes d}(s'_c, s'_d) \qquad \text{and} \qquad f_d(s'_d) = \sum_{s'_c} f_{c\otimes d}(s'_c, s'_d).$$

[2]Write $\mathcal{L}_\mathcal{C}(s_c) \upharpoonright \mathcal{A}$ for $\{W \subseteq \mathcal{A} : W \upharpoonright \mathcal{A}_\mathcal{C} \in \mathcal{L}_\mathcal{C}(s_c)\}$.

Definition 2.35 *The conjunction of two CMCs $\mathcal{C}$ and $\mathcal{D}$ is a CMC $\mathcal{C} \wedge \mathcal{D}$ obtained from $\mathcal{C} \,\widetilde{\wedge}\, \mathcal{D}$ by iteratively applying the β operator until a fixed-point is reached. Thus $\mathcal{C} \wedge \mathcal{D}$ is defined only if the initial state $(s_0^{\mathcal{C}}, s_0^{\mathcal{D}})$ is not pruned from the pseudo-conjunction.*

Lemma 2.36 *Conjunction on CMCs yields the greatest lower bound with respect to weak refinement.*

Parallel composition. Composition can become difficult when there are many different aspects to be considered, such as the combination of probabilistic transitions and the synchronisation of actions. Thus, the authors of [14] use the principle of separation of concerns in defining parallel composition on CMCs. To this end, parallel composition is only defined on CMCs with disjoint action sets. As we shall see, there is no loss of generality from this assumption.

Definition 2.37 *The parallel composition of CMCs $\mathcal{C}$ and $\mathcal{D}$ with disjoint action sets is a CMC $\mathcal{C} \,||\, \mathcal{D}$ with signature $(S, s_0, \mathcal{A}, L, \varphi)$, where:*

- $S = S_{\mathcal{C}} \times S_{\mathcal{D}}$
- $s_0 = (s_0^{\mathcal{C}}, s_0^{\mathcal{D}})$
- $\mathcal{A} = \mathcal{A}_{\mathcal{C}} \cup \mathcal{A}_{\mathcal{D}}$
- $L(s_c, s_d) = \{l_c \cup l_d : l_c \in L_{\mathcal{C}}(s_c), l_d \in L_{\mathcal{D}}(s_d)\}$
- $f_{c||d} \in \varphi(s_c, s_d) \iff f_c \in \varphi_{\mathcal{C}}(s_c)$ *and* $f_d \in \varphi_{\mathcal{D}}(s_d)$, *where* $f_{c||d}(s'_c, s'_d) \triangleq f_c(s'_c) * f_d(s'_d)$.

As we would expect, parallel composition is a congruence with respect to (weak) refinement. Moreover, CMCs support the principle of independent implementability.

Lemma 2.38 *Let $\mathcal{C}$, $\mathcal{C}'$, $\mathcal{D}$ and $\mathcal{D}'$ be CMCs with $\mathcal{C} \preceq \mathcal{C}'$ and $\mathcal{D} \preceq \mathcal{D}'$. Then $\mathcal{C} \,||\, \mathcal{D} \preceq \mathcal{C}' \,||\, \mathcal{D}'$.*

Lemma 2.39 *Let $\mathcal{E}$ and $\mathcal{F}$ be DTMCs, and let $\mathcal{C}$ and $\mathcal{D}$ be CMCs. If $\mathcal{E} \models \mathcal{C}$ and $\mathcal{F} \models \mathcal{D}$, then $\mathcal{E} \,||\, \mathcal{F} \models \mathcal{C} \,||\, \mathcal{D}$.*

As previously remarked, in the case that CMCs $\mathcal{C}$ and $\mathcal{D}$ having non-disjoint alphabets need to be composed, there is a method that can be employed. First, perform a renaming so that the alphabets are disjoint, yielding CMCs $\mathcal{C}'$ and $\mathcal{D}'$ respectively. The parallel composition should now be performed on the resulting CMCs, yielding the CMC $\mathcal{C}' \,||\, \mathcal{D}'$. The conjunction of the resulting CMC $\mathcal{C}' \,||\, \mathcal{D}'$ should now be taken with a special CMC $\mathcal{M}$, known as a synchroniser, which reasserts the relationship between the disjoint actions. Afterwards, renaming can be utilised to restore the alphabet back to the original labellings.

Definition 2.40 *A synchroniser is a CMC $\mathcal{M}$ with a single state and a single probabilistic transition to itself with probability 1. The valuations of such a CMC can be used as constraints, imposing relations between the different actions in use. The conjunction of a synchroniser with a CMC will then restrict the behaviours in the CMC according to the valuations.*

For example, a synchroniser over alphabet $\{a, b\}$ with a single valuation $\{a, b\}$ asserts that $a = b$. In the conjunction of the synchroniser with a CMC, each state will have valuations containing both of a and b, or neither.

Lemma 2.41 *For CMCs $\mathcal{C}_1$, $\mathcal{C}_2$ and $\mathcal{C}_3$ with pairwise disjoint alphabets $\mathcal{A}_1$, $\mathcal{A}_2$ and $\mathcal{A}_3$ respectively, and synchronisers $\mathcal{M}_{1,2}$ and $\mathcal{M}_{1,2,3}$ over alphabets $\mathcal{A}_1 \cup \mathcal{A}_2$ and $\mathcal{A}_1 \cup \mathcal{A}_2 \cup \mathcal{A}_3$ respectively, the following equivalence holds:*

$$\{\mathcal{I} : \mathcal{I} \models (((\mathcal{C}_1 \,||\, \mathcal{C}_2) \wedge \mathcal{M}_{1,2}) \,||\, \mathcal{C}_3) \wedge \mathcal{M}_{1,2,3}\} = \{\mathcal{I} : \mathcal{I} \models (\mathcal{C}_1 \,||\, \mathcal{C}_2 \,||\, \mathcal{C}_3) \wedge \mathcal{M}_{1,2,3}\}.$$

Lemma 2.42 *Let $\mathcal{C}$, $\mathcal{C}'$, $\mathcal{D}$ and $\mathcal{D}'$ be CMCs with $\mathcal{C} \preceq \mathcal{C}'$ and $\mathcal{D} \preceq \mathcal{D}'$, and let $\mathcal{M}$ be a synchroniser. Then $(\mathcal{C} \,||\, \mathcal{D}) \wedge \mathcal{M} \preceq (\mathcal{C}' \,||\, \mathcal{D}') \wedge \mathcal{M}$.*

Although this form of parallel composition may seem quite cumbersome, it does provide the ability to formulate more complex synchronisation patterns than the usual provision from overlapping alphabet synchronisation.

Quotient. As for probabilistic contracts in Section 2.2.3, there has been no attempt at defining quotient for constraint Markov chains.

2.2.5 Modal Specifications

The final formalism that we will consider is that of modal specifications [65], which were introduced by Larsen in [55]. Modal specifications are have two important properties: all of the specification theory operators are defined on them and they are well-suited to handling issues of compositionality. Moreover, the definitions for composition are fairly simple.

Definition 2.43 *A* pseudo-modal specification *is a tuple* $(\mathcal{A}, must, may)$*, where:*

- $\mathcal{A}$ *is a finite set of actions*
- $must : \mathcal{A}^\star \to 2^{\mathcal{A}}$ *is a function yielding mandatorily enabled actions*
- $may : \mathcal{A}^\star \to 2^{\mathcal{A}}$ *is a function yielding permissibly enabled actions.*

Definition 2.44 *A* modal specification *is a pseudo-modal specification* $(\mathcal{A}, must, may)$ *in which* $must(w) \subseteq may(w)$ *for each* $w \in \mathcal{A}^\star$*.*

From hereon let $\mathcal{C} = (\mathcal{A}_\mathcal{C}, must_\mathcal{C}, may_\mathcal{C})$ and $\mathcal{D} = (\mathcal{A}_\mathcal{D}, must_\mathcal{D}, may_\mathcal{D})$ be two pseudo-modal specifications. The operators and relations we consider over pseudo-modal specifications often require that the action sets of the operands are equal. In fact, there is no problem if the action sets do not fully coincide, because we can perform alphabet equalisation, as per the following definition.

Definition 2.45 *The strong extension of a pseudo-modal specification* $\mathcal{C}$ *to* $\mathcal{A}$ *(providing* $\mathcal{A}_\mathcal{C} \subseteq \mathcal{A}$*), written* $\mathcal{C} \uparrow \mathcal{A}$*, is the pseudo-modal specification* $(\mathcal{A}, must, may)$*, where for each* $w \in \mathcal{A}^\star$*:*

- $must(w) = must_\mathcal{C}(w \restriction \mathcal{A}_\mathcal{C}) \cup (\mathcal{A} \setminus \mathcal{A}_\mathcal{C})$
- $may(w) = may_\mathcal{C}(w \restriction \mathcal{A}_\mathcal{C}) \cup (\mathcal{A} \setminus \mathcal{A}_\mathcal{C})$.

Definition 2.46 *The weak extension of a pseudo-modal specification* $\mathcal{C}$ *to* $\mathcal{A}$ *(providing* $\mathcal{A}_\mathcal{C} \subseteq \mathcal{A}$*), written* $\mathcal{C} \Uparrow \mathcal{A}$*, is the pseudo-modal specification equivalent to that for the strong extension, except that for each* $w \in \mathcal{A}^\star$*,* $must(w) = must_\mathcal{C}(w \restriction \mathcal{A}_\mathcal{C})$*.*

The choice of strong or weak equalisation of alphabets when composing systems is actually dependent on the operator under consideration, rather than a choice of a designer. The different forms of extension preserve different properties with respect to the composition operators.

Consistency. The notion of consistency for a pseudo-modal specification relates to whether it has any models or not i.e., whether it is implementable.

Definition 2.47 *A (strong) implementation for a pseudo-modal specification* $\mathcal{C}$ *is a language* $\mathcal{I} \subseteq \mathcal{A}^\star$ *(in which* $\mathcal{A}_\mathcal{C} \subseteq \mathcal{A}$*) such that for each* $w \in \mathcal{I}$*,* $must_{\mathcal{C}\uparrow\mathcal{A}}(w) \subseteq \{a \in \mathcal{A} : w.a \in \mathcal{I}\} \subseteq may_{\mathcal{C}\uparrow\mathcal{A}}(w)$*. We write* $\mathcal{I} \models \mathcal{C}$ *for* $\mathcal{I}$ *is a (strong) model/implementation of* $\mathcal{C}$*.*

A weak implementation is practically the same as a strong implementation, except that the weak extension $\mathcal{C} \Uparrow \mathcal{A}$ *is considered in place of the strong extension* $\mathcal{C} \uparrow \mathcal{A}$*. We write* $\mathcal{I} \models_w \mathcal{C}$ *for* $\mathcal{I}$ *is a weak implementation of* $\mathcal{C}$*.*

Definition 2.48 *A pseudo-modal specification* $\mathcal{C}$ *is said to be consistent (i.e. implementable) if, and only if, there exists a model* $\mathcal{I}$ *such that* $\mathcal{I} \models \mathcal{C}$*. A similar result holds for weak consistency.*

In the case that a pseudo-modal specification $\mathcal{C}$ is consistent, there is an operator ρ that can be applied such that $\rho(\mathcal{C})$ yields a modal specification over the alphabet $\mathcal{A}_\mathcal{C}$ admitting the same set of implementations as $\mathcal{C}$. The construction of ρ is contained in [66].

Parallel composition. The parallel composition operator has a straightforward definition, and moreover preserves consistency, as well as being a congruence with respect to refinement, as we shall see later.

Definition 2.49 *The parallel composition of two consistent modal specifications $\mathcal{C}$ and $\mathcal{D}$ is another consistent modal specification $\mathcal{C} \mid\mid \mathcal{D} = (\mathcal{A}, must_{\mathcal{C}\uparrow\mathcal{A}} \cap must_{\mathcal{D}\uparrow\mathcal{A}}, may_{\mathcal{C}\uparrow\mathcal{A}} \cap may_{\mathcal{D}\uparrow\mathcal{A}})$, where $\mathcal{A} = \mathcal{A}_\mathcal{C} \cup \mathcal{A}_\mathcal{D}$.*

Refinement. The refinement relation is definable on pseudo-modal specifications, and is a pre-order that reduces to set inclusion.

Definition 2.50 *Pseudo-modal specification $\mathcal{C}$ is (strongly) refined by pseudo-modal specification $\mathcal{D}$, written $\mathcal{C} \sqsubseteq \mathcal{D}$, iff $\mathcal{A}_\mathcal{C} \subseteq \mathcal{A}_\mathcal{D}$ and for each $w \in \mathcal{L}(\mathcal{D})$:*

- $may_{\mathcal{C}\uparrow\mathcal{A}_\mathcal{D}}(w) \supseteq may_\mathcal{D}(w)$
- $must_{\mathcal{C}\uparrow\mathcal{A}_\mathcal{D}}(w) \subseteq must_\mathcal{D}(w)$.

Definition 2.51 *Pseudo-modal specification $\mathcal{C}$ is weakly refined by pseudo-modal specification $\mathcal{D}$, written $\mathcal{C} \preceq \mathcal{D}$, iff $\mathcal{A}_\mathcal{C} \subseteq \mathcal{A}_\mathcal{D}$ and for each $w \in \mathcal{L}(\mathcal{D})$:*

- $may_{\mathcal{C}\Uparrow\mathcal{A}_\mathcal{D}}(w) \supseteq may_\mathcal{D}(w)$
- $must_{\mathcal{C}\Uparrow\mathcal{A}_\mathcal{D}}(w) \subseteq must_\mathcal{D}(w)$.

It turns out that refinement respects model inclusion and a number of compositional properties with respect to parallel composition.

Lemma 2.52 *For pseudo-modal specifications $\mathcal{C}$ and $\mathcal{D}$, $\mathcal{C} \sqsubseteq \mathcal{D}$ iff $\{\mathcal{I} : \mathcal{I} \models \mathcal{D}\} \subseteq \{\mathcal{I} : \mathcal{I} \models \mathcal{C}\}$. An analogous result holds for weak refinement. Thus refinement entails model inclusion.*

Lemma 2.53 *For modal specifications $\mathcal{C}, \mathcal{C}', \mathcal{D}$ and $\mathcal{D}'$, if $\mathcal{C} \sqsubseteq \mathcal{C}'$ and $\mathcal{D} \sqsubseteq \mathcal{D}'$, then $\mathcal{C} \mid\mid \mathcal{D} \sqsubseteq \mathcal{C}' \mid\mid \mathcal{D}'$. Thus parallel composition is a congruence with respect to strong refinement.*

Lemma 2.54 *For models $\mathcal{I}$ and $\mathcal{J}$, and modal specifications $\mathcal{C}$ and $\mathcal{D}$, if $\mathcal{I} \models \mathcal{C}$ and $\mathcal{J} \models \mathcal{D}$, then $\mathcal{I} \cap \mathcal{J} \models \mathcal{C} \mid\mid \mathcal{D}$. This result shows that parallel composition supports independent implementability.*

For the last two lemmas, it is not the case that the results hold for weak refinement.

Hiding. The hiding abstraction operator on pseudo-modal specifications preserves consistency.

Definition 2.55 *The hiding of $\mathcal{B} \subseteq \mathcal{A}_\mathcal{C}$ for pseudo-modal specification $\mathcal{C}$ is a pseudo-modal specification $hide_\mathcal{B}(\mathcal{C}) = (\mathcal{A}_\mathcal{C} \setminus \mathcal{B}, must_\mathcal{C} \upharpoonright \mathcal{A}_\mathcal{C} \setminus \mathcal{B}, may_\mathcal{C} \upharpoonright \mathcal{A}_\mathcal{C} \setminus \mathcal{B})$.*

Conjunction. The construction of the conjunction for two modal specifications is based on the weak extension, rather than the strong extension which was used for parallel composition. This is because of the different ways that these operators should behave.

Definition 2.56 *The conjunction of two modal specifications $\mathcal{C}$ and $\mathcal{D}$ is a modal specification $\mathcal{C} \wedge \mathcal{D}$ defined as $\rho(\mathcal{A}, must_{\mathcal{A}_\mathcal{C}\Uparrow\mathcal{A}} \cup must_{\mathcal{A}_\mathcal{D}\Uparrow\mathcal{A}}, may_{\mathcal{A}_\mathcal{C}\Uparrow\mathcal{A}} \cap may_{\mathcal{A}_\mathcal{D}\Uparrow\mathcal{A}})$, where $\mathcal{A} = \mathcal{A}_\mathcal{C} \cup \mathcal{A}_\mathcal{D}$. Note that the ρ operator needs to be applied to the construction in order to obtain a modal specification rather than a pseudo-modal specification.*

Lemma 2.57 *For an implementation $\mathcal{I}$, and modal specifications $\mathcal{C}$ and $\mathcal{D}$ the following holds: $\mathcal{I} \models_w \mathcal{C} \wedge \mathcal{D}$ iff $\mathcal{I} \models_w \mathcal{C}$ and $\mathcal{I} \models_w \mathcal{D}$. It is not the case that an analogous result holds for strong refinement.*

Lemma 2.58 *Conjunction of modal specifications yields the greatest lower bound with respect to weak refinement.*

Quotient. The quotient operator on two modal specifications also has an easy and intuitive construction method. Unlike parallel composition and conjunction, quotient combines both the weak and strong extensions.

Definition 2.59 *The quotient of modal specifications $\mathcal{C}$ and $\mathcal{D}$, written $\mathcal{C} \setminus \mathcal{D}$ is a modal specification obtained by applying the ρ operator to the pseudo-modal specification $(\mathcal{A}, must, may)$, where:*

- $\mathcal{A} = \mathcal{A}_{\mathcal{C} \Uparrow \mathcal{A}} \cup \mathcal{A}_{\mathcal{D} \uparrow \mathcal{A}}$
- $a \in may(w) \cap must(w)$ *providing* $a \in must_{\mathcal{C} \Uparrow \mathcal{A}}(w)$ *and* $a \in must_{\mathcal{D} \uparrow \mathcal{A}}(w)$
- $a \in must(w) \setminus may(w)$ *providing* $a \in must_{\mathcal{C} \Uparrow \mathcal{A}}(w)$ *and* $a \notin must_{\mathcal{D} \uparrow \mathcal{A}}(w)$
- $a \in may(w) \setminus must(w)$ *providing* $a \in may_{\mathcal{C} \Uparrow \mathcal{A}}(w)$ *and* $a \notin must_{\mathcal{C} \Uparrow \mathcal{A}}(w)$
- $a \in may(w) \setminus must(w)$ *providing* $a \notin may_{\mathcal{C} \Uparrow \mathcal{A}}(w)$ *and* $a \notin may_{\mathcal{D} \uparrow \mathcal{A}}(w)$
- $a \notin may(w) \cup must(w)$ *providing* $a \notin may_{\mathcal{C} \Uparrow \mathcal{A}}(w)$ *and* $a \in may_{\mathcal{D} \uparrow \mathcal{A}}(w)$.

The definition of conjunction provided above coincides with the definition for the adjoint of parallel composition, as the following lemma demonstrates.

Lemma 2.60 *For modal specifications $\mathcal{C}$, $\mathcal{D}$ and $\mathcal{E}$ the following holds: $\mathcal{C} \setminus \mathcal{D} \sqsubseteq \mathcal{E}$ iff $\mathcal{C} \sqsubseteq \mathcal{D} \parallel \mathcal{E}$.*

There is also a relationship between the quotient of modal specifications and implementations.

Lemma 2.61 *Let $\mathcal{C}$ and $\mathcal{D}$ be modal specifications, and $\mathcal{I}$ a prefix closed language over $\mathcal{A}_{\mathcal{I}}$ such that $\mathcal{A}_{\mathcal{D}} \subseteq \mathcal{A}_{\mathcal{C}} \subseteq \mathcal{A}_{\mathcal{I}}$. Then $\mathcal{I} \models \mathcal{C} \setminus \mathcal{D}$ iff $\forall \mathcal{J} \cdot \mathcal{J} \models \mathcal{D} \Rightarrow \mathcal{I} \cap \mathcal{J} \models \mathcal{C}$.*

This final result has very close relations to assume-guarantee reasoning for modal specifications. An assume-guarantee contract is a pair of prefix closed languages $(\mathcal{A}, \mathcal{G})$ such that $\mathcal{A} \subseteq \mathcal{G}$. An implementation $\mathcal{I}$ satisfies the contract if for any trace t of $\mathcal{I}$, and for any prefix $t'.\langle a \rangle$ of t: if $t' \in \mathcal{A}$ then $t'.\langle a \rangle \in \mathcal{G}$. Using quotient, one can synthesise a modal specification $\mathcal{G} \setminus \mathcal{A}$, whose implementations all satisfy the contract.

2.2.6 Conclusion

In this section we have surveyed a range of component modelling and specification formalisms. These included non-quantitative component models (I/O automata and interface automata); recently proposed probabilistic extensions of component specification theories (interactive Markov chains and constraint Markov chains); and (non-quantitative) modal specifications. The purpose of the exercise was to classify these formalisms according to the key operations and relations that they possess for supporting compositional development, with the view to identify which of these formalisms are amenable to a quantitative specification theory for connectors that is the goal of CONNECT. A table summarising how the formalisms compare is included below.

Model	Parallel	Refinement	Hiding	Conjunction	Quotient
I/O automata	Yes	Simulation / trace	Yes	Yes	No
Interface automata	Partial	Alternating simulation	Yes	Yes	Partial
Probabilistic I/O automata	Yes	Probabilistic simulation	Yes	No	No
Interactive Markov chains	Yes	Probabilistic simulation	Yes	Partial	No
Constraint Markov chains	Yes	Probabilistic simulation	Yes	Yes	No
Modal specifications	Yes	Trace	Yes	Yes	Yes

It can be seen from the above table that modal specifications have powerful support for compositionality. However, they lack concrete quantitative component models and tool support.

Interface and I/O automata share many similarities, and they both support the notion of a component model that is amenable to the modelling of connectors. A clear advantage of interface automata is that they are not input-enabled, a feature particularly useful in the context of heterogeneous networked systems that

arise in CONNECT. Interface automata also have tool support, in the form of Ticc. In technical terms, the main difference between the two is that I/O automata have refinement defined in terms of simulation or trace containment, whereas for interface automata refinement is based on alternating simulation. The latter supports the notion of substitutivity for components, irrespective of the environment, and is thus more suitable for the heterogeneity of CONNECT. However, a drawback is the complexity of the game-based approach employed by interface automata.

The probabilistic specification theories that we surveyed - probabilistic contracts for IMCs and constraint Markov chains - offer technical arguments that a compositional theory for quantitative (intercative) component behaviours is plausible. For CMCs, parallel composition is not as intuitive as for IMCs because of the concept of a synchroniser for adjusting the state valuations. On the other hand, IMCs have to impose constraints on the definedness of certain compositional operators, such as conjunction, to ensure that the resulting model contains only probabilistic intervals. This is not an issue for CMCs, because the combination of arbitrary constraints is itself a constraint. On the other hand, probabilistic contracts for IMCs appear more amenable to automated verification, since intervals are easier to deal with.

Following on from the above analysis, it is our intention to produce a quantitative specification theory based on a *probabilistic extension of interface automata*. This will be achieved by exploiting the good practices adopted by the frameworks for IMCs and CMCs, which already provide a specification theory for probabilistic systems, but do not as yet support a component-based specification theory. We give the details of our proposal in Section 3.2.

3 Towards a CONNECTor Algebra

Based on the conclusions from the survey in Chapter 2, we intend to develop a connector algebra based on probabilistic interface automata. Interface automata were chosen by the consortium for three key reasons: (i) the concept of the interface is necessary for the modelling of interactions between (sub)systems; (ii) it is not realistic for components to be fully input-enabled in heterogeneous systems; and (iii) existence of tool support (Ticc). This chapter explains how probabilistic interface automata will support us in developing a theory of connectors specifically for CONNECT.

Objectives. We are pursuing this work in two strands described below, which will be subsequently merged.

- **Component framework.** We will develop a framework for components in terms of a quantitative specification theory based on probabilistic interface automata. The framework will support the operations and relations specified in the opening of Section 2.2. CONNECTors will arise, and be characterised in due course, as restricted classes of component, depending on their complexity (finite state, memory, etc).

  Probabilistic interface automata will naturally be amenable to quantitative analysis. We will introduce assume-guarantee reasoning for probabilistic interface automata with respect to ω-regular properties (which includes both safety and liveness properties). This is an extension of the compositional verification techniques developed for probabilistic automata in Chapter 4, which can be used, e.g., to establish key properties of a connector given the description of its environment. To achieve this, we will formulate a collection of proof rules for deducing ω-regular guarantees of systems composed with respect to the different compositional operators defined on the specification theory.

- **Connector algebra.** Within the component framework, we will precisely characterise those components that can be perceived as connectors. To this end, we propose developing an algebra of connectors whose terms have semantics given as probabilistic interface automata from the specification theory. Primitive terms are representable explicitly by probabilistic interface automata, whereas compound terms are obtained by composing the probabilistic interface automata of the operands.

Outline. In this chapter we report on three key tasks we have been working on towards meeting the objectives.

- Besides studying automata-based models for connectors, we developed a preliminary connector algebra based on interface automata, in order to guide us towards a concrete syntax for our connector algebra. The purpose of this algebra was to capture the functional behaviour of connectors that act as *protocol mediators*. Section 3.1 provides all of the technical details, including the syntax, semantics, forms of equivalence and compositional properties of the algebra. An example in the area of universal instant messaging is illustrated and the algebra is used to model a connector for this environment. Details of an implementation for our algebra based on the Ticc tool is discussed. We also comment upon the limitations of our preliminary algebra, to guide the development of the target formalism.

- Finally, Section 3.2 provides a detailed roadmap for the development of our target connector modelling formalism based on a quantitative specification theory. A crucial aspect is the explicit consideration of assumptions and guarantees as constraints on the sequence of actions performed on the interface, thus generalising interface automata, in Section 3.2.2. This theory is in a declarative style and dircetly links to our compositional assume-guarantee work in Chapter 4. We discuss the range of properties that will be supported. In conclusion, we relate the specification theory to the preliminary algebra described in Section 3.1.

