Università degli Studi di Pisa

Dipartimento di Informatica
Dottorato di Ricerca in Informatica
Settore Scientifico Disciplinare: INF/01

Ph.D. Thesis

Specification and Verification of Contract-Based Applications

Davide Basile

Supervisor
Pierpaolo Degano

Supervisor
Gian-Luigi Ferrari

2016

Abstract

Nowadays emerging paradigms are being adopted by several companies, where applications are built by assembling loosely-coupled distributed components, called *services*. Services may belong to possibly mutual distrusted organizations and may have conflicting goals. New methodologies for designing and verifying these applications are necessary for coping with new scenarios in which a service does not adhere with its prescribed behaviour, namely its *contract*.

The thesis tackles this problem by proposing techniques for *specifying* and *verifying* distributed applications. The first contribution is an *automata-based model checking* technique for ensuring both *service compliance* and *security requirements* in a composition of services. We further develop the automata-based approach by proposing a novel *formal model of contracts* based on tailored finite state automata, called *contract automata*. The proposed model features several notions of *contract agreement* described from a language-theoretic perspective, for characterising the modalities in which the duties and requirements of services are fulfilled. Contract automata are equipped with different *composition operators*, to uniformly model both single and composite services, and techniques for synthesising an *orchestrator* to enforce the properties of agreement. *Algorithms* for verifying these properties are introduced, based on *control theory* and *linear programming techniques*. The formalism assumes the existence of possible malicious components trying to break the overall agreement, and techniques for detecting and banning eventually *liable services* are described. We study the conditions for dismissing the central orchestrator in order to generate a *distributed choreography* of services, analysing both *closed* and *open* choreographed systems, with *synchronous* or *asynchronous* interactions. We relate contract automata with different *intutionistic logics* for contracts, introduced for solving mutual *circular dependencies* between the requirements and the obligations of the parties, with either *linear* or *non-linear* availability of resources. Finally, a *prototypical tool* implementing the theory developed in the thesis is presented.

Acknowledgements

These years as a PhD student have somehow changed my life. In this few lines I wish to express my gratitude to all the individuals who helped me in this journey, surely without them this thesis would have not been possible.

Firstly, my special appreciation goes to my PhD advisors, Professor Pierpaolo Degano and Professor Gian-Luigi Ferrari. I warmly thank them for all the hours we spent together, transforming ideas and sketches into formal models, algorithms and all you will find in the next pages. They have been great mentors and friends to me, and working with them has been an honour. They taught me how to pursue scientific research, always being enthusiastic, supportive, very helpful and patient in carefully reading, understanding, and improving all the intricacies and technical details [1]. It is difficult to overstate their impact into this thesis.

I would like to express my deepest gratitude to Professor Emilio Tuosto, whom I had the pleasure to work with during the months I spent at the University of Leicester. I still have stored in some folder all the pictures of his whiteboard, full of nodes, arrows and labels, shaping what has become a chapter into this thesis. I thank him for his contribution into this thesis, and also for helping me settling down in Leicester.

I am indebted to Professor Giancarlo Bigi, who has provided valuable assistance and precious insights that helped me dealing with Linear Programming techniques.

I am much obliged to my dissertation committee, Professor Massimo Bartoletti and Professor Roberto Bruni. Their precious comments, during the various evaluation phases of the PhD programme, enhanced the quality of this thesis.

Moreover, I would like to thank my reviewers, Professor Hugo Torres Vieira and Professor Marco Carbone, that read with such care a preliminary version of this document, returning insightful observations that improved this work.

I would also like to thank Dr. Stefania Gnesi and Dr. Felicita Di Giandomenico, for their support and understanding during this final year of completion of my thesis.

I am really glad I met many nice persons during the months I spent in Leicester. I had a really nice time there, and I was very sad when I came back to Pisa (which was something I could not expect, considering the gloomy English weather). It is impossible to mention all of them, I try with Mariano, Gabriela, Octavian, Al Sokkar, Muz, Sam, Samuel, Matt, Alexey, Xristina, Sarah, Arnab, Alex, all the guys who borrow me their guitars at the various jam sessions, and especially Richard the one third.

[1] I remember when one of the proofs was named "proof by intimidation" :)

Colgo l'occasione per ringraziare i colleghi/amici con cui ho condiviso svariate pause pranzo [2], Matteo, Lillo, Gianluca, Luca, Antonio, Roberto, Simone, Giovanni, Manuela, Antonella e tutti gli altri che mi scuso di non aver menzionato. Vorrei inoltre ringraziare tutti gli amici con cui ho passato piacevoli serate nonostante ci trovassimo a Pisa, chi è andato via e chi è rimasto, citarvi tutti è impossibile, ma ringrazio in particolare Pietro e Alessandra, con cui ho condiviso i (pochi) momenti di svago in questi ultimi mesi.

Ringrazio inoltre Margherita, Piero, Giovanni, Giorgio, Daniele, e Miriam, per il vostro affetto ed i consigli che mi avete dato e continuate a darmi, so di poter sempre contare su di voi e spero di rivedervi presto, grazie di cuore.

Davide

[2] ed il pluri-inflazionato caffé lungo in tazza grande macchiato freddo

"...and now it begins"
Ser Arthur Dayne

Contents

List of Definitions, Lemmata and Theorems

List of Figures

Introduction

Nowadays most of ICT infrastructures, as for example financial, business, defence and healthcare systems, are largely depending on different heterogeneous interorganizational digital systems, that cooperate with each other without human intervention through the Internet. Common examples of these technologies are Service Oriented Computing and Cloud Computing. In the view of software-as-a-service, applications are built by assembling loosely-coupled services provided by possibly mutually distrusted organizations, where services may collaborate or compete in order to reach their goals.

Quoting [TBB03]:

> *"Despite advances in programming language and development environment technologies, the basic paradigm for constructing and maintaining software has altered little since the 1960s (...) Further shifting the focus from providing software to describing and delivering a service moves the focus away from the constraints that traditional software construction, use, and ownership models impose."*

A main challenge for computer scientists is represented by the introduction of new methodologies that are capable of designing distributed applications as well as guaranteeing that their composite behaviour must agree to specified safety requirements. Critical problems on these computing systems, as failures or attacks by hackers, represent a threat for our society. For example, malicious services inside composite applications could perform undesired actions, as stealing sensible information. Serious economic damages could result from failures in business activities, or in an even worst scenario, human lives could be in danger in case of failures of safety-critical applications.

Implementing distributed components that safely interact in an unknown and malicious environment is a hard task. Among the many causes we cite the heterogeneity of the software components that are provided by different entities with possibly conflicting goals, and misbehaviours that may take an unpredictable variety of different forms.

A feasible solution consists in studying specific requirements of a composite application, in order to guarantee them also in the presence of malicious components. This requires the development of new techniques to design, analyse, and implement service-based applications.

Formal methods play an important role in providing models and tools for helping developers in building applications where safety is a main concern from the early design

phases of a system. For example, *model checking* is a widely adopted technique for verifying properties of systems that are modelled through abstract formalisms.

The goal of the thesis is to provide techniques for specifying and verifying properties of service oriented computing-based distributed systems.

Recently, so called *contracts* have been introduced as an abstract formalism for specifying and verifying distributed services. Contracts naturally support the modelling of *service composition*, characterized in terms of orchestration and choreography. In an *orchestrated* approach, a distinguished entity called the *orchestrator* regulates how the composite service evolves. In a *choreographed* approach, the distributed services autonomously interact without a central coordinator. We will study different formalisms for contracts, their compositional operators and coordination mechanisms. The main contributions of the thesis are:

1. We introduce an automata-based model checking technique for ensuring both *service compliance* and *security requirements* in a composition of services, using a well-known abstract model of service contracts. Intuitively, service compliance ensures communication safety, and the security requirements are modelled as regular properties over sequences of security-relevant events. Service compliance is proved to be a safety property, paving the way to its verification through efficient model checking techniques.

2. We propose a novel formal model of contracts based on suitable finite state automata, called contract automata. Contract automata specify what each service is going to guarantee and offer and what in turn it expects and requires. Either single or composite services can be expressed and composed within our formalism. Different compositional operators characterising different policies of orchestration are introduced, to model different interaction patterns. Moreover, properties of *contract agreement* are defined from a language-theoretic perspective, characterising the modalities in which the duties and requirements of services are fulfilled. Algorithms for verifying these properties are also developed. Finally, we assume the existence of possible malicious components trying to break the overall agreement, and we develop techniques for detecting and banning eventually liable services. A distinguished feature of our approach is that verification algorithms are based on control theory and the optimization of network flows. For the last case, we import efficient techniques originally introduced for solving linear programming problems.

3. We relate the contract automata approach with different service coordination mechanisms and logics for contracts. We study the conditions for dismissing the central orchestrator in order to obtain a choreographic interaction of services, thus reducing the communication overhead. We will consider two intuitionistic logics for contracts, introduced for solving circular dependencies among the interacting parties, with either linear or non-linear availability of resources. In particular, we will consider the Horn fragments of these logics, which have a neat interpretation in

terms of contracts. Verifying the agreement in a composition of services and generating proofs of formulae in these logics are proved to be the same problem. This result sheds light on the logical interpretation of contract agreement and opens a novel perspective on the verification of logical properties of contracts.

4. As a proof of concept, we turned the developed theory into a prototype tool, so to completely mechanize our proposal.

Structure of the Thesis The thesis is structured as follows:

- **Chapter 1** briefly introduces the main subjects of the thesis, namely Service Oriented Computing, its coordination mechanisms and contracts. A brief survey on the corresponding literature is also discussed. All the material used throughout the thesis that does not represent an original contribution is introduced here;
- **Chapter 2** outlines a formal theory of contracts that supports verification of *service compliance* and of *security policies* enforcing access control over resources;
- **Chapter 3** introduces *contract automata*, a model for describing, composing and verifying contracts;
- **Chapter 4** establishes a correspondence between logics for contracts and contract automata;
- **Chapter 5** investigates the relations between an *orchestrated* model (contract automata) and a *choreographed* model (*communicating machines*);
- **Chapter 6** describes a prototypical tool-kit supporting contract automata.

The material presented in this thesis have been published in the following international workshops, conferences and journals:

- the content of Chapter 2 has been presented in:
  - Basile, D.: Service interaction contracts as security policies. In: 12th Italian Conference on Theoretical Computer Science, ICTCS 2012, Varese, Italy, Septemper 19-21, 2012, available online at `http://ictcs.di.unimi.it/papers/paper_28.pdf`;
  - Basile, D., Degano, P., Ferrari, G.L.: Secure and unfailing services. In: Proceedings of the 12th International Conference on Parallel Computing Technologies (PaCT), St. Petersburg, Russia, September 30 - October 4, 2013. Malyshkin, V. (ed.) LNCS, vol. 7979, pp. 167–181. Springer (2013);
  - Basile, D., Degano, P., Ferrari, G.L.: A formal framework for secure and complying services. The Journal of Supercomputing, volume 69(1), pp. 43–52 (2014), ISSN: 1573-0484;

- the content of Chapter 3 and Chapter 4 has been presented in:
  - Basile, D., Degano, P., Ferrari, G.L.: Automata for service contracts. In: Hot Issues in Security Principles and Trust - 2nd Workshop, HOTSPOT 2014, Grenoble, France, April 12, 2014;
  - Basile, D., Degano, P., Ferrari, G.L.: Automata for analysing service contracts. In: Trustworthy Global Computing - 9th International Symposium, TGC 2014, Rome, Italy, September 5-6, 2014. Revised Selected Papers, Lecture Notes in Computer Science, vol. 8902, pp. 34–50. Springer (2014);
  - Basile, D., Degano, P., Ferrari, G.L.: Automata for specifying and orchestrating service contracts. To appear in Journal of Logical Methods in Computer Science (2016), ISSN: 1860-5974;
- the content of Chapter 5 has been presented in:
  - Basile, D., Degano, P., Ferrari, G.L., Tuosto, E.: From orchestration to choreography through contract automata. In: Lanese, I., Lluch-Lafuente, A., Sokolova, A., Vieira, H.T. (eds.) Proceedings 7th Interaction and Concurrency Experience, ICE 2014, Berlin, Germany, 6th June 2014. EPTCS, vol. 166, pp. 67–85 (2014);
  - Basile D. , Degano P. , Ferrari G.L. , Tuosto E.: Relating two automata-based models of orchestration and choreography, Journal of Logical and Algebraic Methods in Programming, Volume 85, Issue 3, April 2016, Pages 425-446, ISSN 2352-2208;
- the content of Chapter 6 has been presented in:
  - Basile, D., Degano, P., Ferrari, G.L., Tuosto, E.: Playing with our CAT and Communication-Centric Applications. In: 36th IFIP International Conference on Formal Techniques for Distributed Objects, Components and Systems (FORTE), Held as Part of the 11th International Federated Conference on Distributed Computing Techniques, DisCoTec 2016, Heraklion, Crete, Greece, June 6-9, 2016. Revised Selected Papers, Lecture Notes in Computer Science, volume 9688, pp.62–73. Springer (2016).

Chapter 1

Preliminaries

In this chapter we overview the main concepts addressed in the thesis, and we introduce all the techniques that will be used throughout the thesis and that do not represent an original contribution.

We start by introducing Service Oriented Computing and its coordination mechanisms, namely orchestration and choreography (Section 1.1). We will discuss formal models for specifying service contracts, and the corresponding literature in Section 1.2. Once orchestration and choreography have been introduced, in Section 1.3 we focus on a widely adopted formalism related to choreographed distributed applications, namely communicating machines. Different characterizations of contracts in terms of logic formalisms are discussed in Section 1.4. The remaining sections introduce several techniques that will be used throughout the thesis for checking the properties of interest, namely model checking (Section 1.5), language-based security (Section 1.6), control theory (Section 1.7) and operations research (Section 1.8).

While Sections 1.1 and 1.2 provide a general overview of Service Oriented Computing and contracts, the remaining sections address topics specific to the different notions tackled in the thesis. Hence, in order to improve the overall readability, we link the specific sections to the corresponding chapters:

- Chapter 2: the contract formalism adopted in this chapter is introduced in Section 1.2.1. We will extend History Expressions (Section 1.6) to include operators for describing interactions of services; and we will extend the corresponding model checking techniques (Section 1.5);

- Chapter 3: the techniques borrowed from Control Theory (Section 1.7) and Operations Research (Section 1.8) will be used to verify properties of contracts and to solve circularity issues (Section 1.4), under the assumption that participants may behave maliciously (Section 1.4.3);

- Chapter 4: we will relate our contract formalism with different logics that are introduced in Section 1.4;

- Chapter 5: our orchestrated model will be related to a choreographed model called communicating machines, introduced in Section 1.3.
- Chapter 6: techniques based on optimization research (Section 1.8) will be adopted for mechanising our proposal.

1.1 Service Oriented Computing

The opportunities of exploiting distributed services are becoming an imperative for all organizations and, at the same time, new programming techniques are transforming the ways distributed software architectures are designed and implemented. Distributed applications nowadays are constructed by assembling together computational facilities and resources offered by third-party providers. These applications interact in an unknown and dangerous environment, composed by a heterogeneity of computational entities that are provided by different mutual distrusted organizations. Guaranteeing that specified safety requirements are met represents a challenge for computer scientists.

Service-Oriented Computing [WB10, PG03, Pap12] is a paradigm for designing distributed applications that are built by combining different fine-grained and loosely-coupled distributed components. The basic building blocks are services, which are provided by different organizations, and can be autonomous, platform-independent, reusable and highly portable. Services are equipped with suitable interfaces (roughly) describing the offered computational facilities and the requirements. Services can be combined to accomplish a certain computational task or to form a more complex service. A service exposes both the functionalities it provides and the parameters it requires. Clients exploit service public information to discover and bind the services that better fit their requirements. Services can be discovered according to the information provided in their interface.

Through Service-Oriented Computing companies can implement applications that are low-cost, interoperable, secure, reliable, massively distributed, and can reduce the cost and time for upgrading their IT systems by adding new features or deleting old ones. *Web Services* [ACKM04] are a common example of Service-Oriented Computing approach. The Web Service Description Language (WSDL) is used for describing the messages to be exchanged between services and the Simple Object Access Protocol (SOAP) take care of the interaction between the parties. Web services are published and discovered through the Universal Description Discovery and Integration (UDDI), while the standard for web services composition and orchestration is the Business Process Execution Language (BPEL) [Jur06]. Service Oriented Computing relies on different coordination mechanisms, described below.

1.1.1 Service Coordination: Orchestration and Choreography

Service coordination is a fundamental mechanism of the service-oriented approach, where *coarse-grained* applications are built by assembling together multiple, *fine-grained*

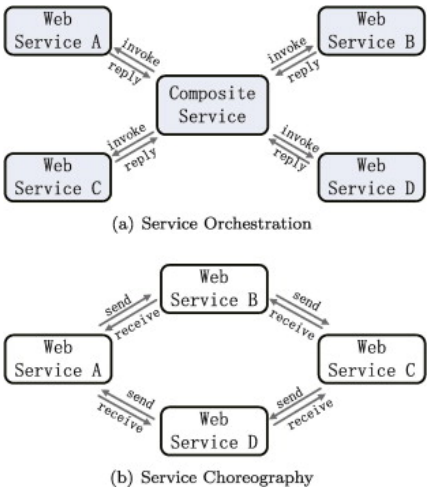


Figure 1.1: Two service coordination mechanisms: orchestration and choreography

and *loosely-coupled* services. Service coordination policies differ on the supports to the interactions adopted to pass information among services. A coordination policy successfully governs the interacting services when no request is left behind, e.g. when a principal is invoking a service that nobody will ever be willing to offer. Therefore distributed services must agree on a shared coordination policy capable of fulfilling all service demands.

Service composition is the aggregation of multiple services into a single service in order to develop more complex functions. Resulting composite services can be used as basic services in further hierarchical service compositions or offered as complete applications to service clients [IGH+11, PTDL07]. There exists two different approaches to services composition, depending on their coordination behaviour [Pel03]:

- service orchestration (centralized coordination);
- service choreography (distributed coordination).

In literature precise definitions of the concepts of *orchestration* and *choreography* are missing, and only intuitive descriptions have been given so far. In Figure 1.1 two diagrams showing the different coordination mechanisms are displayed, namely *service orchestration*, where each service is invoked by the orchestrator, sometimes called the composite service (as in Figure 1.1); and *service choreography*, where services interact autonomously via messages.

1.1.2 Choreography

There is common consensus that the distinguishing element of a disciplined choreographic model is the specification of a so-called *global viewpoint* detailing the interactions among distributed participants and offering a description of their expected communication behaviour in terms of message exchanges.

This intuition is best described in W3C words [KBR+05]:

> *Using the Web Services Choreography specification, a contract containing a global definition of the common ordering conditions and constraints under which messages are exchanged, is produced that describes, from a global viewpoint [...] observable behaviour [...]. Each party can then use the global definition to build and test solutions that conform to it. The global specification is in turn realised by combination of the resulting local systems [...]*

Noteworthy, the excerpt above points out that in a disciplined choreography local behaviour should then be realised by conforming to the global viewpoint in a "top-down" fashion. Hence, the relations among the global and local specifications are paramount.

1.1.3 Orchestration

The concept of orchestration is more controversial. In this thesis we adopt a widely accepted notion of orchestration [DD04, ACKM04, Pap12] envisaging the distributed coordination of services via the mediation of a distinguished participant that – besides acting as provider of some functionalities – regulates the control flow by exchanging messages with partner services according to their exposed communication interface. In Peltz's words [Pel03]:

> *Orchestration refers to an executable business process that can interact with both internal and external Web services. The interactions occur at the message level. They include business logic and task execution order, and they can span applications and organisations to define a long-lived, transactional, multi-step process model. [...] Orchestration always represents control from one party's perspective.*

The "executable process" mentioned by Peltz is called *orchestrator* and specifies the workflow from the "one party's perspective" describing the interactions with other available services, so to yield a new composed service. This description accounts for a composition model enabling developers to combine existing and independently developed services.

The orchestrator then "glues" them together in order to realise a new service, as done for instance in Orc [Mis05]. This is a remarkable aspect since the services combined by an orchestrator are not supposed to have been specifically designed for the service provided by the orchestrator and can in fact be (re)used by other orchestrators for realising different services.

Other authors consider orchestration as the description of message exchanges among participants from the single participants' viewpoint *without assuming the presence of an orchestrator*. For instance, in [QZCY07, YZCQ07] the local specifications of a choreography are considered the orchestration model of the choreography itself. In this thesis we do not consider each local specification of a choreography as an orchestration, because it may obscure the matter; rather local specifications are tailored to (and dependent of) the corresponding party of the choreography instead of being independently designed.

In other words, such local specifications correspond to automata-oriented choreography adopted here (see Section 1.3) and in [LTY15], as well as to the process-oriented choreography of [LGMZ08]. Instead, in an orchestration model, each participant defines and exposes its own communication pattern which is then (somehow) assembled in an orchestration. Abstracting from technological aspects, we can describe our approach using Ross-Talbot's words [Tal]:

> *In the case of orchestration we use a service to broker the interactions between the services including the human agents and in the case of choreography we define the expected observable interactions between the services as peers as opposed to mandating any form of brokering.*

There are several standards for orchestration, such as WS-BPEL (Web Service Business Process Execution Language) an XML-based language used to describe composed Web Services [Jur06].

In Chapter 3 a formal model for describing services will be introduced which adopts orchestration as coordination model. We will investigate the relations between the orchestration model and a choreographic one in Chapter 5.

1.2 Contracts

Software architectures that truly support distributed services require several operational tools to be in place and effective. Indeed, services are provided by possibly mutual distrusted organizations, and they may have conflicting goals. Ensuring the reliability of a composite service is important to avoid economic losses, but also catastrophic failures for safety critical applications.

When services are made available by different companies, understanding and fulfilling the behavioural obligations of services is crucial to determine whether the interactive behaviour is consistent with the requirements.

However, in general standard analysis and verification techniques for distributed systems cannot be applied to third-party services, to which no assumptions can be made. New compositional techniques must thus cope with services that could not fulfil their prescribed behaviour, either unintentionally or maliciously. Moreover, they need to ensure correct interactions even in the case of mutual circular dependencies between the requirements and the obligations of the parties (see Section 1.4).

Recently, so called *contract-oriented design* [BTZ12] has been introduced as a suitable methodology where the interaction between clients and services is specified and regulated by formal specifications, named *contracts* [EKW+09].

Service Contracts are the standard mechanisms to describe the external observable behaviour of a service, as well as the security accountability. Service contracts can be used for guaranteeing that all the services are capable of successfully terminating their tasks without rising security exceptions.

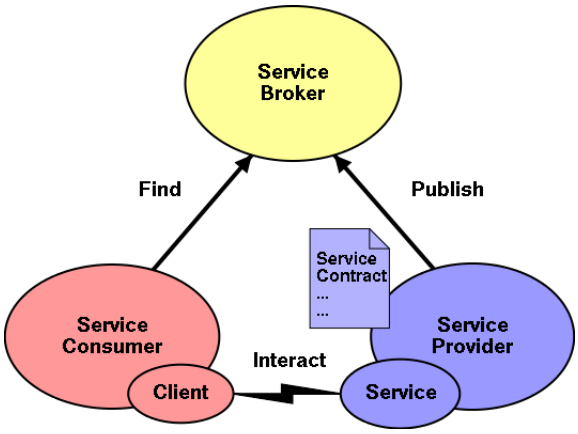


Figure 1.2: Service-Oriented Architecture

They can be used for describing the requests that a service wants from the system, and also the offers that it can provide. Contracts may then help characterize the behavioural conformance of a composition of different services [BCZ15]. This specification is used for guaranteeing that a composition of services does not lead to spurious results. Moreover, it is possible to formally verify that this composition enjoys certain properties, such as progress of interactions, or agreements of the parties. Indeed, a semantic model for contracts is necessary to reason about the behaviour of service assemblies, so as to have formal verification techniques for properties that services must respect. This is useful for the design and the realization of complex service-based applications which must be correct and safe, as otherwise it is extremely hard to certify reliability of such complex services.

The management of service contracts is typically tackled by service providers through the notion of *Service-Level Agreement* [WB10, Pap12], which have to be accepted by the client before using the service. SLAs are documents that specify the level(s) of service being sold in plain language terms.

In Figure 1.2 a Service-Oriented Architecture [PL03] is depicted. Service contracts are published in a trusted repository, and are accessed through service discovery. A Service Broker (i.e. the coordinator) is in charge of finding an agreement required and provided by the service contracts. Obviously, this arrangement is based on the contracts of the involved services, and ensures that all requests are properly served when all the duties are properly kept. The coordinator organises then the overall service coordination policy and proposes the resulting contract agreement to all the parties, so that services can start their interactions. Eventually, if any involved service does not fulfil its duties (i.e. it violates its contract), the coordinator will be capable of finding and blaming those liable participants.

In the literature, several notions of contracts are available.

Outside the world of services, contracts are used in *Component-Based Software Engineering* [Crn02] for specifying the behaviour of components. In Design-by Contract methodologies [Mey92], a contract extends the ordinary definition of abstract data types

with preconditions, postconditions and invariants. They represent the conditions and obligations that software components must respect.

Concerning services, *behavioural contracts* use formal models as *process algebras* [BW90] to describe the abstract behaviour of services and checking their correctness (see Section 1.2.1). Contracts have been applied both for coordination policies based on orchestration (see Section 1.2.2), and choreography (see Section 1.2.3). Several logics for contracts have been described in literature, and are used for identifying the obligations of the parties, as discussed in Section 1.4.

In the next sections we will describe some of the proposals available in the literature for specifying and verifying service contracts, focussing on those closer to the results presented in the thesis.

1.2.1 Behavioural Contracts

Behavioural contracts are expressed in [CGP09] as CCS terms [Mil89], where the interactions between services are modelled via I/O actions. In Chapter 2 we will adopt this contract model for building our model checking technique. Contracts are built with three operators: prefix $\alpha.\sigma$, internal choice $\sigma_1 + \sigma_2$ and external choice $\sigma_1 \oplus \sigma_2$. A contract $\alpha.\sigma$ performs the action α and then continues as σ, where α is an abstraction of an input/output operation from one service to the other. An output action is topped by a bar, i.e. $\overline{a}$, following the CCS notation. We consider a set of communication actions $\mathsf{Comm} = \{a, \overline{a}, b, \overline{b}, \ldots\}$, where as usual we have the involution $\overline{\overline{a}} = a$. The contract $\sigma_1 \oplus \sigma_2$ describes a service that internally decides whether to continue as σ_1 or σ_2. In the contract $\sigma_1 + \sigma_2$ it is the other party that decides whether to continue as σ_1 or σ_2. Contract terms are regular trees, so they can be see as the solution of some recursive definitions. Moreover, 0 is the contract of services that do not perform any action. We follow the standard convention of omitting trailing 0's.

Definition 1 (Behavioural contracts)**.** *A behavioural contract is a finite term generated by the following grammar:*

$$\sigma ::= \alpha.\sigma \mid \sigma_1 \oplus \sigma_2 \mid \sigma_1 + \sigma_2 \mid 0 \qquad \alpha \in \mathit{Comm}$$

We write $\sum_{i\in I} \sigma_i$ for $\sigma_1 + \sigma_2 + \cdots + \sigma_n$ and $\bigoplus_{i\in I} \sigma_i$ for $\sigma_1 \oplus \sigma_2 \oplus \cdots \oplus \sigma_n$ where $|I| = n$, with $(\sigma, +)$ and $(\sigma, \oplus)$ abelian groups with identity 0 and $+, \oplus$ idempotent. The transition relation of behavioural contracts is defined below:

Definition 2 (Transition Relation of Behavioural Contracts)**.** *Let $\sigma \overset{\alpha}{\not\rightarrow}$ be the least relation such that:*

$$0 \overset{\alpha}{\not\rightarrow} \qquad \beta.\sigma \overset{\alpha}{\not\rightarrow} \quad \text{if } \alpha \neq \beta$$

$$\sigma_1 \oplus \sigma_2 \overset{\alpha}{\not\rightarrow} \quad \text{if } \sigma_1 \overset{\alpha}{\not\rightarrow} \text{ and } \sigma_2 \overset{\alpha}{\not\rightarrow} \qquad \sigma_1 + \sigma_2 \overset{\alpha}{\not\rightarrow} \quad \text{if } \sigma_1 \overset{\alpha}{\not\rightarrow} \text{ and } \sigma_2 \overset{\alpha}{\not\rightarrow}$$

The transition relation *of contracts, noted $\xrightarrow{\alpha}$, is the least relation satisfying the rules:*

$$\alpha.\sigma \xrightarrow{\alpha} \sigma \qquad \frac{\sigma_1 \xrightarrow{\alpha} \sigma_1' \quad \sigma_2 \xrightarrow{\alpha} \sigma_2'}{\sigma_1 + \sigma_2 \xrightarrow{\alpha} \sigma_1' \oplus \sigma_2'} \qquad \frac{\sigma_1 \xrightarrow{\alpha} \sigma_1' \quad \sigma_2 \xrightarrow{\alpha} \sigma_2'}{\sigma_1 \oplus \sigma_2 \xrightarrow{\alpha} \sigma_1' \oplus \sigma_2'}$$

$$\frac{\sigma_1 \xrightarrow{\alpha} \sigma_1' \quad \sigma_2 \not\xrightarrow{\alpha}}{\sigma_1 + \sigma_2 \xrightarrow{\alpha} \sigma_1'} \qquad \frac{\sigma_1 \xrightarrow{\alpha} \sigma_1' \quad \sigma_2 \not\xrightarrow{\alpha}}{\sigma_1 \oplus \sigma_2 \xrightarrow{\alpha} \sigma_1'}$$

and closed under symmetric rules for internal and external choices. We write $\sigma \xrightarrow{\alpha}$ if there exists σ' such that $\sigma \xrightarrow{\alpha} \sigma'$.