Conclusions. In this chapter we contribute two key results (i) a preliminary algebra based on (classical) interface automata for capturing the functional behaviour of connectors; and (ii) our proposal for a

quantitative specification theory for probabilistic interface automata for capturing the functional and non-functional properties of connectors. The latter also incorporates assume-guarantee specifications and its declarative format allows us to interchange interface automata with their assume-guarantee specifications when we wish to perform verification.

3.1 Preliminary Algebra for Mediating Protocol Mismatches

Following on from our survey of component models in Section 2.2, we document details of an attempt at defining a preliminary algebra for connectors. As discussed within Work Package WP3 and as detailed in the CONNECT Deliverable D3.2 [18], and in some recent publications related to it [46, 70], a possible approach to protocol mediation involves the categorisation of recurring protocol mismatches that must be solved by means of *mediator patterns*. For each type of mismatch, a pattern can be defined as a solution to the interaction incompatibility. Clearly, a catalogue of such problems and their related solutions would not solve all possible mismatches, but combining multiple sub-solutions should facilitate the coverage of more mismatches.

Inspired by the set of basic mediator patterns described in the CONNECT Deliverable D3.2 [18] (see also the work described in [70]), we present a high-level algebra that reasons about protocol mismatches. Solutions to basic mismatches are modelled as primitives of the algebra, while complex mismatches can be solved by combining suitable primitives in a variety of ways. The semantics of the algebra is given in terms of interface automata of de Alfaro and Henzinger [23, 25], which are described in Section 2.2.2. Composition of terms in the algebra reduces to composition of the underlying interface automata, a number of which are already defined, although we will present a specific composition that we conjecture is necessary for a meaningful connector algebra. We just state the new definitions in this section, and refer to Section 2.2.2 for the comprehensive treatment.

In the CONNECT Deliverable D2.1 [17], we surveyed several formalisms against a number of dimensions deemed to be of importance to CONNECT. These were compositionality, incrementality, scalability, compositional reasoning, reusability, evolution, ability to express and reason about non-functional properties, and the existence of a specialised notation supported by automated tools for architectural analysis. The overall concensus was that none of the surveyed formalisms were suitable for modelling connectors in CONNECT. Consequently, we performed the survey in Section 2.2 looking at general component models. From this survey, we concluded that interface automata were the most suitable formalism for modelling connectors. This was based on the fact that it seems plausible to extend interface automata to support reasoning on non-functional properties, in addition to the standard compositionality results implying reuse and evolution of connectors. Interface automata are by no means complete in satisfying the properties we require of a connector algebra, but they do give us a good starting point and seem extensible enough to gain a good coverage of the dimensions of interest.

The remainder of this section documents our preliminary algebra. In Section 3.1.1, we describe the syntax used to build terms of our algebra. Section 3.1.2 formalizes the interface-automata-based semantics of terms of the algebra, while Section 3.1.3 discusses issues of equivalence and compositionality. By means of an example concerning instant messaging services, Section 3.1.4 shows how to model a connector as a compound term of the algebra. In Section 3.1.5, we give an overview of an implementation of our algebra by a software modelling tool called `IAAnalzyer`, which allows us to reflect upon the expressiveness and usability of our preliminary algebraic formalization.

3.1.1 Syntax

In the CONNECT Deliverable D3.2 [18], and in an its related publication [70], the authors attempt to characterise mismatches between functionally equivalent yet behaviourally different protocols that wish to communicate. For each identified type of communication discrepancy they provide a mediating connector that can handle and resolve the mismatch. This is a high-level approach to analysing and addressing interoperability issues. Similarly, CONNECT focuses on high-level modelling of connectors at the application and middleware layers. For the purposes of our algebra, we are interested in exchanging high-level structured messages between components by abstracting from the exchanged messages' content (i.e., data interpretation).

Following this insight, we present an algebra whose syntax is geared towards modelling mismatches. Basic mismatch solutions are primitives of our algebra, while complex mismatch solutions can be decomposed into combinations of basic ones. Therefore, whenever possible, an algebraic connector for a complex mismatch is obtained by composing primitive connectors. Clearly, we should validate the completeness of our algebra with respect to a meaningful set of possible protocol mismatches (and related patterns) that often occur in the practice. In this way, we would empirically ensure that the decomposition of a complex mismatch solution, into combinations of basic solutions, is always possible. However, the connector algebra presented in this deliverable should be considered as a preliminary version and, so far, we have been more concentrated in defining it by basing on the mismatch categorisation given in the CONNECT Deliverable D3.2 [18] (and in [70]) rather than on rigorously validating it. Thus we let the algebra validation, beyond the mismatches identified by the work of Work Package WP3, as future work.

We refer to the external systems to be connected as components, and model them by interface automata. It is assumed that all components have disjoint action sets[1]. As connectors will have semantics given in terms of interface automata, it follows that connectors are a restricted class of components.

Action sets associated with interface automata are treated as sets of named ports that can send (resp., receive) data depending on whether they are of output (resp., input) type. For this preliminary algebra we do not interpret data sent through ports, so we'll use the words data, message and port interchangeably. Thus, a message a means data exchanged through the port a. Consequently, we focus on the allowed sequences of exchanged data.

From hereon let $\mathcal{A}$ be a global set of message ports. The primitives of the connector algebra $\mathcal{AP}(\mathcal{A})$ corresponding to possible solutions to the mismatches described in the CONNECT Deliverable D3.2 [18], and in [70], are described below.

1. **Extra send.** This first mismatch considers a component that wishes to send a message a not expected by any other component. Such a mismatch may be resolved by introducing a consumer that swallows the superfluous message. We model this by a parameterised primitive $Cons(a)$.

2. **Missing send.** This mismatch describes the case in which a component expects a message a that is not sent by another component. A mismatch of this type may be resolved by introducing a producer that generates the required message. This may be modelled by a parameterised primitive $Prod(a)$.

3. **Signature mismatch.** There are occasions when a message to be exchanged between two components is functionally compatible yet syntactically inconsistent. In the case of CONNECT, the functional equivalence of the messages a and b is assumed to be specified in an ontology. Such a mismatch may be resolved by means of a translating primitive $Trans(a, b)$ that accepts message a as input and produces message b as output.

4. **Split message mismatch.** A component may expect to receive a message a as a sequence of fragments of a. If message a can be decomposed into $a_1, \ldots, a_n$, then the mismatch may be resolved with a primitive $Split(a, [a_1, \ldots, a_n])$ which accepts message a as input and offers $a_1, \ldots, a_n$ as output in that order.

5. **Merge message mismatch.** Similar to the previous case, some components expect to receive a single message a in place of a fragmented version of a. If messages $a_1, \ldots, a_n$ can be composed into a, then the mismatch may be resolved with a primitive $Merge([a_1, \ldots, a_n], a)$ which accepts messages $a_1, \ldots, a_n$ as input in that order, and generates a as output.

6. **Ordering mismatch.** A component can expect to receive messages in an order different from the order used by the sending component. The mismatch can be resolved by introducing an ordering primitive $Order([a_1, \ldots, a_n], \pi, [a'_1, \ldots, a'_n])$, where π is a permutation of $\{1, \ldots, n\}$. The primitive accepts inputs from one component in the order $a_1, \ldots, a_n$, and produces outputs for the other in the order $a'_{\pi(1)}, \ldots, a'_{\pi(n)}$. Note that port a_i is related to port a'_i.

[1] Under the aegis of CONNECT, equivalence of actions is assumed to be specified in an ontology. This allows us to assume disjoint component actions.

Besides the mismatch primitives above, a further primitive is required to force the algebra to work in a sensible way. The primitive does not perform any interactions, so is fittingly called $NoOp$. Equipped with all of the basic primitives, a term s of the algebra is given by the following grammar:

$$s ::= s \odot s \mid s + s \mid s \wedge s \mid s \setminus s \mid s^{\perp} \mid (s) \mid p$$

$$p ::= NoOp \mid Cons(a) \mid Prod(a) \mid Trans(a,b) \mid Split(a,[a_1,\ldots,a_n]) \mid Merge([a_1,\ldots,a_n],a) \mid Order([a_1,\ldots,a_n],\pi,[a'_1,\ldots,a'_n])$$

where $a, a_i, a'_i, b \in \mathcal{A}$ and π is a permutation of $\{1,\ldots,n\}$. The symbols $\odot$, $+$, $\wedge$, and $\setminus$ are binary operators called *plugging*, *alternation*, *conjunction* and *quotienting* respectively, and $^{\perp}$ is a unary operator called *inversion*.

3.1.2 Semantics

For the purposes of this section, we recall from Section 2.2.2 and in particular Definition 2.13 that an interface automaton $\mathcal{C}$ is a tuple $(S_{\mathcal{C}}, s_0^{\mathcal{C}}, \mathcal{A}_I^{\mathcal{C}}, \mathcal{A}_O^{\mathcal{C}}, \mathcal{A}_H^{\mathcal{C}}, \longrightarrow_{\mathcal{C}})$. For the purposes of our preliminary algebra, we restrict to the case of deterministic interface automata with no hidden actions.

The semantics of the algebra $\mathcal{AP}(\mathcal{A})$ is given in terms of a function $[\![\cdot]\!] : \mathcal{AP}(A) \rightarrow \mathcal{IA} \cup \{Err\}$, where $\mathcal{IA}$ is the universal set of interface automata and Err represents the undefined interface automaton. For any term s, the denotation $[\![s]\!]$ is defined inductively.

First and foremost, if s is a primitive, then $[\![s]\!]$ is the corresponding interface automaton defined in Figure 3.1, providing the parameters are well-defined (otherwise the semantics of the primitive is taken to be Err). The parameters of each primitive are single or lists of uninterpreted message ports of $\mathcal{A}$. Lists must have finite length, and message ports in the parameters of a primitive must be pairwise disjoint. In the case of $Merge$ and $Split$, we require that a and $a_1 \ldots a_n$ are equated in the ontology. Furthermore, for the case of the $Order$ primitive, we require that both lists have the same length. In Figure 3.1, we make use of essentially the same graphical notation as the one used by de Alfaro and Henzinger [23, 25].

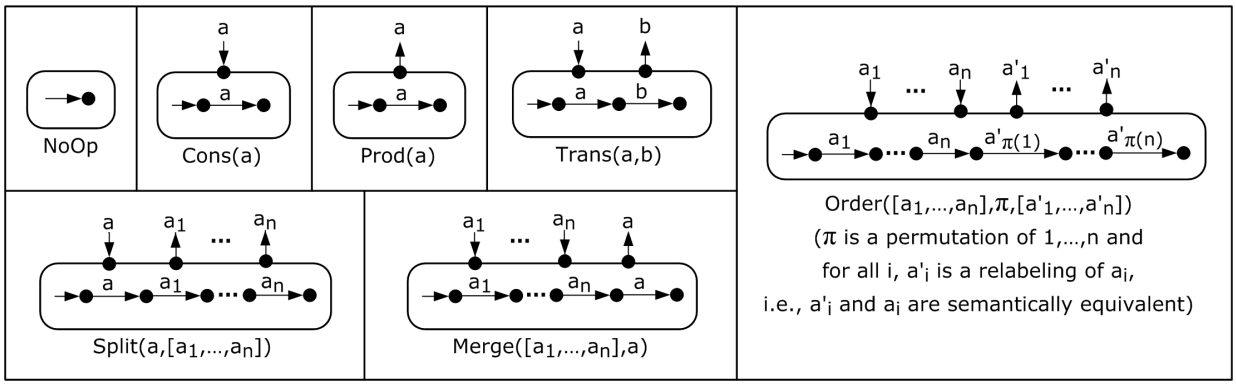

Figure 3.1: Semantics of the primitives.

If s is a compound term (i.e., consists of operators), then $[\![s]\!]$ is given by the mappings below. However, an informal description is in order first. An operator on terms of the algebra induces a behaviour on the behaviours of the operands. The operator $\odot$ connects terms of the algebra on common message ports; this is equivalent to plugging the corresponding interface automata into each other, or synchronising them. On the other hand, the operator $+$ behaves like alternation in regular expressions; a connector defined in terms of $+$ behaves like one of its operands. The operators $\wedge$ and $\setminus$ are both defined in Section 2.2.2. Finally, $^{\perp}$ acts like the inverse of its operand by interchanging inputs and outputs.

- $s = t \odot u$. If either of $[\![t]\!]$ or $[\![u]\!]$ is equal to Err, then $[\![s]\!] = Err$. Alternatively, if $[\![t]\!]$ and $[\![u]\!]$ are not composable or $[\![t]\!] \,||\, [\![u]\!]$ is not defined, then $[\![s]\!] = Err$. Otherwise, $[\![s]\!] = [\![t]\!] \,||\, [\![u]\!]$, where $||$ denotes the parallel composition of interface automata (see Section 2.2.2 for its formal definition).

- $s = t + u$. If either of $[\![t]\!]$ or $[\![u]\!]$ is equal to Err, then $[\![s]\!] = Err$. Otherwise, $[\![t]\!]$ and $[\![u]\!]$ are interface automata $\mathcal{C}$ and $\mathcal{D}$. If $\mathcal{A}_I^{\mathcal{C}} \cup \mathcal{A}_I^{\mathcal{D}}$ and $\mathcal{A}_O^{\mathcal{C}} \cup \mathcal{A}_O^{\mathcal{D}}$ are not disjoint, then the alternation is not defined and $[\![s]\!] = Err$. For the case when the action sets do agree, $[\![s]\!] = Determinise(\mathcal{C} + \mathcal{D})$, where $\mathcal{C} + \mathcal{D}$ is defined to be the interface automaton $(S, s_0, \mathcal{A}_I^{\mathcal{C}} \cup \mathcal{A}_I^{\mathcal{D}}, \mathcal{A}_O^{\mathcal{C}} \cup \mathcal{A}_O^{\mathcal{D}}, \rightarrow)$ such that:

  - $S = S_{\mathcal{C}} \dot{\cup} S_{\mathcal{D}} \dot{\cup} \{s_0\}$ i.e., $S_{\mathcal{C}}$, $S_{\mathcal{D}}$ and $\{s_0\}$ are pairwise disjoint
  - $\rightarrow = \rightarrow_{\mathcal{C}} \cup \rightarrow_{\mathcal{D}} \cup \{(s_0, a, s) : s_0^{\mathcal{C}} \xrightarrow{a}_{\mathcal{C}} s\} \cup \{(s_0, a, s) : s_0^{\mathcal{D}} \xrightarrow{a}_{\mathcal{D}} s\}$.

  $Determinise(\mathcal{C} + \mathcal{D})$ is the deterministic interface automaton equivalent to the possibly non-deterministic interface automaton $\mathcal{C} + \mathcal{D}$. Thus, $Determinise$ is a function on interface automata which implements a suitable variant of the algorithm described in [45], that determinises a finite state automaton.

- $s = t \wedge u$. If either of $[\![t]\!]$ or $[\![u]\!]$ is equal to Err, then $[\![s]\!] = Err$. Otherwise, $[\![t]\!]$ and $[\![u]\!]$ are interface automata $\mathcal{C}$ and $\mathcal{D}$. If $\mathcal{C}$ and $\mathcal{D}$ are not compatible for conjunction, then $[\![s]\!] = Err$. Otherwise, $[\![s]\!] = \mathcal{C} \wedge \mathcal{D}$ as defined in Definition 2.19.

- $s = t \setminus u$. Based on [12], it follows that quotienting is a derived operator of the algebra. Thus, $[\![s]\!] = [\![(t^{\perp} \odot u)^{\perp}]\!]$.

- $s = t^{\perp}$. If $[\![t]\!] = Err$, then $[\![s]\!] = Err$. Otherwise, $[\![s]\!]$ is equal to $[\![t]\!]$ with the input and output sets exchanged in the signature of $[\![t]\!]$.

- $s = (t)$. Simply $[\![s]\!] = [\![t]\!]$.

3.1.3 Equivalence and Compositional Properties

The operators $\odot$, $+$, $\wedge$, $\setminus$ and $^{\perp}$ satisfy a number of axioms, as we briefly elaborate below.

1. Plugging $\odot$ is commutative and associative, but is not idempotent. It has an identity element $NoOp$, so $(\mathcal{AP}(\mathcal{A}),\odot,NoOp)$ is a commutative monoid (i.e., an abelian semigroup with an identity). Plugging does not distribute over $+$, $\wedge$ nor $\setminus$.

2. Alternation $+$ is commutative, associative, and idempotent. The identity of $+$ is also $NoOp$, so $(\mathcal{AP}(\mathcal{A}),+,NoOp)$ is a commutative monoid. Alternation does not distribute over $\odot$ nor $\setminus$, however it does distribute over $\wedge$.

3. Conjunction $\wedge$ is commutative, associative, and idempotent. The operator does not have an identity element in the algebra, and does not distribute over $\odot$ nor $\setminus$, but it does distribute over $+$.

4. Quotienting $\setminus$ is not associative, commutative, nor idempotent. $NoOp$ is a right-identity element for the operator. Quotienting does not left or right distribute over $\odot$, $+$, nor $\wedge$.

5. Inversion $^{\perp}$ distributes over $+$, but not $\odot$, $\wedge$ nor $\setminus$. Double inversion of a term is an identity function on that term.

As we remark in Section 2.2.2, a desirable property of a connector algebra is its ability to support notions of specification. Consequently, there should be a concept of refinement on terms, and indeed our algebra does support this (see Definition 3.1). We recall that a refinement relation, $\sqsubseteq$ (i.e., $P \sqsubseteq Q$ if, and only if, P is refined by Q), on interface automata is defined in terms of alternating simulation [4]. This relation is defined in such a way that refinement on interface automata is a congruence with respect to composition, hence there is a test for checking when a connector can be safely substituted with another.

Definition 3.1 *Let s and t be terms of the algebra, and let $\sqsubseteq$ be the alternating simulation refinement relation defined on interface automata $[\![s]\!]$ and $[\![t]\!]$. Term t refines term s, written as $s \lhd t$, iff the denotation of t refines the denotation of s at the semantic level or $[\![s]\!] = Err$. Formally, $s \lhd t \Leftrightarrow [\![s]\!] \sqsubseteq [\![t]\!] \vee [\![s]\!] = Err$.*

Establishing refinement on terms allows us to define equivalence. Semantic equality of terms is too strong for equivalence of connector behaviours, so we choose to express it in terms of the weaker refinement relation.

Definition 3.2 *Term s is said to be equivalent to term t, written as $s \equiv t$, if, and only if, $s \lhd t$ and $t \lhd s$.*

Our choice of equivalence seems most natural for connector behaviours, as it allows for seamless substitutivity. However, it is unfortunate that the equivalence is expressed in terms of the underlying semantics, rather than the syntax of the terms. Nevertheless, this correspondence with the interface automata semantics means that the algebra is trivially both *sound* and *complete* after equating all incompatible and undefined terms with Err.

Theorem 3.3 *Let $\hat{=}$ denote the equivalence of interface automata (i.e. mutual refinement). For any terms s and t it holds that $s \equiv t \Leftrightarrow ([\![s]\!] \hat{=} [\![t]\!]) \vee ([\![s]\!] = Err = [\![t]\!])$.*

Having defined equivalence in terms of the underlying interface-automata-based semantics (and noticed that, in this way, the algebra is trivially sound and complete), it is an easy consequence that our axiomatisation is correct. The reason for $\odot$ failing to be idempotent is closely related to the reason that $[\![s]\!] \,||\, [\![s]\!]$ is not defined in general, because of restrictions on composability.

The formal definition of the semantics, as well as the example of idempotence failing, has a notable consequence for the algebra. If s and t are well-formed terms whose semantics are not equal to Err, then it is not the case that the semantics of $s \odot t$, $s + t$, $s \wedge t$ and $s \setminus t$ are not equal to Err, because of the restrictions imposed by these operators. This seems undesirable, but it is a direct consequence of interface automata.

This shortcoming might seem unpalatable at first, but we do not believe it to be a problem; in fact, it is an advantage. In the context of CONNECT, we are concerned with generating connectors in a compositional manner. If we take two connectors, each of which is expressed by a term in the algebra, we can combine the two terms and observe if the outcome is equal to Err. If it is, then it follows that the two connectors do not work with each other. Thus the algebra allows for compositional reasoning (if we take an arbitrary number of connectors, we can always proceed pairwise until either we get Err or we successfully combine all the connectors to build a more complex one).

Another requirement of CONNECT is the ability of our algebra to serve as a baseline, within Work Package WP3, for automated connector synthesis. In our algebra setting, by simplifying matters, we can rephrase the connector synthesis problem faced by WP3 (so far, with the work described in the CONNECT Deliverable D3.2 [18]) as stated below. This is closely related to the quotienting operator defined in [12], but requires suitable modification to be applicable to the algebra.

CONNECTOR SYNTHESIS	
Instance:	Components $\mathcal{E}$ and $\mathcal{F}$ represented by interface automata.
Problem:	Find a term $x \in \mathcal{AP}(\mathcal{A})$ such that $inv(\mathcal{E}) \sqsubseteq \mathcal{F} \,||\, [\![x]\!]$ and $inv(\mathcal{F}) \sqsubseteq \mathcal{E} \,||\, [\![x]\!]$, where inv inverts input and output actions.

The connector synthesis problem, as expressed in our algebra setting, aims to find a connector x expressible in our algebra that can mediate interoperability incompatibilities between components represented by arbitrary interface automata. We require that (i) every interaction exhibited by $inv(\mathcal{F})$ is allowed

by $[\![x]\!] \mid\mid \mathcal{E}$, and (ii) every interaction exhibited by $inv(\mathcal{E})$ is permitted by $[\![x]\!] \mid\mid \mathcal{F}$. This allows us to formally characterise *interoperability* between components in our algebra. We recall that, let $\mathcal{F}$ be an interface automaton, $inv(\mathcal{F})$ is equal to $\mathcal{F}$ with the input and output sets exchanged in the signature of $\mathcal{F}$ (it is the inverted interface automaton). In other words, $inv(\mathcal{F})$ represents the trivial "legal" environment for $\mathcal{F}$. With the above algebra-oriented formulation of the connector synthesis problem, we state that a (compound) term x characterizes a "correct" mediator for two components, whose protocols are modelled by interface automata $\mathcal{E}$ and $\mathcal{F}$ respectively, if the parallel composition of $\mathcal{F}$ (resp., $\mathcal{E}$) with $[\![x]\!]$ is a legal environment for $\mathcal{E}$ (resp., $\mathcal{F}$). Note that, a possible legal environment should be a refinement of the trivial legal environment.

3.1.4 Example

To illustrate the efficacy with which our algebra can model and reason about protocol mediators, we introduce a simple yet challenging example of universal instant messaging inspired by [46], and use our algebra to model the required connector.

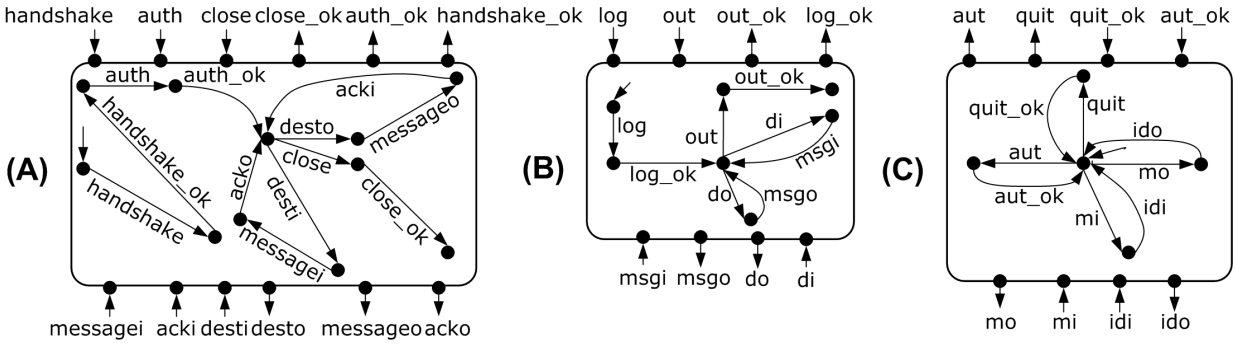

Figure 3.2: (A) MSN messaging service. (B) XMPP messaging service. (C) CFring client.

Fring[2] is an instant messaging program that allows one to exchange text messages between a predefined set of heterogeneous messaging services. At present, the service supports connection to the *MSN Messenger* and *XMPP Messenger* services, amongst many others, with the collection of supported services being static. This contrasts with the evolving world of CONNECT, where messaging services to be bridged should not be known *a priori*. We therefore propose a generalisation of Fring, let it be called *CFring*, where connectors between unknown messaging services are generated on-the-fly. This generalisation will allow us to express the full power of the algebra.

As the interface automaton in Figure 3.2.C shows, the CFring service provides only core functionalities for "abstract" authentication and message exchange. In particular, when receiving (resp., sending) a message mi (resp., mo), CFring expects to receive (resp., send) the identity idi (resp., ido) of the sender (resp., receiver) as well.

The behavioural models of MSN and XMPP, which are unknown to CFring, are expressed as interface automata in Figures 3.2.A and 3.2.B. In contrast to the behavioural model for CFring, both MSN and XMPP when receiving (resp., sending) a message expect to have provided (resp., provide) the identity of the sender (resp., receiver) first. Furthermore, unlike the others, MSN expects to receive (resp., send) an acknowledgement for the message sending (resp., receiving).

It is obvious that the MSN and XMPP services should both be able to interoperate with CFring, since they amount to supporting authentication and then message exchange. This requires "specialising" the CFring communication protocol in order to mediate the communication between the other messaging services. Note that this is far from trivial, especially if one wishes to rigorously characterise the achieved interoperability (e.g., for supporting automated reasoning, detecting possible mismatches, etc.). Nevertheless, we are able to model such a connector that mediates the communication between CFring and

[2] http://www.fring.com.

MSN/XMPP as a term of our algebra. In fact, this connector may be expressed in terms of the algebra $\mathcal{AP}(\mathcal{A})$ as the following compound term:

```
((Trans(aut,handshake) ⊙ Cons(handshake_ok) ⊙ Prod(auth) ⊙ Trans(auth_ok,aut_ok)) ⊙
(Trans(quit,close) ⊙ Trans(close_ok,quit_ok)) ⊙
(Order([mo,ido],(2,1),[mo',ido']) ⊙ Trans(ido',desti) ⊙ Trans(mo',messagei) ⊙ Cons(acko))⊙
(Order([desto,messageo],(2,1),[desto',messageo']) ⊙ Trans(messageo',mi) ⊙
Trans(desto',idi) ⊙ Prod(acki)))
+
((Trans(aut,log) ⊙ Trans(log_ok,aut_ok)) ⊙
(Trans(quit,out) ⊙ Trans(out_ok,quit_ok)) ⊙
(Order([mo,ido],(2,1),[mo',ido']) ⊙ Trans(ido',di) ⊙ Trans(mo',msgi)) ⊙
(Order([do,msgo],(2,1),[do',msgo']) ⊙ Trans(msgo',mi) ⊙ Trans(do',idi)))
```

This expression seems quite complex, but it is worthwhile noticing how it can easily be decomposed into distinct portions corresponding to the original protocols. There is close correspondence between the four branches of the CFring protocol shown in Figure 3.2.C, and the paths of the MSN and XMPP protocols shown in Figures 3.2.A and 3.2.B, respectively. Each of these branches neatly map onto a sub-term of the connector expression above. This suggests that connector terms can be defined by analysis of the corresponding protocols' transition systems. Unfortunately, the connector only allows the sending and receiving of a single message, but we shall elaborate on this observation further in Section 3.2.

Relating the connector synthesis problem to our example, we built our connector by constructing two sub-connectors x' and x''. The term x' was used to mediate MSN and CFring. Note how $inv(MSN) \sqsubseteq [\![x']\!] \,||\, CFring$ and $inv(CFring) \sqsubseteq [\![x']\!] \,||\, MSN$. Analogously, x'' mediates XMPP and CFring. We combined x' and x'' by means of a suitable composition operator (i.e., $+$), thus obtaining x. Automating this kind of reasoning represents a specific area that we wish to explore, in order to develop a comprehensive theory of composable connectors in CONNECT.

3.1.5 Implementation

Our preliminary algebra is implemented into a software tool called `IAAnalyzer`. That is, `IAAnalyzer` allows for expressing connectors as (compound) terms of our algebra, and component protocols (to be mediated) as interface automata. Through model-to-text transformation from the interface automata formalism to the `Ticc` [1, 7] textual notation, `IAAnalyzer` allows also for the automated verification of the modelled mediator w.r.t. composability with the specified component protocols (i.e., interoperability check).

The `IAAnalyzer` source code and binaries can be freely downloaded from the training page on the CONNECT web-site:

`http://connect-forever.eu/training-P-connectoralgebra.html`.

`IAAnalyzer` is implemented as a stand-alone Eclipse product, reducing the need for users to deal with tedious issues concerning missing dependencies. It has been developed as a set of Eclipse plug-ins exploiting both the Eclipse Modeling Framework (EMF) project [31, 64] and the Acceleo 3.0 Model-to-Text (M2T) transformation project [30].

The main reasons for embarking on this implementation of the algebra are: (i) to validate our current formalization; (ii) to have a modelling tool for designing connectors as (compound) terms of our algebra, while having automated support for the generation of the connector interface-automata-based semantics; and (iii) to support the automated analysis (e.g., simulation, refinement, model-checking, etc.) of the modelled connectors in terms of "correctness" checks with respect to the protocol of the Networked Systems (NSs) and the requirements specified for the whole connected system.

Concerning this last purpose, by exploiting Acceleo M2T transformation, `IAAnalyzer` automatically generates the `Ticc` [1, 7] specification of the modelled NSs and connector. `Ticc` (Tool for Interface Compatibility and Composition) is a tool for the prototyping and verification of distributed designs. `Ticc` provides the following features: modelling of components and their interaction in terms of *Sociable Interfaces* [53] (interface automata represent a particular case of them); composition and compatibility checking; simulation and verification of CTL properties [58]. `Ticc` is implemented as a set of functions that

extend the capabilities of the Ocaml command-line. The tool is released under the GPL and its code is freely available.

The current version of `IAAnalyzer` adheres to the current formalization of the algebra and, hence, it should be considered as a base implementation for possible future extensions of the algebra. Consequently, as more features are added to the algebra, our current version of `IAAnalyzer` will be extended to implement the additional functionalities. For example, future areas for extension can include: combining quantitative and interface models, connector (code) synthesis, and integration with Prism, etc.

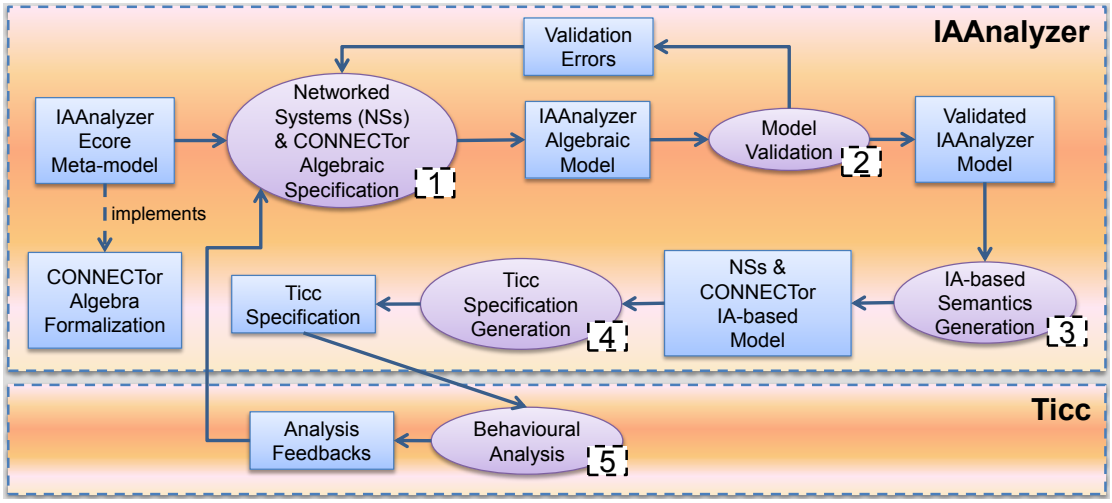

Figure 3.3: The IAAnalyzer modelling and analysis process

Figure 3.3 represents the modelling and analysis process that the users follow while interacting with `IAAnalyzer` (and its integrated tools). Process activities are represented as ellipses, whereas process artefacts are boxes. Some activities are specifically supported by `IAAnalyzer`, others such as analysis can be carried out by external tools, such as `Ticc`, which are integrated through M2T transformations. Dotted-lined boxes with numbers denote the order in which the various activities are carried out. In the following, we briefly describe the process activities shown in Figure 3.3.

1. **Networked Systems (NSs) & Connector Algebraic Specification.** By using the `IAAnalyzer` editor, the user can model the interaction behaviour of the NSs under study and of the connector that make them able to interoperate. The former is expressed by using the interface automata syntax, the latter by combining terms of the underlying algebra (e.g., `Merge`, `Order`, `Trans`, etc.) through its operators (e.g., `Plugging`, `Alternation`, `Conjunction`, etc.). `IAAnalyzer` enables hierarchical connector specification, i.e., two or more simpler connectors can be combined, through the algebra operators, to model a more complex connector. `IAAnalyzer` drives the user in specifying a model that conforms to the underlying ecore meta-model. For instance, given the ecore meta-model, EMF generates validation code for checking whether the value of a required attribute is missing or not. Nevertheless, some domain-specific constraints can be violated but they can be still detected by means of custom model validation.

2. **Model Validation.** `IAAnalyzer` automatically validates the designed model with respect to (i) conformance to the underlying meta-model (validation code generated by EMF), and (ii) domain-specific constraints (hard-coded custom validation). For instance, custom validation is required in order to check whether the set of input actions of an interface automaton model and the set of its output actions are disjoint. If the model suffers some constraint violation, the user is informed and the model specification can be fixed accordingly. Otherwise, the model is validated and its interface-automata-based semantics can be generated.

3. **Interface-automata-based Semantics Generation.** In order to support automated reasoning, `IAAnalyzer` automatically generates the interface-automata-based semantics of the specified NSs

and connector models. NSs are specified by directly exploiting the interface automata syntax and, hence, their interface-automata-based semantics generation is straightforward. The connector is specified by combining terms of the algebra by means of the algebra operators. Thus, the interface-automata-based semantics of the connector is automatically generated by exploring its hierarchical structure and applying the operators' semantics to the respective operands. After interface-automata-based semantics generation, `IAAnalyzer` has produced an interface automaton for each specified NS and connector.

4. **Ticc Specification Generation.** By means of M2T transformation, coded in Acceleo 3.0, `IAAnalyzer` automatically translates the generated interface-automata-based semantics into a `Ticc` textual specification. `Ticc` deals with a generalisation of interface automata (namely, sociable interfaces) and hence, to perform the transformation, no additional information is required beyond the generated interface-automata-based semantics.

5. **Behavioural Analysis.** As mentioned above, once the `Ticc` specification is generated, the user can exploit the `Ticc` functionalities in order to, e.g., automatically check the correctness of the specified connector with respect to the interaction protocol of the specified NSs, i.e., check whether the connector does not introduce wrong interaction behaviours thus making the NSs unable to interoperate.

Concerning the last two activities, it is worthwhile noticing that, being implemented by means of Model Driven Development (MDD) technologies such as EMF and Acceleo 3.0, `IAAnalyer` is indeed independent from the specific analysis we wish to perform. In other words, by referring to the `IAAnalzyer` modules that implement the first four activities, `IAAnalyzer` can be easily integrated with other analysis tools beyond `Ticc`. In fact, integration with other analysis tools is just a matter of possibly extending the meta-model with new domain concepts (depending on the specific analysis tool) and code new M2T Acceleo transformations. Note that, due to the declarative nature of Acceleo, coding transformations are usually easier than writing imperative/procedural transformation code. Other advantages of our algebraic implementation by way of the `IAAnalyzer` tool are listed below.

- `IAAnalyzer` has been useful in validating the effectiveness and expressive power of the current version of our algebra, hence suggesting that new characteristics have to be added in the future to improve, e.g., usability of the algebra. For instance, we'll see shortly that the interaction behaviour of a compound term of the algebra can substantially differ from a user's intuitive thought of what behaviours a term captures. This has suggested that our set of operators should be refined to improve predictability of the modelled connectors.
- The ecore meta-model underlying `IAAnalyzer` can be considered as a base meta-model for interface-automata-based models and, hence, within CONNECT, it may be exploited also outside WP2.
- Being an EMF tool, `IAAnalyzer` can be integrated with other EMF tools that are becoming ever more popular in the practice of modelling tool development.
- By having exploited MDD technologies such as EMF, the `IAAnalzyer`'s code is engineered into a "Model-View-Controller" architecture [41] that supports easy extension of the tool with new features and functionalities.

We have released both the binaries and the source code of `IAAnalyzer`. These are suitable for the two types of users of our system. An end-user (or just user) uses the tool for modelling and analysis purposes, whereas a developer can extend the tool with new functionalities or integrate it with other tools. Thankfully, from the URI above, documentation is available for both users and developers, e.g., see the link titled as "IAAnalyzer Tutorial".

IAAnalyzer as a Debugging Tool. The example discussed in Section 3.1.4 highlights the importance of having a tool like `IAAnalzyer` integrated with other analysis tools. From our experiments, we have identified a number of issues affecting the usability of our algebra. Although `IAAnalyzer` implements a high-level connector formalism conceived with the aim of easing connector modelling, due to the current

formalization of the plugging operator which is mainly based on the interface automata parallel composition operator, the user might not always be able to predict the result of its modelling. A composability check built into `IAAnalyzer` reveals that the connector described in Section 3.1.4 does not actually let "CFring" and "XMPP" interoperate, although from an intuitive glance the connector should always achieve interoperability.

As discussed in Section 2.2.2, when composing two interface automata in parallel, first the product between the two interface automata is performed. It lets the two interface automata synchronize on common input/ouput actions and independently progress on non-common actions. If there is no *error* state[3] in the product, the result of the parallel composition is the same as the one of the product. Otherwise all the traces that always lead to error states are pruned, hence producing the result of the parallel composition. Since the environment can prevent only input actions of a component[4], while pruning these traces, from an error state err, backwards error propagation is performed until encountering the first state from which both err is reachable only by performing an input action and there exist at least an action that does not lead to an error state. For further details and clarifications about interface automata and their parallel composition, we refer to Section 2.2.2.

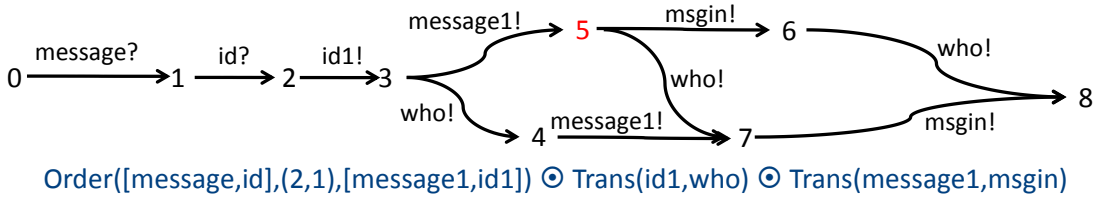

Figure 3.4: Incorrect connector (on the top), correct connector (on the bottom)

Coming back to our example, the top-side of Figure 3.4 shows the interface automaton of the modelled connector[5] plus its algebraic specification. When building the product between the interface automaton of the connector, of "CFring", and of "XMPP", there is a sequence of interactions that would let the two NSs able to interoperate and there is one that would not achieve interoperability, hence detecting an error state. The former would let the connector exchange with the two NSs (and the environment) the messages in the sequence ⟨"message?" "id?" "id1!" "who!" "message1!" "msgin!"⟩, hence making "CFring" and "XMPP" always able to interoperate. The latter would let the connector reach the state '5' that is an error state. Since error states can be prevented by acting only on input actions, the result of the parallel composition is "empty" as also revealed by the message printed as output by the `Ticc` composability check: "`The initial condition of module CFring*XMPP*Plug1 is empty: this is a sign of incompatibility!`" (see the live tutorial).

The bottom-side of Figure 3.4 shows the interface automaton of the correct connector plus its algebraic specification. It is the connector that lets "CFring" and "XMPP" able to interoperate. Although correct, its algebraic specification reveals that for a user it can be unintuitive to write it down since he/she should use the "Order" primitive on the rightmost side of the expression to translate messages rather than to reorder them. Thus, due to the current formalization of the algebra, specifying the correct connector would need to make an inappropriate use of the "Order" primitive. Nevertheless, this conclusion should be considered as a positive one with respect to the overall work done so far about the connector algebra and its related implementation. In fact, `IAAnalzyer` allowed us to raise, at an early stage of our formalization, an expressiveness issue of the current version of the algebra that, without the validation carried on, would

[3] An error state is a sink state or a state from which only error states can be reached.

[4] Within the interface automata setting, output actions are always uncontrollable since, realistically, a component plays an active role with respect to them, whereas it is a passive entity on inputs.

[5] As usual, the suffix '?' (resp., '!') denotes input (resp., output) actions.

not have been trivial to discover or would have been discovered late. Possible ideas to be developed as future work in order to solve the raised issue are discussed in the next section.

3.1.6 Summary and Limitations

In this section, we formalised an initial high-level connector algebra for reasoning about protocol mismatches and we provided an implementation for our algebra in terms of the `IAAnalzyer` modelling tool. Our formulation sets the scene for a yet-to-come comprehensive algebra facilitating the rigorous characterisation of complex interaction behaviours. Such an algebra should allow reasoning with respect to both functional and non-functional properties, in addition to supporting automated reasoning and learning about system interaction behaviour, as well as automated synthesis, matching, refinement, composition, evolution, and (possibly partial) re-use of connectors. Details of this forthcoming work are detailed in Section 3.2.

Limitations. In terms of our current algebra, the case study introduced in Section 3.1.4 has highlighted a number of shortcomings that should be considered in our comprehensive treatment. For starters, we cannot construct a connector that exhibits looping behaviours. The form of our algebra dictates that for any term whose semantics are equivalent to an interface automaton (as opposed to Err), the structure of the automaton is a directed graph in which every state is visited at most once. Such a restriction on the behaviour of the connector is unduly restrictive. This is evident from our case study, where the connector only supports the sending and receiving of a single message. If the number of messages to be sent and received is known in advance, then we can build a connector whose size is related polynomially to the number of messages to be transmitted. This, however, is inadequate.

In a future version of the algebra, the restriction on looping would need to be lifted. We already have an idea of how this could be done, by introducing looping equivalents of the primitives. It also seems likely that we would need a fix-point operator to encode complex looping behaviours in the algebra, beyond those at the primitive level. It would be interesting to see whether looping operators make it easier to model connectors in our algebra or not. Naturally, we would hope so.

Interface automata as a modelling formalism. This shortcoming of the algebra is not to say that interface automata are a bad choice of model for assigning semantics to terms; after all, it is the algebra that is restricting the behaviour of interface automata. As a consequence of having chosen interface automata as the semantic model, our algebra supports specifications of behaviours, which we claim are necessary for building scalable connectors. Furthermore, interface automata support a notion of refinement which is a congruence with respect to a number of our composition operators. Accordingly, substitution (e.g., for connector reuse or evolution) can be performed compositionally. We have also defined a notion of equivalence over the terms of the algebra and, based on this, we established that the algebra is sound and complete. These properties advocate the adoption of interface automata as the semantic model for the algebra.

The algebra as a modelling formalism. Our case study shows that the algebraic term representing a connector, to some extent, maps intuitively onto the models of the protocols to be mediated. The purpose of the algebra was to give system designers a high-level tool for specifying and reasoning about connector behaviours, which is why we favoured the utilisation of high-level primitives rather than developing yet another low-level process calculus. It seems that our high-level algebra allows a designer to easily and intuitively specify complex connectors, although further justification would be required for this claim based on further case studies. For instance, the usability issue raised by the example described in Section 3.1.4 suggests to us that our set of operators should be refined so as to either include an operator that behaves analogously to *sequencing* in regular expressions or conceive a notion of *priority* to suitably filter the interactions exhibited by the plugging operator, in a way similar to what is done in BIP [10].

Besides this future work, it is also our intention to determine whether the set of identified primitives is complete enough. Increasing the set of basic solutions should allow us to increase the types of behaviours that our algebra can capture. Discovery of such primitives is likely to come from analysis of further scenarios.

As broached at the end of Section 3.1.3, we need to take a closer look at automated connector synthesis. This is likely to be a definitive area on which our algebra is judged as to whether it has made a meaningful contribution to component-based design. We already have ideas relating to this in terms of rewriting systems [27], although the details require further fleshing out.

Implementation. Concerning the implementation of our algebra with the `IAAnalzyer` tool, as a software prototype `IAAnalyzer` still needs some improvements. In the future, we should extend the front-end by developing (i) a graphical editor that is more effective, for modelling purposes, than the standard tree-view-based editor generated by EMF; and (ii) a textual editor that is unavoidably needed for the modelling of real-scale examples. The former can be developed by exploiting the GMF Tooling Project [33], which provides a model-driven approach to generating graphical editors in Eclipse. In particular, EuGENia GMF [32] is a convenient tool to automatically generate a GMF editor from an annotated EMF meta-model. Through the CONNECT life-time, we will also keep the current `IAAnalyzer` implementation up-to-date according to each future evolution of our algebraic formalization, e.g., combination of quantitative and interface models, connector (code) synthesis, integration with Prism, etc. Last but not least, we have to extensively validate the tool with more complex examples.

3.2 Future Work: A Quantitative Theory of Connectors

In this section we outline in detail the work that we intend to undertake for generating a framework for modelling connectors. Broadly speaking, the idea is to develop a probabilistic specification theory combining assume-guarantee reasoning for interface automata. This will allows us to bridge this strand of work to the quantitative assume-guarantee verification formulated and implemented in Chapter 4.

3.2.1 Quantitative specification theory

Probabilistic interface automata. Based on observations in Section 2.1.2, probabilistic interface automata will be derived from interface automata by adding reactive-like probabilistic semantics. The semantics will not strictly be reactive, as we will permit multiple distributions from a state with the same action label to allow for local non-determinism. The reason for choosing reactive over generative semantics relates to the necessity of parallel composition being an associative operator for a meaningful specification theory.

Thus, probabilistic interface automata will be syntactically identical to the probabilistic automata presented in Definition 2.5, and hence the compositional verification approach from Chapter 4 will apply following minor adaptations. However, the semantics of composition will be more involved, due to the optimistic nature of composition on non-probabilistic interface automata, where actions are partitioned into input, output and internal varieties.

Definition 3.4 *A* probabilistic interface automaton *(PIA)* $\mathcal{C}$ *is a tuple* $(S, s_0, \mathcal{A}_I, \mathcal{A}_O, \mathcal{A}_H, \longrightarrow)$*, where:*

- S *is a finite collection of states*
- $s_0 \in S$ *is the designated initial state*
- $\mathcal{A}_I$ *is a finite set of input actions*
- $\mathcal{A}_O$ *is a finite set of output actions*
- $\mathcal{A}_H$ *is a finite set of hidden actions*
- $\rightarrow \subseteq S \times \mathcal{A} \times Dist(S)$ *is the probabilistic transition relation.*

Example A system M_2 will safely switch off if it is given an $alert$ before a $switchoff$ request, otherwise if no $alert$ is made the system will power off successfully only 90% of the time. The system M_2 is to be controlled by a controller M_1, but the actions of both systems are syntactically inconsistent, although they are functionally compatible. The systems are to be connected by means of a connector that performs the

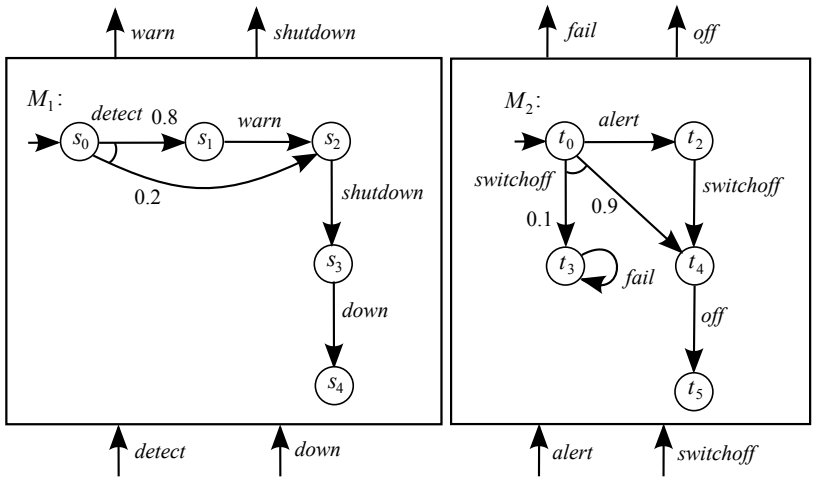

Figure 3.5: Systems M_1 and M_2

necessary mediation. Models of the systems to be connected are represented by probabilistic interface automata in Figure 3.5, while the connector and the composition of the connector and M_1 is shown in Figure 3.6.

Specification theory. Following the lead of Section 2.2.3, specifications of probabilistic interface automata will themselves be probabilistic interface automata in which probabilistic annotations of transitions are replaced by intervals of probabilities. In the long term, we will consider arbitrary probabilistic constraints rather than intervals, as described in Section 2.2.4. The advantage of intervals is that they often allow one to reason in terms of their supremum and infimum.

Definition 3.5 *A* probabilistic contract $\mathcal{D}$ *is a tuple* $(S, s_0, \mathcal{A}_I, \mathcal{A}_O, \mathcal{A}_H, \sigma)$*, where:*

- S *is a finite collection of states*
- $s_0 \in S$ *is the designated initial state*
- $\mathcal{A}_I$ *is a finite set of input actions*
- $\mathcal{A}_O$ *is a finite set of output actions*
- $\mathcal{A}_H$ *is a finite set of hidden actions*
- $\sigma \subseteq S \times \mathcal{A} \times (S \to 2^{[0,1]})$ *is the probabilistic transition relation, in which* $(s, a, \delta) \in \sigma$ *only if* $\delta(s')$ *is an interval on* $[0, 1]$ *for each* $s' \in S$*.*

As in Section 2.2.3, our definitions of implementation (probabilistic interface automata) and contracts are closely related to each other. Implementations can be seen as a restricted class of probabilistic contracts in which all intervals contain a single probability. There is a natural way of lifting implementations to contracts, which means that it is sufficient to define our specification theory operators and relations on probabilistic contracts, although we may wish to have additional operators on implementations.

Definition 3.6 *The associated probabilistic contract for an implementation* $\mathcal{C}$ *(as defined in Definition 3.4) is a tuple* $\lceil \mathcal{C} \rceil = (S, s_0, \mathcal{A}_I, \mathcal{A}_O, \mathcal{A}_H, \sigma')$ *in which* $(s, a, \delta) \in \sigma'$ *iff* $(s, a, d) \in \sigma$ *where* $(s', [p, p]) \in \delta$ *iff* $(s', p) \in d$ *for each* $s' \in S$ *and* $p \in [0, 1]$*.*

Given the above definition of probabilistic interface automata, the goal of this work is to formulate the specification theory operators and relations that were considered in Section 2.2. In particular, we will focus on parallel composition, refinement, abstraction/hiding, conjunction and the quotient of probabilistic interface automata and their specifications.