Note that the transition relation models the evolution of a single service from the viewpoint of the communicating party, hence there is no synchronization rule. The interactions between two services will be characterized in Definition 4. We now recall the notion of *observable ready sets* [CGP09]. Intuitively, these sets represent the communication actions that a service is ready to execute, so characterising the different behaviour of internal and external choice. Roughly, a single action at a time is offered by an internal choice, while in the external choice all the actions are available at the same time.

Definition 3 (Observable Ready Sets of Contracts)**.** *Let σ be a contract, and $\mathcal{P}(\mathsf{Comm})$ be the finite set of parts of Comm, called ready sets. Moreover, let $\sigma \Downarrow S$ be the least relation between contracts σ and ready sets $S \in \mathcal{P}(\mathsf{Comm})$ such that:*

$$0 \Downarrow \emptyset \quad \alpha.\sigma \Downarrow \{\alpha\} \qquad \frac{\sigma_1 \Downarrow S \text{ or } \sigma_2 \Downarrow S}{\sigma_1 \oplus \sigma_2 \Downarrow S} \qquad \frac{\sigma_1 \Downarrow S, \sigma_2 \Downarrow R}{\sigma_1 + \sigma_2 \Downarrow S \cup R}$$

Example 1. *We show some example of observable ready sets:*

- $(\overline{a_1} \oplus \overline{a_2}) \Downarrow \{\overline{a_1}\}$ *and* $(\overline{a_1} \oplus \overline{a_2}) \Downarrow \{\overline{a_2}\}$*;*
- $(a_1 + a_2) \Downarrow \{a_1, a_2\}$*;*

When a service offers some set of actions and the client can synchronize with one of them performing the corresponding co-actions up to the point of client termination, the two contracts are *compliant*. Given a service contract, it is possible to determine the set of clients that comply (written $\vdash$) with it. Examples of compliant services (for simplicity we only consider immediate actions) are: $a \oplus \overline{b} \vdash \overline{a} + b$ and also $a \oplus \overline{b} \vdash \overline{a} + b + c$, moreover $a \vdash \overline{a} + b$ but $a \not\vdash \overline{a} \oplus b$ since $\overline{a} \oplus b$ can internally decide to perform b and the interaction gets stuck. The sub-contract relation, written $\sigma_1 \preceq \sigma_2$, characterizes all services compliant with σ_1 that also comply with σ_2. It is possible to replace a service with one that offers more choices, allowing possible compositions and reuse of services. For example $a \preceq a + b$ (width sub-typing) or $a \preceq a.b$ (depth sub-typing), and with one that is more deterministic, for example $a \oplus b \preceq a$.

We now introduce the notion of service compliance. Note that the definition below does not require both parties to terminate: the client can terminate whenever all its operations have been completed. Hereafter, let $\overline{S} = \{\overline{a} | a \in S\}$.

Definition 4 (Compliance of Behavioural Contracts)**.** *Two contracts σ_c and σ_s are* compliant*, written $\sigma_c \vdash \sigma_s$, if for all observable ready sets C, S:*

(1) $\sigma_1 \Downarrow C$ and $\sigma_2 \Downarrow S$ implies that $C = \emptyset$ or $C \cap \overline{S} \neq \emptyset$, and

(2) $\sigma_1 \xrightarrow{\alpha} \sigma_1' \wedge \sigma_2 \xrightarrow{\overline{\alpha}} \sigma_2'$ *implies* $\sigma_1' \vdash \sigma_2'$.

Note that the compliance relation is indeed defined in terms of the largest relation over contracts enjoying properties (1) and (2) above.

Example 2. *We model an authentication scenario where the server and the client are modelled respectively by contracts* σ_1 *and* σ_2 *below:*

$$\sigma_1 = Login.(\overline{ValidLogin}.\sigma_1' \oplus \overline{InvalidLogin})$$

$$\sigma_2 = \overline{Login}.(ValidLogin.\overline{\sigma_1'} + InvalidLogin)$$

The contracts are now described. After the login action is received by σ_1*, it checks if the user name is valid and returns a Valid login or Invalid login message. If the user is authenticated, the service continues with* σ_1' *. The service whose contract is* σ_2 *sends the login action and then waits for the authentication message. If the login is valid then it continues with* $\overline{\sigma_1'}$*. According to Definition 4, the two contracts are compliant (by assuming* $\overline{\sigma_1'} \vdash \sigma_1'$ *).*

Behavioural contracts in the literature

The approach of Section 1.2.1 is updated to a multi-party version by extending the π-calculus [MPW92] with the notions of non-deterministic choice in [CP09] (see Section 1.2.4).

A CCS-like process calculus, called BPEL *abstract activities*, is used in [LP15] to represent BPEL activities [OTC07], for connecting BPEL processes with contracts in [CGP09]. The calculus is endowed with a notion of compliance and sub-contract relation.

A mechanism for recovering from a stuck computation is introduced in [BDLd15], which is built on top of the contracts of [CGP09]. The external choice is called *retractable*, and a client contract which decides whether to send a message a or b, is compliant with a server which receives only the a message. This is because if the client decides to send b, it can retract the choice and performs the correct operation.

The compliance relations studied in [CGP09, CP09, LP15, Pad10, BvBd15, BDLd15] are mainly inspired by testing equivalence [dNH83]: a CCS process (in our case the service) is tested against an observer (the client), in two different ways. A service *may-satisfy* a client if there exists a computation that ends in a successful state, and a service *must-satisfy* a client if in every maximal trace (an infinite trace or a trace that cannot be prolonged) the client can terminate successfully.

The behaviour of web-services is described in [BSBM05] using automata. Only bi-party interactions are considered, i.e. interactions between a client and a server. Different notion of progress of interactions are introduced. For example, one of them requires that given two states (q_1, q_2), for each outgoing output transition from state q_1 there must be a corresponding outgoing input transition from state q_2 and vice-versa, and there must exist at least one path from the initial state that reaches the final state such that each intermediary state satisfies the above property. Compared to [CGP09] and the discussed extensions, in [BSBM05] there is no distinction between internal and external choice,

by assuming that an output corresponds to an internal decision while a request to an external one.

A notion of abstraction in the contracts of [CGP09] for exposing partial description of the behaviour is discussed in [BM10]. Abstract processes are introduced, which are a version of value-passing CCS [Mil89]. A symbolic semantic for evaluating an expression which possibly contains an abstracted value is described. In the case the value of the if condition is abstracted, then both branches are followed. A simulation-based abstraction is proposed, where the abstract process simulates the concrete one if it performs all the actions that are not hidden by the abstraction. The abstraction operator for contracts is introduced, where the main insight is that an external choice guarded by an input action turns into an internal choice if the input action is hidden by the abstraction operator. It has been proved that the abstraction operator preserves the subcontract relation [CGP09]. Abstract-processes allow multi-party interactions via actions, while this is not the case for the contracts of [CGP09], however recursive processes are not studied.

The problem of checking compliance decidability in different models of contracts is tackled in [ABZ13]. They propose behavioural contracts with bidirectional request-response operations, where request-response operations are decomposed into a sequence of send-receive-reply; and introduce two contract models inspired by WSCL and Abstract WS-BPEL [OTC07]. The difference between the two models is that the first can not describe intermediary activities of the service between the receive and the reply steps, while this is possible in the second. In WSCL contracts, there is an invoke operation and a *recreply* operation, which corresponds to a sequence of consecutively receive and reply. In BPEL contracts there is an invoke operation, a receive operation and a reply operation. The semantics corresponds to an asynchronous pi-calculus. In the invoke operation a channel is sent asynchronously and in parallel the process waits on the channel it has sent the reception of the acknowledgement. In the receive, the process receives the channel and binds it to a variable. In the reply step, the process sends the message in the channel bound at actual time. A process is able to successfully terminate by performing the $\surd$ action. A client is compliant with a service if for every maximal computation there exists a transition labelled with $\surd$. It has been proved that client-server compliance is decidable in WSCL while it is undecidable for BPEL. For the former a reduction of the model to a Petri net is given with an algorithm for deciding compliance. For the latter a reduction to the termination problem in Random Access Machines is provided.

1.2.2 Orchestration in the literature

In [Pad10] the approach of [CGP09] is extended by exploiting an orchestrator for managing the *sub-contract* relation. A contract σ_1 is sub-contract of σ_2 if σ_1 is more deterministic or allows more interactions or is a permutation of the same channels of σ_2. However, it is not always the case that a contract σ, compliant with σ_1, is also compliant with σ_2. A technique for synthesising an orchestrator is presented to enforce compliance of contracts under the sub-contract relation.

This approach is further extended in [BvBd15], where an orchestrator is synthesised from *session contracts*, where actions in a branching can only be all inputs or outputs. Only bi-party contracts are considered, and synthesis is decidable even in the presence of messages never delivered to the receiver (orphan messages). Two notions of compliance are studied: respectful and disrespectful. In the first, orphan messages and circularities are ruled out by the orchestrator, while in the second they are allowed.

The dichotomy orchestration-choreography has been discussed in several papers (see e.g., [Pel03]). A choreography is translated into an orchestration of services in [RDRM12], by adding controllers that make it realisable. It is assumed a possible delay in the transmission of messages that could cause the change of the order in which they are received. They deal with both a synchronous and an asynchronous scenario. In [BGG$^+$06] a bisimulation is used to establish a conformance relation (or its absence) between choreographed and orchestrated computations. We note that the orchestration model of [BGG$^+$06] - unlike the one we adopted in the thesis - envisages systems as the parallel composition of orchestrators.

In Chapter 5 we instead devise conditions to correlate orchestrated computations with local specifications of choreographies so to ensure that the former well-behave and correspond to communication-safe choreographies.

1.2.3 Choreographies in the literature

Different research problems have been identified and investigated for distributed choreographies, for instance *realizability* [LMZ13, BBO12, QZCY07, HB10], *conformance* [LP15, BB11, BZ09b, BZ08b], or *enforcement* [ARS$^+$13].

In [LGMZ08] an analysis of the relations between the *interaction-oriented* choreographies (i.e., global specifications expressed as interactions) and the *process-oriented* ones (i.e., the local specifications expressed as process algebra terms) is presented.

Session types have been introduced to reason over behaviour of choreographic communicating processes, and are used for typing channel names by structured sequences of types [DCD10]. Session types can be global or local. A *global type* represents a formal specification of a choreography of services in terms of their interactions. In [HYC08] a choreography of services is represented in terms of a global type. The projection of a safe global type to its components yields a safe *local type*, which is a term of a process algebra similar to those of [CGP09]. If the processes of a system can be typed with the corresponding local types, then it is proved that the system is also safe.

Conversely, from given safe local types, a choreography is synthesized with a "bottom-up" approach, as a safe global type in [LT12, LTY15]. It is proved that if the global type enjoys safety properties and progress, then this holds also for the end point. Moreover by projecting the global type to its components one obtains terms equivalent to the starting local types. This makes choreography models more flexible (for instance, choreographies have been exploited in [LS13] as a contract model for service composition).

In [LP15] the compliance and sub-contract relations are extended to deal with choreographies. Compliance is obtained by seeing a choreography as a compound service. A

client cannot interact with the choreography on actions already used while synchronising with other services, i.e. the choreography refinement is obtained by preventing the new service to interfere with the others in the network.

A transformation for amending choreographies that do not respect common syntactic conditions for projection correctness is presented in [LMZ13]. The starting point is a global choreography, which represents the system behaviour. Local points are then obtained through projection. In case the local points do not realise the global viewpoint, a transformation which reduces the amount of concurrency and adds hidden interactions is performed. Fresh participants are added to drive the local points with hidden actions, in order to enforce the admitted behaviours, described by the global viewpoint.

The problem of restricting the behaviour of discovered services in order to fulfil the collaboration prescribed by a choreography specification, called *choreography realizability enforcement*, is tackled in [ARS+13]. They propose a technique to automatically synthesise Coordination Delegates (CD), one for each service, in order to prevent a service to perform undesired operations. A distributed coordination algorithm is presented that implements the coordination logic that each CD has to perform in order to solve concurrency issues arising when two events are activated at the same time. Both the model and the implementation of each CD are generated.

In [BZ08b] contracts are represented in a language independent way, by means of labeled transition systems. They address the problem of checking, given a global description of an application (i.e. a choreography specification à la WS-CDL), if a service is conformant with a given role in the choreography, and can be used in any implementation of it. In the literature this problem is known as *choreography conformance*. They also check *contract refinement*, that is checking if a contract is a refinement of another one, and can thus be safely substitute. They operate in an asynchronous settings, where messages are exchanged through queues. The projection of a choreography into a role produces a labeled transition system instead of a contract described in a given language. The Business Process Modelling Notation 2.0 (BPMN2) [(OM11] includes meta-model and related graphical notation for specifying service choreographies.

1.2.4 Contracts and Session Types

Session types are restricted to perform only output actions in internal choices and input actions in external choices, while this is not the case for contracts [BCZ15]. The main difference between contracts and session types is that contracts record the overall behaviour of a process, while session types project this behaviour onto the private channels that a process uses.

Session types impose overly restrictions on their actions while contracts only work for network with fixed topology. A contract language based on the π-calculus is proposed in [CP09], for filling the gap between session types and contracts. Indeed, contracts are also able to send channels in addition to messages. A standard process calculi syntax with parallel composition and scope extrusion is used. Input and output actions are denoted respectively as $\alpha!f$ and $\alpha?f$ where f is a pattern for matching messages or

channels. Contracts include three terminal behaviours: the deadlocked process **0**, the successfully terminated process **1** and Ω that denote indefinite progress without any further interactions with the environment. Compared to [CGP09], a novel notion of compliance is presented: T is compliant with S (written $T \lhd S$) if every computation of $T|S$ leading to a state where the residual of T is stable is such that either the residual of T has successfully terminated and the residual of S will eventually terminate successfully, or the two residuals can eventually synchronize. The synchronization may only be available after some time. Divergence of the test T is ignored, since it implies that T is making progress autonomously; divergence of the contract S being tested is ignored, in the sense that all the visible actions it provides are guaranteed, for example $c?a.1 \lhd c!a.1 + \Omega$.

The contracts of [CGP09] are proved to be a model of first-order session types [GH05] in [BH12]. In particular, a subset of these contracts called *session contracts* is selected, and the main result of this paper shows that first-order session types can be smoothly embedded into the theory of contracts, which thus appears to be a more general formalism. This approach is then extended in [BH14] that introduces a notion of higher-order contracts and relates them to higher-order session types that also handle session delegation. To give a behavioural interpretation in terms of contracts, higher order session contracts are introduced, which are a subset of higher-order service contracts, with a novel behavioural theory. In particular, a notion of mutual compliance is introduced, and the sub-contract relation is extended to set of contracts that are in mutual compliance.

1.2.5 SLA contracts and others

A model capable of expressing Service-Level Agreement requirements on contracts is discussed in [BM11]. They use the notion of constraints systems introduced in the concurrent constraints calculus [SRP91]. In this calculus there is a global constraints store that provides only partial information on the value that variables can take. Instead of writing a value, the store is monotonically redefined by adding constraints with the *tell* construct, a constraint is added only if the resulting constraints system is consistent. In the same fashion, the reading of a value is substituted by the construct *ask* that checks whether the given constraint is entailed by the store. The *cc-pi calculus* admits a *retract* construct for removing previously added constraints, which in turns does not satisfy the monotonicity property. This construct is necessary for the allocation and deallocation of resources. Compared to the cc-calculus, names are handled in a different way. They are ordinary names in the pi-calculus style, and are introduced by means of permutation algebras. The key concept is the *support* of a value, which specifies the set of names such that the permutations which do not affect them do not modify the value. Thus the free names of a constraint c are characterized as the support of c. Finally, two processes can reach a SLA agreement by synchronizing on a public channel, in case all their constraints are satisfied.

Concerning *Component-Based Software Engineering*, several formalisms have been introduced for describing and composing components, which resemble behavioural con-

tract formalisms. Generally these formalisms do not cope with security issues arising in Service Oriented Computing, as for example the possibility of having *liable* participants.

I/O Automata [LT89], *Interface Automata* [dAH01] and *Constraint Automata* [BSAR06] are examples of Component-Based Software Engineering formalisms. I/O automata are input enabled, in that they must be ready to receive any possible input from the environment. Interface Automata do not require to be input-enabled, and feature non-linear behaviours (i.e. broadcasting offers to every possible request), and a notion of *compatibility* [dAH01] between interfaces requiring that all the offers are matched. Constraint Automata [BSAR06] do not require to be input-enabled, and do not distinguish between input and output actions. Their transitions are data-dependent and are activated by logic constraints on those data. They recognise infinite traces and reject those which do not satisfy the imposed constraints. Constraint automata are equipped with an equivalence and a refinement relation. Moreover, they are proposed as an operational model for Reo [Arb04], a coordination language for compositional construction of component connectors based on a calculus of channels.

1.3 Communicating machines

Communicating machines [BZ83] are a simple automata-based model introduced to specify and analyse systems made of agents interacting via asynchronous message passing. In [DY13] a correspondence with choreographies has been proved. In Chapter 5 (adapting the terminology of [LGMZ08]) communicating machines will be used as *automata-oriented* choreographies, as in [LTY15]. These automata interact through FIFO buffers, hence a principal can receive an input only if it was previously enqueued.

We slighty adapt the original definitions and notation from [BZ83] and [CF05]. In particular, the only relevant difference with the original model is that we add the set of final states. Let $\mathcal{P}$ be a finite set of *participants* (ranged over by $\mathtt{p}$, $\mathtt{q}$, $\mathtt{r}$, $\mathtt{s}$, etc.), $C = \{\mathtt{pq} \mid \mathtt{p},\mathtt{q} \in \mathcal{P} \text{ and } \mathtt{p} \neq \mathtt{q}\}$ be the set of *channels*, and *requests* of participants will be built out of $\mathbb{R} = \{a, b, c, \ldots\}$ while their *offers* will be built out of $\mathbb{O} = \{\overline{a}, \overline{b}, \overline{c}, \ldots\}$; with $\mathbb{R} \cap \mathbb{O} = \emptyset$ (offers and requests here should not be confused with the set Comm of Section 1.2.1).

The set of *channel actions* is $\mathsf{Act} = C \times (\mathbb{R} \cup \mathbb{O})$ and it is ranged over by ℓ; we abbreviate $(\mathtt{sr}, \overline{a})$ with $\overline{a}@\mathtt{sr}$ when $\overline{a} \in \mathbb{O}$, representing the action of *sending* a from machine $\mathtt{s}$ to $\mathtt{r}$. Similarly, we shorten $(\mathtt{sr}, a)$ with $a@\mathtt{sr}$ when $a \in \mathbb{R}$, representing the *reception* of a by $\mathtt{r}$.

Definition 5 (CFSM)**.** *Given a finite set of* states Q*, a* communicating finite state machine *is an automaton* $M = (Q, q_0, \mathsf{Act}, \delta, F)$ *where*

- $q_0 \in Q$ *is the* initial *state,*
- $\delta \subseteq Q \times \mathsf{Act} \times Q$ *is the set of* transitions*,*
- $F \subseteq Q$ *is the set of final, accepting states.*

We say that M is deterministic *if and only if for all states $q \in Q$ and all channel actions $\ell \in \mathsf{Act}$, if $(q, \ell, q'), (q, \ell, q'') \in \delta$ then $q' = q''$.*
Finally, we write $\mathscr{L}(M) \subseteq \mathsf{Act}^*$ for the language on Act accepted by the automaton, i.e. the machine M.

The notion of deterministic communicating finite state machines adopted here differs from the standard one (e.g., the one in [CF05]) which requires that, for any state q, if $(q, \overline{a}@\mathtt{sr}, q') \in \delta$ and $(q, \overline{b}@\mathtt{sr}, q'') \in \delta$ then $a = b$ and $q' = q''$. Indeed, hereafter, we will only consider deterministic communicating finite state machines.

The communication model of communicating finite state machines (cf. Definitions 6 and 7) is based on (unbounded) FIFO buffers, which actually are the elements in C. They are to be intended as the channels that the participants use to exchange messages. To spare another syntactic category and cumbersome definitions, we draw the messages appearing in the buffers of communicating finite state machines from the set of requests $\mathbb{R}$. Recall that the set of participants $\mathcal{P}$ is finite.

Definition 6 (Communicating systems)**.** *Given a communicating finite state machine $M_{\mathtt{p}} = (Q_{\mathtt{p}}, q_{0\mathtt{p}}, \mathsf{Act}, \delta_{\mathtt{p}}, F_{\mathtt{p}})$ for each $\mathtt{p} \in \mathcal{P}$, the tuple $S = (M_{\mathtt{p}})_{\mathtt{p} \in \mathcal{P}}$ is a* communicating system, *belonging to the set* CS*.*

A configuration *of S is a pair $s = (\vec{q}; \vec{u})$ where $\vec{q} = (q_{\mathtt{p}})_{\mathtt{p} \in \mathcal{P}}$ with $q_{\mathtt{p}} \in Q_{\mathtt{p}}$ and where $\vec{u} = (u_{\mathtt{pq}})_{\mathtt{pq} \in C}$ with $u_{\mathtt{pq}} \in \mathbb{R}^*$; the component $\vec{q}$ is the* control state *and $q_{\mathtt{p}} \in Q_{\mathtt{p}}$ is the* local state *of machine $M_{\mathtt{p}}$, while $\vec{u}$ represents the contents of the (FIFO buffers of the) channels.*

The initial configuration *of S is $s_0 = (\vec{q_0}; \vec{\varepsilon})$ with $\vec{q_0} = (q_{0\mathtt{p}})_{\mathtt{p} \in \mathcal{P}}$ and $\vec{\varepsilon}$ is the vector of empty channels.*

Hereafter, we fix a machine $M_{\mathtt{p}} = (Q_{\mathtt{p}}, q_{0\mathtt{p}}, \mathsf{Act}, \delta_{\mathtt{p}}, F_{\mathtt{p}})$ for each participant $\mathtt{p} \in \mathcal{P}$ and we let $S = (M_{\mathtt{p}})_{\mathtt{p} \in \mathcal{P}}$ be the corresponding system. The definition below formalises a computation step of a communicating system: if a machine $M_{\mathtt{s}}$ sends a message a to a machine $M_{\mathtt{r}}$ then a is inserted on (the FIFO buffer of) the channel $\mathtt{sr}$ connecting the two, rendered by $\mathtt{sr} \cdot a$ (condition 1 below). If the channel is in the form $a \cdot \mathtt{sr}$ being a the top of the channel (condition 2 below), then the element is read. In both cases, no other machine is affected in the step. In the following let $\vec{q}_{(i)}$ be the projection of the vector $\vec{q}$ on the i-th element.

Definition 7 (Reachable state)**.** *A configuration $s' = (\vec{q'}; \vec{u'})$ is* reachable *from another configuration $s = (\vec{q}; \vec{u})$ by* firing *$\ell \in \mathsf{Act}$, written $s \xrightarrow{\ell} s'$, if there exists $a \in \mathbb{R}$ such that:*

1. *if $\ell = \overline{a}@\mathtt{sr}$ then $(\vec{q}_{(\mathtt{s})}, \ell, \vec{q'}_{(\mathtt{s})}) \in \delta_{\mathtt{s}}$ and for all $\mathtt{p} \neq \mathtt{s}$, $\vec{q'}_{(\mathtt{p})} = \vec{q}_{(\mathtt{p})} \wedge \vec{u'}_{(\mathtt{sr})} = \vec{u}_{(\mathtt{sr})} \cdot a$;*
2. *if $\ell = a@\mathtt{sr}$ then $(\vec{q}_{(\mathtt{r})}, \ell, \vec{q'}_{(\mathtt{r})}) \in \delta_{\mathtt{r}}$ and for all $\mathtt{p} \neq \mathtt{r}$, $\vec{q'}_{(\mathtt{p})} = \vec{q}_{(\mathtt{p})} \wedge \vec{u}_{(\mathtt{sr})} = a \cdot \vec{u'}_{(\mathtt{sr})}$;*
3. *and, in both* (1) *and* (2) *above, for all $\mathtt{pq} \neq \mathtt{sr}$, $\vec{u'}_{(\mathtt{pq})} = \vec{u}_{(\mathtt{pq})}$.*

As usual, the computation *$s_1 \xrightarrow{\ell_1 \cdots \ell_m} s_{m+1}$ shortens $s_1 \xrightarrow{\ell_1} s_2 \cdots s_m \xrightarrow{\ell_m} s_{m+1}$ (for some $s_2, \ldots, s_m$). The set of* reachable configurations *of S is $RS(S) = \{s \mid s_0 \to^* s\}$.*

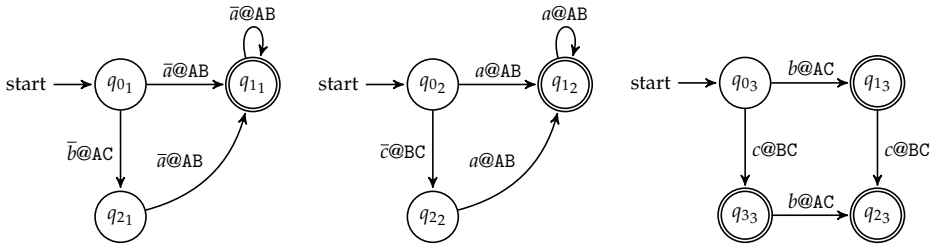


Figure 1.3: A Communicating System

Example 3. *Figure 1.3 shows a graphical representation of a communicating system made of three communicating machines, obtained from the contract automata in Figure 5.1 (see Chapters 3 and 5).*

Similarly to finite state automata, communicating machines have one initial state and a set of final states. A transition has a channel as label, e.g. $\overline{c}$@BC*, indicating that the offer* $\overline{c}$ *is sent by participant* B *to the buffer of the participant* C*; similarly for a receive action. Note that channels of communicating machines specify for each channel action the sender and the receiver.*

Compliance in the asynchronous case is not decidable for communicating finite state machines [BZ83], but become such by using FIFO queues and bags [CHS14]. Moreover in [LTY15] compliance between communicating finite state machines is guaranteed whenever it is possible to synthesise a global choreography from them.

1.4 Logics for Contracts

In this section we assume that the reader is familiar with classic propositional logic (see [nE77] for a comprehensive reading).

Recently, the problem of expressing contracts and of controlling that the principals in a composition fulfil their duties has been studied in *Intuitionistic logic*, where a clause is interpreted as a principal in a contract, in turn rendered as the conjunction of several clauses.

Intuitionistic logic differs from the classic view of logic in that it does require a proof for assigning a truth value to a proposition. In particular, the axioms of the excluded middle and double negation elimination are not present. This turns out to be crucial in modelling circular obligations between contracts, as discussed below.

Actually, the literature only considers fragments of Horn logics because they have an immediate interpretation in terms of contracts. More in detail, these Horn fragments avoid contradiction clauses, as well as formulae with a single Horn clause. These two cases are not relevant because their interpretation as contracts makes little sense, e.g. a composition requires at least two parties.

The first logic we consider is Propositional Contract Logic (PCL) [BZ09a], which is able to deal with circular obligations. Its distinguishing feature is a new implication,

called *contractual implication*, that permits to assume as true the conclusions even before the premises have been proved, provided that they will be in the future.

We then introduce the Intuitionistic Linear Logic with Mix (ILL^{mix})[Ben95]. In this logic one can represent the depletion of resources, and the possibility of recording debits.

In Chapter 4 we will relate a model of contracts with both Horn fragments of these logics.

1.4.1 Propositional Contract Logic

Contracts are modelled as formulae in Intuitionistic logic extended with a contractual form of implication in [BZ09a]. As motivating example the authors propose a toy exchange scenario. Two kids, Alice and Bob, want to share their toys, and their respective contracts are $B = a \to b$ and $A = b \to a$ where a is the atomic proposition "Alice lends her air plane toy" and b is "Bob lends his bike". However, the formula $A \wedge B \to a \wedge b$ does not hold. This problem has been solved by Propositional Contract Logic (PCL).

Definition 8 (PCL)**.** *Assume a denumerable set of atomic formulae $Atoms = \{a, b, c, \ldots\}$; then the formulae of PCL are inductively defined by the following grammar.*

p	::=	$\bot$	*false*
		$\top$	*true*
		a	*prime*
		$\neg p$	*negation*
		$p \vee p$	*disjunction*
		$p \wedge p$	*conjunction*
		$p \to p$	*implication*
		$p \twoheadrightarrow p$	*contractual implication*

PCL extends Intuitionistic logic with a contractual implication ($\twoheadrightarrow$) where $((a \twoheadrightarrow b) \wedge (b \twoheadrightarrow a)) \to a \wedge b$ holds. The starting point is Intuitionistic logic and not the classical one because with the axiom of the excluded-middle the contractual implication coincides with the right projection: $(p \twoheadrightarrow q) \leftrightarrow q$ becomes a theorem. Contractual implication is stronger than standard implication: $\vdash (p \twoheadrightarrow q) \to (p \to q)$, and shares with standard implication a number of properties. A contract that promises true is always satisfied regardless of the precondition $\vdash p \twoheadrightarrow T$. Differently from standard implication, in PCL $\vdash \bot \twoheadrightarrow p$ must not hold ($\bot$ is the false precondition), otherwise the contradiction $(\bot \twoheadrightarrow \bot) \to \bot$ would be deducible. The logic enjoys transitivity properties for implications, moreover the conclusions in a contract can be arbitrarily weakened while the preconditions can be arbitrarily strengthened.