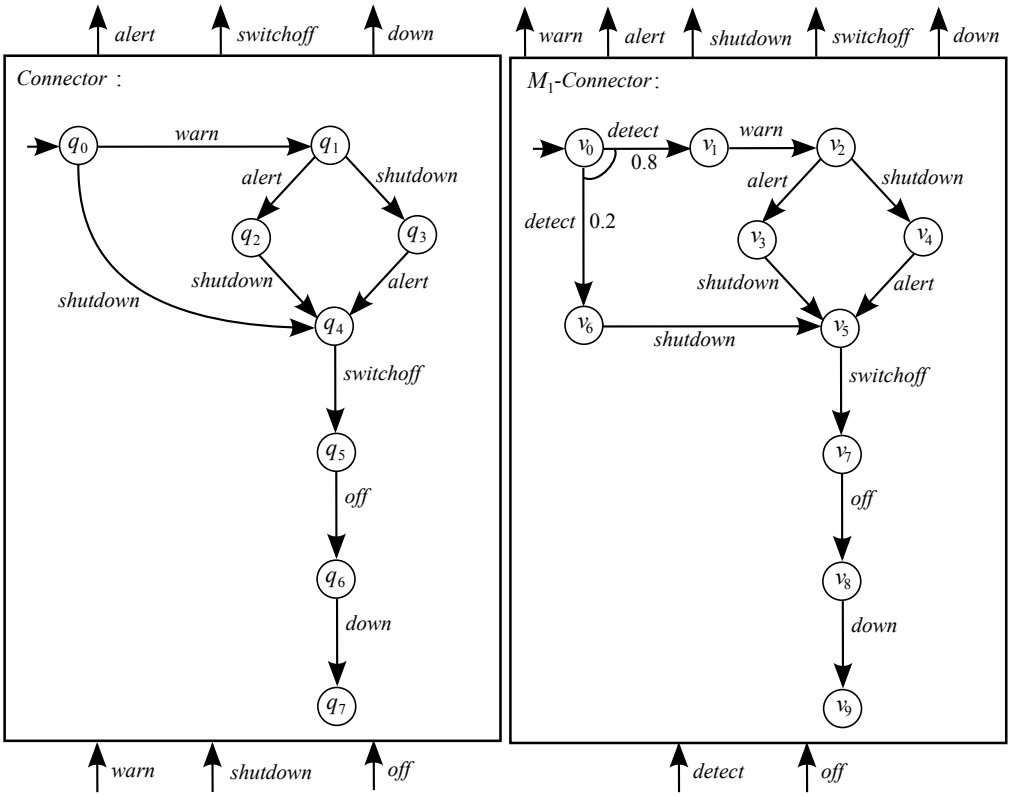

Figure 3.6: A mediating connector for M_1 and M_2, and composition of M_1 and the connector

Operations. An intuitive definition of *refinement* involves a probabilistic extension of the alternating refinement relation defined on interface automata in Definition 2.16. The probabilistic extension of this relation would be based on the probabilistic simulations defined for probabilistic automata in Definition 2.7 and refinement for probabilistic contracts in Definition 2.23. However, we will explore alternative notions of lesser complexity, such as those based on failure or ready equivalence.

Parallel composition of probabilistic contracts should be straightforward, as it will combine parallel composition of the probabilistic contracts defined in Section 2.2.3 with the optimistic composition semantics of interface automata in Definition 2.14. *Compatibility* for parallel composition should be the same as on non-probabilistic interface automata, which is a simple syntactic check on the action sets.

Abstraction/hiding of actions is trivial because it is the same as for interface automata and indeed I/O automata, as presented in Definition 2.11.

Conjunction of contracts is likely to be problematic, as [26] highlights that interval Markov chains are not closed under conjunction. To circumvent this problem for probabilistic contracts as presented in Section 2.2.3, the authors place a number of restrictions on the forms of contracts that can be conjoined. Unfortunately, the restrictions imposed in Definition 2.27 are not intuitive at a first glance, and moreover require costly operations in order to determine whether the conjunction is well-defined. It is not clear from the paper nor the technical report [73] as to whether these constraints are unnecessarily restrictive. Nevertheless, it is clear is that, under such circumstances, the use of probabilistic constraints would ensure a more powerful theory than for intervals, as constraints are closed under conjunction.

The difficulty of defining a *quotient* is difficult to ascertain. The lack of definitions for other probabilistic models makes this challenging. As a result, we will probably start by considering a restricted class of probabilistic interface automata: perhaps the deterministic subclass without hidden actions.

3.2.2 Assume-guarantee reasoning

Overview. Once equipped with a probabilistic specification theory for interface automata, the intention is to augment this with assume-guarantee reasoning. An ordinary interface automaton can be seen as having a weak form of assume-guarantee reasoning: if the environment issues inputs enabled in the current state of the automaton, then the automaton will guarantee to issue only a certain collection of outputs. To empower this idea, we wish to consider assume-guarantee reasoning for the class of safety and liveness properties. This will involve introducing explicit assumptions and guarantees separate from the interface automaton itself, each of which will characterise an ω-regular language.

Restriction to the class of ω-regular properties should be more than sufficient for reasoning about meaningful properties in the specification theory. Although more expressive properties could be considered, here we are aiming to keep the specification theory within an automata-based framework. Besides, reasoning about ω-regular properties on probabilistic contracts in a compositional framework will be no easy task. A recent attempt in [40] successfully adds assume-guarantee reasoning of ω-regular properties to probabilistic automata, but the contracts we will consider are more complex because of the interface automata semantics of composition. It is not easy to see upfront whether these composition semantics would have an effect upon the preservation of properties under composition. Consequently, as is done in [52], we will begin by considering safety properties alone. These should be easier to deal with because it is sufficient to reason in terms of finite traces violating the safety property, rather than infinite words satisfying it. It would then be our intention to consider ω-regular properties after resolving any issues in the safety case. This extension of our specification theory links in very closely with the work documented in Chapter 4.

To position this work, we have proposed how to add arbitrary assume-guarantee specifications of safety properties to non-probabilistic interface automata, something that has not been treated adequately until now. Starting in the next paragraph, we summarize main elements of a possible such theory. After that, we consider how a theory could be lifted to regular, ω-regular, and probabilistic properties.

Assume-Guarantee Specifications for Interface Automata We first outline a theory of assume-guarantee reasoning for interface automata, restricted to safety properties The theory supports the operations on interface automata presented in Section 2.2.2, in the sense that for each operation on interface automata, there is a corresponding operation on specifications. Thus, the theory can be used to specify connectors expressed in the algebra presented in Section 3.1.

A formalism for specifying interface automata should provide constraints on the allowed sequences of input and output actions at its interface, but stated in a less operational way than done by interface automata. Assume-guarantee specifications attain this by consisting of two parts: an assumption and a guarantee. Both the assumption and the guarantee are simply constraints on sequences of actions that may be observed on the interface of a component. A component satisfies such a specification if its any behavior that conforms to the assumption will also conform to the guarantee. Thus, an implementation of an assume-guarantee specification must realize some strategy for fulfilling the guarantee part as long as the assumption part is not violated.

The assumptions and guarantees can be provided in several ways: as temporal formulas over the interface of a component, using some automata formalism to recognize these sets, or in some other way. We will here abstractly just let the assumption and the guarantee be sets of traces (a trace is a sequence of actions). We here consider only safety properties, meaning that these sets will be prefix-closed.

As in the frameworks of I/O-automata and interface automata, an essential assumption is that each observable action is the output action of (i.e., is controlled by) only one component. The actions of a component are partitioned into input and output actions, where output actions can be controlled by the component, and input actions are controlled by the environment.

Before starting, let us just introduce some notation. Let $\mathcal{A}$ and $\mathcal{B}$ be set of actions with $\mathcal{A} \subseteq \mathcal{B}$. For a trace $t \in \mathcal{B}^*$, let $t\lceil_{\mathcal{A}}$ denote the subsequence of t consisting of actions in $\mathcal{A}$. For $T \subseteq \mathcal{B}^*$, define $T\lceil_{\mathcal{A}} = \{t\lceil_{\mathcal{A}} : t \in T\}$. For $T \subseteq \mathcal{A}^*$, define $T\lfloor^{\mathcal{A}} = \{t \in \mathcal{B}^* : t\lceil_{\mathcal{B}} \in T\}$.

We are now ready to define assume-guarantee specifications.

Definition 3.7 *An* Assume-Guarantee specification *(AG-spec for short) is a tuple* $\langle \mathcal{A}_I, \mathcal{A}_O, \mathcal{R}, \mathcal{G} \rangle$*, where*

- $\mathcal{A}_I$ *and* $\mathcal{A}_O$ *are disjoint sets of* input *and* output actions*,*
- $\mathcal{R}$ *and* $\mathcal{G}$ *are prefix-closed subsets of* $(\mathcal{A}_I \cup \mathcal{A}_O)^*$*.*

The intuitive interpretation of an AG-spec is that the component will maintain the past sequence of actions in $\mathcal{G}$ as long as the environment keeps them inside $\mathcal{R}$. More precisely, a component satisfies an assume-guarantee specification $\langle \mathcal{A}_I, \mathcal{A}_O, \mathcal{R}, \mathcal{G} \rangle$ if any behavior of the component, in which the finite trace t is exhibited, satisfies the condition that if $t \in \mathcal{R}$ then $t \in \mathcal{G}$.

In order for a component to fulfill an AG-spec, as given above, it must maintain the invariant that the occurred sequence t of actions (i.e., the current trace), in addition to the requirement that $t \in \mathcal{G}$ if $t \in \mathcal{R}$, satisfies the property that this requirement holds when t is extended by input actions, since they are not controlled by the component. Another way to say this is that the component must keep the past sequence of observable actions t within the largest controllable invariant in $\overline{\mathcal{R}} \cup \mathcal{G}$, i.e., the largest prefix-closed set of traces K such that $K \subseteq (\overline{\mathcal{R}} \cup \mathcal{G})$, and such that for any $t \in K$ and $i \in \mathcal{A}_I$ we have $ti \in K$.

For a set T of traces and set $\mathcal{A}$ of actions, define $\mathcal{K}[\mathcal{A}](T)$ as the set of traces $t \in T$ such that $tu \in T$ for any $u \in \mathcal{A}^*$. Using this notation, we see that a component (e.g., modeled as an I/O-automaton), satisfies $\mathcal{P} \models \mathcal{D}$ if its traces are in $\mathcal{K}[\mathcal{A}_I](\overline{\mathcal{R}} \cup \mathcal{G})$. We say that an AG-spec $\langle \mathcal{A}_I, \mathcal{A}_O, \mathcal{R}, \mathcal{G} \rangle$ is *satisfiable* if $\mathcal{K}[\mathcal{A}_I](\overline{\mathcal{R}} \cup \mathcal{G}) \neq \emptyset$.

A remark on how we define the meaning of Assume-Guarantee specification is in order. In this section, we provide a simple definition, in which the specification is interpreted as an implication between the assumption and the guarantee. It is common to see slightly different formalizations of the intuition that "the component does not violate the guarantee as long as the environment maintains the assumption". A typical definition would be that if to is a trace of a component, then if $t \in \mathcal{R}$ then $to \in \mathcal{G}$. I.e., the definition only constrains traces that end with an output action. This definition coincides with the above if $\mathcal{R}$ is closed under extensions with output actions, which is often the case in applications (the assumption should not constrain the behavior of the component). In the following, we sometimes will use such restrictions. Therefore, say that an assumption is *output-closed* if it is closed under extensions with output actions, and say that a guarantee is *input-closed* if it is closed under extensions with input actions. In our current work, however, the objective is to develop and understand basic principles for assume-guarantee reasoning, whence we have chosen to work with assumptions and guarantees in the form of arbitrary safety properties. This choice can be contrasted with, e.g., the work by Larsen et al. [54], where assumptions and guarantees are provided as I/O-automata, thereby restricting the flexibility of the specifier.

We will now define a satisfaction relation between interface automata and assume-guarantee specifications, similar in spirit to the refinement relation on interface automata. As a preparation, we first have to define precisely what an interface automaton $\mathcal{C} = \langle S, s_0, \mathcal{A}_I, \mathcal{A}_O, \mathcal{A}_H, \longrightarrow \rangle$ assumes about its environment. It follows from the definition of parallel composition for interface automata in Section 2.2.2, that an interface automaton will reject an input if there is any sequence of nondeterministic choices under which this input would not have been accepted. To make this precise, consider an interface automaton that nondeterministically chooses between the sequence $o\ i\ o$, and the sequence o. Since the input action i is allowed only under one outcome of the nondeterministic choice, it is not part of the assumption of the interface automaton. This effect must be considered when deriving the assumption part. Therefore, define a mapping $h_{\mathcal{C}} : (\mathcal{A}_I \cup \mathcal{A}_O)^* \mapsto 2^S$, from traces in $(\mathcal{A}_I \cup \mathcal{A}_O)^*$ to sets of states, by induction over traces, as:

- $h_{\mathcal{C}}(\varepsilon) = \{s_0\}$,
- for $t \in (\mathcal{A}_I \cup \mathcal{A}_O)^*$ we define
  - for $o \in \mathcal{A}_O$, define $h_{\mathcal{C}}(to) = \{s \in S \ : \ \exists s' \in h_{\mathcal{C}}(t) : s' \xrightarrow{o} s\}$
  - for $i \in \mathcal{A}_I$, define $h_{\mathcal{C}}(ti) = \{s \in S \ : \ \exists s' \in h_{\mathcal{C}}(t) : s' \xrightarrow{i} s\}$ if $\forall s' \in h_{\mathcal{C}}(t). \exists s \in S : s' \xrightarrow{i} s$, otherwise $h_{\mathcal{C}}(ti) = \emptyset$

Intuitively, $h_{\mathcal{C}}(t)$ is the set of states that $\mathcal{C}$ can reach after the trace t under the requirement that input actions can be received regardless of how nondeterministic choices have been resolved on the way. Furthermore, the assumption must not constrain the occurrence of output actions: this is taken into account by closing the assumption under extensions by output actions.

We now define the assumption represented by an interface automaton $\mathcal{C}$, denoted $\mathcal{R}^{\mathcal{C}}$, as

$$\mathcal{R}^{\mathcal{C}} = \{tt' \mid h_{\mathcal{C}}(t) \neq \emptyset \wedge t' \in (\mathcal{A}_O)^*\}$$

We can now define when an interface automaton satisfies an AG-specification. Let $T_{\mathcal{C}}$ be the traces of the interface automaton $\mathcal{C}$.

Definition 3.8 *Let $\mathcal{C} = \langle S^{\mathcal{C}}, s_0^{\mathcal{C}}, \mathcal{A}_I^{\mathcal{C}}, \mathcal{A}_O^{\mathcal{C}}, \mathcal{A}_H^{\mathcal{C}}, \longrightarrow_{\mathcal{C}} \rangle$ be an interface automaton, and let $\mathcal{D} = \langle \mathcal{A}_I^{\mathcal{D}}, \mathcal{A}_O^{\mathcal{D}}, \mathcal{R}^{\mathcal{D}}, \mathcal{G}^{\mathcal{D}} \rangle$ be a satisfiable AG-specification. Then $\mathcal{C}$ satisfies $\mathcal{D}$ if $\mathcal{R}^{\mathcal{D}} \subseteq \mathcal{R}^{\mathcal{C}}$ and $(\mathcal{R}^{\mathcal{D}} \cap T_{\mathcal{C}}) \subseteq \mathcal{G}^{\mathcal{D}}$.*

Let us now consider operations on AG-specifications.

Composability. Composability on AG-specifications coincide exactly with the definition on interface automata, i.e., that the sets of output actions should be disjoint.

Parallel composition. There is a well-established composition theorem for AG-specifications, which has been presented in several forms in several papers (e.g., [16, 2, 50, 59]). This rule seems circular, but the circularity can be broken up for safety specifications, using induction over the sequence of steps of a global trace. For safety properties, one only needs the simple property that a safety property cannot be simultaneously violated by two different components. In our framework, this can be captured by the requirements that assumptions must be output-closed and guarantees input-closed.

Proposition 3.9 (Composition Rule for AG-spec) *Let $\langle \mathcal{A}_I^{\mathcal{C}}, \mathcal{A}_O^{\mathcal{C}}, \mathcal{R}^{\mathcal{C}}, \mathcal{G}^{\mathcal{C}} \rangle$ and $\langle \mathcal{A}_I^{\mathcal{D}}, \mathcal{A}_O^{\mathcal{D}}, \mathcal{R}^{\mathcal{D}}, \mathcal{G}^{\mathcal{D}} \rangle$ be composable AG-specifications, in which the assumptions are output-closed and the guarantees are input-closed. Let $\mathcal{A}_O = \mathcal{A}_O^{\mathcal{C}} \cup \mathcal{A}_O^{\mathcal{D}}$ and $\mathcal{A}_I = (\mathcal{A}_I^{\mathcal{C}} \cup \mathcal{A}_I^{\mathcal{D}}) \setminus \mathcal{A}_O$, and let $\mathcal{R}$ and $\mathcal{G}$ be prefix-closed subsets of $(\mathcal{A}_O \cup \mathcal{A}_I)^*$. Let $\mathcal{P}$ and $\mathcal{Q}$ be components (e.g., modeled by interface automata). If*

- *$\mathcal{P}$ satisfies $\langle \mathcal{A}_I^{\mathcal{C}}, \mathcal{A}_O^{\mathcal{C}}, \mathcal{R}^{\mathcal{C}}, \mathcal{G}^{\mathcal{C}} \rangle$ and $\mathcal{Q}$ satisfies $\langle \mathcal{A}_I^{\mathcal{D}}, \mathcal{A}_O^{\mathcal{D}}, \mathcal{R}^{\mathcal{D}}, \mathcal{G}^{\mathcal{D}} \rangle$, and*
- *$(\mathcal{R} \cap \mathcal{G}^{\mathcal{C}} \lfloor^{(\mathcal{A}_I \cup \mathcal{A}_O)}) \lceil_{(\mathcal{A}_I^{\mathcal{D}} \cup \mathcal{A}_O^{\mathcal{D}})} \subseteq \mathcal{R}^{\mathcal{D}}$ and $(\mathcal{R} \cap \mathcal{G}^{\mathcal{D}} \lfloor^{(\mathcal{A}_I \cup \mathcal{A}_O)}) \lceil_{(\mathcal{A}_I^{\mathcal{C}} \cup \mathcal{A}_O^{\mathcal{C}})} \subseteq \mathcal{R}^{\mathcal{C}}$*

then $\mathcal{P} \| \mathcal{Q}$ satisfies $\langle \mathcal{A}_I, \mathcal{A}_O, \mathcal{R}, \mathcal{G}^{\mathcal{C}} \lfloor^{(\mathcal{A}_I \cup \mathcal{A}_O)} \cap \mathcal{G}^{\mathcal{D}} \lfloor^{(\mathcal{A}_I \cup \mathcal{A}_O)} \rangle$. □

It is required that $\mathcal{P}$ and $\mathcal{Q}$ satisfy obvious constraints on their sets of input and output actions. The somewhat clumsy notation in the second bullet intuitively says that (i) $\mathcal{R}$ and $\mathcal{G}^{\mathcal{C}}$ imply $\mathcal{R}^{\mathcal{D}}$ and that (ii) $\mathcal{R}$ and $\mathcal{G}^{\mathcal{D}}$ imply $\mathcal{R}^{\mathcal{C}}$. The intuition in the proof of Proposition 3.9 is to use induction on the length of any computation of $\mathcal{P}\|\mathcal{Q}$. As a base case, the empty trace is in $\overline{\mathcal{R}} \cup \left(\mathcal{G}^{\mathcal{C}} \lfloor^{(\mathcal{A}_I \cup \mathcal{A}_O)} \cap \mathcal{G}^{\mathcal{D}} \lfloor^{(\mathcal{A}_I \cup \mathcal{A}_O)}\right)$ by prefix-closure. For the inductive step, assume that t satisfies $\overline{\mathcal{R}} \cup \left(\mathcal{G}^{\mathcal{C}} \lfloor^{(\mathcal{A}_I \cup \mathcal{A}_O)} \cap \mathcal{G}^{\mathcal{D}} \lfloor^{(\mathcal{A}_I \cup \mathcal{A}_O)}\right)$, and consider an extension to ta such that $ta \in \mathcal{R}$. If a is an output action of one component (say, in $\mathcal{A}^{\mathcal{C}}_O$), then by output-closure we have $ta \in \mathcal{R}^{\mathcal{C}}$, from which it follows by the above rules that ta satisfies $\mathcal{G}^{\mathcal{C}}$, $\mathcal{R}^{\mathcal{D}}$, and $\mathcal{G}^{\mathcal{D}}$. If a is an input action of one component (say, in $\mathcal{A}^{\mathcal{C}}_I$), then by input-closure we have $ta \in \mathcal{G}^{\mathcal{C}}$, from which the same conclusion follows.

In general, there is not a unique AG-specification for the parallel composition of two AG-specifications. However, in Proposition 3.9, we can in fact choose a weakest assumption $\mathcal{R}$ which makes the two last premises true. This allows us to define a composition operation on AG-specifications.

Refinement. Refinement for AG-specifications is quite similar to that for Interface Automata. Let us for simplicity consider AG-specifications $\mathcal{D}$ to refine another AG-spec $\mathcal{C}$ with $\mathcal{A}^{\mathcal{C}}_I = \mathcal{A}^{\mathcal{D}}_I$ and $\mathcal{A}^{\mathcal{D}}_O = \mathcal{A}^{\mathcal{C}}_O$. A natural condition for refinement is then that $\mathcal{R}^{\mathcal{C}} \subseteq \mathcal{R}^{\mathcal{C}}$ and $\mathcal{G}^{\mathcal{D}} \subseteq \mathcal{G}^{\mathcal{D}}$ (This is the condition to be proposed in practice, over the more precise but clumsy $\mathcal{K}[\mathcal{A}^{\mathcal{D}}_I]((\overline{\mathcal{R}^{\mathcal{D}}} \cup \mathcal{G}^{\mathcal{D}})) \subseteq \mathcal{K}[\mathcal{A}^{\mathcal{C}}_I]((\overline{\mathcal{R}^{\mathcal{C}}} \cup \mathcal{G}^{\mathcal{C}}))$).

Regular, ω-regular, and Probabilistic Properties. We now consider how to develop a specification theory, based on assume-guarantee reasoning, for regular, ω-regular, and probabilistic properties.

Definition 3.10 *A* safety assumption/guarantee *for a regular safety property P_s is a deterministic finite automaton $P^{ag}_s = (S, s_0, \mathcal{A}, \delta, F)$, where:*

- *S is a finite collection of states*
- *$s_0 \in S$ is the designated initial state*
- *$\mathcal{A}$ is a collection of actions*
- *$\delta : S \times \mathcal{A} \to S$ is the transition function*
- *$F \subseteq S$ is the set of accepting states.*

The automaton P^{ag}_s induces a language that precisely characterises the finite prefixes of words that violate the safety property P_s. Thus a safety assumption/guarantee Q satisfies a safety property P_s iff the infinite-extension of all finite words in $\mathcal{L}(Q)$ are not contained in $\mathcal{L}(P_s)$.

This representation for safety properties is taken from [52]. We now consider the generalised assumptions and guarantees for ω-regular properties, which are taken from [40].

Definition 3.11 *An* assumption/guarantee *for an ω-regular property P_ω is a deterministic Rabin automaton $P^{ag}_\omega = (S, s_0, \mathcal{A}, \delta, \mathcal{F})$ in which:*

- *S is a finite collection of states*
- *$s_0 \in S$ is the designated initial state*
- *$\mathcal{A}$ is a collection of actions*
- *$\delta : S \times \mathcal{A} \to S$ is the transition function*
- *$\mathcal{F} \subseteq \mathcal{P}(S) \times \mathcal{P}(S)$ is the Rabin acceptance condition.*

Any deterministic Rabin automaton induces a set of infinite words that forms an ω-regular language. For any ω-regular property P_ω, an assumption/guarantee Q satisfies P_ω iff $\mathcal{L}(Q) \subseteq \mathcal{L}(P_l)$.

Equipped with safety and ω-regular assumptions/guarantees, it is possible to define an assume-guarantee triple on the probabilistic contracts defined in Definition 3.5.

Definition 3.12 *A* safety assume-guarantee triple *takes the form $\langle A \rangle^s_{\oplus p} \;\; M \;\; \langle G \rangle^s_{\oplus p'}$, where:*

- M *is a probabilistic contract*
- A *is a safety assumption*
- G *is a safety guarantee*
- $p, p' \in [0, 1]$ *are rational probability bounds*
- $\oplus \in \{<, \leq, >, \geq\}$ *is an ordering relation.*

Definition 3.13 *An* assume-guarantee triple *takes the form* $\langle A \rangle_{\oplus p} \;\; M \;\; \langle G \rangle_{\oplus p'}$*, where:*

- M *is a probabilistic contract*
- A *is an* ω*-regular assumption*
- G *is an* ω*-regular guarantee*
- $p, p' \in [0, 1]$ *are rational probability bounds*
- $\oplus \in \{<, \leq, >, \geq\}$ *is an ordering relation.*

The interpretation of an assume-guarantee triple is very closely related to the interpretation given to Hoare triples. Under the assumption that A holds with probability $\oplus p$ on M, it is guaranteed that G holds on M with probability $\oplus p'$. Given these assume-guarantee triples, it's possible to define a collection of rules in order to support truly compositional verification.

Compositional verification with assume-guarantee reasoning. The state space explosion problem imposes a limitation on the practical use of traditional verification techniques, such as model checking, in verifying properties of systems. In order to circumvent this, we intend to discover a collection of compositional verification rules for probabilistic contracts. The soundness of the rules will provide the ability to infer properties of the entire system by considering the components of which it is made, without having to perform the actual composition.

The paper [34] provides a list of sound and complete compositional verification rules for non-probabilistic interface automata, with respect to compatibility, preservation of safety properties under parallel composition, and refinement. Note that $lts(A)$ defines the labelled transition system equivalent to IA A; this is the same as A except that inputs, outputs and internal actions are not distinguished.

$$\text{IA-COMPAT}\ \frac{M_1 \sim A \qquad A \sqsubseteq M_2}{M_1 \sim M_2}$$

$$\text{IA-PARALLEL-SAFETY}\ \frac{\langle true \rangle\ M_1 \mid\mid A\ \langle P \rangle \qquad \langle true \rangle\ M_2\ \langle lts(A) \rangle}{\langle true \rangle\ M_1 \mid\mid M_2\ \langle P \rangle}$$

$$\text{IA-REFINE}\ \frac{M_1 \sim A \qquad S \sqsubseteq M_1 \mid\mid A \qquad A \sqsubseteq M_2}{S \sqsubseteq M_1 \mid\mid M_2}$$

In order to work towards the goal of introducing assume-guarantee rules for probabilistic contracts, we first intend to introduce the rules for the much simpler setting of non-probabilistic interface automata (more details in Section 3.2.2). Besides the rules mentioned above, we will also introduce rules for abstraction, conjunction and quotient, under the preservation of ω-regular properties, suitably generalised to the probabilistic case. We will aim to prove that the rules are sound and complete.

So far we have considered assume-guarantee rules in the dimension of the specification theory operators. However, there is also a dimension concerned with the order of dependence between different components. For example, here are two assume-guarantee rules for the parallel composition of components treated as probabilistic automata.

$$\text{ASYM}\ \frac{\langle true \rangle\ M_1\ \langle A \rangle_{\geq p_A} \qquad \langle A \rangle_{\geq p_A}\ M_2\ \langle G \rangle_{\geq p_G}}{\langle true \rangle\ M_1 \mid\mid M_2\ \langle G \rangle_{\geq p_G}}$$

$$\textsc{Circ}\ \frac{\langle true\rangle\ M_2\ \langle A_2\rangle_{\geq p_{A_2}} \qquad \langle A_2\rangle_{\geq p_{A_2}}\ M_1\ \langle A_1\rangle_{\geq p_{A_1}} \qquad \langle A_1\rangle_{\geq p_{A_1}}\ M_2\ \langle G\rangle_{\geq p_G}}{\langle true\rangle\ M_1\ ||\ M_2\ \langle G\rangle_{\geq p_G}}$$

The circular rule handles the case in which one of the components must provide a guarantee to the other so that, under the assumption of that guarantee, the other can guarantee an assumption for the original. There are many variants of these rules, which can become quite complex, particularly in the context of generating proofs of correctness. [61] contains a collection of dependence rules that will be applicable.

To give a flavour of what the rules will look like, here is an example rule for conjunction under the preservation of safety properties, although its correctness has not been proved yet. Note that this rule does not deal with assumptions on the sub-components.

$$\textsc{Conj}\ \frac{\langle true\rangle^s\ M_1\ \langle G_1\rangle^s_{\geq p_{G_1}} \qquad \langle true\rangle^s\ M_2\ \langle G_2\rangle^s_{\geq p_{G_2}}}{\langle true\rangle^s\ M_1 \wedge M_2\ \langle G_1 \& G_2\rangle^s_{\geq \min(p_{G_1}, p_{G_2})}}$$

In this rule, $G_1 \& G_2$ is the deterministic automaton that accepts the *union* of G_1 and G_2, as each G_i characterises the violations of the safety properties.

For some assume-guarantee inference rules it may be the case that a guarantee of one triple in the premises is an assumption of another triple in the premises. Automated techniques have been developed for deriving safety assumptions/guarantees for probabilistic automata, an example being [37] which utilises learning (see also Section 4.3). However, the ω-regular case is still open. Work has been done on learning ω-regular assumptions/guarantees in the non-probabilistic case, but the results have very high complexity [15]. As a result, learning assumptions/guarantees will be out of the scope of our work.

Reward structures. The majority of the quantitative models considered in Section 2.1 can have reward structures appended to them for computing and reasoning about expectations, and hence to express non-functional properties. The benefit of adding such structures to probabilistic contracts is considerable.

Recent advances in multi-objective verification [40] (further details in Section 4.2) allows one to consider the simultaneous satisfaction of multiple reward properties or the trade off between them. In the case of a single component, multi-objective verification on a probabilistic contract is closely related to multi-objective verification on MDPs, which is well-established. The work of [40] shows that the satisfiability of an assume-guarantee triple for probabilistic automata, where assumptions and guarantees are Boolean formulas over ω-regular properties and reward conditions, reduces to multi-objective model checking by the following relationship: $M \models A \Rightarrow M \models G$ iff $M \models \neg A \vee G$. We believe this can b adapted to probabilistic interface automata with rewards.

3.2.3 Future directions

Concerning further work that can be done on the quantitative specification theory, there are three immediate areas that come to mind. The first enhancement involves generalising the probabilistic contracts to those that deal with constraints rather than intervals.

Another area of practical importance for the specification theory relates to deriving efficient model checking algorithms. These algorithms should be able to verify properties of probabilistic contracts, but also with respect to the assume-guarantee rules that we will define.

Finally, the remaining consideration involves enriching the quantitative behaviours that can be modelled. As part of the specification theory we have proposed, we have focused on discrete probabilities with non-determinism. However, it would be advantageous to consider a probabilistic dense-timed variant, to express a range of QoS and performance properties, which is a natural extension of this framework.

3.2.4 Relating it back to CONNECT

The specification theory proposed supports the modelling of arbitrary components, rather than connectors. As we consider connectors to be a subclass of components, the formalism trivially permits the modelling of connectors. However, we will also aim to precisely characterise the allowable behaviours of connectors in CONNECT.

To this end, we propose adapting the preliminary (non-quantitative) algebra documented in Section 3.1 so that semantics of algebraic terms are given in terms of our newly proposed underlying specification theory. This would yield two levels of reasoning: the overarching syntactic level dealing with algebraic terms, and the underlying semantic level of quantitative automata and properties. In the long-ter, the role for the algebra will thus be to support not only the modelling of connectors, but also, via translation to appropriate notations studied in other workpackages, their verification, synthesis and dependability analysis.

4 Quantitative compositional verification

Assume-guarantee reasoning is an effective way to support independent development of heterogeneous component-based systems. In the previous chapter we proposed how to extend the preliminary CONNECT algebra to handle non-functional properties via AG-reasoning. The idea is to incorporate assume-guarantee properties within a specification theory for probabilistic interface automata. In this chapter we report on key results to make this possible, and specifically the continued development of AG-reasoning for probabilistic automata originally introduced in the first year deliverable D2.1 [17].

In the deliverable D2.1 [17] (also appeared in [51]), we have reported an assume-guarantee verification technique for CONNECTed systems. This is based on assume-guarantee queries of the form $\langle A\rangle_{\geq p_A} M \langle G\rangle_{\geq p_G}$, where the assumptions $\langle A\rangle_{\geq p_A}$ and guarantees $\langle G\rangle_{\geq p_G}$ are probabilistic safety properties. This technique above was the first fully-automated technique for compositional verification of systems exhibiting both probabilistic and nondeterministic behaviour. However, there are two limitations of this work. Firstly, it cannot handle liveness properties or reward properties. The latter is essential for the verification of non-functional properties. Examples can be found in Chapter 4 of the deliverable D5.2 [20]. Secondly, it requires non-trivial manual effort to create appropriate assumptions. In this chapter, we will address these two limitations. Section 4.1 provides the necessary information for understanding the rest of the chapter. In Section 4.2, we show how to verify probabilistic liveness properties and expected reward properties in a multi-objective assume-guarantee reasoning framework. In Section 4.3, we present a fully automated approach to generate the assumptions for reasoning probabilistic safety properties.

4.1 Preliminaries

In this section, we present the background knowledge about *probabilistic automata* (PAs) [68] and the assume-guarantee reasoning technique developed in the deliverable D2.1 [17]. Probabilistic automata model both nondeterministic and probabilistic behaviour, and are very similar to Markov decision processes (MDPs)[1]. For the purposes of verification, they can often be treated identically; however for compositional analysis (as in Section 4.2.2), the distinction becomes important. We provide more details here than in Section 2.1 on PAs[2]. Moreover, we augment these models with one or more *reward* structures that assign real values to certain transitions of the models in order to capture a wide range of quantitative measures of system behaviour, for example "number of time steps" when we consider latency of a CONNECTed system.

Definition 4.1 (Probabilistic automata) *A* probabilistic automaton *(PA) is a tuple* $\mathcal{M}{=}(S, \overline{s}, \alpha_{\mathcal{M}}, \delta_{\mathcal{M}})$ *where* S *is a set of states,* $\overline{s} \in S$ *is an initial state,* $\alpha_{\mathcal{M}}$ *is an alphabet and* $\delta_{\mathcal{M}} \subseteq S{\times}\alpha_{\mathcal{M}}{\times}Dist(S)$ *is a probabilistic transition relation.*

In a state s, a *transition* $s \xrightarrow{a} \mu$, where a is an *action* and μ is a distribution over states, is available if $(s, a, \mu) \in \delta_{\mathcal{M}}$. The selection of an available transition is nondeterministic and the subsequent choice of successor state is probabilistic, according to the distribution of the chosen transition.

A *path* is a sequence $\omega = s_0 \xrightarrow{a_0,\mu_0} s_1 \xrightarrow{a_1,\mu_1} \cdots$ where $s_0{=}\overline{s}$, $s_i \xrightarrow{a_i} \mu_i$ is an available transition and $\mu_i(s_{i+1}){>}0$ for all $i \in \mathbb{N}$. We denote by $IPaths$ ($FPaths$) the set of infinite (finite) paths. If ω is finite, $|\omega|$ denotes its length and $last(\omega)$ its last state. The trace, $t(\omega)$, of ω is the sequence of actions $a_0 a_1 \ldots$ and we use $t(\omega)\restriction_\alpha$ to indicate the projection of such a trace onto an alphabet $\alpha \subseteq \alpha_{\mathcal{M}}$.

A *reward structure* for $\mathcal{M}$ is a mapping $\rho : \alpha_\rho \to \mathbb{R}_{>0}$ from some alphabet $\alpha_\rho \subseteq \alpha_{\mathcal{M}}$ to the non-negative reals. We sometimes write $\rho(a) = 0$ to indicate that $a \notin \alpha_\rho$. For an infinite path $\omega = s_0 \xrightarrow{a_0,\mu_0} s_1 \xrightarrow{a_1,\mu_1} \cdots$, the *total reward* for ω over ρ is $\rho(\omega) = \sum_{i\in\mathbb{N}, a_i\in\alpha_\rho} \rho(a_i)$.

An *adversary* of $\mathcal{M}$ is a function $\sigma : FPaths{\to}Dist(\alpha_{\mathcal{M}}{\times}Dist(S))$ such that non-zero probabilities are assigned to only action-distribution pairs (a, μ) for which $(last(\omega), a, \mu) \in \delta_{\mathcal{M}}$. Employing standard techniques, for an adversary σ, we can construct a probability measure $Pr^\sigma_{\mathcal{M}}$ over $IPaths$. An adversary σ is *deterministic* if $\sigma(\omega)$ is a point distribution for all $\omega \in FPaths$, *memoryless* if $\sigma(\omega)$ depends only on $last(\omega)$ and has *finite-memory* if there is a finite automaton $\mathcal{A}_\sigma$ reading $FPaths$ such that $\sigma(\omega){=}\sigma(\bar{\omega})$

[1] For MDPs, $\delta_{\mathcal{M}}$ in Definition 4.1 becomes a partial function $S \times \alpha_{\mathcal{M}} \to Dist(S)$.

[2] Also note that we use different notations in this chapter to facilitate the formalisation of theories.

whenever $\mathcal{A}_\sigma$ ends in the same state after reading ω or $\bar{\omega}$. We let $Adv_{\mathcal{M}}$ denote the set of all adversaries for $\mathcal{M}$.

If $\mathcal{M}_i = (S_i, \overline{s}_i, \alpha_{\mathcal{M}_i}, \delta_{\mathcal{M}_i})$ for $i{=}1,2$, then their *parallel composition*, denoted $\mathcal{M}_1 \| \mathcal{M}_2$, is given by the PA $(S_1{\times}S_2, (\overline{s}_1, \overline{s}_2), \alpha_{\mathcal{M}_1} \cup \alpha_{\mathcal{M}_2}, \delta_{\mathcal{M}_1 \| \mathcal{M}_2})$ where $\delta_{\mathcal{M}_1 \| \mathcal{M}_2}$ is defined such that $(s_1, s_2) \xrightarrow{a} \mu_1 {\times} \mu_2$ if and only if one of the following holds:

- $s_1 \xrightarrow{a} \mu_1$, $s_2 \xrightarrow{a} \mu_2$ and $a \in \alpha_{\mathcal{M}_1} \cap \alpha_{\mathcal{M}_2}$
- $s_1 \xrightarrow{a} \mu_1$, $\mu_2 = \mu_{s_2}$ and $a \in \alpha_{\mathcal{M}_1} \backslash \alpha_{\mathcal{M}_2}$
- $s_2 \xrightarrow{a} \mu_2$, $\mu_1 = \mu_{s_1}$ and $a \in \alpha_{\mathcal{M}_2} \backslash \alpha_{\mathcal{M}_1}$.

When verifying systems of PAs composed in parallel, it is often essential to consider *fairness*. In this chapter, we use a simple but effective notion of fairness called *unconditional fairness*, in which it is required that each process makes a transition infinitely often. For probabilistic automata, a natural approach to incorporating fairness (as taken in, e.g., [9, 8]) is to restrict analysis of the system to a class of adversaries in which fair behaviour occurs with probability 1.

Definition 4.2 (Fair adversary) *If $\mathcal{M} = \mathcal{M}_1 \| \ldots \| \mathcal{M}_n$ is a PA comprising n components, then an (unconditionally)* fair path *of $\mathcal{M}$ is an infinite path $\omega \in IPaths$ in which, for each component $\mathcal{M}_i$, there exists an action $a \in \alpha_{\mathcal{M}_i}$ that appears infinitely often. A* fair adversary *σ of $\mathcal{M}$ is an adversary for which $Pr^\sigma_{\mathcal{M}}\{\omega \in IPaths \mid \omega \text{ is fair}\} = 1$. We let $Adv^{\text{fair}}_{\mathcal{M}}$ denote the set of fair adversaries of $\mathcal{M}$.*

Model Checking Probabilistic Automata

Now we present the monolithic model checking techniques for probabilistic and reward properties on PAs, and the assume-guarantee reasoning method for safety properties. In the rest of this section let $\mathcal{M} = (S, \overline{s}, \alpha_{\mathcal{M}}, \delta_{\mathcal{M}})$ be a PA.

Definition 4.3 (Probabilistic predicates) *A* probabilistic predicate *$[\phi]_{\sim p}$ comprises an ω-regular property $\phi \subseteq (\alpha_\phi)^\omega$ over some alphabet $\alpha_\phi \subseteq \alpha_{\mathcal{M}}$, a relational operator $\sim \in \{<, \leq, >, \geq\}$ and a rational probability bound p. Some examples of ω-regular properties are $\Box a$ and $\Diamond a$. The former means that the a is executed permanently and the latter means that eventually a should be executed. Satisfaction of $[\phi]_{\sim p}$ by $\mathcal{M}$, under adversary σ, denoted $\mathcal{M}, \sigma \models [\phi]_{\sim p}$, is defined as follows:*

$$\mathcal{M}, \sigma \models [\phi]_{\sim p} \Leftrightarrow Pr^\sigma_{\mathcal{M}}(\phi) {\sim} p \text{ where } Pr^\sigma_{\mathcal{M}}(\phi) \stackrel{\text{def}}{=} Pr^\sigma_{\mathcal{M}}(\{\omega {\in} IPaths \mid t(\omega)\restriction_{\alpha_\phi} {\in} \phi\}).$$

A *regular safety property* G represents a set of infinite words, denoted $\mathcal{L}(G)$, that is characterised by a regular language of *bad prefixes*, finite words of which any (possibly empty) extension is not in $\mathcal{L}(G)$. In practice, we define the set of bad prefixes for G by a (complete) deterministic finite automaton (DFA) G^{err} over alphabet α_G. The language $\mathcal{L}(G)$ is defined as $\mathcal{L}(G) = \{w \in (\alpha_G)^\omega \mid \text{ no prefix of } w \text{ is in } \mathcal{L}(G^{err})\}$ where $\mathcal{L}(G^{err}) \subseteq (\alpha_G)^+$ is the regular language for DFA G^{err}.

Given a PA $\mathcal{M}$ and regular safety property G with $\alpha_G \subseteq \alpha_{\mathcal{M}}$, an infinite path ω of $\mathcal{M}$ satisfies G, denoted $\omega \models G$, if $tr(\omega)\restriction_{\alpha_G} \in \mathcal{L}(G)$. For a finite path ω' of $\mathcal{M}$, we say that $\omega' \models G$ if some infinite path ω of which ω' is a prefix satisfies G. For an adversary $\sigma \in Adv_{\mathcal{M}}$, we define the probability of $\mathcal{M}$ under σ satisfying G as:

$$Pr^\sigma_{\mathcal{M}}(G) \stackrel{\text{def}}{=} Pr^\sigma_{\mathcal{M}}\{\omega \in Path^\sigma_{\mathcal{M}} \mid \omega \models G\}$$

We then define the minimum probability of satisfying G as:

$$Pr^{\min}_{\mathcal{M}}(G) \stackrel{\text{def}}{=} \inf_{\sigma \in Adv_{\mathcal{M}}} Pr^\sigma_{\mathcal{M}}(G)$$

A *probabilistic safety property* $\langle G \rangle_{\geq p_G}$ comprises a regular safety property G and a rational probability bound p_G. We say that a PA $\mathcal{M}$ satisfies the property, denoted $\mathcal{M} \models \langle G \rangle_{\geq p_G}$, if the probability of satisfying G is at least p_G for any adversary:

$$\begin{array}{rcl} \mathcal{M} \models \langle G \rangle_{\geq p_G} & \Leftrightarrow & \forall \sigma {\in} Adv_{\mathcal{M}} \,.\, Pr^\sigma_{\mathcal{M}}(G) \geq p_G \\ & \Leftrightarrow & Pr^{\min}_{\mathcal{M}}(G) \geq p_G \,. \end{array}$$

Model checking a probabilistic safety property $\langle G \rangle_{\geq p_G}$ on a PA $\mathcal{M}$ requires computation of the minimum probability $Pr^{\min}_{\mathcal{M}}(G)$. This reduces, using standard automata-based techniques for probabilistic model checking, to calculating the maximum probability of reaching a set of accepting states in the PA $\mathcal{M} \otimes G^{err}$ formed as the product of PA $\mathcal{M}$ and the DFA G^{err} representing G (see [51] for details).

To check probabilistic safety properties on PAs, it suffices to consider deterministic, finite-memory adversaries, that is to say there always exists such an adversary σ for which $Pr^{\sigma}_{\mathcal{M}}(G) = Pr^{\min}_{\mathcal{M}}(G)$.

Note also that, in the special case of *qualitative* safety properties, the following holds:

$$\mathcal{M} \models \langle G \rangle_{\geq 1} \quad \Leftrightarrow \quad \forall \omega \in Path_{\mathcal{M}} \,.\, \omega \models G$$

which can be checked with a simple graph analysis.

Example. Consider a simple safety property G "action $fail$ never occurs" for the example in Section 3.2. The property holds in the model with at least probability 0.98, i.e., $M_1 \| Connector \| M_2 \models \langle G \rangle_{\geq 0.98}$. This can be proved by verifying

1. $\langle \texttt{true} \rangle \, M_1 \, \langle A_1 \rangle_{\geq 0.8}$, where property A_1 means that "$warn$ occurs before $shutdown$",
2. $\langle A_1 \rangle_{\geq 0.8} \, Connector \, \langle A_2 \rangle_{\geq 0.8}$, where property A_2 means that "$alert$ occurs before $switchoff$",
3. $\langle A_2 \rangle_{\geq 0.8} \, M_2 \, \langle G \rangle_{\geq 0.98}$,

and applying the (ASYM-N) rule [51]

$$\frac{\begin{array}{c}\langle \texttt{true} \rangle \, \mathcal{M}_1 \, \langle A_1 \rangle_{\geq p_1} \\ \langle A_1 \rangle_{\geq p_1} \, \mathcal{M}_2 \, \langle A_2 \rangle_{\geq p_2} \\ \cdots \\ \langle A_{n-1} \rangle_{\geq p_{n-1}} \, \mathcal{M}_n \, \langle G \rangle_{\geq p_G}\end{array}}{\langle \texttt{true} \rangle \, \mathcal{M}_1 \| \cdots \| \mathcal{M}_n \, \langle G \rangle_{\geq p_G}} \qquad \text{(ASYM-N)}.$$

Definition 4.4 (Reward predicates) *A* reward predicate $[\rho]_{\sim r}$ *comprises a reward structure* $\rho : \alpha_\rho \to \mathbb{R}_{>0}$ *over some alphabet* $\alpha_\rho \subseteq \alpha_{\mathcal{M}}$*, a relational operator* $\sim \in \{<, \leq, >, \geq\}$ *and a rational reward bound* r*. Satisfaction of* $[\rho]_{\sim r}$ *by* $\mathcal{M}$*, under adversary* σ*, denoted* $\mathcal{M}, \sigma \models [\rho]_{\sim r}$*, is defined as follows:*

$$\mathcal{M}, \sigma \models [\rho]_{\sim r} \;\Leftrightarrow\; ExpTot^{\sigma}_{\mathcal{M}}(\rho) \sim r \;\; \textit{where} \;\; ExpTot^{\sigma}_{\mathcal{M}}(\rho) \stackrel{\text{def}}{=} \int_{\omega} \rho(\omega) \, dPr^{\sigma}_{\mathcal{M}}.$$

The standard approach to verifying PAs is to quantify over all adversaries. For example, $\mathcal{M}$ satisfies probabilistic predicate $[\phi]_{\sim p}$, denoted $\mathcal{M} \models [\phi]_{\sim p}$ when:

$$\mathcal{M} \models [\phi]_{\sim p} \;\Leftrightarrow\; \forall \sigma \in Adv_{\mathcal{M}} \,.\, \mathcal{M}, \sigma \models [\phi]_{\sim p} \,.$$

In similar fashion, we can verify a multi-component PA $\mathcal{M}_1 \| \ldots \| \mathcal{M}_n$ *under fairness* by quantifying only over fair adversaries:

$$\mathcal{M}_1 \| \ldots \| \mathcal{M}_n \models_{fair} [\phi]_{\sim p} \;\Leftrightarrow\; \forall \sigma \in Adv^{\text{fair}}_{\mathcal{M}_1 \| \ldots \| \mathcal{M}_n} \,.\, \mathcal{M}_1 \| \ldots \| \mathcal{M}_n, \sigma \models [\phi]_{\sim p} \,.$$

Verifying whether $\mathcal{M}$ satisfies a probabilistic predicate $[\phi]_{\sim p}$ or reward predicate $[\rho]_{\sim r}$ can be done with, for example, the techniques in [21, 22]. In the remainder of this section, we give further details of the case for ω-regular properties, since we need these later in this chapter. We follow the approach of [22], which is based on the use of *deterministic Rabin automata* and *end components*.

An *end component* (EC) of $\mathcal{M}$ is a pair $(S', \delta'_{\mathcal{M}})$ comprising a subset $S' \subseteq S$ of states and a probabilistic transition relation $\delta'_{\mathcal{M}} \subseteq \delta_{\mathcal{M}}$ that is strongly connected when restricted to S' and closed under probabilistic branching, i.e., $\{s \in S \mid \exists (s, a, \mu) \in \delta'_{\mathcal{M}}\} \subseteq S'$ and $\{s' \in S \mid \mu(s') > 0$ for some $(s, a, \mu) \in \delta_{\mathcal{M}}\} \subseteq S'$. An EC $(S', \delta'_{\mathcal{M}})$ is *maximal* if there is no EC $(S'', \delta''_{\mathcal{M}})$ such that $\delta'_{\mathcal{M}} \subsetneq \delta''_{\mathcal{M}}$.

A *deterministic Rabin automaton* (DRA) is a tuple $\mathcal{A} = (Q, \overline{q}, \alpha, \delta, Acc)$ of states Q, initial state $\overline{q}$, alphabet α, transition function $\delta : Q \times \alpha \to Q$, and acceptance condition $Acc = \{(L_i, K_i)\}_{i=1}^{k}$ with $L_i, K_i \subseteq Q$. Any infinite word $w \in (\alpha)^{\omega}$ has a unique corresponding run $\overline{q} \, q_1 q_2 \ldots$ through $\mathcal{A}$ and we say that $\mathcal{A}$

accepts w if the run contains, for some $i \in \mathbb{N}$, finitely many states from L_i and infinitely many from K_i. For any ω-regular property $\phi \subseteq (\alpha_\phi)^\omega$ we can construct a DRA, say $\mathcal{A}_\phi$, over α_ϕ that accepts precisely ϕ.

Verification of $[\phi]_{\sim p}$ on $\mathcal{M}$ is done by constructing the *product* of $\mathcal{M}$ and $\mathcal{A}_\phi$, and then identifying *accepting end components*. The *product* $\mathcal{M} \otimes \mathcal{A}_\phi$ of $\mathcal{M}$ and DRA $\mathcal{A}_\phi = (Q, \overline{q}, \alpha_\mathcal{M}, \delta, \{(L_i, K_i)\}_{i=1}^k)$ is the PA $(S \times Q, (\overline{s}, \overline{q}), \alpha_\mathcal{M}, \delta_{\mathcal{M}\otimes\mathcal{A}_\phi})$ where for all $(s, a, \mu) \in \delta_\mathcal{M}$ there is $((s, q), a, \mu') \in \delta_{\mathcal{M}\otimes\mathcal{A}_\phi}$ such that $\mu'(s', q') = \mu(s')$ for $q' = \delta(q, a)$ and all $s' \in S$. An *accepting EC* for ϕ in $\mathcal{M} \otimes \mathcal{A}_\phi$ is an EC $(S', \delta'_{\mathcal{M}\otimes\mathcal{A}_\phi})$ for which, there exists an $1 \leq i \leq k$ such that the set of states S', when projected onto Q, contain some state from K_i, but no states from L_i. To verify, for example, a probabilistic predicate $[\phi]_{\sim p}$ where $\sim \in \{<, \leq\}$, we have:

$$\mathcal{M} \models [\phi]_{\sim p} \quad \Leftrightarrow \quad \mathcal{M} \otimes \mathcal{A}_\phi \models [\Diamond T]_{\sim p}$$

where T is the union of states of accepting ECs for ϕ in $\mathcal{M} \otimes \mathcal{A}_\phi$.