For example, let Customer be the contract $bookFlight \twoheadrightarrow pay$ and AirLine be the contract $pay \twoheadrightarrow bookFlight \wedge freeDrink$. We can weaken the Airline contract $\vdash Airline \to (pay \twoheadrightarrow bookFlight)$ or strengthen the Customer contract $\vdash Customer \to ((bookFlight \wedge freeDrink) \twoheadrightarrow pay)$. The contract still gives rise to an agreement. The properties $\vdash q \to (p \twoheadrightarrow q)$ and $\nvdash (p \twoheadrightarrow q) \to q$ also hold. In

the former, if the conclusion is true then it is also for the inner contract which implies q. The second property states that a contract does not imply its conclusion. An Hilbert-style axiomatization for PCL is presented, which comprises all the axioms of Intuitionistic logic, modus ponens rule and three axioms $T \twoheadrightarrow T$, $(p \twoheadrightarrow p) \rightarrow p$ and $(p' \rightarrow p) \rightarrow (p \twoheadrightarrow q) \rightarrow (q \rightarrow q') \rightarrow (p' \twoheadrightarrow q')$. The last rule combines the rule for the weakening of the conclusions and the strengthening of the premises. The set of axioms are sound and complete with respect to all the properties stated above. It has been proved that PCL is consistent and negation free formulae do not lead to inconsistencies. The provability of formulae specified in PCL is decidable, indeed a Gentzen-style sequent calculus for PCL equivalent to the Hilbert-style axiomatization that enjoys cut elimination has been provided. The rules for its sequent calculus are in Figure 1.4. The contractual implication rules are *Zero*, Fix and *Prepost* while the others are the standards for Intuitionistic logic. Rules $(weakR)$ and (cut) are proved to be redundant in [BZ09a].

Moreover, a binding mechanism between contracts and principals who issue the contracts is proposed in [BZ09a]. This allows to single out the principal who is responsible for a violation. In Chapter 3 this problem will be tackled for several properties of agreement between contracts. PCL is extended with the formula *a says p*, representing the fact that the principal *a* has issued the contract p. PCL axioms are extended accounting with the new formula and it has been proved that the extended logic preserves all the main results, in particular it is decidable. The toy example becomes:

p_{toy} =*Alice says ((Bob says b)* $\twoheadrightarrow$ *a)*$\wedge$*Bob says ((Alice says a)*$\twoheadrightarrow$ *b)*.

Such contract implies the expected duties: $\vdash p_{toy} \rightarrow$ *Alice says a* $\wedge$ *Bob says b*. In case of a violation, a third party can infer who is responsible of the violation. For example, suppose Alice has not respected her contract, than a judge could infer the formula $p_{toy} \wedge \neg a \rightarrow$ *(Alice says a)* $\wedge \neg a \rightarrow$ *Alice says* $\bot$.

The Horn fragment of PCL (H-PCL) [BCP13, BCPZ15] that has a neat interpretation in terms of contracts is defined below, under the assumption that a principal cannot offer and require the same action.

Definition 9 (H-PCL)**.** *The* H-PCL formulae $p, p', \ldots$ *and the clauses* $\alpha, \alpha_i, \ldots$ *are generated by the following BNF grammar*

$$p ::= \bigwedge_{i \in I} \alpha_i \qquad \alpha ::= \bigwedge_{j \in J} a_j \mid (\bigwedge_{j \in J} a_j) \rightarrow b \mid (\bigwedge_{j \in J} a_j) \twoheadrightarrow b$$
$$\text{where } |I| \geq 2, |J| \geq 1, i \neq j \text{ implies } a_i \neq a_j, \text{ and } \forall j \in J.\, a_j \neq b$$

where I and J are finite set of indexes. Also, let $\lambda(p)$ *be the conjunction of all atoms in p.*

A standard contract calculus similar to the π-calculus is proposed in [BZ10a], with primitives for managing constraints (the PCL formulae), which resemble those of [BM11].

The scope delimitation $(a)P$ is used for identifying a session; while the primitive $fuse_x c$ implements a contract-based multi-party agreement. It checks the entailment of the constraint c, and binds the variable x to an actual session identifier, shared among the parties involved in the contract. For example, continuing the toy-exchange scenario, Alice is modelled as $(x)(tell\ b(x) \twoheadrightarrow a(x).fuse_x a(x).lendAirplane)$. The intended use of $fuse_x c$ is to initiate a new session, by accepting a contract which implies c. To do that,

x is instantiated to a fresh session ID and some variables are instantiated to actual ones, representing the binding of the unknown principals to the actual ones. The $ask_x c$ construct stops a process until a formula c can be deduced from the context. This construct models a participant joining an already initiated session, since no fresh identifier is generated. A fusion is driven by the local minimal fusion policy, which only requires to instantiate those variables actually involved in the entailment of c. Thanks to locality, it is not necessary to explore the whole system, but it suffices to inspect any set of known contracts to decide if a set of contracts leads to an agreement.

A simplified version of [BZ10a] has been proposed in [BZ10b]. Principals names are not specified, and there is a simplified version of the local minimal fusion policy, providing the calculus with a $join_x$ construct for joining an already initiated session. The expressiveness of the proposed approach has been shown by encoding some concurrent idioms into their calculus. They express semaphore, memory cells and synchronous π-calculus in their model. For example, concerning π-calculus the output action $\overline{a}(b).P$ becomes $(x)(msg(x,b)|fuse_x in(a,x).P)$, and input action $a(b).Q$ becomes $(y)(in(a,y)|(z)join_z msg(y,z).Q)$. In particular, synchronous communications are expressed through the $fuse$ construct: in an output action, the process sends the message b, codified in the predicate $msg(x,b)$, and waits with $fuse_x in(a,x)$ where a is the identifier of the channel. The input action provides the channel identifier with the predicate $in(a,y)$ and reads the message by fusing the variable z with the message itself. At runtime x and y will be fused to the same session identifier.

1.4.2 Intuitionistic Linear Logic with Mix

Linear logic [Gir87] has been proposed as a refinement of classic and Intuitionistic logic; where logical deduction becomes a way of manipulating resources that cannot always be duplicated or contracted at will. This logic has been shown to be effective in modelling contracts where actual resources are consumed and produced.

Intuitionistic linear logic with mix [Ben95] is used for modelling exchange of resources between partners with the possibility of recording debits (requests satisfied by a principal offer but not yet paid back by honouring one of its requests), and has been recently given a model in terms of Petri Nets [BDGZ15]. As an example, consider the following contracts expressed in ILL^{mix}:

$$Alice \;=\; ia, ia \multimap (ca \otimes cb^{\perp}) \qquad\qquad Bob \;=\; ib, ib \multimap (cb \otimes ca^{\perp})$$

The above formulas model the scenario where Alice wants a birthday cake (cb), but she only has the ingredients to make an apple cake (ia); Bob wants an apple cake (ca), but he only has the ingredients to make a birthday cake (ib). They make a deal: each one will cook for the other, and then they will exchange cakes (and eat them). Intuitively, a positive atom ca represents a *credit*, while a negative atom $ca^{\perp}$ represents a *debit*. The composition (via tensor product) of the three contracts is successful, in that all resources are exchanged and all debits honoured. Indeed with ILL^{mix}, the entailment

$$\frac{}{\Gamma, p \vdash p}\, id \qquad \frac{\Gamma, p \wedge q, p \vdash r}{\Gamma, p \wedge q \vdash r} \wedge L1 \qquad \frac{\Gamma, p \wedge q, q \vdash r}{\Gamma, p \wedge q \vdash r} \wedge L2 \qquad \frac{\Gamma \vdash p \quad \Gamma \vdash q}{\Gamma \vdash p \wedge q} \wedge R$$

$$\frac{\Gamma, p \vee q, p \vdash r \quad \Gamma, p \vee q, q \vdash r}{\Gamma, p \vee q \vdash r} \vee L \qquad \frac{\Gamma \vdash p}{\Gamma \vdash p \vee q} \vee R1 \qquad \frac{\Gamma \vdash q}{\Gamma \vdash p \vee q} \vee R2$$

$$\frac{\Gamma \vdash p \quad \Gamma, p \vdash q}{\Gamma \vdash q}\, cut \qquad \frac{\Gamma, p \to q \vdash p \quad \Gamma, p \to q, q \vdash r}{\Gamma, p \to q \vdash r} \to L \qquad \frac{\Gamma, p \vdash q}{\Gamma \vdash p \to q} \to R$$

$$\frac{\Gamma, \neg p \vdash p}{\Gamma, \neg p \vdash r} \neg L \qquad \frac{\Gamma, p \vdash \bot}{\Gamma \vdash \neg p} \neg R \qquad \frac{}{\Gamma, \bot \vdash p} \bot L$$

$$\frac{}{\Gamma \vdash \top} \top R \qquad \frac{\Gamma \vdash \bot}{\Gamma \vdash p}\, weakR \qquad \frac{\Gamma \vdash q}{\Gamma \vdash p \twoheadrightarrow q}\, Zero$$

$$\frac{\Gamma, p \twoheadrightarrow q, r \vdash p \quad \Gamma, p \twoheadrightarrow q, q \vdash r}{\Gamma, p \twoheadrightarrow q \vdash r}\, Fix \qquad \frac{\Gamma, p \twoheadrightarrow q, a \vdash p \quad \Gamma, p \twoheadrightarrow q, q \vdash b}{\Gamma, p \twoheadrightarrow q \vdash a \twoheadrightarrow b}\, PrePost$$

Figure 1.4: The rules of the sequent calculus for PCL

Alice, *Bob* $\vdash 1$ is proved, which stands for an agreement between Alice and Bob where all the produced resources are consumed.

Deadlock situations in which the two parties are waiting each other for making the first step are avoided by allowing Alice to give an apple cake to Bob, provided that contextually Bob is charged with a debit to give her a birthday cake. This is due to the axiom $a \otimes a^{\perp} \vdash 1$, called the *annihilation principle*, which allows a credit and a debit of the same resource to be cancelled out. The rule *Mix* (see Figure 1.5) of the sequent calculus allows for cancelling debits, without freely generating credits and debits, as done by similar logics such as cancellative linear logic [RRW91].

We recall the full grammar of ILL^{mix}:

Definition 10 (ILL^{mix})**.** *The formulas $A, B, \ldots$ of ILL^{mix} are defined as follows:*

$$A ::= a \mid A^{\perp} \mid A \otimes A \mid A \multimap A \mid A \& A \mid A \oplus A \mid !A \mid 1 \mid 0 \mid \top \mid \bot$$

The full sequent calculus for ILL^{mix} is displayed in Figure 1.5, where A, B stand for a Horn formula p or clause α, while γ may also be empty (note that in rule $(NegL)$, $A = a$ and so $A^{\perp} = a^{\perp}$); Γ and Γ' stand for multi-sets containing Horn formulae or clauses; and Γ, Γ' is the multi-set union of Γ and Γ', assuming $\Gamma, \emptyset = \Gamma$. In Chapter 4 we will only consider proofs without the rule *Cut*, which is redundant by [Ben95].

We now recall the basics of ILL^{mix}. Let $\mathbf{A}, \mathbf{A}^{\perp}$ be respectively the set of *positive* and *negative atoms*, ranged over by $a, b, c, \ldots \in \mathbf{A}$ and by $a^{\perp}, b^{\perp}, c^{\perp}, \cdots \in \mathbf{A}^{\perp}$. Let $\mathbf{L} = \mathbf{A} \cup \mathbf{A}^{\perp}$

$$\frac{}{A \vdash A}\ Ax \qquad \frac{\Gamma \vdash \quad \Gamma' \vdash \gamma}{\Gamma, \Gamma' \vdash \gamma}\ Mix \qquad \frac{\Gamma \vdash A}{\Gamma, A^{\perp} \vdash}\ NegL \qquad \frac{\Gamma, A, B \vdash \gamma}{\Gamma, A \otimes B \vdash \gamma}\ \otimes L$$

$$\frac{\Gamma \vdash A \quad \Gamma' \vdash B}{\Gamma, \Gamma' \vdash A \otimes B}\ \otimes R \qquad \frac{\Gamma \vdash A \quad \Gamma', B \vdash \gamma}{\Gamma, \Gamma', A \multimap B \vdash \gamma}\ \multimap L \qquad \frac{\Gamma, A \vdash B}{\Gamma \vdash A \multimap B}\ \multimap R$$

$$\frac{\Gamma \vdash A \quad \Gamma', A \vdash \gamma}{\Gamma, \Gamma' \vdash \gamma} Cut \qquad \frac{\Gamma, A \vdash}{\Gamma \vdash A^{\perp}} NegR \qquad \frac{\Gamma \vdash}{\Gamma \vdash \perp} \perp R \qquad \frac{}{\perp \vdash} \perp L$$

$$\frac{}{\vdash 1} 1R \qquad \frac{\Gamma \vdash \gamma}{\Gamma, 1 \vdash \gamma} 1L \qquad \frac{}{\Gamma \vdash \top} \top \qquad \frac{}{\Gamma, 0 \vdash A} 0L$$

$$\frac{\Gamma, A \vdash \gamma \quad \Gamma, B \vdash \gamma}{\Gamma, A \oplus B \vdash \gamma} \oplus L \qquad \frac{\Gamma \vdash A}{\Gamma \vdash A \oplus B} \oplus R1 \qquad \frac{\Gamma \vdash B}{\Gamma \vdash A \oplus B} \oplus R2$$

$$\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \& B} \& R \qquad \frac{\Gamma, A \vdash \gamma}{\Gamma, A \& B \vdash \gamma} \& L1 \qquad \frac{\Gamma, B \vdash \gamma}{\Gamma, A \& B \vdash \gamma} \& L2$$

$$\frac{\Gamma, A \vdash \gamma}{\Gamma, !A \vdash \gamma} !L \qquad \frac{!\Gamma \vdash A}{!\Gamma \vdash !A} !R \qquad \frac{\Gamma \vdash \gamma}{\Gamma, !A \vdash \gamma} weakL \qquad \frac{\Gamma, !A, !A \vdash \gamma}{\Gamma, !A \vdash \gamma} coL$$

Figure 1.5: The sequent calculus for ILL^{mix}

be the set of *literals*, and assume $Y \subseteq \mathbf{A}, X \subseteq \mathbf{L}$, where X contains no atom a and its negation $a^{\perp}$. A *positive* tensor product is a tensor product of positive atoms.

As said, we will only consider a fragment of Horn ILL^{mix} called H-ILL^{mix}, defined below. It only has tensor product and *Horn implication*: $\bigotimes_{b \in Y} b \multimap \bigotimes_{a \in X} a$. Note that the premises of the Horn implication are always positive tensor products, and the conclusion is a tensor product of literals, possibly negative.

Definition 11 (H-ILL^{mix})**.** *The Horn formulae $p, p_i, \ldots$ and the clauses $\alpha, \alpha_i, \ldots$ of* H-ILL^{mix} *are defined by*

$$p ::= \bigotimes_{i \in I} \alpha_i \qquad \alpha ::= \bigotimes_{a \in X} a \mid \bigotimes_{b \in Y} b \multimap \bigotimes_{a \in X} a$$

where $|I| \geq 2$; $|X|, |Y| \geq 1$; $\{a, a^{\perp}\} \not\subseteq X$; and $b \in Y$ implies $b \notin X$.

1.4.3 Logic for Contracts in the literature

Different models of PCL have been proposed in the literature, as event structures, Petri nets, process algebra [BCZ13, BCP13, BTZ12].

In [BCPZ16] [BCZ13] event structures endowed with certain notions from game theory are used to represent contracts. The property of agreement is studied, which ensures safe interactions among participants. A principal is culpable if it has not yet fired an

enabled event, and is otherwise innocent. In particular a principal agrees to a contract if it has a positive pay-off in case all the principals are innocent, or if someone else is found culpable. Additionally the authors study protection: a protected principal has a non-losing strategy in every possible context. It is shown that for a particular type of contracts, with Offer Request pay-off, it is not possible to obtain both agreement and protection for all principals. Indeed a participant is protected if every offer is performed only after the corresponding request is obtained. Hence in circular contracts every participant is stuck waiting for the other parties to perform the corresponding offers. A new enabling action, called *circular enabling action*, is introduced for solving this problem. It is showed how protection and agreement can coexist by using the circular enabling relation. Finally two encodings from session types to event structures are proposed. Two session types are compliant if the client can successfully interact with the server. It is proved how compliance between binary session types corresponds to agreement via an eager strategy.

Processes and contracts are two separate entities in [BTZ12]. In this formalism contracts can be represented as formulae or as process algebras. A process can fulfil its duty by obeying its contract or it behaves dishonestly and becomes *culpable* — and become honest again by performing later on the prescribed actions. A generic calculus for Contract-Oriented COmputing (CO_2) is proposed, where it is possible that contracts can not be fulfilled after an agreement has been reached. Contracts are used to drive computations after sessions have been established for detecting violations. These contracts have the form A *says* c, where A is the principal that advertises the contract c. A set of observables Φ, which are properties of contracts, and an entailment relation $\vdash$ between contracts and observables are given. There is a contract fulfilment relation $c\textcircled{\smile}A$ between contracts and principals, and contracts are represented as CCS-like processes.

The execution of a contract dictates obligations to principals. The processes have input activities, output activities and autonomous activities. Moreover they have branching, parallel composition and recursive definition. Φ is the set of LTL formulae. $c \vdash \phi$ holds when $c \models_{LTL} \phi$. Also $c\textcircled{\smile}A \longleftrightarrow \forall c', c''.c \equiv (A \ says \ c')|c'' \rightarrow c' \equiv 0$, that is A has fulfilled all his duties. They also provide contract-as-formulae where contracts are modelled as PCL formulae [BZ09a]. In this framework we have $\Phi = C$ and $C\textcircled{\smile}A \longleftrightarrow \forall a.C \vdash (A \ says \ a) \rightarrow C \vdash (A \ says \ !a)$, that is each obligation for A entailed by C has been fulfilled.

An encoding of H-PCL formulae without nesting contractual implications into contracts-as-processes is provided. Contractual implications are encoded into parallel composition, in order to fulfil circular dependencies asynchronously. CO_2 is proposed as a generalization of the calculus presented in [BZ10a], where an abstract contract model based on cc-calculus [SRP91] is considered. It differs from cc-calculus for not assuming a global constraint store, instead it uses sessions. Moreover, constraints are multi-sets of contracts. The construct $fuse$ initiates a new session according to a local minimal fusion policy, likewise of PCL. The construct *tell* advertises a new contract. The construct $ask\phi$ blocks the process until the formula ϕ is entailed. The construct *do* makes a multi-set of

contracts evolve. A system is composed by a framing $A[P]$ for each agent that contains his process and the contracts in which it is involved, and by a frame $s[C]$ for the session that contains all the contracts instantiated by a $fuse$. With $fuse$ the contracts involved are shifted from the agent's frame to the actual frame of the session.

The system evolves when a session contains the contract A *says* a and meanwhile in the frame of the agent, it performs the action *do* a. In a typical usage, the parties first publish their contracts with *tell*, then one of them opens a new session with $fuse$ while the others are waiting with *ask* to discover their duties in the session, and finally perform their duties with *do*. A principal is honest if it performs all the duties provided by its contract. If in the frame of the agent there are no actions, while in the session it still has duties to performs, then it is dishonest. An agent can protect himself from frauds by providing a contract with a contractual implication which implies to perform his duties only if it is guaranteed that the parties involved fulfil their duties as well.

1.5 Model Checking

Model checking [BK08] is a technique for automatically verifying correctness properties, which is exhaustive for finite-state systems. It consists in proving that a model M, i.e. a suitable abstraction of the system under analysis, satisfies a particular property ϕ, written $M \models \phi$. An example of a property ϕ could be the absence of deadlock states, i.e. the system never gets stuck. The model is generally described by a Kripke structure [Kri63] and the property of interest ϕ by a modal temporal logic. In this logic each formula has a truth value in each possible state of the system. States are temporally ordered: if a state q' is reachable from a state q then q temporally precedes q'. Modal operators allow to express properties that must hold in every possible future state or in one future state. Since the system is finite, the procedure is decidable: an exhaustive search on the states space suffices to find states that eventually violate the property ϕ, if any.

In this thesis we will use automata-based model checking techniques [VW86, VW08]. In particular, the problem of deciding whether a formula satisfies a given model can be reduced to the problem of checking the emptiness of the intersection of two languages, since the set of models of a temporal logic formula can be interpreted as a language (see [VW86, VW08] for details). In Chapter 2 we will extend the model checking technique of [BDFZ11, BDF09]. Briefly, the model is described by an abstraction H whose denotational semantic is a context-free language; its words represent all the possible execution traces. The property ϕ is rendered as a regular language representing the offending traces. In this case, the problem of deciding whether $M \models \phi$ reduces to the problem of checking the emptiness of the intersection of the language of H and the complement of the language of ϕ (i.e. the language of $\neg\phi$). This problem is known to be decidable since regular languages are closed under complements, context-free languages are closed under intersection with regular languages, and emptiness of context free languages is decidable [HMU06], yielding an effective model-checking procedure.

Properties under which a model is verified are traditionally:

- *invariant properties*: this type of properties are memory-less, it suffices to check if the property holds in each state separately;
- *safety properties*: these properties are history-dependent; in order to check if a state q satisfies a safety property ϕ all the states that are traversed for reaching q are needed (i.e. the prefix). In particular, ϕ is characterized by its set of *bad prefixes*, that are all those finite traces that lead to a violation of the property;
- *liveness properties*: these properties cannot be violated by any finite prefix of an execution. An example of liveness properties are fairness properties, which are used for ensuring that if a state q is visited infinitely often, then all possible transitions from q must be traversed.

1.6 Language-based Security

Security has been seen commonly as a property that relevant programs must fulfil. It is a composite of three attributes [ALRL04]:

- *confidentiality* absence of unauthorized disclosure of information;
- *availability* readiness for correct service;
- *integrity* absence of improper alterations.

Mechanisms for enforcing security have been developed at operating system level, and consist mainly of firewalls, monitors, cryptography, etc... With the advent of mobile and ubiquitous computing, security has become a relevant concern and thus the need for introducing security mechanisms from the early phases of the development process.

Language-based security [Koz99, SMH01] has been proposed for introducing security mechanisms at programming language-level. The introduction of languages constructs capable of expressing security issues allow the development of static analysis techniques to verify and validate security properties. In Chapter 2 we will rely on the approaches proposed by Bartoletti et al. [BDF09, BDFZ11].

In [BDFZ11, BDF09] services are modelled as expressions of a typed λ-calculus extended to handle services orchestration, where types are an abstraction of the behaviours of services. A set of primitive *access events* are introduced to abstract from activities with possible security concerns. Security policies are regular properties of execution histories (i.e. sequences of access events). A service request is modelled by a particular construct that uniquely identifies the request and the type of the requested service, including the safety properties that explain how the caller protects itself from the invoked service. Safety properties are modelled as regular nominal automata recognising data words, called *Usage Automata* [BDFZ15], which are a sort of parametric finite state automata. Services are assumed to be published in a global trusted repository.

Plans represent the assignment of services to requests, hence describing each possible orchestration. There is a type and effect system for the calculus; types are standard, while effects, called *history expressions*, are an approximation of all the possible behaviours of services. History expressions are a sort of context-free grammar, and include the empty history ε, access events α, sequencing $H \cdot H'$, non-deterministic choice $H + H'$, safety framings $\varphi[H]$, recursion $\mu h.H$ (μ binds the occurrences of the variable h in H), and planned selection which abstracts a branch of different history expressions generated by different plans.

Definition 12 (History Expression with Planned Selection)**.** *A* History Expression H *is a term generated by the following grammar:*

H, H'	$::=$	ε	*empty*
		h	*variable*
		α	*access event*
		$H \cdot H'$	*sequence*
		$H + H'$	*choice*
		$\varphi[H]$	*safety framing*
		$\mu h.H$	*recursion*
		$\{\pi_1 \triangleright H_1 \cdots \pi_k \triangleright H_k\}$	*planned selection*

The denotational semantics of H is the set of histories $\mathcal{H}$ that represents all the possible computations of a service expression.

Static analysis techniques are built to determine plans that drive service executions enjoying safety properties. A history expression H of a service is *valid* when it respects all its security framings, and eventually those of the caller. The effect of a service composition is obtained by suitably assembling the effects of the component services, and of those services they may invoke in a nested fashion.

The validity of a service composition depends thus on the global orchestration that selects a service for each request. A main result of [BDFZ11, BDF09] shows how to construct a plan that associates requests with offers so to guarantee that no executions will violate the security requirements. Indeed, if a service composition is valid, then the corresponding plan will safely drive the execution without resorting to any run-time monitor, and guaranteeing all the safety properties required.

Validity of history expressions is ascertained by model checking Basic Process Algebras [BW90] with finite state automata; a history expression H is naturally rendered as a BPA process, while a finite state automaton models the validity of H. In Chapter 2 these techniques have been applied to an automata based representation of the contracts of [CGP09], recovering the same notion of progress.

1.7 Control Theory

Ramadge and Wonham's *Theory of Supervisory Control for Discrete Event Systems* [CL06] is concerned with the synthesis of a *controller* which drives the execution of the system

while enforcing some security properties. The theory was introduced in [RW87, Thi96].

A discrete event system G is a finite state automaton, called the plant, where *accepting* states represent successful terminations of a task, e.g. the completion of manufacturing processes comprised of series of tasks; while *forbidden* states should never be traversed in "good" computations. In forbidden states control problems, the purpose of supervisory control theory is to synthesise a controller to enforce that forbidden states cannot actually be reached while marked states are always reachable. To do so, this theory distinguishes between *controllable* events, that may be disabled by the controller even though they are enabled by the plant; and *uncontrollable* events, i.e. those always enabled.

In general, the supervisor cannot observe all events. Hence events are partioned into *observable* and *unobservable*. It is assumed that all controllable events are observable.

A solution to the Supervisory Control for Discrete Event Systems may be given by a finite deterministic transition system called a *supervisory controller* $\mathcal{K}$, such that the following conditions are satisfied in the finite transition system $\mathcal{K}/G$ obtained by composing the plan with the controller, called the *controlled system*:

- all states reachable in the controlled system are not forbidden;
- all uncontrollable events enabled in a state of the plant are also enabled in the corresponding state of the controlled system, if reachable;
- from every reachable state of the controlled system, there exists a path leading to some final state of the plant.

If all events are observable then a most permissive controller exists that never blocks a good computation [CL06]. The most permissive controller can be computed iteratively, starting from $\mathcal{K}_0 = G$ and using a set Bad_0 containing at first step all the forbidden states. At step n in the iteration, one computes $\mathcal{K}_n$ and Bad_n as modified versions of $\mathcal{K}_{n-1}$ and Bad_{n-1} updated as follows:

- if there is a controllable transition t in K_n with target state in Bad_{n-1} and source state q not in Bad_{n-1}, then q is added to the updated set Bad_n and the transition t is removed from K_n;
- all the states from which it is not possible to reach a marked state are removed in the updated controller K_n.

When the iteration stops, i.e., when $\mathcal{K}_n = \mathcal{K}_{n-1}$, one finally removes from $\mathcal{K}_n$ all states in Bad_n. If the initial state is removed, then the control problem has no solution. Otherwise, $\mathcal{K}_n$ yields the most permissive controller $\mathcal{K}$.

In [DDM10, DDM08] the opacity control problem is discussed. Given a finite transition system and a regular predicate, one is required to compute a controller enforcing the opacity of a predicate against an attacker that partially observes the system supposedly trying to break the behaviour. In the general case where the property to enforce has a μ-calculus definable behaviour [Koz83], a general synthesis procedure is proposed

in [PR05], which always computes a maximal permissive controller of deterministic systems under partial observation when it exists. The approach is based on labelling processes, and characterizes the maximally permissive controllers as models accepted by some tree automaton, so that their existence (and synthesis when possible) comes down to the non-emptiness problem of the tree automaton.

In Chapter 3, an orchestration of services in agreement will be generated with a simplifed construction of the most permissive controller for the agreement property under analysis.

1.8 Operations Research, Flow problem

Operations research [HL86] is a discipline that deals with calculating optimal or near-optimal solutions to complex decision-making problems, by using techniques such as mathematical optimization. In Chapter 3 we will borrow techniques from Operation research, namely optimization of *network flow problem*, in order to decide if a composition of service contracts admits a particular type of agreement.