Verification of such properties under fairness, e.g. checking $\mathcal{M} \models_{fair} [\phi]_{\sim p}$, can be done by further restricting the set of accepting ECs. For details, see [8], which describes verification of PAs under strong and weak fairness conditions, of which unconditional fairness is a special case.

Probabilistic Assume-Guarantee Reasoning for Safety Properties

In this section, we use the compositional verification framework of [51]. This is based on the assume-guarantee paradigm, where assumptions and guarantees are probabilistic safety properties. The key idea is the notion of *probabilistic assume-guarantee triples* of the form $\langle A \rangle_{\geq p_A} \mathcal{M} \langle G \rangle_{\geq p_G}$, in which $\langle A \rangle_{\geq p_A}$ and $\langle G \rangle_{\geq p_G}$ are probabilistic safety properties and $\mathcal{M}$ is a PA. Informally, the meaning of this is "whenever $\mathcal{M}$ is part of a system satisfying A with probability at least p_A, then the system will satisfy G with probability at least p_G". Formally:

Definition 4.5 *If $\langle A \rangle_{\geq p_A}$ and $\langle G \rangle_{\geq p_G}$ are probabilistic safety properties, $\mathcal{M}$ is a PA and $\alpha_G \subseteq \alpha_A \cup \alpha_\mathcal{M}$, then:*

$$\langle A \rangle_{\geq p_A} \mathcal{M} \langle G \rangle_{\geq p_G}$$
$$\Leftrightarrow$$
$$\forall \sigma \in Adv_{\mathcal{M}[\alpha_A]}. \left(Pr^\sigma_{\mathcal{M}[\alpha_A]}(A) \geq p_A \Rightarrow Pr^\sigma_{\mathcal{M}[\alpha_A]}(G) \geq p_G \right)$$

where $\mathcal{M}[\alpha_A]$ is $\mathcal{M}$ with its alphabet extended to include α_A.

Determining whether an assume-guarantee triple holds reduces to *multi-objective* probabilistic model checking [51, 35], which can be solved efficiently by solving an LP problem. In the absence of an assumption (denoted by $\langle \texttt{true} \rangle$) checking a triple reduces to normal model checking:

$$\langle \texttt{true} \rangle \, \mathcal{M} \, \langle G \rangle_{\geq p_G} \Leftrightarrow \mathcal{M} \models \langle G \rangle_{\geq p_G}$$

This reduction holds because $\langle G \rangle_{\geq p_G}$ is a safety property. We also note that, for the case of *qualitative* assumptions of the form $\langle A \rangle_{\geq 1}$, the LP problem can be bypassed and checking the triple $\langle A \rangle_{\geq 1} \mathcal{M} \langle G \rangle_{\geq p_G}$ also reduces to normal (probabilistic) model checking.

Using these definitions, [51] presents several proof rules for compositional probabilistic verification. In this deliverable, we focus on the following *asymmetric* rule (ASYM). For PAs $\mathcal{M}_1, \mathcal{M}_2$, probabilistic safety properties $\langle A \rangle_{\geq p_A}, \langle G \rangle_{\geq p_G}$ such that $\alpha_A \subseteq \alpha_{\mathcal{M}_1}$ and $\alpha_G \subseteq \alpha_{\mathcal{M}_2} \cup \alpha_A$:

$$\frac{\begin{array}{c}\langle \texttt{true} \rangle \, \mathcal{M}_1 \, \langle A \rangle_{\geq p_A} \\ \langle A \rangle_{\geq p_A} \, \mathcal{M}_2 \, \langle G \rangle_{\geq p_G}\end{array}}{\langle \texttt{true} \rangle \, \mathcal{M}_1 \parallel \mathcal{M}_2 \, \langle G \rangle_{\geq p_G}} \qquad \textit{(ASYM)}$$

Thus, given an appropriate probabilistic assumption $\langle A \rangle_{\geq p_A}$, verifying that $\mathcal{M}_1 \| \mathcal{M}_2 \models \langle G \rangle_{\geq p_G}$ can be done compositionally by model checking a safety property on $\mathcal{M}_1$ (for premise 1) and an assume guarantee triple on $\mathcal{M}_2$ (for premise 2).

We can also check *quantitative* assume-guarantee triples. As shown in [51], for a PA $\mathcal{M}$, regular safety properties A, G and a fixed value of p_A, we can compute (again through multi-objective model checking) the tightest lower-bounded interval $I_G \subseteq [0, 1]$ for which the triple $\langle A \rangle_{\geq p_A} \mathcal{M} \langle G \rangle_{I_G}$ holds. Conversely, for

a fixed value of p_G, we can compute the widest lower-bounded interval $I_A \subseteq [0,1]$ for which the triple $\langle A \rangle_{I_A} \mathcal{M} \langle G \rangle_{\geq p_G}$ holds. We denote these two queries, respectively, as:

$$\langle A \rangle_{\geq p_A} \mathcal{M} \langle G \rangle_{I_G=?} \text{ and } \langle A \rangle_{I_A=?} \mathcal{M} \langle G \rangle_{\geq p_G}$$

Intuitively, these allow us to compute the strongest possible guarantee that can be obtained for some assumption $\langle A \rangle_{\geq p_A}$ and the weakest possible assumption[3] that guarantees a particular $\langle G \rangle_{\geq p_G}$. Note that, for the latter type of query, I_A can in fact be empty. This occurs when there are adversaries of $\mathcal{M}$ that satisfy $\langle A \rangle_{\geq 1}$ but violate $\langle G \rangle_{\geq p_G}$, i.e. even under the strongest possible assumption $\langle A \rangle_{\geq 1}$ for A, we cannot guarantee that $\langle G \rangle_{\geq p_G}$ holds.

4.2 Assume-guarantee reasoning for non-functional properties

In this section, we handle the first limitation of the framework proposed in deliverable D2.1 [17], i.e., an extension to allow the verification of probabilistic liveness properties and expected reward properties, by considering verification problems for probabilistic automata on properties with *multiple*, *quantitative* objectives [39].

Most of stochastic model checking techniques only deal with one property at a time. A natural extension of these techniques is to consider *multiple objectives* [35]. For example, rather than verifying two separate properties such as "message loss occurs with probability at most 0.001" and "the expected latency is below 50 units of time", we might ask whether it is possible to satisfy both properties *simultaneously*, or to investigate the *trade-off* between the two objectives as some parameters of the system are varied. In Section 4.2.1, we first define a language that expresses Boolean combinations of probabilistic ω-regular properties (which subsumes e.g. LTL) and expected total reward measures. We then present, for properties expressed in this language, techniques both to *verify* that a property holds for all adversaries (strategies) of a PA and to *synthesise* an adversary of a PA under which a property holds. We also consider *numerical* queries, which yield an optimal value for one objective, subject to constraints imposed on one or more other objectives. This is done via reduction to a linear programming problem, which can be solved efficiently. It takes time polynomial in the size of the model and doubly exponential in the size of the property (for LTL objectives), i.e. the same as for the single-objective case [21]. In Section 4.2.2, we use the multi-objective queries to verify probabilistic liveness properties and reward properties via assume-guarantee reasoning.

4.2.1 Quantitative Multi-Objective Verification

We define a language for expressing multiple quantitative objectives of a probabilistic automaton. We then describe, for properties expressed in this language, techniques both to *verify* that the property holds for all adversaries of a PA and to *synthesise* an adversary of a PA under which the property holds. We also consider *numerical* queries, which yield an optimal value for one objective, subject to constraints imposed on one or more other objectives.

Definition 4.6 (Quantitative multi-objective properties) *A* quantitative multi-objective property *(qmo-property) for a PA $\mathcal{M}$ is a Boolean combination of probabilistic and reward predicates, i.e. an expression produced by the grammar:*

$$\Psi ::= \texttt{true} \mid \Psi \wedge \Psi \mid \Psi \vee \Psi \mid \neg\Psi \mid [\phi]_{\sim p} \mid [\rho]_{\sim r}$$

where $[\phi]_{\sim p}$ and $[\rho]_{\sim r}$ are probabilistic and reward predicates for $\mathcal{M}$, respectively. A simple *qmo-property comprises a single conjunction of predicates, i.e. is of the form $(\wedge_{i=1}^{n}[\phi_i]_{\sim_i p_i}) \wedge \left(\wedge_{j=1}^{m}[\rho_j]_{\sim_j r_j}\right)$. We refer to the predicates occurring in a formula as* objectives*. For property Ψ_P, we use α_P to denote the set of actions used in Ψ_P, i.e. the union of α_ϕ and α_ρ over $[\phi]_{\sim p}$ and $[\rho]_{\sim r}$ occurring in Ψ_P.*

A quantitative multi-objective property Ψ is evaluated over a PA $\mathcal{M}$ and an adversary σ of $\mathcal{M}$. We say that $\mathcal{M}$ satisfies Ψ under σ, denoted $\mathcal{M}, \sigma \models \Psi$ if Ψ evaluates to $\texttt{true}$ when substituting each predicate x with the result of $\mathcal{M}, \sigma \models x$. *Verification* of Ψ over a PA $\mathcal{M}$ is defined as follows.

[3] Not to be confused with the "weakest assumption" defined in [62].

Definition 4.7 (Verification queries) *For a PA $\mathcal{M}$ and a qmo-property Ψ, a* verification query *asks whether Ψ is satisfied under all adversaries of $\mathcal{M}$:*

$$\mathcal{M} \models \Psi \;\Leftrightarrow\; \forall \sigma \in Adv_{\mathcal{M}} \,.\, \mathcal{M}, \sigma \models \Psi$$

For a *simple* qmo-property Ψ, we can verify whether $\mathcal{M} \models \Psi$ using standard techniques [21, 22] (since each conjunct can be verified separately). To treat the general case, we will use multi-objective model checking, proceeding via a reduction to the dual notion of *achievability queries*.

Definition 4.8 (Achievability queries) *For a PA $\mathcal{M}$ and qmo-property Ψ, an* achievability query *asks if there exists a satisfying adversary of $\mathcal{M}$, i.e. whether there exists $\sigma \in Adv_{\mathcal{M}}$ such that $\mathcal{M}, \sigma \models \Psi$.*

Remark. Since qmo-properties are closed under negation, we can convert any *verification* query into an equivalent (negated) *achievability* query. Furthermore, any qmo-property, can be translated to an equivalent disjunction of *simple* qmo-properties (obtained by converting to disjunctive normal form and pushing negation into predicates, e.g. $\neg([\phi]_{>p}) \equiv [\phi]_{\leq p}$).

In practice, it is also often useful to obtain the minimum/maximum *value* of an objective, subject to constraints on others. For this, we use *numerical queries*.

Definition 4.9 (Numerical queries) *For a PA $\mathcal{M}$, qmo-property Ψ and ω-regular property ϕ or reward structure ρ, a* (maximising) numerical query *is:*

$$\begin{array}{rcl} Pr^{\max}_{\mathcal{M}}(\phi \,|\, \Psi) & \stackrel{\text{def}}{=} & \sup\{Pr^{\sigma}_{\mathcal{M}}(\phi) \mid \sigma \in Adv_{\mathcal{M}} \wedge \mathcal{M}, \sigma \models \Psi\}, \\ or\ ExpTot^{\max}_{\mathcal{M}}(\rho \,|\, \Psi) & \stackrel{\text{def}}{=} & \sup\{ExpTot^{\sigma}_{\mathcal{M}}(\rho) \mid \sigma \in Adv_{\mathcal{M}} \wedge \mathcal{M}, \sigma \models \Psi\}. \end{array}$$

If the property Ψ is not satisfied by any adversary of $\mathcal{M}$, these queries return $\perp$. A minimising numerical query *is defined similarly.*

Before describing our techniques to check verification, achievability and numerical queries, we first need to discuss some *assumptions* made about PAs. One of the main complications when introducing rewards into multi-objective queries is the possibility of *infinite* expected total rewards. For the classical, single-objective case (see e.g. [22]), it is usual to impose assumptions so that such behaviour does not occur. For the multi-objective case, the situation is more subtle, and requires careful treatment. We now outline what assumptions should be imposed; later we describe how they can be checked algorithmically.

A key observation is that, if we allow arbitrary reward structures, situations may occur where extremely improbable (but non-zero probability) behaviour has infinite reward. Consider e.g. the PA $(\{s_0, s_1, s_2\}, s_0, \{a, b\}, \delta)$ with $\delta = \{(s_0, b, \mu_{s_1}), (s_0, b, \mu_{s_2}), (s_1, a, \mu_{s_1}), (s_2, b, \mu_{s_2})\}$, reward structure $\rho = \{a \mapsto 1\}$, and the qmo-property $\Psi = [\Box b]_{\geq p} \wedge [\rho]_{\geq r}$. For p arbitrarily close to 1, there is an adversary satisfying Ψ for any $r \in \mathbb{R}_{\geq 0}$, because it suffices to take action a with non-zero probability. This rather unnatural behaviour would lead to misleading verification results, masking possible errors in the model design.

Motivated by such problems, we enforce the restriction below on multi-objective queries. To match the contents of the next subsection, we state this for a maximising numerical query on rewards. We describe *how* to check the restriction holds in the next subsection.

Assumption 1 *Let $ExpTot^{\max}_{\mathcal{M}}(\rho \,|\, \Psi)$ be a numerical query for a PA $\mathcal{M}$ and qmo-property Ψ which is a disjunction*[4] *of simple qmo-properties $\Psi_1, \ldots, \Psi_l$. If $\Psi_k = (\wedge_{i=1}^{n} [\phi_i]_{\sim_i p_i}) \wedge \left(\wedge_{j=1}^{m} [\rho_j]_{\sim_j r_j}\right)$, then we require that:*

$$\sup\{ExpTot^{\sigma}_{\mathcal{M}}(\zeta) \mid \mathcal{M}, \sigma \models \textstyle\bigwedge_{i=1}^{n} [\phi]_{\sim_i p_i}\} < \infty$$

for all $\zeta \in \{\rho\} \cup \{\rho_j \,|\, 1 \leq j \leq m \wedge\ \sim_j \in \{>, \geq\}\}$.

[4] This assumption extends to arbitrary properties Ψ by, as described earlier, first reducing to disjunctive normal form.

Checking Multi-Objective Queries

We now describe techniques for checking the multi-objective queries described previously. For presentational purposes, we focus on *numerical* queries. It is straightforward to adapt this to *achievability* queries by introducing, and then ignoring, a dummy property to maximise (with no loss in complexity). As mentioned earlier, *verification* queries are directly reducible to achievability queries.

Let $\mathcal{M}$ be a PA and $ExpTot^{\max}_{\mathcal{M}}(\rho \,|\, \Psi)$ be a maximising numerical query for reward structure ρ (the cases for minimising queries and ω-regular properties are analogous). As discussed earlier, we can convert Ψ to a disjunction of simple qmo-properties. Clearly, we can treat each element of the disjunction separately and then take the maximum. So, without loss of generality, we assume that Ψ is simple, i.e. $\Psi = (\wedge_{i=1}^{n}[\phi_i]_{\sim_i p_i}) \wedge (\wedge_{j=1}^{m}[\rho_j]_{\sim_j r_j})$. Furthermore, we assume that each $\sim_i$ is $\geq$ or $>$ (which we can do by changing e.g. $[\phi]_{<p}$ to $[\neg\phi]_{>1-p}$).

Our technique to compute $ExpTot^{\max}_{\mathcal{M}}(\rho \,|\, \Psi)$ proceeds via a sequence of modifications to $\mathcal{M}$, producing a PA $\hat{\mathcal{M}}$. From this, we construct a linear program $L(\hat{\mathcal{M}})$, whose solution yields both the desired numerical result and a corresponding adversary $\hat{\sigma}$ of $\hat{\mathcal{M}}$. Crucially, $\hat{\sigma}$ is *memoryless* and can thus be mapped to a matching *finite-memory* adversary of $\mathcal{M}$. The structure of $L(\hat{\mathcal{M}})$ is very similar to the one used in [36], but many of the steps to construct $\hat{\mathcal{M}}$ and the techniques to establish a memoryless adversary are substantially different. We also remark that, although not discussed here, $L(\hat{\mathcal{M}})$ can be adapted to a multi-objective linear program, or used to approximate the Pareto curve between objectives.

In the remainder of this section, we describe the process in detail, which comprises 4 steps: **1.** checking Assumption 1; **2.** building a PA $\bar{\mathcal{M}}$ in which unneccessary actions are removed; **3.** converting $\bar{\mathcal{M}}$ to a PA $\hat{\mathcal{M}}$; **4.** building and solving the linear program $L(\hat{\mathcal{M}})$.

Step 1. We start by constructing a PA $\mathcal{M}^{\phi} = \mathcal{M} \otimes \mathcal{A}_{\phi_1} \otimes \cdots \otimes \mathcal{A}_{\phi_n}$ which is the product of $\mathcal{M}$ and a DRA $\mathcal{A}_{\phi_i}$ for each ω-regular property ϕ_i appearing in Ψ. We check Assumption 1 by analysing $\mathcal{M}^{\phi}$: for each *maximising* reward structure ζ (i.e. letting $\zeta{=}\rho$ or $\zeta{=}\rho_j$ when $\sim_j \in \{>, \geq\}$) we use the proposition below. This requires a simpler multi-objective achievability query on probabilistic predicates only. In fact, this can be done with the techniques of [36].

Proposition 4.10 *We have* $\sup\{ExpTot^{\sigma}_{\mathcal{M}}(\zeta) \mid \mathcal{M}, \sigma \models \bigwedge_{i=1}^{n}[\phi]_{\sim_i p_i}\} = \infty$ *for a reward structure* ζ *of* $\mathcal{M}$ *iff there is an adversary* σ *of* $\mathcal{M}^{\phi}$ *such that* $\mathcal{M}^{\phi}, \sigma \models [\Diamond pos]_{>0} \wedge \bigwedge_{i=1}^{n}[\phi]_{\sim_i p_i}$ *where "pos" labels any transition* (s, a, μ) *that satisfies* $\zeta(a){>}0$ *and is contained in an EC.*

Step 2. Next, we build the PA $\bar{\mathcal{M}}$ from $\mathcal{M}^{\phi}$ by removing actions that will not be used by any adversary which satisfies Ψ and maximises the expected value for the reward ρ. Let $\mathcal{M}^{\phi} = (S^{\phi}, \overline{s}, \alpha_{\mathcal{M}}, \delta^{\phi}_{\mathcal{M}})$. Then $\bar{\mathcal{M}} = (\bar{S}, \overline{s}, \alpha_{\mathcal{M}}, \bar{\delta}_{\mathcal{M}})$ is the PA obtained from $\mathcal{M}^{\phi}$ as follows. First, we remove (s, a, μ) from $\delta^{\phi}_{\mathcal{M}}$ if it is contained in an EC and $\zeta(a){>}0$ for some *maximising* reward structure ζ. Second, we repeatedly remove states with no outgoing transitions and transitions that lead to non-existent states, until a fixpoint is reached.

Proposition 4.11 *There is an adversary* σ *of* $\mathcal{M}^{\phi}$ *where* $ExpTot^{\sigma}_{\mathcal{M}}(\rho){=}x$ *and* $\mathcal{M}^{\phi}, \sigma \models \Psi$ *iff there is an adversary* $\bar{\sigma}$ *of* $\bar{\mathcal{M}}$ *where* $ExpTot^{\bar{\sigma}}_{\bar{\mathcal{M}}}(\rho){=}x$ *and* $\bar{\mathcal{M}}, \bar{\sigma} \models \Psi$.

Step 3. Then, we construct PA $\hat{\mathcal{M}}$ from $\bar{\mathcal{M}}$, by converting the n probabilistic predicates $[\phi_i]_{\sim_i p_i}$ into n reward predicates $[\lambda_i]_{\sim_i p_i}$. For each $R \subseteq \{1, \ldots, n\}$, we let S_R denote the set of states that are contained in an EC $(S', \delta'_{\bar{\mathcal{M}}})$ that: (i) is accepting for all $\{\phi_i \mid i \in R\}$; (ii) satisfies $\rho_j(a) = 0$ for all $1 \leq j \leq m$ and $(s, a, \mu) \in \delta'_{\bar{\mathcal{M}}}$. Thus, in each S_R, no reward is gained and almost all paths satisfy the ω-regular properties ϕ_i for $i \in R$.

We then construct $\hat{\mathcal{M}}$ by adding a new terminal state s_{dead} and adding transitions from states in each S_R to s_{dead}, labelled with a new action a^R. Intuitively, taking an action a^R in $\hat{\mathcal{M}}$ corresponds to electing to remain forever in the corresponding EC of $\bar{\mathcal{M}}$. Formally, $\hat{\mathcal{M}} = (\hat{S}, \overline{s}, \hat{\alpha}_{\mathcal{M}}, \hat{\delta}_{\mathcal{M}})$ where $\hat{S} = \bar{S} \cup \{s_{dead}\}$, $\hat{\alpha}_{\mathcal{M}} = \alpha_{\mathcal{M}} \cup \{a^R \mid R \subseteq \{1, \ldots, n\}\}$, and $\hat{\delta}_{\mathcal{M}} = \bar{\delta}_{\mathcal{M}} \cup \{(s, a^R, \eta_{s_{dead}}) \mid s \in S_R\}$. Finally, we create, for each $1 \leq i \leq n$, a reward structure $\lambda_i : \{a^R \mid i \in R\} \to \mathbb{R}_{>0}$ with $\lambda_i(a^R) = 1$ whenever λ_i is defined.

Proposition 4.12 *There is an adversary* $\bar{\sigma}$ *of* $\bar{\mathcal{M}}$ *such that* $ExpTot^{\bar{\sigma}}_{\bar{\mathcal{M}}}(\rho){=}x$ *and* $\bar{\mathcal{M}}, \bar{\sigma} \models \Psi$ *iff there is a memoryless adversary* $\hat{\sigma}$ *of* $\hat{\mathcal{M}}$ *such that* $ExpTot^{\hat{\sigma}}_{\hat{\mathcal{M}}}(\rho){=}x$ *and* $\hat{\mathcal{M}}, \hat{\sigma} \models (\wedge_{i=1}^{n}[\lambda_i]_{\sim_i p_i}) \wedge (\wedge_{j=1}^{m}[\rho_j]_{\sim_j r_j}) \wedge ([\Diamond s_{dead}]_{\geq 1})$.

Maximise $\sum_{(s,a,\mu)\in\hat{\delta}_{\mathcal{M}}, s\neq s_{dead}} \rho(a)\cdot y_{(s,a,\mu)}$ *subject to:*

$$\begin{array}{rll}
\sum_{(s,a^R,\mu)\in\hat{\delta}_{\mathcal{M}}, s\neq s_{dead}} y_{(s,a^R,\mu)} & = 1 & \\
\sum_{(s,a,\mu)\in\hat{\delta}_{\mathcal{M}}, s\neq s_{dead}} \lambda_i(a)\cdot y_{(s,a,\mu)} & \sim_i p_i & \text{for all } 1\leq i\leq n \\
\sum_{(s,a,\mu)\in\hat{\delta}_{\mathcal{M}}, s\neq s_{dead}} \rho_j(a)\cdot y_{(s,a,\mu)} & \sim_j r_j & \text{for all } 1\leq j\leq m \\
\sum_{(s,a,\mu)\in\hat{\delta}_{\mathcal{M}}} y_{(s,a,\mu)} - \sum_{(\hat{s},\hat{a},\hat{\mu})\in\hat{\delta}_{\mathcal{M}}} \mu'(s)\cdot y_{(\hat{s},\hat{a},\hat{\mu})} & = init(s) & \text{for all } s\in\hat{S}\backslash\{s_{dead}\} \\
y_{(s,a,\mu)} & \geq 0 & \text{for all } (s,a,\mu)\in\hat{\delta}_{\mathcal{M}}
\end{array}$$

where $init(s)$ *is* 1 *if* $s=\bar{s}$ *and* 0 *otherwise.*

Figure 4.1: The linear program $L(\hat{\mathcal{M}})$

Step 4. Finally, we create a linear program $L(\hat{\mathcal{M}})$, given in Figure 4.1, which encodes the structure of $\hat{\mathcal{M}}$ as well as the objectives from Ψ. Intuitively, in a solution of $L(\hat{\mathcal{M}})$, the variables $y_{(s,a,\mu)}$ express the expected number of times that state s is visited and transition $s \xrightarrow{a} \mu$ is taken subsequently. The expected total reward w.r.t. ρ_i is then captured by $\sum_{(s,a,\mu)\in\hat{\delta}_{\mathcal{M}}, s\neq s_{dead}} \rho_i(a)\cdot y_{(s,a,\mu)}$. The result of $L(\hat{\mathcal{M}})$ yields the desired value for our numerical query.

Proposition 4.13 *For* $x \in \mathbb{R}_{\geq 0}$*, there is a memoryless adversary* $\hat{\sigma}$ *of* $\hat{\mathcal{M}}$ *where* $ExpTot^{\hat{\sigma}}_{\hat{\mathcal{M}}}(\rho)=x$ *and* $\hat{\mathcal{M}},\hat{\sigma}\models(\wedge_{i=1}^{n}[\lambda_i]_{\sim_i p_i})\wedge(\wedge_{j=1}^{m}[\rho_j]_{\sim_j r_j})\wedge([\diamond s_{dead}]_{\geq 1})$ *iff there is a feasible solution* $(y^{\star}_{(s,a,\mu)})_{(s,a,\mu)\in\hat{\delta}_{\mathcal{M}}}$ *of the linear program* $L(\hat{\mathcal{M}})$ *such that* $\sum_{(s,a,\mu)\in\hat{\delta}_{\mathcal{M}}, s\neq s_{dead}} \rho_i(a)\cdot y^{\star}_{(s,a,\mu)} = x$*.*

In addition, a solution to $L(\hat{\mathcal{M}})$ gives a memoryless adversary σ_{prod} defined by $\sigma_{prod}(s)(a,\mu) = \frac{y_{(s,a,\mu)}}{\sum_{a',\mu'}(s,a',\mu')}$. This can be converted into an adversary σ' for $\mathcal{M}^{\phi}$ by following σ up to the point where a^R is to be taken and, instead of taking a^R, staying in an EC witnessing that the state preceding a^R in the history is in S_R. Adversary σ' can be translated into an adversary σ of $\mathcal{M}$ by $\sigma(\omega)=\sigma'((\omega,q_1,\ldots,q_n))$ where q_i is the state where $\mathcal{A}_{\phi_i}$ ends on reading $t(\omega)$.
The following is then a direct consequence of Propositions 4.11, 4.12 and 4.13.

Theorem 4.14 *Given a PA* $\mathcal{M}$ *and numerical query* $ExpTot^{\max}_{\mathcal{M}}(\rho\,|\,\Psi)$ *satisfying Assumption 1, the result of the query is equal to the solution of the linear program* $L(\hat{\mathcal{M}})$ *(see Figure 4.1). Furthermore, this requires time polynomial in the size of* $\mathcal{M}$ *and doubly exponential in the size of the property (for LTL objectives).*

An analogous result holds for numerical queries of the form $ExpTot^{\min}_{\mathcal{M}}(\phi\,|\,\Psi)$, $Pr^{\max}_{\mathcal{M}}(\phi\,|\,\Psi)$ or $Pr^{\min}_{\mathcal{M}}(\phi\,|\,\Psi)$. As discussed previously, this also yields a technique to solve both achievability and verification queries in the same manner.

4.2.2 Quantitative Assume-Guarantee Verification

We now present novel compositional verification techniques for probabilistic automata, based on the quantitative multi-objective properties defined in the previous section. The key ingredient of this approach is the *assume-guarantee triple*, whose definition, like in [51], is based on quantification over adversaries. However, whereas [51] uses a single *probabilistic safety property* as an assumption or guarantee, we permit *quantitative multi-objective* properties. Another key factor is the incorporation of *fairness*.

Definition 4.15 (Assume-guarantee triples) *If* $\mathcal{M}=(S,\bar{s},\alpha_{\mathcal{M}},\delta_{\mathcal{M}})$ *is a PA and* Ψ_A,Ψ_G *are qmo-properties such that* $\alpha_G\subseteq\alpha_A\cup\alpha_M$*, then* $\langle\Psi_A\rangle\,\mathcal{M}\,\langle\Psi_G\rangle$ *is an* assume-guarantee triple *with the following semantics:*

$$\langle\Psi_A\rangle\,\mathcal{M}\,\langle\Psi_G\rangle \;\Leftrightarrow\; \forall\sigma\in Adv_{\mathcal{M}[\alpha_A]}\,.\;(\,\mathcal{M},\sigma\models\Psi_A\rightarrow\mathcal{M},\sigma\models\Psi_G)\,.$$

where $\mathcal{M}[\alpha_A]$ *denotes the* alphabet extension *[51] of* $\mathcal{M}$*, which adds* a*-labelled self-loops to all states of* $\mathcal{M}$ *for each* $a\in\alpha_A\backslash\alpha_{\mathcal{M}}$*.*

Informally, an assume-guarantee triple $\langle \Psi_A \rangle \, \mathcal{M} \, \langle \Psi_G \rangle$, means "if $\mathcal{M}$ is a component of a system such that the environment of $\mathcal{M}$ satisfies Ψ_A, then the combined system (under fairness) satisfies Ψ_G".

Verification of an assume guarantee triple, i.e. checking whether $\langle \Psi_A \rangle \, \mathcal{M} \, \langle \Psi_G \rangle$ holds, reduces directly to the verification of a qmo-property since:

$$(\mathcal{M}, \sigma \models \Psi_A \rightarrow \mathcal{M}, \sigma \models \Psi_G) \;\Leftrightarrow\; \mathcal{M}, \sigma \models (\neg \Psi_A \vee \Psi_G) \, .$$

Thus, using the techniques of Section 4.2.1, we can reduce this to an achievability query, solvable via linear programming. Using these assume-guarantee triples as a basis, we can now formulate several proof rules that permit compositional verification of probabilistic automata. We first state two such rules, then explain their usage, illustrating with an example.

Theorem 4.16 *If $\mathcal{M}_1$ and $\mathcal{M}_2$ are PAs, and Ψ_{A_1}, Ψ_{A_2} and Ψ_G are quantitative multi-objective properties, then the following proof rules hold:*

$$\frac{\mathcal{M}_1 \models_{fair} \Psi_{A_1} \quad \langle \Psi_{A_1} \rangle \, \mathcal{M}_2 \, \langle \Psi_G \rangle}{\mathcal{M}_1 \| \mathcal{M}_2 \models_{fair} \Psi_G} \qquad \text{(ASYM)}$$

$$\frac{\mathcal{M}_2 \models_{fair} \Psi_{A_2} \quad \langle \Psi_{A_2} \rangle \, \mathcal{M}_1 \, \langle \Psi_{A_1} \rangle \quad \langle \Psi_{A_1} \rangle \, \mathcal{M}_2 \, \langle \Psi_G \rangle}{\mathcal{M}_1 \, \| \, \mathcal{M}_2 \models_{fair} \Psi_G} \qquad \text{(CIRC)}$$

where, for well-formedness, we assume that, if a rule contains an occurrence of the triple $\langle \Psi_A \rangle \, \mathcal{M} \, \langle \Psi_G \rangle$ in a premise, then $\alpha_G \subseteq \alpha_A \cup \alpha_{\mathcal{M}}$; similarly, for a premise that checks Ψ_A against $\mathcal{M}$, we assume that $\alpha_A \subseteq \alpha_{\mathcal{M}}$

Theorem 4.16 presents two assume-guarantee rules. The simpler, (ASYM), uses a single assumption Ψ_{A_1} about $\mathcal{M}_1$ to prove a property Ψ_G on $\mathcal{M}_1 \| \mathcal{M}_2$. This is done compositionally, in two steps. First, we verify $\mathcal{M}_1 \models_{fair} \Psi_{A_1}$. If $\mathcal{M}_1$ comprises just a single PA, the stronger (but easier) check $\mathcal{M}_1 \models \Psi_{A_1}$ suffices; the use of fairness in the first premise is to permit recursive application of the rule. Second, we check that $\langle \Psi_{A_1} \rangle \, \mathcal{M}_2 \, \langle \Psi_G \rangle$ holds. Again, optionally, we can consider fairness here.[5] In total, these two steps have the potential to be significantly cheaper than verifying $\mathcal{M}_1 \| \mathcal{M}_2$. The other rule, (CIRC), operates similarly, but using assumptions about both $\mathcal{M}_1$ and $\mathcal{M}_2$.

4.3 Automated assumption learning

As mentioned before, the second limitation when applying the assume-guarantee reasoning approach in the deliverable D2.1 [17] (also in [51]) is the manual generation of assumptions. In this section, we address this limitation by proposing a fully automated approach to the generation of such assumptions, based on *learning* techniques. Learning, and in particular the well-known L* learning algorithm [6], have proved to be well suited to generating assumptions for compositional verification of non-probabilistic systems [62], and are currently attracting considerable interest in the verification community.

We propose a novel learning technique, based on L*, for generating *probabilistic* assumptions [38]. Given components $\mathcal{M}_1, \mathcal{M}_2$ and probabilistic safety property $\langle G \rangle_{\geq p_G}$, our framework attempts to build an assumption $\langle A \rangle_{\geq p_A}$ that can be used to prove that $\mathcal{M}_1 \| \mathcal{M}_2$ satisfies $\langle G \rangle_{\geq p_G}$, without constructing the full model $\mathcal{M}_1 \| \mathcal{M}_2$.

Like in [62], this is done by generating a series of possible assumptions which are analysed by a (probabilistic) model checker. The results of this analysis, in the form of (probabilistic) counterexamples, are used to guide the learning process by refining the current assumption. During this process, we may also discover a counterexample illustrating that $\mathcal{M}_1 \| \mathcal{M}_2$ does *not* satisfy $\langle G \rangle_{\geq p_G}$.

Because the assume-guarantee framework of [51] is incomplete, there may be no suitable assumption that can be learnt. To address this problem, we adopt a *quantitative* approach: our learning technique also generates *lower and upper bounds* on the (minimum) probability of satisfying G.

The rest of the section is structured as follows. We provide some background on the L* algorithm in Section 4.3.1. We introduce the generation of probabilistic counterexamples in the context of probabilistic assume-guarantee verification in Section 4.3.2, and present our learning framework in Section 4.3.3.

[5] Adding fairness to checks of both qmo-properties and assume-guarantee triples is achieved by encoding the unconditional fairness constraint as additional objectives.

4.3.1 The L* Learning Algorithm

The L* algorithm, which was proposed by Angluin [6], is one of the most influential, cited and extended online learning algorithms for regular languages. L* learns a *minimal* DFA accepting an unknown regular language $\mathcal{L}$, by interacting with a *teacher*. The teacher responds to two kinds of questions: *membership queries* (i.e., whether some word is in the target language) and *conjectures* (i.e., whether a hypothesised DFA H accepts the target language). In the latter case, if the conjecture is not correct, the teacher must provide a counterexample illustrating this (i.e. a word in the symmetric difference between $\mathcal{L}$ and $\mathcal{L}(H)$). L* is attractive because it is guaranteed to produce a minimal DFA and it runs in polynomial time.

Over the past decade, L* has become popular in the context of verification. In particular, as proposed by Giannakopoulou, Pasareanu et al. (see e.g. [62]), it has been successfully applied to the automatic generation of assumptions for assume-guarantee verification of non-probabilistic systems. The key innovation was to rephrase the problem of generating an assumption as the problem of learning a (prefix-closed) regular language characterising the *weakest assumption* about a component that suffices for verification.

In this approach, the questions asked of the teacher are translated into problems that can be executed by a model checker. In the case of conjectures, counterexamples produced during model checking are used to generate the required counterexamples for L*. Actually, in practice, the weakest assumption is rarely learnt: the process either finds a simpler (stronger) assumption that suffices for verification, or discovers (by analysing the counterexamples produced) that the property being checked does not in fact hold.

4.3.2 Probabilistic Counterexamples for Compositional Verification

Counterexamples are an essential ingredient of (non-probabilistic) model checking. They provide valuable feedback to the user of a model checker about the reason why a property is violated. They are also crucial to the success of verification techniques such as counterexample-guided abstraction refinement and learning-based assume-guarantee model checking. In the context of probabilistic verification, generation of counterexamples is more complex and is currently an active area of research. The basic difficulty is that, whereas a non-probabilistic property such as "an error never occurs" can be refuted with a single finite path to an error state, a probabilistic property such as "an error state is reached with probability at most p" is refuted by a *set* of such paths whose total probability exceeds p.

The fundamental techniques in this area were proposed in [42], which considers generation of counterexamples for probabilistic reachability properties (expressed with the logic PCTL) of discrete-time Markov chains (DTMCs). They show, for example, that counterexamples for PCTL properties of the form $\mathcal{P}_{\leq p}[\Diamond a]$, i.e. with non-strict, upper probability bounds, can be constructed as a finite set of finite paths that reach a. Furthermore, they characterise the notion of *smallest counterexample* which illustrates the violation with as few paths as possible. They also show how to handle properties with lower probability bounds (by identifying strongly connected components) and strict probability bounds (by using regular expressions).

Counterexamples for more complex models and properties can often be reduced to this basic case. For example, model checking of more expressive logics such as LTL on DTMCs reduces to computing reachability probabilities on a larger, product DTMC. Extending to probabilistic automata is also relatively straightforward (see e.g. [5]) since the first step is to find an adversary causing a violation; the problem then reduces to generating a counterexample for a DTMC.

Since probabilistic counterexamples are crucial to this work, and our needs are rather specific, we describe in the remainder of this section how these basic techniques can be adapted and extended to our setting.

Counterexamples for Safety Properties

We focus first on the case of counterexamples for safety properties of PAs. Recall that, for PA $\mathcal{M}$ to satisfy $\langle G \rangle_{\geq p_G}$, we require that $Pr^{\sigma}_{\mathcal{M}}(G) \geq p_G$ for all adversaries σ. So, to refute this, we require an adversary for which this does not hold and a set of paths, under this adversary, that illustrates this. As mentioned in Section 4.1, deterministic finite-memory adversaries suffice to verify (or refute) safety properties. Further-

more, we can represent the set of paths of $\mathcal{M}$ under such an adversary σ by a larger PA, which we denote $\mathcal{M}^\sigma$. For every path ω in $\mathcal{M}^\sigma$, we have $\omega\restriction_{\mathcal{M}} \in Path^\sigma_{\mathcal{M}}$, where $\omega\restriction_{\mathcal{M}}$ denotes the projection of ω onto $\mathcal{M}$.

Definition 4.17 *For a PA $\mathcal{M}$ and a probabilistic safety property $\langle G\rangle_{\geq p_G}$ with $\mathcal{M} \not\models \langle G\rangle_{\geq p_G}$, a* counterexample *for $\langle G\rangle_{\geq p_G}$ is a pair (σ, c) of a (deterministic, finite-memory) adversary σ for $\mathcal{M}$ with $Pr^\sigma_{\mathcal{M}}(G) < p_G$ and a set c of finite paths in $\mathcal{M}^\sigma$ such that $Pr(c) > 1-p_G$ and $\omega\restriction_{\mathcal{M}} \not\models G$ for all $\omega \in c$.*

The process of obtaining a counterexample (σ, c) for $\mathcal{M} \not\models \langle G\rangle_{\geq p_G}$ is as follows. As described in Section 4.1, model checking $\langle G\rangle_{\geq p_G}$ on $\mathcal{M}$ requires computation of $Pr^{\min}_{\mathcal{M}}(G)$, which reduces to the problem of computing the maximum probability of reaching an accepting state in the product PA $\mathcal{M}\otimes G^{err}$. The (deterministic, finite-memory) adversary σ for $\mathcal{M}$ is obtained directly from the (deterministic, memoryless) adversary of $\mathcal{M}\otimes G^{err}$ under which this reachability probability is above $1-p_G$. The set of paths c is taken from $\mathcal{M}^\sigma$ which, since σ is deterministic, is a DTMC. Since c equates to a counterexample for a non-strict, upper-bounded reachability property, as [42] shows, a finite set of finite paths suffices.

In fact, our learning techniques require not just a set of violating paths c, but also the *fragment* of the PA $\mathcal{M}$ which contains these paths. This fragment, which we denote $\mathcal{M}^{\sigma,c}$, is a (sub-stochastic) PA obtained from $\mathcal{M}^\sigma$ by removing all transitions that do not appear in any path of c.

Definition 4.18 *Let $\mathcal{M}$ be a PA, σ a deterministic, finite-memory adversary and $\mathcal{M}^\sigma{=}(S, \overline{s}, \alpha_{\mathcal{M}}, \delta_{\mathcal{M}})$ be the PA $\mathcal{M}^\sigma$. If c is a set of (finite or infinite) paths of $\mathcal{M}^\sigma$, the corresponding* PA fragment*, denoted by $\mathcal{M}^{\sigma,c}$, is the sub-stochastic PA $(S, \overline{s}, \alpha_{\mathcal{M}}, \delta'_{\mathcal{M}})$ where $\delta'_{\mathcal{M}}$ is defined as follows. For distribution $\mu \in Dist(S)$, we define the sub-distribution μ^c over S as $\mu^c(s') = \mu(s')$ if the state s' appears in some path in c, and $\mu^c(s') = 0$ otherwise. For each $s \xrightarrow{a} \mu$ in $\delta_{\mathcal{M}}$, then $\delta'_{\mathcal{M}}$ contains $s \xrightarrow{a} \mu^c$ if and only if μ^c is non-empty.*

Note that, due to the possibility of loops in $\mathcal{M}^\sigma$, a fragment $\mathcal{M}^{\sigma,c}$ may contain paths that are not in c. However, PA fragments have the following useful properties.

Proposition 4.19 *For PAs $\mathcal{M}$ and $\mathcal{M}'$, PA fragment $\mathcal{M}^{frag}$ of $\mathcal{M}$ and probabilistic safety property $\langle G\rangle_{\geq p_G}$, we have:*

(a) $\mathcal{M} \models \langle G\rangle_{\geq p_G} \Rightarrow \mathcal{M}^{frag} \models \langle G\rangle_{\geq p_G}$

(b) $\mathcal{M}\|\mathcal{M}' \models \langle G\rangle_{\geq p_G} \Rightarrow \mathcal{M}^{frag}\|\mathcal{M}' \models \langle G\rangle_{\geq p_G}$

(c) $\mathcal{M}^{frag}\|\mathcal{M}' \not\models \langle G\rangle_{\geq p_G} \Rightarrow \mathcal{M}\|\mathcal{M}' \not\models \langle G\rangle_{\geq p_G}$.

Counterexamples for Assume-Guarantee Triples

We also need to consider counterexamples for probabilistic assume-guarantee triples. Recall (see Definition 4.5) that a triple $\langle A\rangle_{\geq p_A}\ \mathcal{M}\ \langle G\rangle_{\geq p_G}$ is false if some adversary of $\mathcal{M}[\alpha_A]$ satisfies assumption $\langle A\rangle_{\geq p_A}$ but violates the guarantee $\langle G\rangle_{\geq p_G}$. We first need the notion of a *witness*.

Definition 4.20 *For a PA $\mathcal{M}$ and probabilistic safety property $\langle A\rangle_{\geq p_A}$ for which $\mathcal{M} \models \langle A\rangle_{\geq p_A}$, a* witness *is a pair (σ, w) comprising a (deterministic, finite-memory) adversary σ for $\mathcal{M}$ with $Pr^\sigma_{\mathcal{M}}(A) \geq p_A$ and a set w of infinite paths in $\mathcal{M}^\sigma$ such that $Pr(w) \geq p_A$ and $\omega\restriction_{\mathcal{M}} \models A$ for all $\omega \in w$.*

To compute a witness (σ, w), adversary σ is obtained exactly as for generating a counterexample (σ, c): through probabilistic reachability on the product PA $\mathcal{M}\otimes A^{err}$. Finding paths w in $\mathcal{M}^\sigma$ that show $Pr^\sigma_{\mathcal{M}}(A){\geq}p_A$ is then equivalent to building a counterexample in a DTMC for a strict, lower-bounded reachability property. This can be done using the techniques of [42] (analysis of strongly connected components, followed by construction of a regular expression).

Definition 4.21 *For a PA $\mathcal{M}$ and probabilistic safety properties $\langle A\rangle_{\geq p_A}$ and $\langle G\rangle_{\geq p_G}$ such that $\langle A\rangle_{\geq p_A}\ \mathcal{M}\ \langle G\rangle_{\geq p_G}$ is false, a* counterexample *for $\langle A\rangle_{\geq p_A}\ \mathcal{M}\ \langle G\rangle_{\geq p_G}$ is a tuple (σ, w, c) such that (σ, w) is a witness for $\langle A\rangle_{\geq p_A}$ in $\mathcal{M}[\alpha_A]$ and (σ, c) is a counterexample for $\langle G\rangle_{\geq p_G}$ in $\mathcal{M}[\alpha_A]$.*

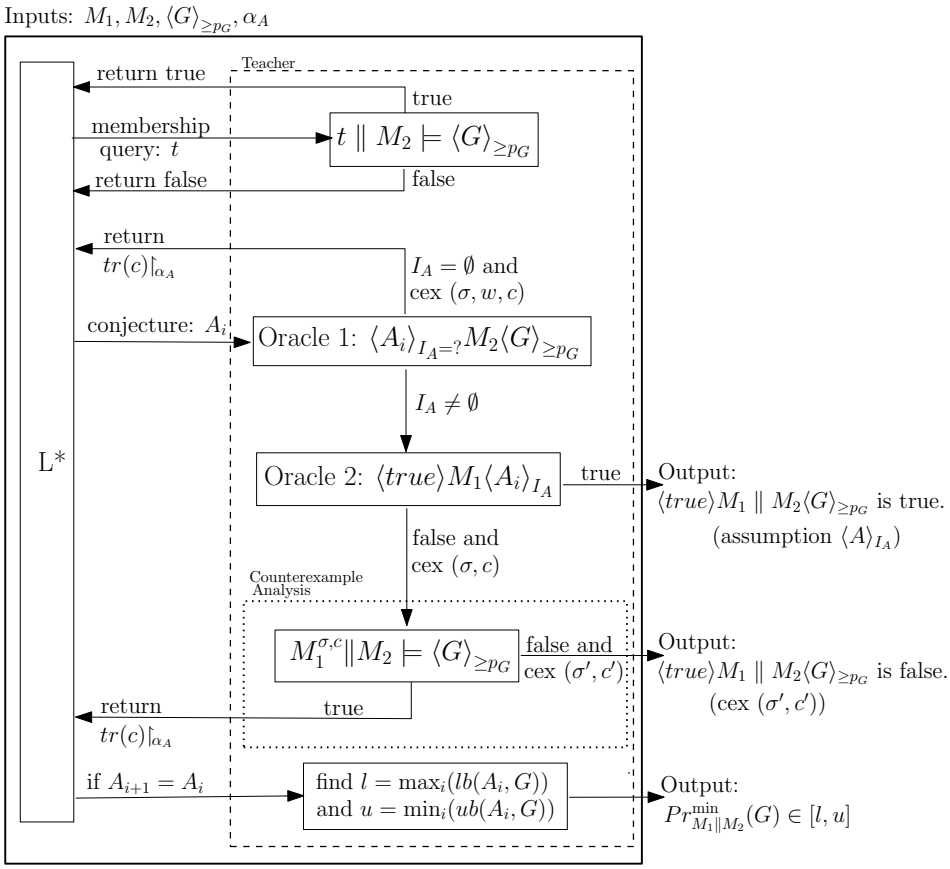

Figure 4.2: L*-based learning framework for the rule (ASYM)

Construction of a counterexample (σ, w, c) for a triple $\langle A \rangle_{\geq p_A} \mathcal{M} \langle G \rangle_{\geq p_G}$ proceeds as follows. The process of checking whether the query is true (done using multi-objective model checking [51, 35] on the product of $\mathcal{M}[\alpha_A]$ with A^{err} and G^{err}) also yields the adversary σ when the query is false. The counterexample c and witness w can be obtained from the product in similar fashion to the cases described above. Note that, in the case where the assumption is qualitative (i.e. $p_A{=}1$), the witness w comprises *all* paths of $\mathcal{M}^\sigma$ and so its explicit construction can be avoided.

4.3.3 The Learning Framework

In this section, we present a framework that learns assumptions for two-component probabilistic systems based on the asymmetric probabilistic assume-guarantee rule (ASYM) of [51]. The inputs are component models $\mathcal{M}_1, \mathcal{M}_2$, a probabilistic safety property $\langle G \rangle_{\geq p_G}$ and an alphabet α_A. The aim is to verify (or refute) $\mathcal{M}_1 \| \mathcal{M}_2 \models \langle G \rangle_{\geq p_G}$ by generating a probabilistic assumption $\langle A \rangle_{\geq p_A}$ over α_A. Our approach is *quantitative*, in that it also yields lower and upper bounds on the minimum probability of satisfying G.

Overview

A summary of the framework is shown in Figure 4.2. It is built on top of the L* algorithm, and inspired by techniques that apply this to assume-guarantee verification of labelled transition systems [62]. The introduction of probabilities brings several complications. Perhaps the most important is the issue of completeness. The approach of [62] is *complete*, meaning that if the property being verified of the system is satisfied, then there always exists an assumption that can be used to verify the property compositionally. This so-called *weakest assumption* can be formally defined, as a regular language, and used as the target language for L*. The probabilistic assume-guarantee framework of [51] is *incomplete*: even if the property is satisfied, there may be no assumption that permits a compositional verification.

Crucially, though, L*-based implementations of assume-guarantee verification do not, in practice, actually construct the weakest assumption. Instead, the aim is to, in the process of learning it, either find a simpler, stronger assumption that is sufficient to verify the property being checked or generate a counterexample that refutes it. We adopt a similar approach here: we use L* to generate a succession of conjectured assumptions and, for each one, execute probabilistic model checking queries that may either verify or refute the property $\langle G \rangle_{\geq p_G}$ on $\mathcal{M}_1 \| \mathcal{M}_2$. If neither is possible, information from model checking, in the form of counterexamples, guides the learning process towards a new, refined assumption.

In our context, we need to learn a *probabilistic* assumption, i.e. a probabilistic safety property $\langle A \rangle_{\geq p_A}$. However, as we will show, we can reduce this task to the problem of learning a *non-probabilistic* assumption, i.e. a regular safety property A. This is because, for a fixed A, it is possible to determine whether there is any probability threshold p_A for which $\langle A \rangle_{\geq p_A}$ suffices as a probabilistic assumption. To avoid confusion, in the remainder of this section, we refer to A and $\langle A \rangle_{\geq p_A}$ as the "assumption" and "probabilistic assumption", respectively.

This means that our framework can be built around the standard L* algorithm, which generates a (regular language) assumption A, guided by a teacher. The task of determining whether a corresponding probabilistic assumption $\langle A \rangle_{\geq p_A}$ can be created is performed by the teacher.