We fix some useful notation. A flow network [FF10] (also known as a transportation network) is a directed graph where each edge has a capacity and each edge receives a flow. Let $G = (V, E)$ be a graph with set of nodes V and edges E, that are pair of nodes. Generally there are two types of special nodes: *source nodes*, that are generating flow, and *sink nodes*, that are consuming the flow. We assume the presence of a single source q_s node and one sink node q_f, and that all nodes are reachable from the source node. In general, if there are more sink nodes, it is possible to obtain a new graph G' with only one sink node by simply adding artificial, dummy edges from all the original sink nodes to the new single sink node. Given a node $q \in V$, the *forward star* $FS(q)$ is the set of out-coming edges of q, while the *backward star* $BS(q)$ is the set of incoming edges in the node q.

For each edge $e \in E$, the *flow variable* x_e represents the flow that is passing through the edge e. Generally, a maximum capacity u_e is assigned to each edge e, representing the maximum amount of flow of e, and a cost c_e representing the cost of utilising the edge e.

A network flow problem is a type of network optimization problem where the objective function requires to optimize a flow such that the solution respects the following constraints:

- the amount of flow on an edge cannot exceed the capacity of the edge (*capacity constraints*), written $\forall e \in E.x_e \leq u_e$;

- the amount of flow into a node equals the amount of flow out of it, unless it is a source, which has only outgoing flow d, or sink, which has only incoming flow d (*flow conservation*),written:

$$\forall q \in V. \sum_{e \in BS(q)} x_e - \sum_{e \in FS(q)} x_e = \begin{cases} -d & \text{if } q = q_s \\ 0 & \text{if } q \neq q_s, q_f \\ d & \text{if } q = q_f \end{cases}$$

- depending on the studied problem, it can be required that the computed flow must be an integer value (*integrity constraints*), written: $\forall e \in E. x_e \in \mathbb{N}$.

Examples of network flow problems are the *Maximum flow problem* [FF57] or the *Minimum-cost flow problem* [Kle67]. The first problem consists in maximizing the amount of flow that can be send from the source nodes to the sink nodes. The objective function is then `max` d.

In the second problem a cost is associated to each edge of the network, and the objective function is minimised in order to find the optimal cost for sending a given amount of flow from the source nodes to the sink nodes, that is `min` $\sum_{e \in E} x_e c_e$.

These problems are solved by using *Integer Linear programming* [HKLW10, Wal89]. Indeed, all constraints are represented by linear inequalities, and the objective function is linear. Several solvers are available for solving linear optimization problems automatically and efficiently, by using for example the simplex algorithm [FGK89]. Linear programming techniques have been used for verifying system properties in [CA95, EM00]. In this papers a set of linear constraints, called *the state equation*, provide an approximation of the behaviour of the system (e.g. modelled as a Petri Net [EM00]). A main benefit of this approach is that properties (expressed as linear constraints) are checked without generating the whole state space of the model. However the solutions may not be exact, false positives may be generated. In Chapter 6 we will rely on integer linear programming problems solvers for mechanising our verification techniques.

1.9 Concluding Remarks

In this chapter we have introduced Service Oriented Computing, its coordination mechanisms, namely orchestration and choreography, and service contracts that are useful for guaranteeing security properties of a composition of services. We have described different formalisms for service contracts, and communicating machines, a formalism that has been used for representing choreographies of services. We have also introduced two logics for expressing contract properties, and techniques that will be exploited in the next chapters for checking properties of a composition of services.

Chapter 2

Contract Compliance as a Safety Property

In this chapter we outline a formal theory of contracts that supports the verification of *service compliance* and of *security policies* enforcing access control over resources.

Indeed, communications between services occur along specific channels, and it is equally important to guarantee that the interactions between a client and a server never get blocked (i.e. services are compliant), and that communications and data exchanged among services are secure.

Services are compliant when their interactive behaviour eventually progresses, i.e. all the service invocations are guaranteed to be eventually served. Services are secure when data are exchanged and accessed according to specific rules, called *policies*.

Our starting point is the language-based methodology supporting static analysis of security policies developed in [BDF09], and described in Section 1.6.

We extend this approach to check security and service compliance at the same time. The first contribution is to extend *history expressions* with suitable communication facilities to model the interactive behaviour of services, including possibly nested service sessions in a multiparty fashion. In particular, we extend history expressions to include communications along channels, and internal/external choice for combining the notions of security of resource accesses and progress of interactions.

The second contribution is sharpening the verification phase. Intuitively, we prove that service compliance is a *safety property*: when it holds, all the involved parties are able to successfully complete their interactions without getting stuck. Reducing services compliance to a safety property makes it efficiently to model-check. Finally, we extract from a history expression all the *viable* plans for serving a request, i.e. those orchestrations that successfully drive secure and compliant executions. Adopting a valid plan guarantees that the involved services never go wrong at run-time, in that they are capable of successfully accomplishing their tasks (progress property) without rising any security exception. Therefore, no run-time monitor is needed, to control the execution of the network of services.

The main insight is to reduce the problem of checking compliance to the problem of

checking a safety property over a suitable finite state automaton, obtained by tailoring the notion of product automaton to contracts. Indeed, we will apply an automata-based model checking technique for verifying compliance between services.

As a further advantage, this transformation allows us to apply all the techniques and tools developed for checking safety properties to efficiently verify the absence of communication errors between services, as well as security policies.

The idea of checking the correctness of a composition of services through a suitable notion of product automaton will be further developed in the next chapters. We will introduce a compositional automata-based formalism for specifying service contracts, used for verifying several properties of agreement adopting different coordination mechanisms.

Structure of the chapter. The chapter is organised as follows. The next section intuitively presents our formal machineries, the problems we address, and our goals through an illustrative example. The definition of compliance, its reduction to a safety property and the technicalities needed to model-check it are in Section 2.3. In Section 2.4, we summarise our results and future work.

2.1 An Example

To illustrate our approach and help intuition, we consider a simple cloud-based scenario. We take into account *federated cloud services* [BYV+09] where a broker is responsible for collecting the clients' requests and for sending the information to the multiple cloud providers. The broker is also granted the rights to negotiate the service contracts with the providers on behalf of the clients, and to distribute and coordinate requests of clients across the multiple cloud providers. Finally, formalising and guaranteeing security of federated cloud services is a major challenge [BJMU11].

When a client submits its service requests to the broker, it also specifies the constraints on the required quality of service. In our approach, this negotiation is managed by issuing the policy $\varphi_{(c,p,t)}$, where its parameters are the configuration profile c, the pricing p and the workload threshold t. Roughly, policies are regular properties over the execution progress of service requests (*execution histories*).

The policy φ displayed in Figure 1 will be used in our running example to determine and minimise the possible bad behaviours, that we will discuss later on.

Since in the cloud federation each provider comprises multiple computing services, the information stored in the configuration profile is exploited to identify the computing facilities according to the need of the clients. The configuration profile can also contain information about the required security constraints to model the fact that cloud architectures have well-defined security policies and enforcement mechanisms in place. E.g., the profile could include the requirement to provide a level of isolation over the virtualization infrastructure.

The pricing information is used to decide how service requests are charged. E.g., pricing can be based on the submission time, it can be fixed or depend on the availability

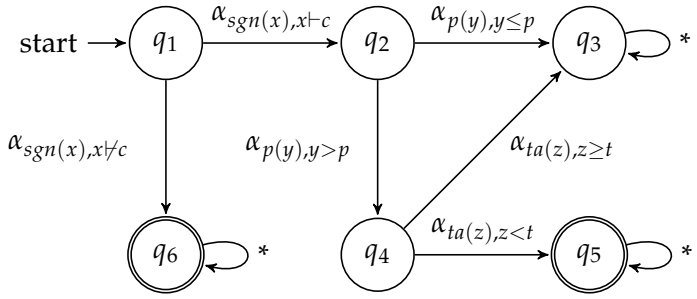


Figure 2.1: The automaton for the policy $\varphi_{(c,p,t)}$

$$C_1 = \texttt{open}_{1,\varphi_{(\{c1\},45,100)}}\overline{Req}(Ack.\overline{Pay} + NoAv)\texttt{close}_{1,\varphi_{(\{c1\},45,100)}}$$
$$C_2 = \texttt{open}_{2,\varphi_{(\{c2\},40,70)}}\overline{Req}(Ack.\overline{Pay} + NoAv)\texttt{close}_{2,\varphi_{(\{c2\},40,70)}}$$
$$B = Req.\texttt{open}_{3,\emptyset}\overline{IdC}.(AgOk + UnA)\texttt{close}_{3,\emptyset}(\overline{Ack}.Pay \oplus \overline{NoAv})$$
$$S_1 = \alpha_{sgn(1)}.\alpha_{p(45)}.\alpha_{t(80)}.IdC(\overline{AgOk} \oplus \overline{UnA})$$
$$S_2 = \alpha_{sgn(2)}.\alpha_{p(70)}.\alpha_{t(100)}.IdC(\overline{AgOk} \oplus \overline{UnA} \oplus \overline{Del})$$
$$S_3 = \alpha_{sgn(3)}.\alpha_{p(90)}.\alpha_{t(100)}.IdC(\overline{AgOk} \oplus \overline{UnA})$$
$$S_4 = \alpha_{sgn(4)}.\alpha_{p(50)}.\alpha_{t(90)}.IdC(\overline{AgOk} \oplus \overline{UnA})$$
$$S1 \notin c1 \qquad S1, S3 \notin c2$$

Figure 2.2: Two clients, a broker and four cloud providers

of certain resources.

The last parameter t specifies the minimum amount of work expected to be done. E.g., in a data center this value could be the amount of data processing performed in a given time. The parameters of the service contract are crucial to identify and supply the actual demand of computing resources on the cloud.

When a cloud provider accepts the service requests, it signs the contracts, i.e. it supplies the information x about the profile of the available resources issuing the *event* $\alpha_{sgn(x)}$. The cloud provider also publishes its pricing and workload information by issuing the events $\alpha_{p(y)}, \alpha_{t(z)}$. If the profile of the available resources does not match the given client profile (formally expressed by the guard $x \nvdash c$) the policy is violated and the final state q_6 is reached. Note that the forbidden traces belong then to the language *accepted* by the automaton, as prescribed by the so called "default-accept" approach. A violation of the policy also occurs if the pricing of the cloud provider does not fulfil client's requirements and the provided workload is lower then t. In this case the policy is violated and the final offending state q_5 is reached. Finally, note that the policy allows to have a pricing different from the given one whenever a higher workload performance is ensured.

We abstractly and formally specify the scenario discussed above through a process

calculus, namely an extension of History Expressions defined in Section 3.1. We consider a setting (see Figure 2.2) with two clients C_1, C_2, a cloud broker B and a cloud federation including computational service providers S_1, S_2, S_3, S_4. An important difference between the computational model considered in our approach and the standard process calculi is that policies are first class citizens of the calculus. The two clients only differ in the way they instantiate their policies.

A client opens a session and sends his request to the broker, who must respect the policy φ. Sending the request is modelled by the action $\overline{Req}$, while receiving the request is done through Req, the complementary action (for brevity and simplicity we omit the data information). The client is then willing to receive the confirmation of the policy agreement and to settle the payment ($Ack.\overline{Pay}$). The client is also ready to receive a negative message in the case where no resources are available ($NoAv$). When either message is received, the session with the broker is closed.

As discussed above, the broker receives the service request Req and then opens a session with the service providers. Here for simplicity we only model the interactions with the providers and not the actual deployment of the service. The broker sends the client Id and all the related data by issuing the event $\overline{IdC}$, and then waits for either the agreement or for the negative messages with $(AgOk + UnA)$ (note that now different channels are used). Then the session is closed, and the response message is forwarded to the client.

The cloud providers perform the events of signing and publishing the pricing and the workload; and they then interact with the broker. Here we assume a fixed pricing strategy and a basic description for the workload. Note that all services, except for S_2, have the *internal* choice $\overline{AgOk} \oplus \overline{UnA}$. This abstractly represents that the cloud providers can decide on their own which message to send depending on their state of affairs. Being purely non-deterministic, in our computational model the internal choice behaves differently than the external choice, e.g. $AgOk + UnA$, that is instead driven by the message received.

Since the broker B is ready to receive each sent message, we say that the mentioned providers are *compliant* with B. Instead, the provider S_2 is *not compliant* with B. Indeed, the broker can also send the message $\overline{Del}$ (meaning that there will be available services later) but the broker cannot handle it, and therefore the interaction gets stuck.

As far as security is concerned, assuming that S_1 does not match the configuration profile $c1$, then it turns out that the providers S_1 and S_4 violate the policy settled by C_1, since S_1 has not the required profile and S_4 does not respect the required workload and price. Finally, note also that the services S_1 and S_3 do not satisfy the configuration parameters of C_2.

Figure 2.3 displays a fragment of a computation. It is a sequence of configurations χ and of transitions $\chi \xrightarrow{\gamma} \chi'$, where γ records either an event relevant to security or progress, or a communication made of two complementary actions (disregard for a while the indexes $\vec{\pi}, R$ of the arrows).

A configuration is made of tuples $\eta, \ell : S$, put in parallel (through $\|$), where η is a

sequence of events, and ℓ is the location of the service/client S. In our example, the starting configuration has the two clients, one at location ℓ_{c1}, the other at ℓ_{c2}. Both performed no actions, so their execution history is empty (ε).

The first step opens a session between C_1 and B and registers in the history that the whole session, in particular B, is subject to the policy φ, duly instantiated (call it $\varphi_1 = \varphi_{(\{c1\},45,100)}$).

The second step shows that the request of the client C_1 has been accepted by the broker, via a communication.

Now a nested session is opened involving the broker and S_3, in the third step, and no policy is imposed over the called service S_3.

Concurrently, C_2 can perform its service request, as expressed by the fourth step, that registers that the policy $\varphi_2 = \varphi_{(\{c_2\},40,70)}$ is active. Note that we assume that the broker can replicate its code at will.

The two parallel sessions can evolve concurrently. For simplicity, we proceed with service S_3, that signs, shows the pricing and its workload threshold (all displayed in the same line).

The broker is ready to send the data of the client to S_3, and to receive back an answer, say "no services are available" (S_3 is now ε, because it has no further activities to do).

The session is then closed in step 10; the broker resumes its conversation with the client C_1, and forwards the non-availability message in step 11.

The next steps close the session numbered 1 and the security framing implementing and enforcing the policy φ_1. The last transition continues the session involving the second client. The index R of the arrows shows that the transitions depend on the service providers constituting the cloud federation.

The index $\vec{\pi}$ is a vector of functions that indicates how the requests are bound to services, i.e. a *plan*. The plan π_1 for the first client maps the request 1, originated by $\texttt{open}_1$ of the client, to ℓ_{br}, and the request 3 from $\texttt{open}_3$ of the broker, to ℓ_{s3}. We call π_1 *valid*, because it drives a computation where both the security constraints and compliance of clients/services are guaranteed.

Suppose now that the plan π_2 for the second client maps request 2 to ℓ_{br} and request 3 (from the second instance of the broker) to ℓ_{s2}. Since S_2 does not comply with B, at run-time a communication involving the action $\overline{Del}$ cannot be performed because the broker has no action Del. Our assumption that the service can decide what to send on its own is violated. For this reason, we say that this plan is *not valid*. Finally, consider a plan that maps request 3 to ℓ_{s3}, that this time is compliant with the broker. However S_3 does not satisfy the configuration profile settled by C_2, and so a policy violation occurs; also this plan is not valid.

2.2 Programming Model

Here we define the syntax and the semantics of services and of networks of services. The abstract behaviour of services is represented in the form of *history expressions*.

$$\varepsilon, \ell_{c1} : C_1 \| \varepsilon, \ell_{c2} : C_2 \xrightarrow{open_{1,\varphi_1}}_{\vec{\pi},R} \quad (1)$$

$$(\!|\varphi_1, [\ell_{c1} : \overline{Req}.(Ack.\ldots)\mathtt{close}_{1,\varphi_1}, \ell_{br} : Br] \| \varepsilon, \ell_{c2} : C_2 \xrightarrow{\tau}_{\vec{\pi},R} \quad (2)$$

$$(\!|\varphi_1, [\ell_{c1} : (Ack.\ldots)\mathtt{close}_{1,\varphi_1}, \ell_{br} : \mathtt{open}_{3,\varnothing}\overline{IdC}\ldots] \| \varepsilon, \ell_{c2} : C_2 \xrightarrow{open_{3,\varnothing}}_{\vec{\pi},R} \quad (3)$$

$$\overbrace{(\!|\varphi_1, [\ell_{c1} \ldots, [\ell_{br} : \overline{IdC} \ldots, \ell_{s3} : \alpha_{sgn(3)} \ldots]]}^{P} \| \varepsilon, \ell_{c2} : C_2 \xrightarrow{open_{2,\varphi_2}}_{\vec{\pi},R} \quad (4)$$

$$P \| \overbrace{(\!|\varphi_2, [\ell_{c2} : \overline{Req} \ldots \mathtt{close}_{2,\varphi_2}, \ell_{br} : Br]}^{Q} \xrightarrow{\alpha_{sgn(3)}}_{\pi,R} \xrightarrow{\alpha_{p(90)}}_{\pi,R} \xrightarrow{\alpha_{ta(100)}}_{\vec{\pi},R} \quad (5\text{-}7)$$

$$\overbrace{(\!|\varphi_1 \alpha_{sgn(3)} \alpha_{p(90)} \alpha_{ta(100)}}^{\eta}, [\ell_{c1} : \ldots, [\ell_{br} : \overline{IdC} \ldots, \ell_{s3} : Idc \ldots]] \| Q \xrightarrow{\tau}_{\pi,R} \xrightarrow{\tau}_{\vec{\pi},R} \quad (8\text{-}9)$$

$$\eta, [\ell_{c1} : \ldots, [\ell_{br} : \mathtt{close}_{3,\varnothing} \ldots, \ell_{s3} : \varepsilon]] \| Q \xrightarrow{close_{3,\varnothing}}_{\vec{\pi},R} \quad (10)$$

$$\eta, [\ell_{c1} : (Ack.\overline{Pay} + NoAv)\mathtt{close}_{1,\varphi_1}, \ell_{br} : (\overline{Ack}.Pay \oplus \overline{NoAv})] \| Q \xrightarrow{\tau}_{\vec{\pi},R} \quad (11)$$

$$\eta, [\ell_{c1} : \mathtt{close}_{1,\varphi_1}, \ell_{br} : \varepsilon] \| Q \xrightarrow{close_{1,\varphi_1}}_{\vec{\pi},R} \quad (12)$$

$$\eta |\!)_{\varphi_1}, \ell_{c1} : \varepsilon \| (\!|\varphi_2, [\ell_{c2} : \overline{Req}. \ldots, \ell_{br} : B] \xrightarrow{\tau}_{\vec{\pi},R} \quad \ldots (13)$$

Figure 2.3: A fragment of a computation

We further extend the basic model of history expressions (see Definition 12) with standard I/O operations of contracts of Definition 1, because we want to explicitly represent through communications also the interactions between clients and services. Moreover, we explicitly deal with sessions, and our history expressions will therefore record also the operations of opening and closing them. Our final extension permits to have several sessions in parallel.

Compared to [BDF09], we address neither the analogous extensions to the λ-calculus, nor the definition of a type and effect system for it.

Some auxiliary notions are in order. We assume to have a set of security relevant operations described by events $\alpha \in \mathsf{Ev}$, and a set of policies $\varphi \in \mathsf{Pol}$, i.e. a regular language over Ev. Opening and closing a session is modelled through communication actions, labelled by a request identifier $r \in \mathsf{Req}$ and a policy φ. These special activities will be logged in computations by framing actions $\mathsf{Frm} = \{(\!|\varphi, |\!)_\varphi | \varphi \in \mathsf{Pol}\}$. We also assume the presence of channels along which clients and services communicate. Hereafter, the set of communication actions of Section 1.2.1 is extended as $\mathsf{Comm} = \{a, \overline{a}, \ldots, \tau, \mathtt{open}_{r,\varphi}, \mathtt{close}_{r,\varphi}\}$, and let $\lambda \in \mathsf{Comm} \cup \mathsf{Ev} \cup \mathsf{Frm}$. Finally, we assume services and clients be hosted in locations $\ell \in \mathsf{Loc}$.

Definition 13 (History Expression)**.** *A history expression is a term generated by the following grammar:*

$$H ::= \varepsilon \mid h \mid \mu h.H \mid (\sum_{i \in I} a_i.H_i) \mid (\bigoplus_{i \in I} \overline{a}_i.H_i) \mid \alpha \mid H \cdot H \mid \mathtt{open}_{r,\varphi}\, H\, \mathtt{close}_{r,\varphi}$$

$$\bigoplus_{i\in I} \overline{a}_i.H_i \overset{\overline{a}_i}{\twoheadrightarrow} H_i \quad \text{(I-Choice)} \qquad \sum_{i\in I} a_i.H_i \overset{a_i}{\twoheadrightarrow} H_i \quad \text{(E-Choice)}$$

$$\alpha \overset{\alpha}{\twoheadrightarrow} \varepsilon \quad \text{(Acc)} \qquad \texttt{open}_{r,\varphi}\, H\, \texttt{close}_{r,\varphi} \overset{open_{r,\varphi}}{\twoheadrightarrow} H\, \texttt{close}_{r,\varphi} \quad \text{(S-Open)}$$

$$\frac{H \overset{\lambda}{\twoheadrightarrow} H'}{H \cdot H'' \overset{\lambda}{\twoheadrightarrow} H' \cdot H''} \quad \text{(Conc)} \qquad \frac{H\{^{\mu h.H}/_h\} \overset{\lambda}{\twoheadrightarrow} H'}{\mu h.H \overset{\lambda}{\twoheadrightarrow} H'} \quad \text{(Rec)}$$

Figure 2.4: Operational Semantics of History Expressions

where I is a set of indexes such that if $i, j \in I$ then $i \neq j$.

Intuitively, ε is the history expression that cannot do anything, and thus we stipulate $\varepsilon \cdot H \equiv H \equiv H \cdot \varepsilon$.

Infinite behaviour is denoted by $\mu h.H$, restricted to be tail-recursive and contractive, i.e. guarded by communication actions $\bar{a}$ or a. Events α can occur, if they do not violate any active policy. The expression $H_1 \cdot H_2$ is the sequential composition.

An expression can send/receive on a channel messages. This model of contracts have some restrictions, that is internal choices on outputs and external choices on inputs (see Section 1.2.4), while in the next chapter these restrictions will be relaxed. Intuitively, each internal choice needs to be communicated through an output action to the other communicating party, and an input is necessary to evaluate an external choice. In this way a tailored automata-based model checking technique can be applied. To stress that the non-deterministic choice of the output $\overline{a}_i$ is up to the sender only (internal choice), we use $\oplus$, while the external choice only involves inputs a_i and is denoted by Σ. Note that these requirements are fulfilled by the prefixing for summations in Definition 13.

A service is engaged in a session with another through $\texttt{open}_{r,\varphi} H \texttt{close}_{r,\varphi}$, where r is a unique identifier and φ is the policy to be enforced while the responding service is active.

As anticipated, a policy is a parametric finite state automaton (see [BDFZ15]) that accepts those strings of access events that violate it, in the default-accept paradigm. An example is "never write (α_{write}) after read (α_{read})", and a trace that violates it is $\alpha_{read}\alpha_{write}$. When entering a security framing, all the histories, i.e. the sequence of events previously fired, must respect the policy: ours is a *history-dependent* approach. We remark that policies are safety properties: something bad will never happen.

Later on we will require that a client must be able to synchronize with the server and correctly terminate the session, i.e. client and service have to be compliant.

The operational semantics of stand-alone history expressions is defined inductively by rules in Figure 2.4.

We now turn our attention to our specification of networks of services N. In the following definition, we also introduce the notion of plan π.

Definition 14 (Network and Plan)**.** *A network N, session S and plan π are terms defined by the following grammars:*

$$N ::= N \| N \mid S \qquad S ::= \ell : H \mid [S, S]$$
$$\vec{\pi} = [\pi_1, \ldots, \pi_n], \text{ where } \pi_i, \pi_i' ::= \emptyset \mid r[\ell] \mid \pi \cup \pi'$$

A network N is composed of the parallel composition of different clients H, each hosted at a location $\ell \in \mathsf{Loc}$, and of sessions S involving a client (or a service) and a service. Services are published in a global trusted repository $R = \{\ell_j : H_j \mid j \in J\}$, and they are always available for joining sessions (while clients are not).

We assume that the operator $\|$ is associative, but not commutative, so a network can be written as a vector $\vec{N}$. Instead, we stipulate that $[S, S'] \equiv [S', S]$.

We can have nested sessions, modelling that a service involved in a session can open a new session with another service. In this case the previous session will be restored upon termination of the new one.

The semantic of networks is the transition system inductively defined by the rules in Figure 2.5. Network configurations have the form $\|_{i \in I} \eta_i, S_i$ abbreviated by $\overrightarrow{\eta, N}$, where η_i is the *history* of S_i. As a matter of fact, access events α and policy framings $\langle\!|_\varphi, |\!\rangle_\varphi$ are logged into the history η_i. A session can evolve only if its history respects all the active policies in η_i, denoted by $\models \eta_i$.

We briefly comment on the rules of the operational semantic of networks. The first rule is for opening a session: the service at ℓ_i fires an event $\mathtt{open}_{r,\varphi}$ (in the stand-alone semantics); the plan π_i selects the service at ℓ_j; and the client and the server get involved in a new session. However, this only occurs if the history η, updated with $\langle\!|_\varphi$ recording the policy imposed by the client, satisfies all the policies that are active.

Symmetrically, the rule *Close* ends a session. The client continues computing on its own, while the server H_j'' is terminated. The history of the client is updated with the closing frame of the policy φ imposed over the session.

Rule *Session* governs the independent evolution of an element within a session.

Similarly, rule *Net* updates the network according to the evolution of one of its components.

Rule *Access* fires an access event γ, appends it to the current history η; and checks $\eta\gamma$ for validity.

The premises of rule *Synch* require a service to send/receive a message $\bar{a}/a$, and its partner to receive/send it, written as the co-action $a/\bar{a}$. The resulting communication is labelled with the (non observable) action τ. Note that a communication can only take place if both services are inside the same session.

As usual, a computation starts from the initial configuration $N_0 = \|_{j \in J} \varepsilon, H_j$, and it is a sequence $N_0 \xrightarrow{\lambda}_{\vec{\pi},R} \|_{i \in I} \eta_i, N_i \xrightarrow{\lambda'}_{\vec{\pi},R} \|_{i \in I} \eta_i', N_i' \ldots$ We now give some remarks on the two ways its participants may get stuck. The first is when all the access events a service H may perform violate the security policies that are active. In this case, a resource monitor, formalised by the validity relation $\models \eta$, aborts the execution of H. Note however

$$\frac{H \overset{open_{r,\varphi}}{\twoheadrightarrow} H' \quad r[\ell_j] \in \pi \quad \{\ell_j : H_j\} \subseteq R \quad \models \eta\!\Downarrow_\varphi}{\eta, \ell_i : H \xrightarrow{open_{r,\varphi}}_{\pi,R} \eta\!\Downarrow_\varphi, [\ell_i : H', \ell_j : H_j]} \ (\textsc{Open})$$

$$\frac{H \overset{close_{r,\varphi}}{\twoheadrightarrow} H'}{\eta, [\ell_i : H, \ell_j : H''_j] \xrightarrow{close_{r,\varphi}}_{\pi,R} \eta\,\Uparrow_\varphi, \ell_i : H'} \ (\textsc{Close})$$

$$\frac{\eta, S \xrightarrow{\lambda}_{\pi,R} \eta', S' \quad \models \eta'}{\eta, [S, S''] \xrightarrow{\lambda}_{\pi,R} \eta', [S', S'']} \ (\textsc{Session})$$

$$\frac{\eta_i, N_i \xrightarrow{\lambda}_{\pi_i,R} \eta'_i, N'_i \quad \models \eta'_i \quad (\vec{\pi})_i = \pi_i \quad \overrightarrow{(\eta, N)}_i = \eta_i, N_i}{\overrightarrow{(\eta, N)} \xrightarrow{\lambda}_{\vec{\pi},R} \overrightarrow{(\eta, N)[\eta'_i, N'_i \mapsto \eta_i, N_i]}} \ (\textsc{Net})$$

$$\frac{H \overset{\gamma}{\twoheadrightarrow} H' \quad \models \eta\gamma}{\eta, \ell_i : H \xrightarrow{\gamma}_{\pi,R} \eta\gamma, \ell_i : H'} \ \gamma \in \mathsf{Ev} \ (\textsc{Access})$$

$$\frac{H_i \overset{a}{\twoheadrightarrow} H'_i \quad H_j \overset{co(a)}{\twoheadrightarrow} H'_J}{\eta, [\ell_i : H_i, \ell_j : H_j] \xrightarrow{\tau}_{\pi,R} \eta, [\ell_i : H'_i, \ell_j : H'_j]} \ a \in \mathsf{Comm} \ (\textsc{Synch})$$

Figure 2.5: Operational Semantics of Network

that the computation proceeds if there is an event that H can fire without violating any active policies: our semantics implements the so-called *angelic* non-determinism.

The second way for deadlocking a component of a network is when two services in a session want to communicate, but the output of one of them is not matched by an input of the other, in other words, the two services are *not compliant*. Also here our semantics is angelic, in that it does not respect the requirement saying that the choice among various outputs is done regardless of the environment and of its capability of accepting the sent message.

The main task of this chapter is proposing an automated technique to construct plans that drive executions that do not deadlocks, namely *valid plans*. We present a static analysis, that guarantees that the networks *only* have computations that can always proceed, i.e. that at run-time a component of a network neither violates any security policies, nor does it get stuck because of missing communications.

Both safety policies and services compliance are verified through automata-based model checking techniques. For verifying the security properties, here we adopt a simplified version of the approach of [BDFZ11, BDF09] (see Section 1.6), which is presented in the next section. The compliance between services is addressed through a novel

$$\mathcal{AP}(\varepsilon) = \emptyset \qquad \mathcal{AP}(\alpha\eta) = \mathcal{AP}(\eta)$$
$$\mathcal{AP}(\langle\!|_\varphi \eta) = \mathcal{AP}(\eta) \uplus \{\varphi\} \qquad \mathcal{AP}(|\!\rangle_\varphi \eta) = \mathcal{AP}(\eta) \setminus \{\varphi\}$$

Figure 2.6: the function $\mathcal{AP}$ for computing the active policies in a history η

model checking technique, discussed in Section 2.3.

2.2.1 Statically Checking Validity

We first intuitively define when a history $\eta \in (\mathsf{Ev} \cup \mathsf{Frm})^*$ is valid, written $\models \eta$.

Let $\eta^\flat$ be the history obtained by erasing all the framing events from it. For example, if $\eta_0 = \gamma\alpha\langle\!|_\varphi \beta |\!\rangle_\varphi$ then $\eta_0^\flat = \gamma\alpha\beta$. A history $\eta^\flat$ respects a policy φ, in symbols $\eta^\flat \models \varphi$, if it is not recognized by the automaton φ; and it is valid if it respects *all* the polices that are opened, but not closed, i.e. the policies active in η.

Since our approach to security is history-dependent, we actually require that *all* the prefixes of $\eta^\flat$ respect the relevant policies. For example, consider again the history η_0 above, and let φ require that no α occurs after γ. Then, η_0 is *not* valid according to our intended meaning, because when firing β, the policy φ is activated and the prefix $\gamma\alpha$ does not obey φ — note instead that $\langle\!|_\varphi \gamma\, |\!\rangle_\varphi \alpha\beta$ would be valid, as φ is no longer active after γ is fired.

The definition of validity follows.

Definition 15 (Validity of Histories)**.** *A history η is* valid *(written $\models \eta$) when:*

$$\forall \eta_0\eta_1 \text{ s.t. } \eta_0\eta_1 = \eta, \varphi \in \mathcal{AP}(\eta_0).\eta_0^\flat \models \varphi$$

where the auxiliary function $\mathcal{AP}$ used for computing the multi-set of active policies in η is given in Figure 2.6, , where $\uplus$ is multi-set union.

Now the problem is verifying if all the histories generated by a given network lead to a final configuration, with no security violations. This can be done by separately checking if all its clients H are valid, i.e. that all the histories generated when it is executed are valid. Most likely, H will contain some requests, and serving them will open and eventually close possibly nested sessions with other services $H', H'', \ldots$, made available by the repository R. The idea is to suitably assemble the history expressions $H, H', H'', \ldots$, and recording in a plan for H which service to invoke for each request, as to obtain the pair $\hat{H}, \pi$.

Note that $\hat{H}$ may be *non-valid*, even if the selected services are valid, each in isolation. Indeed, the impact on the execution history of selecting a service H_r for a request r is not confined to the execution of H_r, but it spans over the whole execution, because security is history-dependent. The validity of the composed service $\hat{H}$ depends thus on the global orchestration, i.e. on the plan π.

In order to ascertain the validity of $\hat{H}$, we resort to model checking.

Definition 16 (Validity of History Expressions)**.** *Let H be a history expression of a client, R be a repository of services and π be a plan that associates each request of H to a service in R, then the set of all the prefixes of histories generated by H under π, R is:*

$$[\![H]\!]^{\pi}_{R} = \{\eta \mid \varepsilon, \ell : H \rightarrow_{\pi,R} \ldots \rightarrow_{\pi,R} \eta, S\}$$

Then the client H is valid under π, R, written $[\![H]\!]^{\pi}_{R} \models$, iff $\forall \eta \in [\![H]\!]^{\pi}_{R}, \models \eta$.

Since history expressions are regular, likewise the policies φ, we know that $[\![H]\!]^{\pi}_{R}$ is a regular language and can be rendered as a finite state automaton. Defining the function which computes the finite state automaton generating $[\![H]\!]^{\pi}_{R}$ is out of the scope of this chapter, as we are interested in verifying compliance of history expressions.

A policy φ to be enforced is rendered as an automaton $\varphi_{(\!|}$ with two layers recording the activation of the policy – event $(\!|_{\varphi}$, and the deactivation – event $|\!)_{\varphi}$. Each layer contains a replica of the policy φ. However, only the layer corresponding to an activated policy contains accepting states.

Since all policies are regular, the intersection of the two automaton $[\![H]\!]^{\pi}_{R} \cap \varphi_{(\!|}$ is regular. Checking its emptiness is then decidable, and suffices to verify that the policy φ will be never violated by the client H under the plan π. By repeating this technique for all the policies φ in H it is possible to verify that $[\![H]\!]^{\pi}_{R} \models$.

2.3 Checking Service Compliance

We introduce a technique to construct a plan that provides the assurance that compliance between clients and services is guaranteed at run-time. Again, we can consider a client (or server) at a time, and, for each of its requests, we determine the compliant services.

First we manipulate the syntactic structure of a service in order to identify and pick up all the requests, i.e. the subterms of the form $\texttt{open}_{r,\varphi} H_1 \texttt{close}_{r,\varphi}$. Then, to check compliance of the service request r against an available service $\ell_2 : H_2$, we compute the projection of H_1 and H_2 on their communication actions. This projection removes from H_1 and H_2 all the access events α, as well as all the *inner* service requests, i.e. the subterms of the form $\texttt{open}_{r',\varphi'} \ldots \texttt{close}_{r',\varphi}$ occurring inside H_1 and H_2. The inductive definition of the projection on communication actions is given in Figure 2.7.

Note that if H is a closed history expression then $H^!$ is closed. Moreover the projection function $H^!$ yields a behavioural contract as defined in Definition 1, for this reason we feel free to call *contracts* these kind of history expressions.

More precisely, the projection function produces a subset of those contracts, since in our history expressions the internal choice is always guarded by output action and the external choice is always guarded by input actions. Finally, we only have guarded tail recursion.

Because of the last restriction, it turns out that the transition system of $H^!$ is *finite state*; in other words there only is a finite number of expressions that are reachable from $H^!$ through the transitions defined by the operational semantics of history expressions in

$$(H \cdot H')^! = H^! \cdot H'^! \qquad h^! = h \qquad (\mu h.H)^! = \mu h.(H)^!$$

$$(\sum_{i\in I} a_i.H_i)^! = \sum_{i\in I} a_i.(H_i^!) \qquad (\bigoplus_{i\in I} \overline{a}_i.H_i)^! = \bigoplus_{i\in I} \overline{a}_i.(H_i)^!$$

$$(\mathtt{open}_{r,\varphi}.H.\mathtt{close}_{r,\varphi})^! = \varepsilon^! = \alpha^! = \varepsilon$$

Figure 2.7: Projection on Communication Actions

isolation. We now adapt the notion of *observable ready sets* introduced in Section 1.2.1(see Definition 3) to our history expressions.

Definition 17 (Observable Ready Sets of History Expressions)**.** *Let $H = H_1^!$ be (the projection of) a history expression. The* observable ready set *of H is the finite set $S \in \mathcal{P}(Comm)$ given by the relation $H \Downarrow S$ inductively defined below.*

$$\varepsilon \Downarrow \emptyset \quad h \Downarrow \emptyset \qquad \bigoplus_{i\in I} \overline{a}_i.H_i \Downarrow \{\overline{a}_i\} \qquad \sum_{i\in I} a_i.H_i \Downarrow \bigcup_{i\in I}\{a_i\}$$

$$\frac{H \Downarrow S}{\mu h.H \Downarrow S} \qquad \frac{H \Downarrow S \quad S \neq \emptyset}{H \cdot H' \Downarrow S} \qquad \frac{H \Downarrow \emptyset \quad H' \Downarrow S}{H \cdot H' \Downarrow S}$$

Example 4. *We show some example of ready set:*

- *let $H = \mu h.(\overline{a_1} \oplus \overline{a_2}) \cdot b \cdot h$, then $H \Downarrow \{\overline{a_1}\}$ and $H \Downarrow \{\overline{a_2}\}$;*
- *$\varepsilon \cdot (a + b) \cdot (\overline{d} \oplus \overline{e}) \Downarrow \{a, b\}$.*

We now extend the notion of compliance (see Definition 4) to history expressions. Given the service request $\mathtt{open}_{r,\varphi} H_1 \mathtt{close}_{r,\varphi}$ and the service H_2, we say that the two contracts $H_1^!$ and $H_2^!$ are compliant if for every possible internal action of a party, the other is able to perform the corresponding coaction (recall that $\overline{S} = \{\overline{a} | a \in S\}$).

Definition 18 (Compliance of History Expressions)**.** *Two history expressions H_c and H_s are* compliant, *written $H_c \vdash H_s$, if for all $C, S \in \mathcal{P}(Comm)$, $H_1 = H_c^!$ and $H_2 = H_s^!$ are such that*

(1) $H_1 \Downarrow C$ and $H_2 \Downarrow S$ implies that $C = \emptyset$ or $C \cap \overline{S} \neq \emptyset$, and

(2) $H_1 \overset{a}{\twoheadrightarrow} H_1' \wedge H_2 \overset{co(a)}{\twoheadrightarrow} H_2'$ implies $H_1' \vdash H_2'$.

We introduce a model-checking technique for verifying if two contracts are compliant. We remark that the key idea is to reduce the problem of checking compliance to the problem of checking a safety property over a suitable finite state automaton, obtained by tailoring the notion of product automaton to contracts. Notice that this transformation will allow us to apply all the techniques and tools developed for checking safety properties.

The product automaton $\mathcal{A} = H_1^! \otimes H_2^!$ of two contracts $H_1^!$ and $H_2^!$ models the behaviour of contracts composition. Final states represent stuck configurations: these states are reached whenever the two contracts are not compliant.

Definition 19 (Product of History Expressions)**.** *Let H_1' and H_2' be history expressions. The product automata $H_1 \otimes H_2$ of $(H_1')^! = H_1$ and $(H_2')^! = H_2$ is defined as follows.*
$H_1 \otimes H_2 = \langle Q_1 \times Q_2, \{\tau\}, \delta, \langle H_1, H_2\rangle, F\rangle$ where

- *Q_1 and Q_2 are the states of the transition systems of H_1 and H_2;*
- *$\{\tau\}$ is the alphabet and $\langle H_1, H_2\rangle$ is the initial state;*
- *the set of final states is $F = \{\langle H_1, H_2\rangle | H_1 \neq \varepsilon \wedge \neg(i) \vee \neg(ii)\}$ where :*

  (i) $\exists \overline{a}.(H_1 \overset{\overline{a}}{\twoheadrightarrow} H_1' \vee H_2 \overset{\overline{a}}{\twoheadrightarrow} H_2')$

  (ii) $(\forall H_1 \overset{\overline{a}}{\twoheadrightarrow} H_1', \exists H_2 \overset{a}{\twoheadrightarrow} H_2') \wedge (\forall H_2 \overset{\overline{a}}{\twoheadrightarrow} H_2', \exists H_1 \overset{a}{\twoheadrightarrow} H_1')$

- *the transition function δ is:*

$$\delta = \{(\langle H_1, H_2\rangle, \tau, \langle H_1', H_2'\rangle) | H_1 \overset{a}{\twoheadrightarrow} H_1' \wedge H_2 \overset{co(a)}{\twoheadrightarrow} H_2' \wedge \langle H_1, H_2\rangle \notin F\}$$

The correctness of the definition of the product automaton relies on the fact that the projection function yields finite state contracts since recursive behaviour is obtained only via tail recursion.

Note that condition (i) ensures that both services are not waiting on input actions. Condition (ii), instead, ensures that for all the possible output actions that a service is ready to fire, the other party is ready to perform the corresponding input action. Moreover a final state has no outgoing transition.

Intuitively, H_1 and H_2 are compliant if and only if the language of the product automaton $\mathcal{A}$ is empty.

Lemma 1 (Ready Sets and Product)**.** *Let H_c and H_s be closed history expressions with $H_1 = H_c^!$ and $H_2 = H_s^!$. For all $C, S \in \mathcal{P}(\mathsf{Comm})$ such that $H_1 \Downarrow C$, $H_2 \Downarrow S$ we have that $C \cap \overline{S} \neq \emptyset$ if and only if conditions (i) and (ii) of Definition 19 hold.*

Proof. $(\rightarrow)$ We know that either H_1 or H_2 performs an output action, because $C \cap \overline{S} \neq \emptyset$, so condition (i) holds. W.l.o.g. assume that H_1 performs an output. The proof in the other case is symmetric. By definition of observable ready sets and $C \cap \overline{S} \neq \emptyset$, we have that H_1 must be of the form $H \cdot H'$ with H either $\bigoplus_{i\in I} \overline{a}_i.H_i$ or $\mu h.\bigoplus_{i\in I} \overline{a}_i.H_i$, (recall that $\varepsilon \cdot H \equiv H \equiv H \cdot \varepsilon$), with $I \neq \emptyset$. Also H_2 must be of the form $H_2' \cdot H_2''$ with H_2' either $\sum_{j\in J} a_j.H_j$ or $\mu h.\sum_{j\in J} a_j.H_j$ with $J \neq \emptyset$. For H_1 we have $|I|$ different ready sets forming $IC = \{\{\overline{a_i}\} | H_1 \Downarrow \{\overline{a_i}\}\}$. Instead H_2 has a single ready set of the form $S = \{a_j | j \in J\}$. Now by hypothesis $\forall C \in IC.\ C \cap \overline{S} \neq \emptyset$, that implies $(\forall H_1 \overset{\overline{a}}{\twoheadrightarrow} H_1', \exists H_2 \overset{a}{\twoheadrightarrow} H_2') \wedge (\forall H_2 \overset{\overline{a}}{\twoheadrightarrow} H_2', \exists H_1 \overset{a}{\twoheadrightarrow} H_1')$ because H_2 performs no output actions.
$(\leftarrow)$ By (i) we have that either C or S contains an output action. W.l.o.g. assume that

H_1 performs an output. The proof in the other case is symmetric. By definition of operational semantics of history expressions, H_1 must be of the form $H \cdot H'$ with H either $\bigoplus_{i\in I} \overline{a}_i.H_i$ or $\mu h. \bigoplus_{i\in I} \overline{a}_i.H_i$ with $I \neq \emptyset$. Hence by definition H_1 has $|I|$ different ready sets in $IC = \{\{\overline{a_i}\} | H_1 \Downarrow \{\overline{a_i}\}\}$. By condition (ii), H_2 must be able to execute any corresponding coaction. Therefore by definition of operational semantic of history expressions, H_2 must be of the form $H_2' \cdot H_2''$ with H_2' either $\sum_{j\in J} a_j.H_j$ or $\mu h. \sum_{j\in J} a_j.H_j$ with $|J| \neq \emptyset$. Hence by definition H_2 has a single ready set of the form $S = \{a_j | j \in J\}$. Condition (ii) ensures that $\forall C \in IC$ it must be $\overline{C} \cap S \neq \emptyset$. □

We now state our main theorem that guarantees compliance of two services whenever their languages have an empty intersection.

Theorem 1 (Product Emptiness and Compliance)**.** *Let H_1' and H_2' be closed history expressions and let $(H_1')^! = H_1$ and $(H_2')^! = H_2$. Then $H_1 \vdash H_2$ if and only if $\mathscr{L}(H_1 \otimes H_2) = \emptyset$.*

Proof. $(\rightarrow)$ assume $H_1 \vdash H_2$. We firstly show that $H_1 \vdash H_2 \rightarrow \langle H_1, H_2\rangle \notin F$. We have two possible cases:

- $H_1 \Downarrow \emptyset$: by definition it must be $H_1 = \varepsilon$, since $H_1 = h$ is not closed. Hence by construction the set of final states of the product is empty.
- $H_1 \Downarrow C, H_2 \Downarrow S$ with $C \cap \overline{S} \neq \emptyset$: by Lemma 1 $(i) \wedge (ii)$ holds, and therefore $\langle H_1, H_2\rangle \notin F$.

We prove by induction on the length of the path of the finite state product automaton that no final states are reachable. From this follows that $\mathscr{L}(H_1 \otimes H_2) = \emptyset$. For the base case (the empty path) we have already showed $\langle H_1, H_2\rangle \notin F$.

Suppose now that $\langle H_{1i}, H_{2i}\rangle$ is a reachable non final state through a path of length n. We prove that $\forall(\langle H_{1i}, H_{2i}\rangle, \tau, \langle H_{1i}', H_{2i}'\rangle) \in \delta$ there must be $\langle H_{1i}', H_{2i}'\rangle \notin F$. By hypothesis $H_1 \vdash H_2$ we know that $H_1 \overset{a}{\twoheadrightarrow} H1' \wedge H_2 \overset{co(a)}{\twoheadrightarrow} H_2'$ implies $H_1' \vdash H_2'$, i.e. compliance is preserved under synchronization. An easy inductive reasoning guarantee both $H_{1i} \vdash H_{2i}$ and $H_{1i}' \vdash H_{2i}'$. We already proved that $H_{1i}' \vdash H_{2i}'$ implies $\langle H_{1i}', H_{2i}'\rangle \notin F$. Note that if H is closed and $H \twoheadrightarrow H'$ then also H' is closed.
$(\leftarrow)$ Let I to be the set:

$$I = \{\langle H_1', H_2'\rangle | \langle H_1', H_2'\rangle \notin F \wedge \langle H_1', H_2'\rangle \text{ reachable state of the product automaton } \}$$

We show that I enjoys the properties characterizing the notion of compliance relation and the thesis follows from the fact that relation $\vdash$ is the largest compliance relation. By hypothesis it follows that $\varepsilon \notin \mathscr{L}(H_1 \otimes H_2)$, hence the initial state is not final and belongs to I.

Assume that $\langle H_1', H_2'\rangle \in I$, and therefore is reachable from the initial state. We have to show that $H_1' \Downarrow C$ and $H_2' \Downarrow S$ implies $C = \emptyset$ or $C \cap \overline{S} \neq \emptyset$. Now we have two sub cases:

- $C = \emptyset$, the result follows; or

- $C \neq \emptyset$ we apply Lemma 1.

It remains to prove that $H_1' \stackrel{a}{\twoheadrightarrow} H_1'' \wedge H_2' \stackrel{co(a)}{\twoheadrightarrow} H_2''$ implies $\langle H_1'', H_2'' \rangle \in I$. By construction of the product automaton $(\langle H_1', H_2' \rangle, \tau, \langle H_1'', H_2'' \rangle) \in \delta$, by hypothesis $\langle H_1'', H_2'' \rangle \notin F$ and since $\langle H_1', H_2' \rangle$ is reachable it has to be the case also for $\langle H_1'', H_2'' \rangle$, hence $\langle H_1'', H_2'' \rangle \in I$. □

An important consequence of our model-checking technique is that the property of progress of a session (even for infinite executions) is not a liveness property, but an invariant property (see Section 1.5). An invariant property P_{inv} only inspects a state at time, without looking at all the past history, i.e. $P_{inv} = \{q_0 q_1 q_2 \ldots \mid \forall j \geq 0.\ q_j \models \Phi\}$, where $q_0 q_1 q_2 \ldots$ is the sequence of visited states. Since conditions (i) and (ii) of Definition 19 do not inspect the past states, it turns out that compliance is an invariant property: a subset of the safety properties.

Theorem 2 (Compliance as Invariant Property)**.** *Compliance is an invariant property.*

Proof. Given two services $(H_1')^! = H_1$ and $(H_2')^! = H_2$ by Theorem 1 $H_1 \vdash H_2$ iff $\mathscr{L}(H_1 \otimes H_2) = \emptyset$, i.e. all states $\langle H_1', H_2' \rangle$ reachable from $\langle H_1, H_2 \rangle$ are such that $\langle H_1', H_2' \rangle \models (H_1' = \varepsilon \vee (i) \wedge (ii))$ ((i) and (ii) conditions of Definition 19). □

Since all invariant properties are safety properties, the following corollary holds.

Corollary 1 (Safety Compliance)**.** *Compliance is a safety property.*

This result paves the way to the application of techniques and tools developed for checking safety properties to efficiently verify the compliance of services.

2.4 Concluding Remarks

In this chapter, we provided the means to statically verify whether a network of services will evolve without incurring in security nor compliance violations. Given a repository R and a vector of clients, pick up one of them, say H, at a time; generate a valid plan π_H for H; for each request $\texttt{open}_{r,\varphi} H_1 \texttt{close}_{r,\varphi}$ occurring in the composed service check if $H_1 \vdash H_2$, where $\pi_H(r) = \ell_2$ and $\ell_2 \in R$. If all these steps succeed, we have guaranteed that nothing bad will happen, allowing to safely switch off any run-time monitor.

This result relies on suitable extensions of the methodology proposed in [BDF09] with standard operations of contracts, and on a careful definition of service sessions, possibly nested, and of compliance. Indeed, Theorem 2 shows that compliance of behavioural contracts is a safety property, paving the way to its verification via standard model-checking, for example with existing tools like [BZ08a].

As a matter of fact, the results presented in this chapter establish a novel connection between the world of service contracts compliance and the world of security.

An interesting line of research concerns studying a restricted form of service availability, that now can replicate themselves boundlessly many times. We are confident that more detailed rules for opening and closing sessions can be easily given.

The formalization of contracts compliance in terms of product of finite state automata will be deepened in the next chapters, where a novel formalism called contract automata will be presented. Security concerns will focus on detecting the services responsible of failures.

Chapter 3

Contract Automata

In this chapter an approach to the formal description of service contracts in terms of automata is presented.

We address two main questions. First, we propose a rigorous formal technique for describing and composing contracts. Second, we develop techniques capable of determining when a contract composition is correct and leads to a successful service composition. Thus reaching the contract agreement is the basic ingredient to design successful orchestrations.

More in detail, we introduce an automata-based model for contracts called *contract automata*, that are a special kind of finite state automata, endowed with two composition operations. A contract automaton may represent a single service or a composition of several services. Its language describes the possible executions of the services bound in it. The traces accepted by a contract automaton show the possible interactions among the principals, by recording which offers and requests are performed, and by which principals in the composition. This provides the basis to define criteria that guarantee a composed service to well behave.

These services are oblivious of their communicating partners, and the underlying coordination mechanism of contract automata is *orchestration*. The messages have to be thought of as directed to an orchestrator synthesised out of the principals; the orchestrator directs the interactions in such a way that only good executions actually happen.

We then rephrase in our model three notions presented in the literature that characterise when contracts are honoured. The first one is called *agreement* and considers the case when all the requests made are synchronously matched by offers, and thus satisfied, while the second one called *strong agreement* requires that all the requests and offers are satisfied. The third property, *weak agreement*, is more liberal, in that requests can be asynchronously matched, and an offer can be delivered even before a corresponding request, and vice-versa. We say that a contract automaton is *safe* (*strongly/weakly safe*, respectively) when all the interactions among principals are carried out as required by the (strong/weak) agreement.

The notions of safety presented above may appear too strict since they require that all the words belonging to the language recognised by a contract automaton must sat-

isfy (strong/weak) agreement. We thus introduce a more flexible notion that characterises when a service composition may be successful, i.e. at least one among all the possible service interactions well behaves. We say that a contract automaton *admits* (strong/weak) agreement when such interaction exists.

When a contract automaton admits (strong/weak) agreement, but it is not (strongly/weakly) safe, we define those principals in a contract that are (strongly/weakly) *liable*, i.e. those responsible for leading a contract composition into a failure. This may happen either because a final state cannot be reached or because some requests have not been satisfied.

The main idea behind our notion of agreement is the assumption that possible "good" executions exist together with "bad" computations, that are those breaking the overall agreement. In our case the orchestrator eventually cuts off "bad" computations, while the literature has also notions that require composition of participants to have "good" computations, only.

For checking when a contract automaton enjoys the properties sketched above, we propose two formal verification techniques that have been implemented (see Chapter 6).

The first amounts to build the so-called controllers in Control Theory [CL06]. We show that controllers are powerful enough to synthesise a correct orchestrator enforcing (strong) agreement and to detect the (strongly) liable principals. In order to check weak agreement and detect weak liability we resort to techniques borrowed from Operational Research [HKLW10], namely optimisation of network flows (see Section 1.8). The intuitive idea is that service coordination is rendered as an optimal flow itinerary of offers and requests in a network, automatically constructed from the contract automaton.

A brief introduction to Control Theory and Flow Problems can be found in Chapter 1.

Structure of the chapter.

In Section 3.1 we introduce contract automata and two composition operators. Section 3.2 discusses the properties of agreement and safety. The technique for checking and enforcing them are also presented here, along with the notion of liability. Strong agreement is briefly discussed in Section 3.3. Weak agreement and weak liability are defined in Section 3.4, along with a technique to check them. A case study is proposed in Section 3.5. Finally, concluding remarks are in Section 3.6.

3.1 The Model

This section formally introduces the notion of contract automata, that are finite state automata with a partitioned alphabet. We start with some notation and preliminary definitions.

A contract automaton (cf. Definition 23 below) represents the behaviour of a set of principals (possibly a singleton) capable of performing some *actions*; more precisely, as formalised in Definition 20, the actions of contract automata allow them to "advertise"

offers, "make" requests, or "handshake" on simultaneous offer/request actions. The number of principals in a contract automaton is called *rank*, and a vectorial representation is used for tracking the moves of each principal in the composition.

Let $\mathbb{L} = \mathbb{R} \cup \mathbb{O} \cup \{\square\}$ be the alphabet of *basic actions* such that:

- *requests* of principals will be built out of $\mathbb{R} = \{a, b, c, \ldots\}$ while their *offers* will be built out of $\mathbb{O} = \{\overline{a}, \overline{b}, \overline{c}, \ldots\}$;
- $\mathbb{R} \cap \mathbb{O} = \emptyset$;
- $\square \notin \mathbb{R} \cup \mathbb{O}$ is a distinguished label to represent components that stay idle.

We define an involution $co(\bullet) : \mathbb{L} \mapsto \mathbb{L}$ such that:

- $co(\mathbb{R}) = \mathbb{O}$;
- $co(\mathbb{O}) = \mathbb{R}$;
- $\forall \alpha \in \mathbb{R} \cup \mathbb{O} : \ co(co(\alpha)) = \alpha$;
- $co(\square) = \square$.

Let $\vec{v} = (a_1, ..., a_n)$ be a vector of *rank* $n \geq 1$, in symbols r_v, whose elements are actions and belong to $\mathbb{L}$, then $\vec{v}_{(i)}$ denotes the i-th element. We write $\vec{v}_1\vec{v}_2 \ldots \vec{v}_m$ for the concatenation of m vectors $\vec{v}_i$, while $|\vec{v}| = n$ is the rank (length) of $\vec{v}$ and $\vec{v}^n$ is the vector obtained by n concatenations of $\vec{v}$. The alphabet of a contract automaton consists of vectors, each element of which intuitively records the activity, i.e. the occurrence of a basic action of a single principal in the contract.

In a vector there is either a single offer or a single request, or there is a single pair of request-offer that matches, i.e. there exists exactly i, j such that $\vec{v}_{(i)}$ is an offer and $\vec{v}_{(j)}$ is the complementary request or vice-versa; all the other elements of the vector contain the symbol $\square$, meaning that the corresponding principals stay idle. In the following let $\square^m$ denote a vector of rank m, all elements of which are $\square$. Formally:

Definition 20 (Actions)**.** *Given a vector $\vec{a} \in \mathbb{L}^n$, if*

- $\vec{a} = \square^{n_1}\alpha\square^{n_2}, n_1, n_2 \geq 0$, *then $\vec{a}$ is a* request (action) on α *if $\alpha \in \mathbb{R}$, and is an* offer (action) on α *if $\alpha \in \mathbb{O}$*
- $\vec{a} = \square^{n_1}\alpha\square^{n_2}co(\alpha)\square^{n_3}, n_1, n_2, n_3 \geq 0$, *then $\vec{a}$ is a* match (action) on α, *where $\alpha \in \mathbb{R} \cup \mathbb{O}$.*

We now define complementary actions.

Definition 21 (Complementary Actions)**.** *Two actions $\vec{a}$ and $\vec{b}$ are* complementary, *in symbols $\vec{a} \bowtie \vec{b}$ if and only if*

- $\exists \alpha \in \mathbb{R} \cup \mathbb{O} : \ \vec{a}$ *is either a request or an offer on α;*
- $\vec{a}$ *is an offer on $\alpha \implies \vec{b}$ is a request on $co(\alpha)$ and*

- $\vec{a}$ *is a request on* $\alpha \implies \vec{b}$ *is an offer on* $co(\alpha)$.

We now extract from an action the request or offer made by a principal, and the matching of a request and an offer, and then we lift this notion to a sequence of actions, i.e. to a trace of a contract automaton that intuitively corresponds to an execution of a service composition.