The overall structure of the interaction between L* and the teacher is similar to the non-probabilistic case [62]. The teacher first responds to several *membership queries*. Then, L* provides a *conjecture* for A to the teacher. The teacher uses two oracles (executed by a probabilistic model checker) to analyse the conjecture and determine whether it can verify or refute $\mathcal{M}_1 \| \mathcal{M}_2 \models \langle G \rangle_{\geq p_G}$. These correspond to the top two outputs on the right hand side of Figure 4.2. If neither is possible, the teacher generates and returns traces to L* based on counterexamples generated during model checking.

Here, there is another important difference with the use of L* for non-probabilistic assume-guarantee verification. In [62], the learning is driven by the weakest assumption. When the teacher finds a conjecture to be unsuitable, it is guaranteed to be able to find a trace illustrating an inconsistency between the current assumption and the weakest assumption. In our framework, there may not exist such an assumption so this is not always possible. The feedback provided by the teacher should be seen as *heuristics* that guide L* towards an appropriate assumption.[6]

[6]Since we use L* in a non-standard fashion, we use the original version of [6], rather than optimisations e.g. due to Rivest & Schapire.

To address this limitation, we equip our framework with the ability to, at any point in its execution, produce a lower and an upper bound on the minimum probability of G holding, based on the assumptions generated so far. This means that, if the algorithm reaches a point where the teacher is unable to provide feedback for L* to produce a new assumption, it can still provide valuable quantitative information to the user (this is indicated by the third possible output at the bottom of Figure 4.2). Furthermore, we can choose to interrupt the learning process at any point and obtain this information.

In the following sections, we describe each aspect of the learning algorithm in more detail.

Answering Membership Queries

The L* learning procedure is guided by the results of membership queries as to whether a given finite trace t should be included in the assumption A being generated. Since, as highlighted above, an assumption may not exist, we cannot guarantee definitive answers to these queries. The criterion we use is to check whether $t \| \mathcal{M}_2 \models \langle G \rangle_{\geq p_G}$, where t here denotes a PA representing the trace, i.e a linear, $|t|+1$ state PA in which each transition has probability 1.

Note that if t *does* cause a violation, i.e. $t \| \mathcal{M}_2 \not\models \langle G \rangle_{\geq p_G}$, then t should certainly not be in A. This is because any assumption A that contains t will thus not satisfy premise 2 $\langle A \rangle_{\geq p_A} \mathcal{M}_2 \langle G \rangle_{\geq p_G}$ of (ASYM) for any value of p_A.

The converse does not hold, i.e. $t \| \mathcal{M}_2 \models \langle G \rangle_{\geq p_G}$ does not imply that t should be in A. This is because multiple traces that do not lead to a violation individually may do so when combined into a single assumption. Unfortunately, we cannot establish this with an analysis of t in isolation. As we will show later in the paper, however, the proposed scheme for answering membership queries works well in practice.

Answering Conjectures

The second job of the teacher is to answer conjectures, i.e. to check whether a generated assumption A can be used to apply rule (ASYM). For this, A needs to, for some probability bound p_A, satisfy both premise 1, $\langle \texttt{true} \rangle \mathcal{M}_1 \langle A \rangle_{\geq p_A}$, and premise 2, $\langle A \rangle_{\geq p_A} \mathcal{M}_2 \langle G \rangle_{\geq p_G}$. As shown in Figure 4.2, the teacher checks this using two separate *oracles*, one for each premise.

Oracle 1 checks the quantitative assume-guarantee query $\langle A \rangle_{I_A = ?} \mathcal{M}_2 \langle G \rangle_{\geq p_G}$ corresponding to premise 2 of (ASYM), i.e. it determines the widest interval $I_A \subseteq [0,1]$ for which the premise holds using assumption A. As described in Section 4.1, either I_A is a non-empty, closed, lower-bounded interval, in which case $\langle A \rangle_{I_A}$ is a valid safety property, or $I_A = \emptyset$. In the former case, we will proceed to Oracle 2 to check premise 1 with $\langle A \rangle_{I_A}$.

In the latter case, $I_A = \emptyset$ indicates that, even under the assumption $\langle A \rangle_{\geq 1}$, $\mathcal{M}_2$ would still not satisfy $\langle G \rangle_{\geq p_G}$ so A must be refined, regardless of the validity of premise 1. To refine A we generate a probabilistic counterexample (σ, w, c) to show that $\langle A \rangle_{\geq 1} \mathcal{M}_2 \langle G \rangle_{\geq p_G}$ does not hold and then use this to generate traces for L*. More precisely, we take the set $\mathcal{T} = tr(c)\restriction_{\alpha_A}$ of traces for paths in c, restricted to the alphabet α_A. Intuitively, we want to find a trace which is currently included in A but should in fact be excluded since it causes a violation of $\langle G \rangle_{\geq p_G}$. By construction, all paths in c satisfy A (since A is satisfied with probability 1 under adversary σ).

Consider first the case where $\mathcal{T}$ comprises a single trace t. In this instance, reasoning as for membership queries above, since t corresponds to a path through $\mathcal{M}_2$ that causes $\langle G \rangle_{\geq p_G}$ to be false, any assumption A that contains t will not satisfy premise 2 of (ASYM). Hence, t should not be in the learnt assumption A. Unfortunately, if $\mathcal{T}$ includes multiple traces, it is unclear whether the same is true. In this case, we return all traces in $\mathcal{T}$ to L*. However, we can increase the likelihood that c contains a single trace by choosing c to be the *smallest* counterexample [42] violating G.

Oracle 2, which is invoked when the interval I_A from Oracle 1 is non-empty, checks premise 1 of (ASYM), i.e. it verifies whether $\langle \texttt{true} \rangle \mathcal{M}_1 \langle A \rangle_{I_A}$ is satisfied. If so, then we have found an assumption $\langle A \rangle_{I_A}$ that satisfies both premises of (ASYM), thus proving that $\mathcal{M}_1 \| \mathcal{M}_2$ satisfies $\langle G \rangle_{\geq p_G}$, and we can terminate the learning algorithm.

If, on the other hand, premise 1 is not satisfied, then we construct a counterexample (σ, c) showing $\mathcal{M}_1 \not\models \langle A \rangle_{I_A}$. Since, from Oracle 1, we know that $\mathcal{M}_2$ is only guaranteed to satisfy $\langle G \rangle_{\geq p_G}$ if $\langle A \rangle_{I_A}$ is true, (σ, c) is potentially a counterexample for $\mathcal{M}_1 \| \mathcal{M}_2$.

Counterexample analysis is then applied to determine whether (σ, c) is a real counterexample for $\mathcal{M}_1 \| \mathcal{M}_2$. To do so, we construct the PA fragment $\mathcal{M}_1^{\sigma,c}$ and check if $\mathcal{M}_1^{\sigma,c} \| \mathcal{M}_2 \models \langle G \rangle_{\geq p_G}$ holds. If not, we can conclude that $\mathcal{M}_1 \| \mathcal{M}_2 \not\models \langle G \rangle_{\geq p_G}$ (see Proposition 4.19(c)) and the algorithm terminates. Furthermore, a counterexample from the verification of $\mathcal{M}_1^{\sigma,c} \| \mathcal{M}_2$ also serves as a counterexample to illustrate the violation of $\langle G \rangle_{\geq p_G}$ by $\mathcal{M}_1 \| \mathcal{M}_2$.

Otherwise, (σ, c) is not a real counterexample and we again need to refine the assumption A by returning appropriate traces to L*. Here, the situation is the opposite to that of Oracle 1. Intuitively, our aim is to find a trace t that is not currently in the assumption A but should be. Consider as above the case where $\mathcal{T} = tr(c)\restriction_{\alpha_A}$ comprises a single trace t. Since this trace comes from a counterexample showing $\mathcal{M}_1 \not\models \langle A \rangle_{\geq I_A}$, we know t is not in A. Furthermore, from the results of the counterexample analysis, it is likely (but not guaranteed) that $t \| \mathcal{M}_2 \models \langle G \rangle_{\geq p_G}$.

Generation of Lower and Upper Bounds

As discussed previously, there is a third possible output from our learning algorithm (shown at the bottom of Figure 4.2). To refine a conjecture, L* requires that any trace returned as a counterexample by the teacher has not already been tested with a membership query. In the probabilistic setting, this cannot be guaranteed, so if this occurs, we terminate the algorithm without generating further conjectures.

Fortunately, as we will now show, for any conjectured assumption A, it is possible to produce lower and upper bounds on the (minimum) probability of satisfying the safety property G. We denote these $lb(A, G)$ and $ub(A, G)$. Computation of these bounds proceeds as follows. First we compute $p_A^* = Pr^{\min}_{\mathcal{M}_1}(A)$ and, simultaneously, generate an adversary $\sigma \in Adv_{\mathcal{M}_1}$ that achieves this minimum probability. Next, we check the quantitative assume-guarantee query $\langle A \rangle_{\geq p_A^*} \mathcal{M} \langle G \rangle_{I_G = ?}$ and, from the resulting interval, take:

$$lb(A, G) = min(I_G)$$

For the upper bound, we compute:

$$ub(A, G) = Pr^{\min}_{\mathcal{M}_1^\sigma \| \mathcal{M}_2}(G)$$

using the adversary σ from above. Then:

Proposition 4.22 $lb(A, G) \leq Pr^{\min}_{\mathcal{M}_1 \| \mathcal{M}_2}(G) \leq ub(A, G)$.

Proof: For the lower bound, by construction, both $\mathcal{M}_1 \models \langle A \rangle_{\geq p_A^*}$ and $\langle A \rangle_{\geq p_A^*} \mathcal{M} \langle G \rangle_{\geq lb(A,G)}$ hold. Thus, $lb(A, G) \leq Pr^{\min}_{\mathcal{M}_1 \| \mathcal{M}_2}(G)$ follows from rule (ASYM). For the upper bound, since $\mathcal{M}_1^\sigma$ is a fragment of $\mathcal{M}_1$, Proposition 4.19(b) yields that: $Pr^{\min}_{\mathcal{M}_1 \| \mathcal{M}_2}(G) \leq Pr^{\min}_{\mathcal{M}_1^\sigma \| \mathcal{M}_2}(G) = ub(A, G)$. □

This means that, if the algorithm terminates because no further conjectures are possible, we can provide bounds from the current A. In fact, an interesting property of L* is that the series of conjectures produced is not monotonic (in terms of language inclusion). So, the lower/upper bounds from earlier assumptions may produce tighter bounds and we actually return the tightest bounds produced from any assumption.

It is worth pointing out that the steps outlined above to produce the bounds are carried out in many cases anyway. So, generating this information comes at little extra cost.

Correctness and Termination

As we have seen, there are three possible outputs from the execution of the learning algorithm: (i) verification of $\langle G \rangle_{\geq p_G}$; (ii) refutation of $\langle G \rangle_{\geq p_G}$; and (iii) provision of lower/upper bounds on the minimum probability of satisfying G. The correctness of these three conclusions has been explained in the sections above. It is important to emphasise that this correctness is *independent* of the choices made by Oracles 1 and 2 (with regards to the traces that they return to L*) *and* of the conjectures generated by L*.

As discussed earlier in this section, due to the incompleteness of the underlying compositional verification framework, we cannot hope to guarantee that the learning algorithm will terminate and produce a definitive answer as to whether the property is satisfied or not. Instead, we provide the option to obtain lower and upper bounds at any point.

4.4 Future work

In this deliverable, we introduced multi-objectives properties of PAs, using a language that combines ω-regular properties, expected total reward and multiple objectives. We described how to verify a property over all adversaries of a PA, synthesise an adversary that satisfies and/or optimises objectives, and compute the minimum or maximum value of an objective, subject to constraints. Finally, we proposed a multi-objectives-based assume-guarantee framework for PAs that significantly improves existing ones [51]. We also proposed a fully automated assume-guarantee framework for verification of PAs. Assumptions, which are represented as probabilistic safety properties in the style of [51], are constructed automatically using an L*-based learning algorithm. The results produced are quantitative: in the case where an assumption cannot be produced that verifies or refutes the property being verified, lower and upper bounds on the probability of satisfying the property are produced.

There are various possible directions for future work. One is to investigate extending our compositional verification approach with automated assumption generation, using learning for ω-automata, in a similar style to [38] for the simpler framework of [51]. It is also interesting to extend the learning framework to support more assume-guarantee rules, e.g. the circular rule and the N-component rule of [51], since these patterns are useful for composing CONNECTors, and to optimise the efficiency of the techniques, e.g. using alphabet refinement [62] and symbolic (BDD-based) implementations of L*.

The L* algorithm has also been employed in the Work Package 4 (WP4) to learn behaviours of network systems. Though simple non-functional properties, such as latency and throughput, can be handled by an extension of learning techniques developed in WP4 (see the deliverable D4.2 [19]), the probabilistic assumptions needed for AG-reasoning cannot be generated by the techniques in WP4. In our learning framework, we have modified the L* algorithm to deal with probabilistic counterexamples, which is crucial for assumption generation. However, it is very interesting indeed to investigate the potential collaboration between WP2 and WP4 on learning non-functional properties and assumptions in the future, which we believe will be of great benefit to the project.

5 Conclusion

In this document we have presented the research completed in the second year as part of Work Package 2 (WP2). We are pursuing two streams of work, (i) *compositional theory of connector behaviours* and (ii) *compositional assume-guarantee verification for probabilistic automata*, with the view to integrate these together next year.

Compositional theory of CONNECTor behaviours We have laid the foundations for a comprehensive theory of connectors by proposing a quantitative specification theory based on probabilistic interface automata, as a framework for modelling and combining components. We have also provided a preliminary algebra focusing on the functional aspects of connectors that resolve protocol mismatches. The goal now is to establish key results for the compositional quantitative specification theory, as outlined in Section 3.2, and to finalise an overarching algebra to characterise the behaviours of connectors in CONNECT.

Compositional assume-guarantee verification techniques. We have continued our previous work in Deliverable D2.1 [17] concerning the quantitative assume-guarantee reasoning method, addressing the following two limitations. We have formulated a quantitative multi-objective framework to support quantitative liveness and expected reward properties, and a fully automatic learning framework to generate quantitative assumptions for assume-guarantee reasoning.

Putting it all together Our proposal for a quantitative specification theory is based on probabilistic interface automata, which are essentially probabilistic automata with interfaces, as defined in Section 3.2, extended with assume-guarantee specifications, see Section 4.2. Building on the compositional verification techniques developed in Chapter 4, we aim to formulate a collection of compositional proof rules to infer properties of composed CONNECTors from their constituents. We expect to be able to adapt the verification techniques and tools implemented in Chapter 4 to this new setting, by providing translation mechanisms from component models to the PRISM notation. Nevertheless, the formulation of a comprehensive quantitative specification theory poses a number of research challenges.

The preliminary algebra for protocol mismatches presented in Section 3.1.6 enables the modelling of functional aspects of the network systems (NSs) and CONNECTor interaction with support from the IAAnalyser tool. In Deliverable D5.2 [20] we have presented a property meta-model that allows the modelling of non-functional aspects of the NSs and CONNECTor interaction. In future, we will integrate the property meta-model with the IAAnalyzer tool. This is feasible, as both are defined in ECore. More widely, we will also explore the potential for interaction between IAAnalyser and other workpackages, such as synthesis (WP3).

Bibliography

[1] A. Legay and L. de Alfaro and M. Faella. An Introduction to the Tool Ticc. In *Workshop "Trustworthy Software" 2006*. Internationales Begegnungs- und Forschungszentrum f"ur Informatik (IBFI), Schloss Dagstuhl, Germany, 2006.

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

[3] R. Alur, T. A. Henzinger, O. Kupferman, and M. Y. Vardi. Alternating refinement relations. In D. Sangiorgi and R. de Simone, editors, *CONCUR '98: Concurrency Theory*, volume 1466 of *Lecture Notes in Computer Science*, pages 163–178. Springer-Verlag, 1998.

[4] R. Alur, T. A. Henzinger, O. Kupferman, and M. Y. Vardi. Alternating Refinement Relations. In *CONCUR '98*, pages 163–178, London, UK, 1998. Springer-Verlag.

[5] M. Andrés, P. D'Argenio, and P. van Rossum. Significant diagnostic counterexamples in probabilistic model checking. In *Proc. HVC'08*, pages 129–148, 2008.

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

[7] B. T. Adler and L. De Alfaro and R. D. Da Silva and M. Faella and A. Legay and V. Raman and P. Roy. Ticc, a tool for interface compatibility and composition. In *In Proceedings 18th International Conference on Computer Aided Verification (CAV), volume 4144 of Lecture Notes in Computer Science*, pages 59–62. Springer, 2006.

[8] C. Baier, M. Größer, and F. Ciesinski. Quantitative analysis under fairness constraints. In *Proc. ATVA'09*, volume 5799 of *LNCS*, pages 135–150. Springer, 2009.

[9] C. Baier and M. Kwiatkowska. Model checking for a probabilistic branching time logic with fairness. *Distributed Computing*, 11(3):125–155, 1998.

[10] A. Basu, M. Bozga, and J. Sifakis. Modeling heterogeneous real-time components in bip. In *SEFM '06*, pages 3–12, 2006.

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

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

[13] A. Bianco and L. de Alfaro. Model checking of probabilistic and nondeterministic systems. In P. S. Thiagarajan, editor, *Foundations of Software Technology and Theoretical Computer Science*, volume 1026 of *Lecture Notes in Computer Science*, pages 499–513. Springer-Verlag, 1995.

[14] B. Caillaud, B. Delahaye, K. G. Larsen, A. Legay, M. L. Pedersen, and A. Wasowski. Compositional Design Methodology with Constraint Markov Chains. In *Proc. 7th International Conference on Quantitative Evaluation of Systems (QEST '10)*, pages 123–132. IEEE Computer Society, 2010.

[15] S. Chaki and A. Gurfinkel. Automated Assume-Guarantee Reasoning for Omega-Regular Systems and Specifications. In *Proc. 2nd NASA Formal Methods Symposium (NFM)*, pages 57–66, April 2010.

[16] 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.

[17] 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/.

[18] CONNECT consortium. CONNECT Deliverable D3.2: Reasoning about and Harmonizing the Interaction Behavior of Networked Systems at Application- and Middleware-Layer. FET IP CONNECT EU project, FP7 grant agreement number 231167, http://connect-forever.eu/.

[19] CONNECT consortium. CONNECT Deliverable D4.2: Further development of learning techniques. FET IP CONNECT EU project, FP7 grant agreement number 231167, http://connect-forever.eu/.

[20] 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/.

[21] C. Courcoubetis and M. Yannakakis. Markov decision processes and regular events. *IEEE Transactions on Automatic Control*, 43(10):1399–1418, 1998.

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

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

[24] 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.

[25] L. de Alfaro and T. A. Henzinger. Interface-based Design. In *In Engineering Theories of Software-intensive Systems*, volume 195 of *NATO Science Series: Mathematics, Physics, and Chemistry*, pages 83–104. Springer, 2005.

[26] B. Delahaye. *Modular Specification and Compositional Analysis of Stochastic Systems*. PhD thesis, Ecole doctorale MATISSE, Université de Rennes 1, IRISA, Rennes, France, October 2010.

[27] N. Dershowitz and J.-P. Jouannaud. *Rewrite Systems, Chapter 6 in Jan van Leeuwen (Ed.), Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics, pp. 243–320*. Elsevier and MIT Press, 1990.

[28] J. Desharnais. *Labelled Markov Processes*. PhD thesis, McGill University, 1999.

[29] 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.

[30] Eclipse Platform, Acceleo 3.0 Model to Text transforamtion language. http://www.eclipse.org/acceleo/.

[31] Eclipse Platform, Eclipse Modeling Project. http://www.eclipse.org/modeling/.

[32] Eclipse Platform, EuGENia GMF Tutorial. http://www.eclipse.org/gmt/epsilon/doc/articles/eugenia-gmf-tutorial/.

[33] Eclipse Platform, Graphical Modeling Project (GMP). http://www.eclipse.org/modeling/gmp/.

[34] M. Emmi, D. Giannakopoulou, and C. Păsăreanu. Assume-Guarantee Verification for Interface Automata. In J. Cuellar, T. Maibaum, and K. Sere, editors, *FM 2008: Formal Methods*, volume 5014 of *Lecture Notes in Computer Science*, pages 116–131. Springer-Verlag, 2008.

[35] K. Etessami, M. Kwiatkowska, M. Vardi, and M. Yannakakis. Multi-objective model checking of Markov decision processes. In O. Grumberg and M. Huth, editors, *Proc. 13th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'07)*, volume 4424 of *LNCS*, pages 50–65. Springer, 2007.

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

[37] L. Feng, M. Kwiatkowska, and D. Parker. Compositional Verification of Probabilistic Systems using Learning. In *Proc. 7th International Conference on Quantitative Evaluation of Systems (QEST'10)*, pages 133–142. IEEE CS Press, September 2010.

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

[39] V. Forejt, M. Kwiatkowska, G. Norman, D. Parker, and H. Qu. Quantitative multi-objective verification for probabilistic systems. In *Proc. 17th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'11)*, LNCS. Springer, 2011. To appear.

[40] V. Forejt, M. Kwiatkowska, G. Norman, D. Parker, and H. Qu. Quantitative Multi-Objective Verification for Probabilistic Systems. In *Tools and Algorithms for the Construction and Analysis of Systems*, Lecture Notes in Computer Science. Springer-Verlag, To appear 2011.

[41] S. P. G.E. Krasner. A cookbook for using the model-view controller user interface paradigm in Smalltalk-80. *JOOP*, 1(3):26–49, 1988.

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

[43] H. Hermanns. *Interactive Markov Chains*. PhD dissertation, Universität Erlangen-Nürnberg, Institut für Mathematische Maschinen und Datenverarbeitung, September 1999.

[44] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice Hall, 1985.

[45] J. E. Hopcroft, R. Motwani, and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Prentice Hall, 2007.

[46] P. Inverardi, V. Issarny, and R. Spalazzese. A Theory of Mediators for Eternal Connectors. In *In proceedings of ISOLA 2010*, 2010.

[47] B. Jonsson. Modular verification of asynchronous networks. In *Proc. 6th Annual ACM Symposium on Principles of Distributed Computing*, PODC87, pages 152–166, Vancouver, Canada, 1987. ACM.

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

[49] B. Jonsson and K. G. Larsen. Specification and Refinement of Probabilistic Processes. In *Proc. 6th Annual IEEE Symposium on Logic in Computer Science*, LICS '91, pages 266–277. IEEE, 1991.

[50] 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.

[51] M. Kwiatkowska, G. Norman, D. Parker, and H. Qu. Assume-guarantee verification for probabilistic systems. In *Proc. TACAS'10*, volume 6105 of *LNCS*, pages 23–37, 2010.

[52] M. Kwiatkowska, G. Norman, D. Parker, and H. Qu. Assume-Guarantee Verification for Probabilistic Systems. In J. Esparza and R. Majumdar, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 6015 of *Lecture Notes in Computer Science*, pages 23–37. Springer-Verlag, 2010.

[53] L. de Alfaro and L. D. da Silva and M. Faella and A. Legay and P. Roy and M. Sorea. Sociable Interfaces. In *FroCos'05*, pages 81–105, 2005.

[54] K. Larsen, U. Nyman, and A. Wasowski. Interface input/output automata. In *FM 2006: Proc. 14th Int. Symp. on Formal Methods, Hamilton, Canada*, volume 4085 of *Lecture Notes in Computer Science*, pages 82–97. Springer, 2006.

[55] K. G. Larsen. Modal specifications. In J. Sifakis, editor, *Automatic Verification Methods for Finite State Systems*, volume 407 of *Lecture Notes in Computer Science*, pages 232–246. Springer-Verlag, 1990.

[56] N. Lynch and F. Vaandrager. Forward and Backward Simulations Part I: Untimed Systems. *Inf. Comput.*, 121(2):214–233, September 1995.

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

[58] M. Ben-Ari and Z. Manna and A. Pnueli. The temporal logic of branching time. In *Proc. of the 8th ACM SIGPLAN-SIGACT symposium on principles of programming languages (POPL81)*, page 164176, 1981.

[59] P. Maier. A set-theoretic framework for assume-guarantee reasoning. In F. Orejas, P. G. Spirakis, and J. Leeuwen, editors, *Proceedings of the 28th International Colloqium on Automata, Languages, and Programming (ICALP), LNCS 2076*, volume 2076 of *Lecture Notes in Computer Science*, pages 821–834. Springer, 2001.

[60] R. Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer-Verlag, 1980.

[61] C. Păsăreanu, 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.

[62] 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. *FMSD*, 32(3):175–205, 2008.

[63] M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley, 1994.

[64] R. C. Gronback. *Eclipse Modeling Project, A Domain-Specific Language (DSL) Toolkit*. Addison-Wesley, 2009.

[65] 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.

[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. *Modeling and verification of randomized distributed real-time systems*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 1995.

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

[69] R. Segala. Probability and Nondeterminism in Operational Models of Concurrency. In C. Baier and H. Hermanns, editors, *CONCUR 2006 – Concurrency Theory*, volume 4137 of *Lecture Notes in Computer Science*, pages 64–78. Springer-Verlag, 2006.

[70] R. Spalazzese and P. Inverardi. Mediating connector patterns for components interoperability. In *ECSA2010*, 2010, LNCS - To appear.

[71] R. J. VanGlabbeek, S. A. Smolka, and B. Steffen. Reactive, generative, and stratified models of probabilistic processes. *Inf. Comput.*, 121:59–80, August 1995.

[72] M. Y. Vardi. Automatic verification of probabilistic concurrent finite state programs. In *Proc. 26th Annual Symposium on Foundations of Computer Science*, pages 327–338. IEEE Computer Society, 1985.

[73] D. Xu, G. Gössler, and A. Girault. Probabilistic contracts for component-based design. In A. Bouajjani and W.-N. Chin, editors, *Automated Technology for Verification and Analysis*, volume 6252 of *Lecture Notes in Computer Science*, pages 325–340. Springer-Verlag, 2010.