Definition 22 (Observable)**.**

Let $w = \vec{a}_1 \dots \vec{a}_n$ *be a sequence of actions, and let* ε *be the empty one, then its* observable *is given by the partial function* $Obs(w) \in (\mathbb{R} \cup \mathbb{O} \cup \{\tau\})^*$ *where:*

$$Obs(\varepsilon) = \varepsilon$$

$$Obs(\vec{a}\, w') = \begin{cases} \vec{a}_{(i)}\, Obs(w') & \text{if } \vec{a} \text{ is an offer/request and } \vec{a}_{(i)} \neq \square \\ \tau\, Obs(w') & \text{if } \vec{a} \text{ is a match} \end{cases}$$

We now define contract automata, the actions and states of which are vectors of basic actions and of states of principals, respectively.

Definition 23 (Contract Automata)**.** *Assume as given a finite set of states* $\mathcal{Q} = \{q_1, q_2, \dots\}$. *Then a* contract automaton $\mathcal{A}$ *(CA for short) of rank* n *is a tuple* $\langle Q, \vec{q_0}, A^r, A^o, T, F \rangle$, *where*

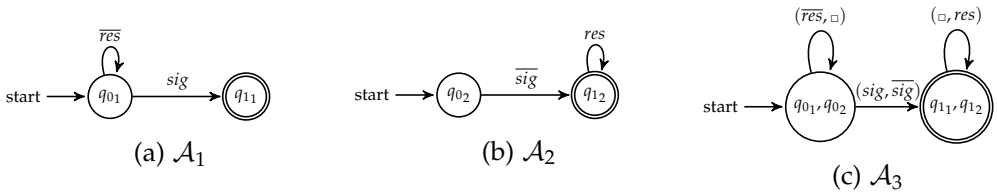
- $Q = Q_1 \times \dots \times Q_n \subseteq \mathcal{Q}^n$
- $\vec{q_0} \in Q$ *is the initial state*
- $A^r \subseteq \mathbb{R}$, $A^o \subseteq \mathbb{O}$ *are finite sets (of requests and offers, respectively)*
- $F \subseteq Q$ *is the set of final states*
- $T \subseteq Q \times A \times Q$ *is the set of transitions, where* $A \subseteq (A^r \cup A^o \cup \{\square\})^n$ *and if* $(\vec{q}, \vec{a}, \vec{q'}) \in T$ *then both the following conditions hold:*
  - $\vec{a}$ *is either a request or an offer or a match*
  - *if* $\vec{a}_{(i)} = \square$ *then it must be* $\vec{q}_{(i)} = \vec{q'}_{(i)}$

A principal contract automaton *(or simply* principal*) is a contract automaton of rank 1 such that* $A^r \cap co(A^o) = \emptyset$.

A benefit of adopting contract automata for describing services is the possibility to express as a single automaton both individual principals, where no action is matched, closed systems (i.e. composition of principals) where all actions are matched, and open systems, where not all actions are matched.

Given a contract automaton $\mathcal{A}$ of rank n, the standard definitions and constructions of finite-state automata apply. In particular,

- the configurations of $\mathcal{A}$ are pairs in $A^* \times Q$;



(a) $\mathcal{A}_1$ (b) $\mathcal{A}_2$ (c) $\mathcal{A}_3$

Figure 3.1: Three contract automata

- $\mathcal{A}$ *moves a step from* $(w, \vec{q})$ *to* $(w', \vec{q'})$, written $(w, \vec{q}) \xrightarrow{\vec{a}} (w', \vec{q'})$, if and only if $w = \vec{a}w'$, $w' \in A^*$ and $(\vec{q}, \vec{a}, \vec{q'}) \in T$; we write $(w, \vec{q}) \to (w', \vec{q'})$ when $\vec{a}$ is immaterial and $\vec{q} \xrightarrow{\vec{a}} \vec{q'}$ when w is immaterial (note that transitions are triples, instead);
- the *language of* $\mathcal{A}$ is $\mathscr{L}(\mathcal{A}) = \{w \mid (w, \vec{q_0}) \to^* (\varepsilon, \vec{q}), \vec{q} \in F\}$ where $\to^*$ is the reflexive transitive closure of $\to$. As usual $\vec{q}_1 \xrightarrow{\vec{a}_1 \cdots \vec{a}_m} \vec{q}_{m+1}$ shortens $\vec{q}_1 \xrightarrow{\vec{a}_1} \vec{q}_2 \cdots \vec{q}_m \xrightarrow{\vec{a}_m} \vec{q}_{m+1}$ (for some $\vec{q}_2, \ldots, \vec{q}_m$) and we say that $\vec{q}_1$ is reachable in $\mathcal{A}$ if $\vec{q}_0 \xrightarrow{w} \vec{q}_1$; finally $\vec{q} \not\to$ if and only if for no $\vec{q'}$ it is the case that $\vec{q} \to \vec{q'}$.

Note that for principals we have the restriction $A^r \cap co(A^o) = \emptyset$. Indeed, a principal who offers what he requires makes little sense.

Example 5. *Figure 3.1 shows three contract automata. The automaton* $\mathcal{A}_1$ *may be understood as producing a certain number of resources through one or more offers* $\overline{res}$ *and it terminates with the request of receiving a signal sig. The contract* $\mathcal{A}_2$ *starts by sending the signal* $\overline{sig}$ *and then it collects the resources produced by* $\mathcal{A}_1$*. The contract* $\mathcal{A}_3$ *represents the contract automaton where* $\mathcal{A}_1$ *and* $\mathcal{A}_2$ *interact as discussed below. Both* $\mathcal{A}_1$ *and* $\mathcal{A}_2$ *have rank 1 while* $\mathcal{A}_3$ *has rank 2.*

Contract automata can be composed, by making the cartesian product of their states and of the labels of the joined transitions, with the additional possibility of labels recording matching request-offer. This is the case for the action $(sig, \overline{sig})$ of the contract automaton $\mathcal{A}_3$ in Figure 3.1.

Below, we introduce two different operators for composing contract automata. Both products interleave all the transitions of their operands. We only force a synchronisation to happen when two contract automata are ready on their respective request/offer action. These operators represent two different policies of orchestration. The first operator is called simply *product* and it considers the case when a service S joins a group of services already clustered as a single orchestrated service S'. In the product of S and S', the first can only accept the still available offers (requests, respectively) of S' and vice versa. In other words, S cannot interact with the principals of the orchestration S', but only with it as a whole component, that is already matched actions in S' are not split. This is obtained in Definition 24 through the relation $\bowtie$ (see Definition 21), which is only defined for actions that are not matches.

This is not the case with the second operation of composition, called *a-product*: it puts instead all the principals of S at the same level of those of S'. Any matching request-offer of either contract can be split, and the offers and requests, that become available again, can be re-combined with complementary actions of S, and vice-versa.

The a-product turns out to satisfactorily model coordination policies in dynamically changing environments, because the a-product supports a form of *dynamic orchestration*, that adjusts the workflow of messages when new principals join the contract.

We now introduce our first operation of composition; recall that we implicitly assume the alphabet of a contract automaton of rank m to be $A \subseteq (A^r \cup A^o \cup \{\square\})^m$. Note that the first case of the definition of T below is for the matching of actions of two automata, while the other considers the action of a single automaton.

Definition 24 (Product). *Let $\mathcal{A}_i = \langle Q_i, \vec{q}_{0_i}, A_i^r, A_i^o, T_i, F_i \rangle, i \in 1 \dots n$ be contract automata of rank r_i. The* product $\bigotimes_{i \in 1 \dots n} \mathcal{A}_i$ *is the contract automaton $\langle Q, \vec{q}_0, A^r, A^o, T, F \rangle$ of rank $m = \sum_{i \in 1 \dots n} r_i$, where:*

- $Q = Q_1 \times ... \times Q_n, \quad$ *where* $\vec{q}_0 = \vec{q}_{0_1} \dots \vec{q}_{0_n}$
- $A^r = \bigcup_{i \in 1 \cdots n} A_i^r, \quad A^o = \bigcup_{i \in 1 \cdots n} A_i^o$
- $F = \{\vec{q}_1 \dots \vec{q}_n \mid \vec{q}_1 \dots \vec{q}_n \in Q, \vec{q}_i \in F_i, i \in 1 \dots n\}$
- *T is the least subset of $Q \times A \times Q$ s.t. $(\vec{q}, \vec{c}, \vec{q}') \in T$ iff, when $\vec{q} = \vec{q}_1 \dots \vec{q}_n \in Q$,*

  either *there are $1 \leq i < j \leq n$ s.t. $(\vec{q}_i, \vec{a}_i, \vec{q}_i') \in T_i$, $(\vec{q}_j, \vec{a}_j, \vec{q}_j') \in T_j$, $\vec{a}_i \bowtie \vec{a}_j$ and*

  $$\begin{cases} \vec{c} = \square^u \vec{a}_i \square^v \vec{a}_j \square^z \text{ with } u = r_1 + \ldots + r_{i-1},\ v = r_{i+1} + \ldots + r_{j-1}, |\vec{c}| = m \\ \text{and} \\ \vec{q}' = \vec{q}_1 \dots \vec{q}_{i-1}\ \vec{q}_i'\ \vec{q}_{i+1} \dots\ \vec{q}_{j-1}\ \vec{q}_j'\ \vec{q}_{j+1} \dots \vec{q}_n \end{cases}$$

  or *there is $1 \leq i \leq n$ s.t. $(\vec{q}_i, \vec{a}_i, \vec{q}_i') \in T_i$ and*
  $\vec{c} = \square^u \vec{a}_i \square^v$ *with $u = r_1 + \ldots + r_{i-1}$, $v = r_{i+1} + \ldots + r_n$, and*
  $\vec{q}' = \vec{q}_1 \dots \vec{q}_{i-1}\ \vec{q}_i'\ \vec{q}_{i+1} \dots \vec{q}_n$ *and*
  $\forall j \neq i, 1 \leq j \leq n, (\vec{q}_j, \vec{a}_j, \vec{q}_j') \in T_j$ *it does not hold that $\vec{a}_i \bowtie \vec{a}_j$.*

There is a simple way of retrieving the principals involved in a composition of contract automata obtained through the product introduced above: just introduce projections $\prod^i$ as done below. For example, for the contract automata in Figure 3.1, we have $\mathcal{A}_1 = \prod^1(\mathcal{A}_3)$ and $\mathcal{A}_2 = \prod^2(\mathcal{A}_3)$.

Definition 25 (Projection). *Let $\mathcal{A} = \langle Q, \vec{q}_0, A^r, A^o, T, F \rangle$ be a contract automaton of rank n, then the* projection *on the i-th principal is:*

$$\prod^i(\mathcal{A}) = \langle \prod^i(Q), \vec{q}_{0_{(i)}}, \prod^i(A^r), \prod^i(A^o), \prod^i(T), \prod^i(F) \rangle$$

where $i \in 1 \dots n$ and:

$$\prod^i(Q) = \{\vec{q}_{(i)} \mid \vec{q} \in Q\} \qquad \prod^i(F) = \{\vec{q}_{(i)} \mid \vec{q} \in F\}$$

$$\prod^i(A^r) = \{a \mid a \in A^r, (q, a, q') \in \prod^i(T)\} \qquad \prod^i(A^o) = \{\overline{a} \mid \overline{a} \in A^o, (q, \overline{a}, q') \in \prod^i(T)\}$$

$$\prod^i(T) = \{(\vec{q}_{(i)}, \vec{a}_{(i)}, \vec{q'}_{(i)}) \mid (\vec{q}, \vec{a}, \vec{q}') \in T \wedge \vec{a}_{(i)} \neq \square\}$$

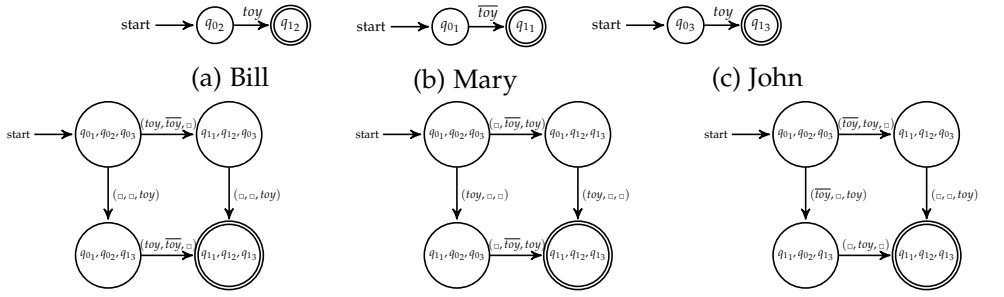


Figure 3.2: Three contract automata, their product and a-product

The following property states that decomposition is the inverse of product.

Property 1 (Product Decomposition)**.** *Let $\mathcal{A}_1, \ldots, \mathcal{A}_n$ be a set of principal contract automata, then $\prod^i(\bigotimes_{j \in 1 \ldots n} \mathcal{A}_j) = \mathcal{A}_i$.*

Proof. We have $\mathcal{A}_i = \langle Q_i, q_{0_i}, A_i^r, A_i^o, T_i, F_i \rangle$, and by Definition 24 $\prod^i(Q) = Q_i, \prod^i(F) = F_i$, also $\prod^i(T) = T_i$ since by Definition 25 we take only the transition where $\mathcal{A}_i$ makes a step, while avoiding the transitions in which $\mathcal{A}_i$ stays idle. Finally $\prod^i(A^r) = A_i^r, \prod^i(A^o) = A_i^o$ because we select only the actions that appear as labels in the transitions of $\mathcal{A}_i$. □

Our second operation of composition first extracts from its operands the principals they are composed of, and then reassembles them.

Definition 26 (a-Product)**.** *Let $\mathcal{A}_1, \mathcal{A}_2$ be two contract automata of rank n and m, respectively, and let $I = \{\prod^i(\mathcal{A}_1) \mid 0 < i \leq n\} \cup \{\prod^j(\mathcal{A}_2) \mid 0 < j \leq m\}$. Then the* a-product *of $\mathcal{A}_1$ and $\mathcal{A}_2$ is $\mathcal{A}_1 \boxtimes \mathcal{A}_2 = \bigotimes_{\mathcal{A}_i \in I} \mathcal{A}_i$.*

Note that if $\mathcal{A}, \mathcal{A}'$ are principal contract automata, then $\mathcal{A} \otimes \mathcal{A}' = \mathcal{A} \boxtimes \mathcal{A}'$, and that the complexity of this operation is similar to the complexity of the product. E.g. in Figure 3.1, we have that $\mathcal{A}_3 = \mathcal{A}_1 \otimes \mathcal{A}_2 = \mathcal{A}_1 \boxtimes \mathcal{A}_2$.

From now onwards we assume that every contract automaton $\mathcal{A}$ of rank $r_{\mathcal{A}} > 1$ is composed by principal contract automata using the operations of product and a-product.

Both compositions are commutative, up to the expected rearrangement of the vectors of actions, and $\boxtimes$ is also associative, while $\otimes$ is not, as shown by the following example.

Example 6. *In Figure 3.2 Mary offers a toy that both Bill and John request. In the product $(Bill \otimes Mary) \otimes John$ the toy is assigned to Bill who first enters into the composition with Mary, no matter if John performs the same move. Equally, in the product $Bill \otimes (Mary \otimes John)$ the toy is assigned to John. In Figure 3.2f we have the a-product that represents a dynamic re-orchestration: no matter of who is first composed with Mary, the toy will be non-deterministically assigned to either principal.*

Proposition 1. *The following properties hold:*
– $\exists \mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3.(\mathcal{A}_1 \otimes \mathcal{A}_2) \otimes \mathcal{A}_3 \neq \mathcal{A}_1 \otimes (\mathcal{A}_2 \otimes \mathcal{A}_3)$
– $\forall \mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3.(\mathcal{A}_1 \boxtimes \mathcal{A}_2) \boxtimes \mathcal{A}_3 = \mathcal{A}_1 \boxtimes (\mathcal{A}_2 \boxtimes \mathcal{A}_3)$

Proof. Example 6 suffices to prove the first statement. For the second statement one has $\mathcal{A} = (\mathcal{A}_1 \boxtimes \mathcal{A}_2) \boxtimes \mathcal{A}_3 = \bigotimes_{\mathcal{A}_i \in I} \mathcal{A}_i = \mathcal{A}_1 \boxtimes (\mathcal{A}_2 \boxtimes \mathcal{A}_3)$ where $I = \{\Pi^i(\mathcal{A}) \mid i \in 1,2,3\}$. □

3.2 Enforcing Agreement

It is common to say that some contracts are in agreement when all the requests they make have been fulfilled by corresponding offers [CGP09, CP09, BSBM05, LP15, BvBd15, BDLd15, dNH83, BH14, BCPZ16, BTZ12] (see Section 1.2). In terms of contract automata, this is rendered in two different ways, the first of which is introduced below and resembles the notion of compliance introduced in [CGP09, CP09]. We say that two or more contract automata are in *agreement* when the final states of their product are reachable from the initial state by traces only made of matches and offer actions. Our goal is to enforce the behaviour of principals so that they only follow the traces of the automaton which lead to agreement. Additionally, it is easy to track every action performed by each principal, because we use vectors of actions as the elements of the alphabet of contract automata. It is equally easy finding who is liable in a bad interaction, i.e. the principals who perform a transition leaving a state from which agreement is possible, reaching a state where instead agreement is no longer possible.

We now introduce the notion of *agreement* as a property of the language recognised by a contract automaton.

Definition 27 (Agreement)**.** *A trace accepted by a contract automaton is in* agreement *if it belongs to the set*

$$\mathfrak{A} = \{w \in (\mathbb{L}^n)^* \mid Obs(w) \in (\mathbb{O} \cup \{\tau\})^*, n > 1\}$$

Note that, if an action observable in w is a request, i.e. it belongs to $\mathbb{R}$, then w is not in agreement. Intuitively, a trace is in agreement if it is only composed of offer and match actions, that is no requests are left unsatisfied.

Example 7. *The automaton $\mathcal{A}_3$ in Figure 3.1 has a trace in agreement: $Obs((\overline{res}, \Box)(sig, \overline{sig})) = \overline{res}\,\tau \in \mathfrak{A}$, and one not in agreement: $Obs((sig, \overline{sig})(\Box, res)) = \tau\, res \notin \mathfrak{A}$.*

A contract automaton is safe when all the traces of its language are in agreement, and admits agreement when at least one of its traces is in agreement. Formally:

Definition 28 (Safety)**.** *A contract automaton $\mathcal{A}$ is* safe *if $\mathscr{L}(\mathcal{A}) \subseteq \mathfrak{A}$, otherwise it is* unsafe*.*
Additionally, if $\mathscr{L}(\mathcal{A}) \cap \mathfrak{A} \neq \emptyset$ then $\mathcal{A}$ admits agreement*.*

Example 8. *The contract automaton $\mathcal{A}_3$ of Figure 3.1 is unsafe, but it admits agreement since $\mathscr{L}(\mathcal{A}_3) \cap \mathfrak{A} = (\overline{res}, \Box)^*(sig, \overline{sig})$. Consider now the contract automata Bill and Mary in Figure 3.2; their product Bill $\otimes$ Mary is safe because $\mathscr{L}(Bill \otimes Mary) = (toy, \overline{toy}) \subset \mathfrak{A}$.*

Note that the set $\mathfrak{A}$ can be seen as a safety property in the default-accept approach [Sch00], where the set of bad prefixes of $\mathfrak{A}$ contains those traces ending with a trailing request, i.e. $\{w\vec{a} \mid w \in \mathfrak{A}, Obs(\vec{a}) \in \mathbb{R}\}$. One could then consider a definition of product that disallows the occurrence of transitions labelled by requests only. However, this choice would not prevent a product of contracts to reach a deadlock. In addition, compositionality would have been compromised, as shown in the following example.

Example 9. *In what follows, we feel free to present contract automata through a sort of extended regular expressions. Consider a simple selling scenario involving two parties Ann and Bart.*

Bart starts by notifying Ann that he is ready to start the negotiation, and waits from Ann to select a pen or a book. In case Ann selects the pen, he may decide to withdraw and restart the negotiation again, or to accept the payment. As soon as Ann selects the book, then Bart cannot withdraw any longer, and waits for the payment. The contract of Bart is:

$$Bart = (\overline{init}.pen.\overline{cancel})^*.(\overline{init}.book.pay + \overline{init}.pen.pay)$$

The contract of Ann is dual to Bart's. Ann waits to receive a notification from Bart when ready to negotiate. Then Ann decides what to buy. If she chooses the pen, she may proceed with the payment unless a withdrawal from Bart is received. In this case, Ann can repeatedly try to get the pen, until she succeeds and pays for it, or buys the book but maliciously omits to pay it.

The contract of Ann is:

$$Ann = (init.\overline{pen}.cancel)^*.(init.\overline{pen}.\overline{pay} + init.\overline{book})$$

The contract $\mathcal{A} = Ann \otimes Bart$ is in Figure 3.3a.

Assume now to change the product $\otimes$ so to disallow transitions labelled by requests. The composition of Ann and Bart is in Figure 3.3c, and contains the malformed trace in which Bart does not reach a final state:

$$(init, \overline{init})(\overline{book}, book)$$

In addition, if a third principal Carol $= \overline{pay}$ were involved, willing to pay for everybody, the following trace in agreement would not be accepted

$$(init, \overline{init}, \square)(\overline{book}, book, \square)(\square, pay, \overline{pay})$$

because Bart's request was discarded by the wrongly amended composition operator. So, compositionality would be lost.

To avoid the two unpleasant situations of deadlock and lack of compositionality, we introduce below a technique for driving a safe composition of contracts, in the style of the Supervisory Control for Discrete Event Systems (see Section 1.7).

The purpose of contracts is to declare all the activities of a principal in terms of requests and offers. Therefore all the actions of a (composed) contract are controllable and observable. Clearly, the behaviours that we want to enforce upon a given contract automaton $\mathcal{A}$ are exactly the traces in agreement, and so we assume that a request leads to a forbidden state. A most permissive controller exists for contract automata and is defined below.

Definition 29 (Controller for Agreement)**.** *Let $\mathcal{A}$ and $\mathcal{K}$ be contract automata, we call $\mathcal{K}$* controller *of $\mathcal{A}$ if and only if $\mathscr{L}(\mathcal{K}) \subseteq \mathfrak{A} \cap \mathscr{L}(\mathcal{A})$.*
A controller $\mathcal{K}$ of $\mathcal{A}$ is the most permissive controller (mpc) *if and only if for all $\mathcal{K}'$ controller of $\mathcal{A}$ it is $\mathscr{L}(\mathcal{K}') \subseteq \mathscr{L}(\mathcal{K})$.*

Proposition 2. *Let $\mathcal{K}$ be the mpc of the contract automaton $\mathcal{A}$, then $\mathscr{L}(\mathcal{K}) = \mathfrak{A} \cap \mathscr{L}(\mathcal{A})$.*

Proof. The existence of $\mathcal{K}$ is guaranteed since all actions are controllable and observable and $\mathscr{L}(\mathcal{A})$ is regular, as well as $\mathfrak{A}$ [CL06]. By contradiction assume $\mathscr{L}(\mathcal{K}) \subset \mathfrak{A} \cap \mathscr{L}(\mathcal{A})$, then there exists another controller $\mathcal{K}'_{\mathcal{A}}$ such that $\mathscr{L}(\mathcal{K}) \subset \mathscr{L}(\mathcal{K}') = \mathfrak{A} \cap \mathscr{L}(\mathcal{A})$. □

In order to effectively build the most permissive controller, we introduce below the notion of hanged state, i.e. a state from which no final state can be reached.

Definition 30 (Hanged state)**.** *Let $\mathcal{A} = \langle Q, \vec{q_0}, A^r, A^o, T, F \rangle$ be a contract automaton, then $\vec{q} \in Q$ is* hanged*, and belongs to the set $Hanged(\mathcal{A})$, if for all $\vec{q}_f \in F, \nexists w.(w, \vec{q}) \to^* (\varepsilon, \vec{q}_f)$.*

Definition 31 (Mpc construction)**.** *Let $\mathcal{A} = \langle Q, \vec{q_0}, A^r, A^o, T, F \rangle$ be a contract automaton, $\mathcal{K}_1 = \langle Q, \vec{q_0}, A^r, A^o, T \setminus (\{t \in T \mid t \text{ is a request transition } \}, F \rangle$ and define*

$$\mathcal{K}_{\mathcal{A}} = \langle Q \setminus Hanged(\mathcal{K}_1), \vec{q_0}, A^r, A^o, T_{\mathcal{K}_1} \setminus \{(\vec{q}, a, \vec{q'}) \mid \{\vec{q}, \vec{q'}\} \cap Hanged(\mathcal{K}_1) \neq \emptyset\}, F \rangle$$

Proposition 3 (Mpc)**.** *The controller $\mathcal{K}_A$ of Definition 31 is the most permissive controller of the contract automaton $\mathcal{A}$.*

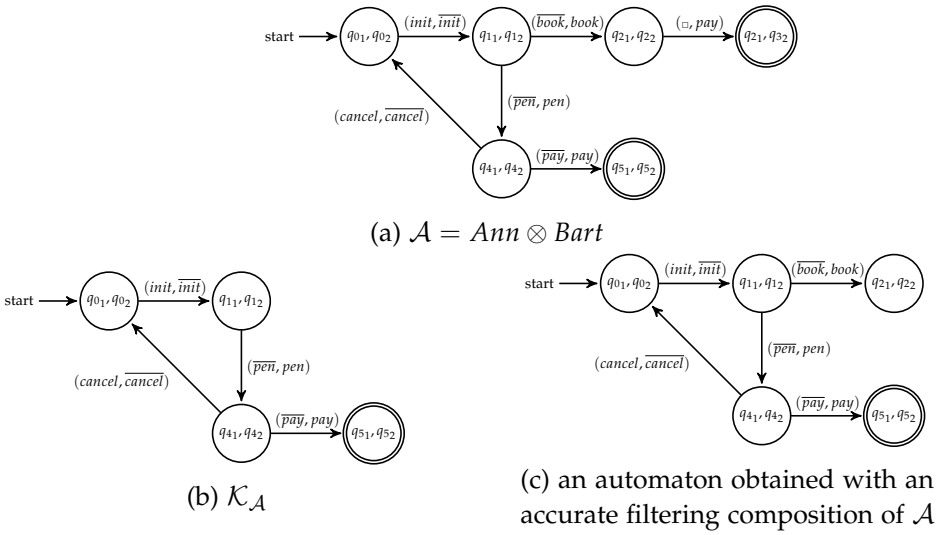
Proof. In $\mathcal{K}_{\mathcal{A}}$ every request transition is removed in the first step, so it must be $\mathscr{L}(\mathcal{K}_{\mathcal{A}}) \subseteq \mathfrak{A} \cap \mathscr{L}(\mathcal{A})$. We will prove that $\mathscr{L}(\mathcal{K}_{\mathcal{A}}) = \mathfrak{A} \cap \mathscr{L}(\mathcal{A})$, from this follows that $\mathcal{K}_{\mathcal{A}}$ is the most permissive controller. By contradiction assume that exists a trace $w \in \mathfrak{A} \cap \mathscr{L}(\mathcal{A}), w \notin \mathscr{L}(\mathcal{K}_{\mathcal{A}})$. Then there exist a transition $t = (\vec{q}, \vec{a}, \vec{q'}) \notin T_{\mathcal{K}_{\mathcal{A}}}$ in the accepting path of w (i.e. the sequence of transitions used to recognise w). The transition t is not a request since $w \in \mathfrak{A} \cap \mathscr{L}(\mathcal{A})$, and $\vec{q}, \vec{q'} \notin Hanged(\mathcal{K}_{\mathcal{A}})$ because the transition belongs to an accepting path. Since the only transitions removed to obtain $\mathcal{K}_{\mathcal{A}}$ are requests and those involving hanged states, it follows that $t \in T_{\mathcal{K}_{\mathcal{A}}}$. □

Example 10. *Consider again Example 9. For obtaining the most permissive controller we first compute the auxiliary set $\mathcal{K}_1$ that does not contain the transition $((q_{2_1}, q_{2_2}), (\square, pay), (q_{2_1}, q_{3_2}))$ because it represents a request from Bart which is not fulfilled by Ann. As a consequence, some states are hanged:*

$$Hanged(\mathcal{K}_1) = \{(q_{2_1}, q_{2_2})\}$$

By removing them, we eventually obtain $\mathcal{K}_{\mathcal{A}}$, the most permissive controller of $\mathcal{A}$ depicted in Figure 3.3b (unreachable states are not considered).

The following property rephrases the notions of safe, unsafe and admits agreement on automata in terms of their most permissive controllers.



(a) $\mathcal{A} = Ann \otimes Bart$

(b) $\mathcal{K}_{\mathcal{A}}$

(c) an automaton obtained with an inaccurate filtering composition of $\mathcal{A}$

Figure 3.3: The contract automata of Example 9

Property 2. *Let $\mathcal{A}$ be a contract automaton and let $\mathcal{K}_{\mathcal{A}}$ be its mpc, the following hold:*

- *if $\mathscr{L}(\mathcal{K}_{\mathcal{A}}) = \mathscr{L}(\mathcal{A})$ then $\mathcal{A}$ is safe, otherwise if $\mathscr{L}(\mathcal{K}_{\mathcal{A}}) \subset \mathscr{L}(\mathcal{A})$ then $\mathcal{A}$ is unsafe;*
- *if $\mathscr{L}(\mathcal{K}_{\mathcal{A}}) \neq \emptyset$, then $\mathcal{A}$ admits agreement.*

Proof. If $\mathscr{L}(\mathcal{K}_{\mathcal{A}}) = \mathscr{L}(\mathcal{A})$ then by Proposition 2 we have $\mathscr{L}(\mathcal{A}) = \mathfrak{A} \cap \mathscr{L}(\mathcal{A})$, hence $\mathscr{L}(\mathcal{A}) \subseteq \mathfrak{A}$ and $\mathcal{A}$ is *safe*.

If $\mathscr{L}(\mathcal{K}_{\mathcal{A}}) \subset \mathscr{L}(\mathcal{A})$ then $\mathfrak{A} \cap \mathscr{L}(\mathcal{A}) \subset \mathscr{L}(\mathcal{A})$, hence $\mathscr{L}(\mathcal{A}) \not\subseteq \mathfrak{A}$ and $\mathcal{A}$ is *unsafe*.

Finally if $\mathscr{L}(\mathcal{K}_{\mathcal{A}}) \neq \emptyset$ then $\mathfrak{A} \cap \mathscr{L}(\mathcal{A}) \neq \emptyset$, and $\mathcal{A}$ *admits agreement*. □

We introduce now an original notion of *liability*, that characterises those principals potentially responsible of the divergence from the expected behaviour. The liable principals are those who perform a transition t from a state $\vec{q}$ such that $\vec{q}$ is reachable in the mpc but t is not allowed in it. As noticed above, after this step is done, a successful state cannot be reached any longer, and so the principals who performed it will be blamed. (Note in passing that hanged states play a crucial role here: just removing the request transitions from $\mathcal{A}$ would result in a contract automaton language equivalent to the mpc, but detecting liable principals would be much more intricate).

Definition 32 (Liability)**.** *Let $\mathcal{A}$ be a contract automaton and $\mathcal{K}_{\mathcal{A}}$ be its mpc of Definition 31; let $(v\vec{a}w, \vec{q_0}) \rightarrow^* (\vec{a}w, \vec{q})$ be a run of both automata and let $\vec{q}$ be such that $(\vec{a}w, \vec{q}) \rightarrow (w, \vec{q'})$ is possible in $\mathcal{A}$ but not in $\mathcal{K}_{\mathcal{A}}$.*

The principals $\Pi^i(\mathcal{A})$ such that $\vec{a}_{(i)} \neq \square, i \in 1 \ldots r_{\mathcal{A}}$ are liable for *$\vec{a}$ and belong to $Liable(\mathcal{A}, v\vec{a}w)$. The set of* liable *principals in $\mathcal{A}$ is $Liable(\mathcal{A}) = \{i \mid \exists w.i \in Liable(\mathcal{A}, w)\}$.*

Note that by using as a controller the intersection of the bad prefixes of the safety property $\mathfrak{A}$ with $\mathcal{A}$ we would not be able to compute the set of participants which are responsible for the violation of the agreement. Indeed in this case the liable participants would be the first ones to perform a request in a run.

Example 11. *In Figure 3.3b, we have $Liable(\mathcal{A}) = \{1,2\}$, hence both Ann and Bart are possibly liable, because the match transition with label $(\overline{book}, book)$ can be performed, that leads to a violation of the agreement. Note that by computing $BadPrefix \cap \mathcal{A}$ (Figure 3.3c), the set of liable participants becomes $\{2\}$ (Bart). Indeed in this way it is not possible to track the first transition which diverges from the agreement.*

The following proposition relates safety and liability.

Proposition 4. *A contract automaton $\mathcal{A}$ is safe if and only if $Liable(\mathcal{A}) = \emptyset$.*

Proof. If $Liable(\mathcal{A}) = \emptyset$ then $\mathscr{L}(\mathcal{K}_{\mathcal{A}}) = \mathscr{L}(\mathcal{A})$. The thesis follows by applying Proposition 2. □

The set $Liable(\mathcal{A})$ is easily computable as follows.

Proposition 5 (Liable Sets)**.** *Let $\mathcal{A}$ be a CA and $\mathcal{K}_{\mathcal{A}}$ be its mpc as in Definition 31, then*

$$Liable(\mathcal{A}) = \{i \mid (\vec{q}, \vec{a}, \vec{q'}) \in T_{\mathcal{A}}, \vec{a}_{(i)} \neq \square, \vec{q} \in Q_{\mathcal{K}_{\mathcal{A}}}, \vec{q'} \notin Q_{\mathcal{K}_{\mathcal{A}}}\}$$

Proof. Let $Liable(\mathcal{A})$ be the set of Definition 32 and $I = \{i \mid (\vec{q}, \vec{a}, \vec{q'}) \in T_{\mathcal{A}}, \vec{a}_{(i)} \neq \square, \vec{q} \in Q_{\mathcal{K}_{\mathcal{A}}}, \vec{q'} \notin Q_{\mathcal{K}_{\mathcal{A}}}\}$.

- $I \subseteq Liable(\mathcal{A})$: assume $i \in I$, hence there exists a string $w\vec{a}$ such that $\vec{a}_{(i)} \neq \square$ and $(w\vec{a}, \vec{q_0}) \rightarrow^*_{\mathcal{K}_{\mathcal{A}}} (\vec{a}, \vec{q}) \rightarrow_{\mathcal{A}} (\varepsilon, \vec{q'})$ and $(\vec{a}, \vec{q}) \not\rightarrow_{\mathcal{K}_{\mathcal{A}}} (\varepsilon, \vec{q'})$, hence $i \in Liable(\mathcal{A})$.
- $Liable(\mathcal{A}) \subseteq I$: assume $i \in Liable(\mathcal{A})$, we have $(w\vec{a}, \vec{q_0}) \rightarrow^*_{\mathcal{K}_{\mathcal{A}}} (\vec{a}, \vec{q}) \rightarrow_{\mathcal{A}} (\varepsilon, \vec{q'})$ and $(\vec{a}, \vec{q}) \not\rightarrow_{\mathcal{K}_{\mathcal{A}}} (\varepsilon, \vec{q'})$, for some $w\vec{a}$ with $\vec{a}_{(i)} \neq \square$. We have $(\vec{q}, \vec{a}, \vec{q'}) \in T_{\mathcal{A}}$ such that $\vec{q} \in Q_{\mathcal{K}_{\mathcal{A}}}, \vec{q'} \notin Q_{\mathcal{K}_{\mathcal{A}}}$, hence $i \in I$.

□

If the composition of contracts is *safe*, then it is possible to turn off the controller: an unsafe trace will never belong to $\bigotimes_{i \in I} \mathcal{A}_i$. To statically decide if a contract automaton is safe it suffices to compute its mpc and to check whether it is equivalent to $\mathcal{A}$.

Some properties of $\otimes$ and $\boxtimes$ follow, that enable us to predict under which conditions a composition is safe without actually computing it, so avoid generating the whole state space. This supports modular and compositional verification of service contracts.

We first introduce the notions of collaborative and competitive contracts. Intuitively, two contracts are *collaborative* if some requests of one meet the offers of the other, and are *competitive* if both can satisfy the same request. An example follows.

Example 12. *Consider the contract automata Bill, Mary, John in Figure 3.2. In Figure 3.4 the contract automaton Bill $\otimes$ John is displayed. The two contract automata Mary and Bill $\otimes$ John are collaborative and not competitive, indeed the offer $\overline{toy}$ of Mary is matched in Bill $\otimes$ John, and no other principals interfere with this offer. Moreover, let $\mathcal{A}_1 = \overline{apple} + cake \otimes apple + \overline{cake}$ and $\mathcal{A}_2 = \overline{apple}$. The pair $\mathcal{A}_1, \mathcal{A}_2$ is competitive since $\mathcal{A}_2$ interferes with $\mathcal{A}_1$ on the $\overline{apple}$ offer.*

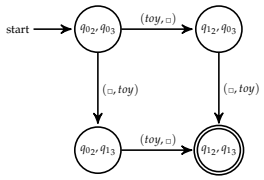


Figure 3.4: The contract automaton $Bill \otimes John$ of Example 12

.

Definition 33 (Competitive, Collaborative)**.** *The pair of contract automata*

$$\mathcal{A}_1 = \langle Q_1, \vec{q}_{0_1}, A_1^r, A_1^o, T_1, F_1 \rangle \qquad \mathcal{A}_2 = \langle Q_2, \vec{q}_{0_2}, A_2^r, A_2^o, T_2, F_2 \rangle$$

are

- competitive *if* $A_1^o \cap A_2^o \cap co(A_1^r \cup A_2^r) \neq \emptyset$
- collaborative *if* $(A_1^o \cap co(A_2^r)) \cup (co(A_1^r) \cap A_2^o) \neq \emptyset$.

Note that *competitive* and *collaborative* are not mutually exclusive, as stated in the first and second item of Theorem 3 below. Moreover if two contract automata are *non-competitive* then all their match actions are preserved in their composition, indeed we have $\mathcal{A}_1 \boxtimes \mathcal{A}_2 = \mathcal{A}_1 \otimes \mathcal{A}_2$.

The next theorem says that the composition of safe and non-competitive contracts prevents all principals from harmful interactions, unlike the case of safe competitive contracts. In other words, when $\mathcal{A}_1$ and $\mathcal{A}_2$ are safe, no principals will be found liable in $\mathcal{A}_1 \otimes \mathcal{A}_2$ (i.e. $Liable(\mathcal{A}_1 \otimes \mathcal{A}_2) = \emptyset$), and the same happens for $\mathcal{A}_1 \boxtimes \mathcal{A}_2$ if the two are also non-competitive (i.e. $Liable(\mathcal{A}_1 \boxtimes \mathcal{A}_2) = \emptyset$).

Theorem 3 (Competitive, Collaborative and Agreement)**.** *If two contract automata* $\mathcal{A}_1$ *and* $\mathcal{A}_2$ *are*

1. *competitive then they are collaborative,*
2. *collaborative and safe, then they are competitive,*
3. *safe then* $\mathcal{A}_1 \otimes \mathcal{A}_2$ *is safe,* $\mathcal{A}_1 \boxtimes \mathcal{A}_2$ *admits agreement,*
4. *non-collaborative, and one or both unsafe, then* $\mathcal{A}_1 \otimes \mathcal{A}_2$, $\mathcal{A}_1 \boxtimes \mathcal{A}_2$ *are unsafe,*
5. *safe and non-competitive, then* $\mathcal{A}_1 \boxtimes \mathcal{A}_2$ *is safe.*

Proof. 1) Assume by contradiction that $\mathcal{A}_1$ and $\mathcal{A}_2$ are non-collaborative, that is

$$(A_1^o \cap co(A_2^r)) \cup (co(A_1^r) \cap A_2^o) = \emptyset$$

Since the two automata are competitive, we have

$$A_1^o \cap A_2^o \cap (co(A_1^r) \cup co(A_2^r)) \neq \emptyset$$

By the distributive law

$$(A_1^o \cap (co(A_1^r) \cup co(A_2^r))) \cap (A_2^o \cap (co(A_1^r) \cup co(A_2^r))) \neq \emptyset$$

By hypothesis the two automata are non-collaborative, hence the above term can be rewritten as

$$(A_1^o \cap co(A_1^r)) \cap (co(A_2^r) \cap A_2^o) \neq \emptyset$$

By associative and commutative laws

$$(A_1^o \cap co(A_2^r)) \cap (co(A_1^r) \cap A_2^o) \neq \emptyset$$

Which implies

$$(A_1^o \cap co(A_2^r)) \cup (co(A_1^r) \cap A_2^o) \neq \emptyset$$

obtaining a contradiction.

2) By hypothesis the automata are collaborative:

$$(A_1^o \cap co(A_2^r)) \cup (A_2^o \cap co(A_1^r)) \neq \emptyset$$

By hypothesis $\mathcal{A}_1$ and $\mathcal{A}_2$ are safe, hence for each request there is a corresponding action, that is $co(A_i^r) \subseteq A_i^o$ where $i = 1, 2$. Then the following holds

$$A_i^o \cap co(A_i^r) = co(A_i^r) \qquad i = 1, 2$$

By substitution in the previous term we obtain

$$(A_1^o \cap A_2^o \cap co(A_2^r)) \cup (A_2^o \cap A_1^o \cap co(A_1^r)) \neq \emptyset$$

Which implies

$$(A_1^o \cap A_2^o \cap (co(A_1^r) \cup co(A_2^r))) \cup (A_2^o \cap A_1^o \cap (co(A_1^r) \cup co(A_2^r))) \neq \emptyset$$

By simplification we have

$$(A_1^o \cap A_2^o \cap (co(A_1^r) \cup co(A_2^r))) \neq \emptyset$$

Hence $\mathcal{A}_1$ and $\mathcal{A}_2$ are competitive.

3) Note that the labels of $\mathcal{A}_1 \otimes \mathcal{A}_2$ are the union of the labels of $\mathcal{A}_1$ and $\mathcal{A}_2$ (extended with idle actions for fitting the rank), hence no request transitions are added, and $\mathcal{A}_1 \otimes \mathcal{A}_2$ is *safe*.

Since the traces of $\mathcal{A}_1 \otimes \mathcal{A}_2$ are a subset of $\mathcal{A} = \mathcal{A}_1 \boxtimes \mathcal{A}_2$, $\mathcal{A}$ has at least a trace in agreement. Example 12 shows that not all the traces of $\mathcal{A}$ admit agreement.

4) Without loss of generality assume that $\mathcal{A}_1$ is unsafe, hence there exists a request $\vec{a}$, and traces w, v such that $w\vec{a}v \in \mathscr{L}(\mathcal{A}_1)$.

Since $\mathcal{A}_1$ and $\mathcal{A}_2$ are non-collaborative there will be no match between the actions of $\mathcal{A}_1$ and $\mathcal{A}_2$, hence we have $w_1\vec{a'}v_1 \in \mathscr{L}(\mathcal{A}_1 \otimes \mathcal{A}_2), w_2\vec{a'}v_2 \in \mathscr{L}(\mathcal{A}_1 \boxtimes \mathcal{A}_2)$ for some

w_1, w_2, v_1, v_2, where $\vec{a}'$ is obtained from $\vec{a}$ by adding the idle actions to principals from $r_{\mathcal{A}_1} + 1$ to $r_{\mathcal{A}_1} + r_{\mathcal{A}_2}$.

5) The proof is similar to 3, indeed it suffices to prove that no new matches between principals in $\mathcal{A}_1$ and $\mathcal{A}_2$ are introduced in $\mathcal{A}_1 \boxtimes \mathcal{A}_2$. By 2 it follows that $\mathcal{A}_1$ and $\mathcal{A}_2$ are non-collaborative:

$$(A_1^o \cap co(A_2^r)) \cup (A_2^o \cap co(A_1^r)) \neq \emptyset$$

This suffices to prove that no matches will be introduced in their composition.

□

Note that in item 3 of Theorem 3 it can be that $\mathcal{A}_1 \boxtimes \mathcal{A}_2$ is not *safe*. Moreover consider the contract automata $\mathcal{A}_1$ and $\mathcal{A}_2$ of Example 12. We have that $\mathcal{A}_1 \boxtimes \mathcal{A}_2$ is unsafe because the trace $(\square, apple, \overline{apple})(cake, \square, \square)$ belongs to $\mathscr{L}(\mathcal{A}_1 \boxtimes \mathcal{A}_2)$.

The following property relates liability with collaborative and competitive contracts. When $\mathcal{A}_1$ and $\mathcal{A}_2$ are safe, no participants will be found liable in $\mathcal{A}_1 \otimes \mathcal{A}_2$, and the same happens for $\mathcal{A}_1 \boxtimes \mathcal{A}_2$ if the two are also non-competitive.

Property 3. *Let $\mathcal{A}_1, \mathcal{A}_2$ be two* safe *CA, then $Liable(\mathcal{A}_1 \otimes \mathcal{A}_2) = \emptyset$. If additionally $\mathcal{A}_1, \mathcal{A}_2$ are non-competitive, then $Liable(\mathcal{A}_1 \boxtimes \mathcal{A}_2) = \emptyset$.*

Proof. The first item is a direct consequence of Theorem 3(3), while the second is a consequence of Theorem 3(5). □

3.3 Strong Agreement

This section introduces the notion of *strong agreement* on contract automata, that is a tighter version of the property of agreement. It requires *synchronous* fulfilment of all offers *and* requests. Strong agreement is introduced to prove the correspondence with communicating machines that always succeed in Chapter 5. The notions of *strong agreement, strong safety, strong controller* and *strong liability* are similar to those of the agreement property, and similar results can be obtained by simply substituting $\mathfrak{A}$ with $\mathfrak{Z}$ (defined below).

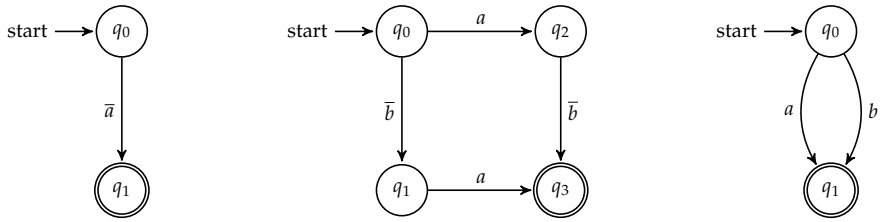
In the remaining of this section we report the main definitions.

Definition 34 (Strong Agreement and Strong Safety)**.** *A* strong agreement *on $(\mathbb{L}^n)^*$, $n > 1$, is a finite sequence of match actions. We let $\mathfrak{Z}$ denote the set of all strong agreements on $(\mathbb{L}^n)^*$.*

A contract automaton $\mathcal{A}$ is strongly safe *if $\mathscr{L}(\mathcal{A}) \subseteq \mathfrak{Z}$. We say that $\mathcal{A}$* admits strong agreement *when $\mathscr{L}(\mathcal{A}) \cap \mathfrak{Z} \neq \emptyset$.*

Definition 35 (Strong Controller)**.** *A* (strong) controller *of $\mathcal{A}$ is a contract automaton $\mathcal{KS}_{\mathcal{A}}$ such that $\mathscr{L}(\mathcal{KS}_{\mathcal{A}}) \subseteq \mathfrak{Z} \cap \mathscr{L}(\mathcal{A})$.*

The most permissive (strong) controller of $\mathcal{A}$ *is the controller $\mathcal{KS}_{\mathcal{A}}$ such that $\mathscr{L}(\mathcal{KS}'_{\mathcal{A}}) \subseteq \mathscr{L}(\mathcal{KS}_{\mathcal{A}})$ for all $\mathcal{KS}'_{\mathcal{A}}$ controllers of $\mathcal{A}$.*



(a) Automaton of Alan (b) Automaton of Betty (c) Automaton of Carol

Figure 3.5: The contract automata of Alan, Betty and Carol

In the next definition, we introduce a notion of strong liability, to single out the principals that are potentially responsible of the divergence from the expected behaviour.

Definition 36 (Strong Liability)**.** *Given the most permissive strong controller $\mathcal{KS}_{\mathcal{A}}$ of the contract automaton $\mathcal{A}$, let $(v\vec{a}w, \vec{q_0}) \rightarrow^* (\vec{a}w, \vec{q})$ be a run of both automata and let $\vec{q}$ be such that $(\vec{a}w, \vec{q}) \rightarrow (w, \vec{q'})$ is possible in $\mathcal{A}$ but not in $\mathcal{KS}_{\mathcal{A}}$.*

The principals $\Pi^i(\mathcal{A})$ such that $\vec{a}_{(i)} \neq \square, i \in 1 \dots r_{\mathcal{A}}$ are strongly liable *for $\vec{a}$ and belong to $SLiable(\mathcal{A}, \vec{a})$. Then, the set of* strongly liable *principals in $\mathcal{A}$ is $SLiable(\mathcal{A}) = \{i \mid \exists w.i \in SLiable(\mathcal{A}, w)\}$. Finally, let $TSLiable(\mathcal{A})$ denote the set of transitions of $\mathcal{A}$ that make principals strongly liable.*

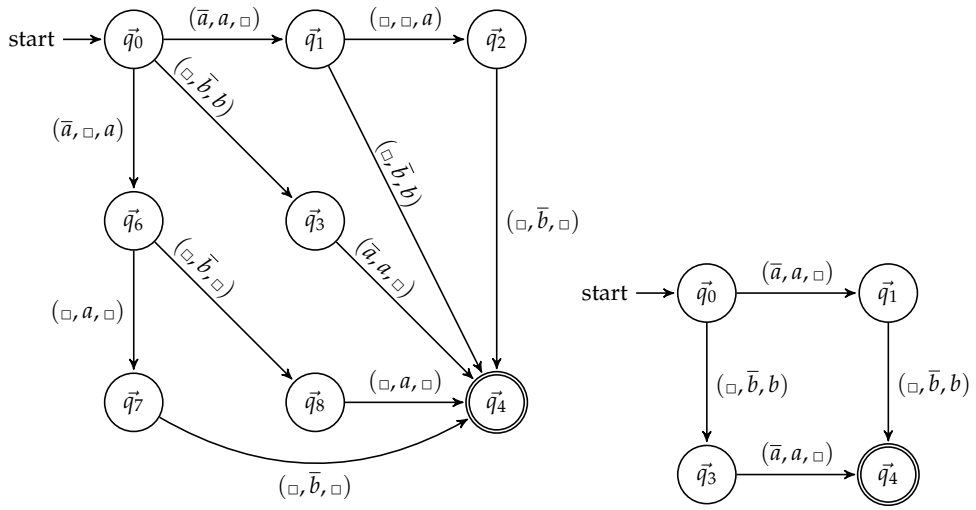
Example 13. *We illustrate the strong agreement property with the following simple example. Alan is willing to lend an apple, Betty offers a blueberry in order to exchange it with an apple, while Carol wants to eat either the apple or the blueberry. Let $\overline{a}$ and $\overline{b}$ denote respectively the actions of offering an apple or a blueberry and, dually, a and b denote the corresponding request actions.*

The principal automata of Alan, Betty, and Carol are in Figure 3.5, and their product is in in Figure 3.6a. Their most permissive strong controller is in Figure 3.6b; it is composed by states $\vec{q}_0$, $\vec{q}_1$, $\vec{q}_3$, and $\vec{q}_4$ and transitions $(\vec{q}_0, (\overline{a}, a, \square), \vec{q}_1)$, $(\vec{q}_3, (\overline{a}, a, \square), \vec{q}_4)$, $(\vec{q}_0, (\square, \overline{b}, b), \vec{q}_3)$, and $(\vec{q}_1, (\square, \overline{b}, b), \vec{q}_4)$.

The strongly liable indexes are 1 and 3, corresponding to Alan and Carol respectively; the transitions that make them liable are respectively $(\vec{q}_0,\ (\overline{a}, \square, a),\ \vec{q}_6)$ and $(\vec{q}_1,\ (\square, \square, a),\ \vec{q}_2)$. The former liable transition is a match that leads to non-matching transitions.

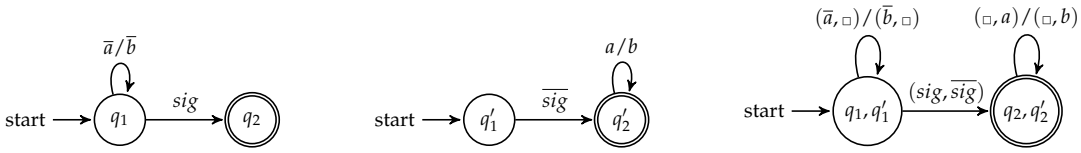
3.4 Weak Agreement

We will now consider a more liberal notion of agreement, where offers can be asynchronously fulfilled by matching requests, even though either of them occur beforehand. In other words, some actions can be taken on credit, assuming that in the future the obligations will be honoured. According to this notion, called here *weak agreement*,



(a) Alan⊗Betty⊗Carol (b) $\mathcal{KS}_{\text{Alan}\otimes\text{Betty}\otimes\text{Carol}}$

Figure 3.6: The product and most permissive strong controller of Example 13



(a) $\mathcal{A}_4$ (b) $\mathcal{A}_5$ (c) $\mathcal{A}_4 \otimes \mathcal{A}_5$

Figure 3.7: The contract automata of Example 17

computations well behave when all the requests are matched by offers, in spite of lack of synchronous agreement, in the sense of Section 3.2. This may lead to a circularity, as shown by the example below, because, e.g. one principal first requires something from the other and then is willing to fulfil the request of the other principal, who in turn behaves in the same way. This is a common scenario in contract composition, and variants of weak agreement have been studied using many different formal techniques, among which Process Algebras, Petri Nets, non-classical Logics, Event Structures [BZ09a, BCZ13, BCP13, BTZ12].

Example 14. *We recall the toy exchange scenario presented in Section 1.4.1. Suppose Alice and Bob want to share a bike and an airplane, but neither trusts the other. Before providing their offers they first ask for the complementary requests. As regular expressions:* $Alice = bike.\overline{airplane}$ *and* $Bob = airplane.\overline{bike}$*. The language of their composition is:* $\mathscr{L}(Alice \otimes Bob) =$

$$\{(\square, airplane)(bike, \overline{bike})(\overline{airplane}, \square), (bike, \square)(\overline{airplane}, airplane)(\square, \overline{bike})\}$$

In both possible traces the contracts fail in exchanging the bike or the airplane synchronously, hence $\mathscr{L}(Alice \otimes Bob) \cap \mathfrak{A} = \emptyset$ *and the composition does not admit agreement.*

The circularity in the requests/offers is solved by weakening the notion of agreement, allowing an action to be performed on credit and making sure that in the future the complementary action will occur, giving rise to a trace in weak agreement. As for agreement, we have an auxiliary definition.

Definition 37 (Weak Agreement)**.** *A trace accepted by a contract automaton of rank $n > 1$ is in* weak agreement *if it belongs to $\mathfrak{W} = \{w \in (\mathbb{L}^n)^* \mid w = \vec{a}_1 \dots \vec{a}_m, \exists$ a function $f : [1..m] \to [1..m]$ total and injective on the requests of w, and such that $f(i) = j$ only if $\vec{a}_i \bowtie \vec{a}_j\}$.*

Needless to say, a trace in agreement is also in weak agreement, so $\mathfrak{A}$ is a proper subset of $\mathfrak{W}$, as shown below.

Example 15. *Consider $\mathcal{A}_3$ in Figure 3.1, whose trace $(\overline{res}, \square)(sig, \overline{sig})(\square, res)$ is in $\mathfrak{W}$ but not in $\mathfrak{A}$, while $(\overline{res}, \square)(sig, \overline{sig})(\square, res)(\square, res) \notin \mathfrak{W}$.*

Definition 38 (Weak Safety)**.** *Let $\mathcal{A}$ be a contract automaton. Then*

- *if $\mathscr{L}(\mathcal{A}) \subseteq \mathfrak{W}$ then $\mathcal{A}$ is* weakly safe*, otherwise is* weakly unsafe*;*
- *if $\mathscr{L}(\mathcal{A}) \cap \mathfrak{W} \neq \emptyset$ then $\mathcal{A}$* admits weak agreement*.*

Example 16. *In Example 14 we have $\mathscr{L}(Alice \otimes Bob) \subset \mathfrak{W}$, hence the composition of Alice and Bob is weakly safe.*

The following theorem states the conditions under which weak agreement is preserved by our operations of contract composition.

Theorem 4 (Competitive Collaborative and Weak Agreement)**.** *Let $\mathcal{A}_1, \mathcal{A}_2$ be two contract automata, then if $\mathcal{A}_1, \mathcal{A}_2$ are*

1. *weakly safe then $\mathcal{A}_1 \otimes \mathcal{A}_2$ is weakly safe, $\mathcal{A}_1 \boxtimes \mathcal{A}_2$ admits weak agreement*
2. *non-collaborative and one or both unsafe, then $\mathcal{A}_1 \otimes \mathcal{A}_2, \mathcal{A}_1 \boxtimes \mathcal{A}_2$ are weakly unsafe*
3. *safe and non-competitive, then $\mathcal{A}_1 \boxtimes \mathcal{A}_2$ is weakly safe.*

Proof. Let req_a^w, of_a^w be the number of requests and offers of an action $a \in \mathbb{R} \cup \mathbb{O}$ in a trace w.

1. For $\otimes$: we will prove that in every trace of $\mathcal{A}_1 \otimes \mathcal{A}_2$, for each action the number of requests are less than or equal to the number of offers, and the thesis follows. By contradiction, assume that there exists a trace w in $\mathcal{A}_1 \otimes \mathcal{A}_2$ and an action a with $req_a^w > of_a^w$. Assume that w is obtained combining two traces w_1, w_2 of $\mathcal{A}_1$ and $\mathcal{A}_2$, that is each principal in each automaton performs the moves prescribed by its trace. Since both automata are weakly safe, we have $req_a^{w_1} \leq of_a^{w_1}$ and $req_a^{w_2} \leq of_a^{w_2}$ for all actions a.

   Independently of how many matches occur, in w we still have more requests than offers: $req_a^{w1} + req_a^{w2} - k \leq of_a^{w1} + of_a^{w2} - k$ where k are the new matches.

For $\boxtimes$ it suffices to take a trace w in $\mathcal{A}_1 \boxtimes \mathcal{A}_2$ obtained by combining two traces w_1, w_2 of respectively $\mathcal{A}_1$ and $\mathcal{A}_2$, where the match actions of both automata are maintained in w (the matches are performed by the same principals). In this case, the trace w will be present also in $\mathcal{A}_1 \otimes \mathcal{A}_2$, hence $w \in \mathfrak{W}$.

2. Without loss of generality assume that $\mathcal{A}_1$ is weakly unsafe, hence there exists an action a and a trace w_1 in $\mathcal{A}_1$ such that $req_a^{w_1} > of_a^{w_1}$. Since $\mathcal{A}_1$ and $\mathcal{A}_2$ are non-collaborative, in every trace w of $\mathcal{A}_1 \otimes \mathcal{A}_2$ or $\mathcal{A}_1 \boxtimes \mathcal{A}_2$ obtained by shuffling w_1 with an arbitrary w_2 in $\mathcal{A}_2$ we will have $req_a^w > of_a^w$.

3. from Theorem 3.5 $\mathcal{A}_1 \boxtimes \mathcal{A}_2$ is safe and since $\mathfrak{A} \subset \mathfrak{W}$ the thesis follows.

□

The following proposition helps the proof of Theorem 5.

Proposition 6. *Let* $WA(\mathfrak{W}) = \{w \in (\mathbb{R} \cup \mathbf{O} \cup \{\tau\})^* \mid \exists f : [1 \dots |w|] \to [1 \dots |w|]$ *injective and such that* $f(i) = j$ *only if* $w_{(i)} = co(w_{(j)})$*, total on the requests of* $w\}$.
Then, $Obs(w) \in WA(\mathfrak{W})$ *implies* $w \in \mathfrak{W}$.

Proof. Let $\sigma = Obs(w) \in WA(\mathfrak{W})$, and let f be a function that certifies that $\sigma \in WA(\mathfrak{W})$, i.e. that all the requests in w are fulfilled. Then f certifies $w \in \mathfrak{W}$. □

The example below shows that weak agreement is not a context-free notion, in language theoretical sense; rather we will prove it context-sensitive. Therefore, we cannot define the most permissive controller for weak agreement in terms of contract automata, because they are finite state automata.

Example 17. *Let* $\mathcal{A}_4$, $\mathcal{A}_5$ *and* $\mathcal{A}_4 \otimes \mathcal{A}_5$ *be the automata in Figure 3.7, then we have that* $L = \mathfrak{W} \cap \mathscr{L}(\mathcal{A}_4 \otimes \mathcal{A}_5) \neq \emptyset$ *is not context-free. Consider the following regular language*

$$L' = \{(\overline{a}, \square)^*(\overline{b}, \square)^*(sig, \overline{sig})(\square, a)^*(\square, b)^*\}$$

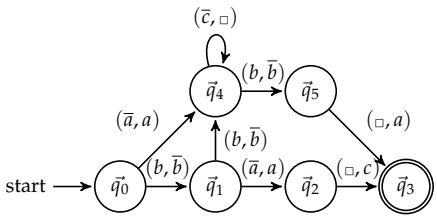
We have that

$$L \cap L' = \{(\overline{a}, \square)^{n_1}(\overline{b}, \square)^{m_1}(sig, \overline{sig})(\square, a)^{n_2}(\square, b)^{m_2} \mid n_1 \geq n_2 \geq 0, m_1 \geq m_2 \geq 0\}$$

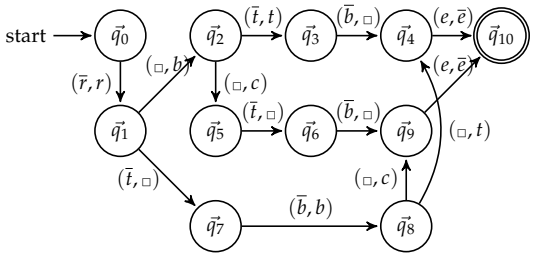
is not context-free (by pumping lemma), and since L' *is regular,* L *is not context-free.*

Theorem 5 (Weak Agreement Context-Sensitive)**.** $\mathfrak{W}$ *is a context-sensitive language, but not context-free. Word decision can be done in* $O(n^2)$ *time and* $O(n)$ *space.*

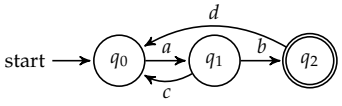
Proof. Example 17 shows that the property is not context-free. For proving that $\mathfrak{W}$ is context-sensitive we now outline a Linear Bounded Automata (LBA) [Kur64] that decides whether a trace w belongs to $\mathfrak{W}$, giving us time and space complexity for the membership problem. Roughly, a LBA is a Turing machine with a tape, linearly bounded by the size of the input. Since we have an infinite alphabet due to the (unbounded) rank



(a) The product of two contract automata of Examples 18 and 19



(b) The booking service of Example 18



(c) The principal contract automaton whose flow constraints generate many traces, as discussed at the end of Example 18

Figure 3.8: The contract automata discussed in Examples 18 and 19

of vector $\vec{a}$, we compute $Obs(w)$ and decide if $Obs(w) \in WA(\mathfrak{W})$. By Proposition 6 we obtain the thesis. Below is the scheme of the algorithm:

for $i = 0; i < length(w); i++$ **do**
  if $w_i \in \mathbb{R}$ **then**
    for $j = 0; j < length(w); j++$ **do**
      if $w_j = co(w_i)$ **then**
        $w_j \leftarrow \#$
        break
      else
        if $j = length(w) - 1$ **then return** false
        end if
      end if
    end for
  end if
end for
return true

The length of the tape equals the length of w, so the algorithm is $O(n)$ space, while it is $O(n^2)$ time, because of the two nested **for** cycles. □

In general, it is undecidable checking whether a regular language L is included in a context-sensitive one, as well as checking emptiness of the intersection of a regular language with a context-sensitive one. However in our case these two problems are decidable: we will introduce an effective procedure to check whether a contract automaton $\mathcal{A}$ is weakly safe, or whether it admits weak agreement. The technique we propose amounts to find optimal solutions to network flow problems [HKLW10], and

will be used also for detecting weak liability.

As an additional comment, note that the membership problem is polynomial in time for mildly context-sensitive languages [JSW90], but it is PSPACE-complete for arbitrary ones. In the first case, checking membership can be done in polynomial time through *two way deterministic pushdown automata* [GHI67], that have a read-only input tape readable backwards and forwards. It turns out that $\mathfrak{W}$ is mildly context-sensitive, and checking whether $w \in \mathfrak{W}$ can be intuitively done by repeating what follows for all the actions occurring in w. Select an action α; scroll the input; and push all the requests on α on the stack; scroll again the input and pop a request, if any, when a corresponding offer is found. If at the end the stack is empty the trace w is in $\mathfrak{W}$.

The following property characterize the language of weak agreement as the shuffling of context-free languages.

Property 4. *The language $\mathfrak{W}$ is the shuffle of a set of context-free languages.*

Proof. First consider only the language $WA(\mathfrak{W})$ over an alphabet consisting of the single action a, i.e. $\mathbb{R} = \{a\}, \mathbb{O} = \{\overline{a}\}$, and without τ. Note that if $Obs(w) \in WA(\mathfrak{W})$, also the string obtained by adding in or removing τ from σ belongs to $WA(\mathfrak{W})$. In this case, $WA(\mathfrak{W})$ is generated by the context-free productions $G ::= aG\overline{a}G \mid \overline{a}GaG \mid \overline{a}G \mid G\overline{a} \mid \varepsilon$. Consider now the case of many actions, and the context-free languages associated with each of them. It is easy to show that the overall language for $WA(\mathfrak{W})$ is obtained by shuffling all these context-free languages. Additionally, the shuffle of context-free languages is context-sensitive [Gis81]. □

3.4.1 Flow Optimization Problems for Weak Agreement

Before presenting our decision procedure we recall the notation introduced in Section 1.8, and we adapt it to contract automata. Assume as given a contract automaton $\mathcal{A}$, with a single final state $\vec{q}_f \neq \vec{q}_0$. If this is not the case, one simply adds artificial, dummy transitions from all the original final states to the new single final state. Clearly, if the modified contract automaton admits weak agreement, also the original one does — and the two will have the same liable principals. We assume that all states are reachable from $\vec{q}_0$ and so is $\vec{q}_f$ from each of them. In addition, we enumerate the requests of $\mathcal{A}$, i.e. $A^r = \{a^i \mid i \in I_l = \{1, 2, \ldots, l\}\}$, as well as its transitions $T = \{t_1, \ldots, t_n\}$. Also, let $FS(\vec{q}) = \{(\vec{q}, \vec{a}, \vec{q}') \mid (\vec{q}, \vec{a}, \vec{q}') \in T\}$ be the *forward star* of a state $\vec{q}$, and let $BS(\vec{q}) = \{(\vec{q}', \vec{a}, \vec{q}) \mid (\vec{q}', \vec{a}, \vec{q}) \in T\}$ be its *backward star*. For each transition t_i we introduce the *flow variables* $x_{t_i} \in \mathbb{N}$, and $z^{\vec{q}}_{t_i} \in \mathbb{R}$ where $\vec{q} \in Q, \vec{q} \neq \vec{q}_0$.

We are ready to define the set $F_{\vec{s},\vec{d}}$ of *flow constraints*, an element of which $\vec{x} = (x_{t_1}, \ldots, x_{t_n}) \in F_{\vec{s},\vec{d}}$ defines traces from the source state $\vec{s}$ to the target state $\vec{d}$. The intuition is that each variable x_{t_i} represents how many times the transition t_i is traversed in the traces defined by $\vec{x}$. Hereafter, we will abbreviate $F_{\vec{q}_0,\vec{q}_f}$ as F_x, and we identify a transition through its source and target states.

An example follows.

Example 18. *Figure 3.8b shows a simple service of booking, which is the composition of a client and a hotel contracts.*

The contract of the client requires to book a room (r), including breakfast (b) and a transport service, by car (c) or taxi (t); finally it sends a signal of termination ($\overline{e}$). The contract of the client is then:

$$C = r.b.(c + t).\overline{e}$$

The hotel offers a room, breakfast and taxi. Its contract is:

$$H = \overline{r}.\overline{t}.\overline{b}.e$$

Four traces accepted by the automaton $H \otimes C$ are:

$$w_1 = (\overline{r}, r)(\square, b)(\overline{t}, t)(\overline{b}, \square)(e, \overline{e})$$
$$w_2 = (\overline{r}, r)(\square, b)(\square, c)(\overline{t}, \square)(\overline{b}, \square)(e, \overline{e})$$
$$w_3 = (\overline{r}, r)(\overline{t}, \square)(\overline{b}, b)(\square, t)(e, \overline{e})$$
$$w_4 = (\overline{r}, r)(\overline{t}, \square)(\overline{b}, b)(\square, c)(e, \overline{e})$$

We now detail the flows associated with each trace giving the set of variables with value 1, all the others having value 0, because there are no loops. The associated flows are:

$$w_1 : \{x_{\vec{q}_0,\vec{q}_1}, x_{\vec{q}_1,\vec{q}_2}, x_{\vec{q}_2,\vec{q}_3}, x_{\vec{q}_3,\vec{q}_4}, x_{\vec{q}_4,\vec{q}_{10}}\}$$
$$w_2 : \{x_{\vec{q}_0,\vec{q}_1}, x_{\vec{q}_1,\vec{q}_2}, x_{\vec{q}_2,\vec{q}_5}, x_{\vec{q}_5,\vec{q}_6}, x_{\vec{q}_6,\vec{q}_9}, x_{\vec{q}_9,\vec{q}_{10}}\}$$
$$w_3 : \{x_{\vec{q}_0,\vec{q}_1}, x_{\vec{q}_1,\vec{q}_7}, x_{\vec{q}_7,\vec{q}_8}, x_{\vec{q}_8,\vec{q}_4}, x_{\vec{q}_4,\vec{q}_{10}}\}$$
$$w_4 : \{x_{\vec{q}_0,\vec{q}_1}, x_{\vec{q}_1,\vec{q}_7}, x_{\vec{q}_7,\vec{q}_8}, x_{\vec{q}_8,\vec{q}_9}, x_{\vec{q}_9,\vec{q}_{10}}\}$$

Note that a flow $\vec{x}$ may represent many traces that have the same balance of requests and offers for each action occurring therein. For example, in the contract automatonof Figure 3.8c, the same flow $x_{q_0,q_1} = 3, x_{q_1,q_2} = 2, x_{q_2,q_0} = x_{q_1,q_0} = 1$ represents both $w_1 = acabdab$ and $w_2 = abdacab$. The following auxiliary definition introduces a notation for flow constraints. It is beneficial in the statements of Theorems 6, 7 and 8 below.

Definition 39 (Flow Problem for Weak Agreement)**.** *Given a source state $\vec{s}$ and a destination state $\vec{d}$, the set of* flow constraints $F_{\vec{s},\vec{d}}$ *from $\vec{s}$ to $\vec{d}$ is defined as:*

$$F_{\vec{s},\vec{d}} = \{(x_{t_1}, \ldots, x_{t_n}) \mid \forall \vec{q} : (\sum_{t_i \in BS(\vec{q})} x_{t_i} - \sum_{t_i \in FS(\vec{q})} x_{t_i}) = \begin{cases} -1 & \text{if } \vec{q} = \vec{s} \\ 0 & \text{if } \vec{q} \neq \vec{s}, \vec{d} \\ 1 & \text{if } \vec{q} = \vec{d} \end{cases}$$

$$\forall \vec{q} \neq \vec{s}, t_i. \quad 0 \leq z^{\vec{q}}_{t_i} \leq x_{t_i},$$

$$\forall \vec{q} \neq \vec{s}, \forall \vec{q'} : (\sum_{t_i \in BS(\vec{q'})} z^{\vec{q}}_{t_i} - \sum_{t_i \in FS(\vec{q'})} z^{\vec{q}}_{t_i}) = \begin{cases} -p^{\vec{q}} & \text{if } \vec{q'} = \vec{s} \\ 0 & \text{if } \vec{q'} \neq \vec{s}, \vec{q} \\ p^{\vec{q}} & \text{if } \vec{q'} = \vec{q} \end{cases}$$

$$\text{where} \quad p^{\vec{q}} = \begin{cases} 1 & \text{if } \sum_{t_i \in FS(\vec{q})} x_{t_i} > 0 \\ 0 & \text{otherwise} \end{cases} \}$$

In the definition above, the variables $z^{\vec{q}}_{t_i}$ represent $|Q|-1$ auxiliary flows and make sure that a flow $\vec{x}$ represents valid runs only, that is they guarantee that there are no disconnected cycles with a positive flow. A more detailed discussion is in Example 19 below. Note that the values of $z^{\vec{q}}_{t_i}$ are *not* integers, and so we are defining Mixed Integer Linear Programming problems that have efficient solutions [HKLW10].

We eventually define a set of variables $a^i_{t_j}$ for each action and each transition, that take the value -1 for requests, 1 for offers, and 0 otherwise; they help counting the difference between offers and requests of an action in a flow.

$$\forall t_j = (\vec{q}, \vec{a}, \vec{q'}) \in T, \forall i \in I_l : \quad a^i_{t_j} = \begin{cases} 1 & \text{if } Obs(\vec{a}) = \overline{a^i} \\ -1 & \text{if } Obs(\vec{a}) = a^i \\ 0 & \textit{otherwise} \end{cases}$$

Example 19. *Figure 3.8a depicts the contract $A \otimes B$, where*

$$A = \overline{a}.\overline{c}^*.b + b.(b.\overline{c}^*.b + \overline{a}) \qquad B = a.\overline{b}.a + \overline{b}.(\overline{b}.\overline{b}.a + a.c)$$

To check whether there exists a run recognising a trace w with less or equal requests than offers (for each action) we solve $\sum_{t_j} a^i_{t_j} x_{t_j} \geq 0$, for $\vec{x} \in F_x$.

We illustrate how the auxiliary variables $z^{\vec{q}}_{t_i}$ ensure that the considered solutions represent valid runs. Consider the following assignment to $\vec{x}$: $x_{\vec{q}_0,\vec{q}_1} = x_{\vec{q}_1,\vec{q}_2} = x_{\vec{q}_2,\vec{q}_3} = 1, x_{\vec{q}_4,\vec{q}_4} \geq 1$, and null everywhere else. It does not represent valid runs, because the transition $(\vec{q}_4, (\overline{c}, \square), \vec{q}_4)$ cannot be fired in a run that only takes transitions with non-null values in $\vec{x}$. However, the constraints on the flow $\vec{x}$ are satisfied (e.g. we have $\sum_{t_j \in FS(\vec{q}_4)} x_{t_j} = \sum_{t_j \in BS(\vec{q}_4)} x_{t_j}$). Now the constraints on the auxiliary $z^{\vec{q}}_{t_i}$ play their role, checking if a node is reachable from the initial state on a run defined by $\vec{x}$. The assignment above is not valid since for $z^{\vec{q}_4}$ we have:

$$0 \leq z^{\vec{q}_4}_{(\vec{q}_0,\vec{q}_4)} \leq x_{(\vec{q}_0,\vec{q}_4)} = 0$$

$$0 \leq z^{\vec{q}_4}_{(\vec{q}_1,\vec{q}_4)} \leq x_{(\vec{q}_1,\vec{q}_4)} = 0$$

$$0 \leq z^{\vec{q}_4}_{(\vec{q}_4,\vec{q}_5)} \leq x_{(\vec{q}_4,\vec{q}_5)} = 0$$

Hence $\sum_{t_j \in BS(\vec{q}_4)} z^{\vec{q}_4}_{t_j} = z^{\vec{q}_4}_{(\vec{q}_4,\vec{q}_4)}$, $\sum_{t_j \in FS(\vec{q}_4)} z^{\vec{q}_4}_{t_j} = z^{\vec{q}_4}_{(\vec{q}_4,\vec{q}_4)}$ and we have:

$$\sum_{t_j \in BS(\vec{q}_4)} z^{\vec{q}_4}_{t_j} - \sum_{t_j \in FS(\vec{q}_4)} z^{\vec{q}_4}_{t_j} = 0 \neq 1 = p^{\vec{q}_4}$$

Finally, note in passing that there are no valid flows $\vec{x} \in F_x$ for this problem.

More importantly, note that the auxiliary variables $z^{\vec{q}}_{t_i}$ are not required to have integer values, which is immaterial for checking that those solutions represent valid runs, but makes finding them much easier.

The following result is auxiliary to Theorems 6, 7 and 8 below.

Lemma 2 (Flow and Traces)**.** *Let $\mathcal{A}$ be a contract automaton such that $\vec{x} \in F_x$, then there exists a run $(w, \vec{q}_0) \rightarrow^* (\varepsilon, \vec{q}_f)$ that passes through each $t_j \in T$ exactly x_{t_j} times.*

Proof. We outline an algorithm that visits all the transitions t_j with $x_{t_j} > 0$, starting from $\vec{q}_f$ and proceeding backwards to $\vec{q}_0$.

We use auxiliary variables $\overline{x}_{t_j}, t_j \in T$, initialised to zero, for storing how many times we have passed through a transition t_j. At each iteration the algorithm selects non deterministically a transition $\hat{t}$ in the backward star of the selected node such that $x_{\hat{t}} - \overline{x}_{\hat{t}} > 0$, and increases by one unit the variable $\overline{x}_{\hat{t}}$ for the selected $\hat{t}$. The next node will be the starting state of $\hat{t}$. The algorithm terminates when for all the transitions t_j in the backward star we have $x_{t_j} - \overline{x}_{t_j} = 0$.

We prove that the algorithm terminates and constructs a trace that passes through each t_j exactly x_{t_j} times, and the last transition considered leaves the initial state. For the first step we have $\sum_{t_j \in BS(\vec{q}_f)} x_{t_j} - \sum_{t_j \in FS(\vec{q}_f)} x_{t_j} = 1$ hence there exists at least one $t_i \in BS(\vec{q}_f)$ such that $x_{t_i} > 0$ (and $\overline{x}_{t_i} = 0$).

Pick up one of these transitions, say t_i, and assign it to the iteration variable $\hat{t}$. Two cases may arise, depending on the source of $\hat{t}$:

1. the source of $\hat{t}$ is $\vec{q} \neq \vec{q}_0$: we have $\sum_{t_j \in BS(\vec{q})} x_{t_j} - \sum_{t_j \in FS(\vec{q})} x_{t_j} \geq 0$ and we know that $\sum_{t_j \in FS(\vec{q})} x_{t_j} > 0$, because $\hat{t} \in FS(\vec{q})$ and $x_{\hat{t}} > 0$, hence $\sum_{t_j \in BS(\vec{q})} x_{t_j} > 0$.

   We now show that there is at least one $t \in BS(\vec{q})$ such that $(x_t - \overline{x}_t) > 0$. By contradiction, assume $\sum_{t_j \in BS(\vec{q})} x_{t_j} - \sum_{t_j \in BS(\vec{q})} \overline{x}_{t_j} = 0$. We distinguish two cases:

   - $\vec{q} = \vec{q}_f$: we have $\sum_{t_j \in FS(\vec{q})} \overline{x}_{t_j} = \sum_{t_j \in BS(\vec{q})} \overline{x}_{t_j}$, since at every iteration we increase of one unit the value of $\overline{x}_{t_i}$ for $\hat{t}$ and we are proceeding backwards starting from $\vec{q}_f$ (the flow variable of a loop belongs to both backward and forward star). Since $\sum_{t_j \in BS(\vec{q})} x_{t_j} > \sum_{t_j \in FS(\vec{q})} x_{t_j}$, we have $\sum_{t_j \in FS(\vec{q})} x_{t_j} - \sum_{t_j \in FS(\vec{q})} \overline{x}_{t_j} < 0$. Contradiction, since by definition the value $\overline{x}_{t_j}$ for a transition t_j will never be greater then the corresponding value x_{t_j}.
   - $\vec{q} \neq \vec{q}_f$: we have $\sum_{t_j \in FS(\vec{q})} \overline{x}_{t_j} > \sum_{t_j \in BS(\vec{q})} \overline{x}_{t_j}$. Since $\sum_{t_j \in BS(\vec{q})} x_{t_j} = \sum_{t_j \in FS(\vec{q})} x_{t_j}$, we have $\sum_{t_j \in FS(\vec{q})} x_{t_j} - \sum_{t_j \in FS(\vec{q})} \overline{x}_{t_j} < 0$ obtaining a contradiction as above.

   Then, we iterate the algorithm taking the above t as $\hat{t}$.

2. the source of t_i is $\vec{q}_0$: we have $\sum_{t_j \in BS(\vec{q}_0)} x_{t_j} - \sum_{t_j \in FS(\vec{q}_0)} x_{t_j} = -1$.
   Let $k_1 = \sum_{t_j \in FS(\vec{q}_0)} x_{t_j} - \sum_{t_j \in FS(\vec{q}_0)} \overline{x}_{t_j}$, $k_2 = \sum_{t_j \in BS(\vec{q}_0)} x_{t_j} - \sum_{t_j \in BS(\vec{q}_0)} \overline{x}_{t_j}$, and note that since we are proceeding backwards starting from $\vec{q}_f$ it must be that $\sum_{t_j \in FS(\vec{q}_0)} \overline{x}_{t_j} = 1 + \sum_{t_j \in BS(\vec{q}_0)} \overline{x}_{t_j}$. Hence, from the previous equations it must be that $k_2 - k_1 = 0$. We have that:

   - if $k_1 = 0$, we have $k_2 = 0$ and the algorithm terminates;
   - if $k_1 > 0$, we have $k_2 > 0$ and the algorithm continues by selecting a transition $\hat{t} \in BS(\vec{q}_0)$ such that $x_{\hat{t}} - \overline{x}_{\hat{t}} = 0$.

Since at every iteration we increase the value $\overline{x}_{\hat{t}}$, the constraints on F_x guarantee that the algorithm will eventually terminate. Moreover there exists an execution of the algorithm that traverses all the possible cycles of the trace induced by $\vec{x}$. Hence we have a trace from $\vec{q}_0$ to $\vec{q}_f$ that passes through each transition t_j visited by the algorithm exactly x_{t_j} times.

It remains to prove that for all the transitions t_j not visited by the algorithm we have $x_{t_j} = 0$. By contradiction assume that there exists a transition $t_i = (\vec{q}_s, \vec{a}, \vec{q}_d)$ with $x_{t_i} - \overline{x}_{t_i} > 0$ for all the possible executions of the algorithm.

This is possible only if $\vec{q}_d$ it is not connected to $\vec{q}_f$ by the flow $\vec{x}$. Moreover in this case by the flow constraints on $\vec{x}$ it follows that $\vec{q}_s$ is not reachable from $\vec{q}_0$ by the flow $\vec{x}$, i.e. t_i is not part of the trace induced by $\vec{x}$. Then there must exist a cycle $C = \{t_{c1}, \ldots, t_{cm}\}$ with $t_i \in C$ and disconnected from $\vec{q}_0$ and $\vec{q}_f$ with positive flow. Let Q_C be the set of nodes having ingoing or outgoing transitions in C. The constraints $\sum_{t \in BS(\vec{q})} x_t - \sum_{t \in FS(\vec{q})} x_t = 0$ are satisfied for all $\vec{q} \in C$.

We show that C will eventually violate the constraints defined by the variables $z_{t_j}^{\vec{q}_s}$. We have:

$$\forall \vec{q'} \in Q : \sum_{t_j \in BS(\vec{q'})} z_{t_j}^{\vec{q}_s} - \sum_{t_j \in FS(\vec{q'})} z_{t_j}^{\vec{q}_s} = \begin{cases} -p^{\vec{q}_s} & \text{if } \vec{q'} = \vec{q}_0 \\ 0 & \text{if } \vec{q'} \neq \vec{q}_0, \vec{q}_s \\ p^{\vec{q}_s} & \text{if } \vec{q'} = \vec{q}_s \end{cases}$$

$$\forall t_j \in T.\ z_{t_j}^{\vec{q}_s} \in \mathbb{R}, \quad 0 \leq z_{t_j}^{\vec{q}_s} \leq x_{t_j}$$

We have $\sum_{t_j \in FS(\vec{q}_s)} x_{t_j} > 0$ and $p^{\vec{q}_s} = 1$, hence $\sum_{t_j \in BS(\vec{q}_s)} z_{t_j}^{\vec{q}_s} - \sum_{t_j \in FS(\vec{q}_s)} z_{t_j}^{\vec{q}_s} = 1$ and for all $\vec{q} \in Q_C, \vec{q} \neq \vec{q}_s : \sum_{t_j \in BS(\vec{q})} z_{t_j}^{\vec{q}_s} - \sum_{t_j \in FS(\vec{q})} z_{t_j}^{\vec{q}_s} = 0$. Note that is not possible to satisfy these constraints since for all $t \in C$, x_t are all equal and positive and $0 \leq z_t^{\vec{q}_s} \leq x_t$. □

The main results of this section follow.

Theorem 6 (Flow Problem for Weak Safety)**.**
Let $\vec{v}$ be a binary vector. Then a contract automaton $\mathcal{A}$ is weakly safe *if and only if min $\gamma \geq 0$ where:*

$$\sum_{i \in I_l} v_i \sum_{t_j \in T} a_{t_j}^i x_{t_j} \leq \gamma \quad \sum_{i \in I_l} v_i = 1 \quad \forall i \in I_l.\ v_i \in \{0,1\} \quad (x_{t_1} \ldots x_{t_n}) \in F_x \quad \gamma \in \mathbb{R}$$

Proof. ($\Rightarrow$) By contradiction assume that $\min\ \gamma < 0$. Hence there exists an action a^j such that $v_j = 1, \forall i \in I_l, i \neq j . v_i = 0$ and $\gamma = \sum_{t_j \in T} a_{t_j}^j x_{t_j} < 0$. By Lemma 2 we know that $\vec{x}$ builds a trace recognising $w \in \mathscr{L}(\mathcal{A})$, and the number of offers for a^j in w are less than the corresponding number of requests since $\sum_{t_j \in T} a_{t_j}^j x_{t_j} < 0$, hence $w \notin \mathfrak{W}$.

($\Leftarrow$) By contradiction there exists $w \in \mathscr{L}(\mathcal{A}) \setminus \mathfrak{W}$. Hence there exists an action a^j that occurs in w fewer times as an offer than as a request. Let $\vec{x}$ be the flow induced in the obvious way by the trace w, counting the number of times each transition occurs in the path accepting w. We have $\sum_{t_j \in T} a_{t_j}^j x_{t_j} < 0$, hence it must be $\min\ \gamma < 0$. □

The minimum value of γ selects the trace and the action a for which the difference between the number of offers and requests is the minimal achievable from $\mathcal{A}$. If this difference is non-negative, there will always be enough offers matching the requests, and so $\mathcal{A}$ will never generate a trace not in $\mathfrak{W}$. In other words, $\mathcal{A}$ is *weakly safe*, otherwise it is not.

Example 20. *Consider again Example 18 and let $a^1 = r$, $a^2 = b$, $a^3 = t$, $a^4 = c$, $a^5 = e$. If $v_1 = 1$, for each flow $\vec{x} \in F_x$, we have that $\sum_{t_j} a^1_{t_j} x_{t_j} = 0$ (for $i \neq 1$, we have $v_i = 0$). This means that the request of a room is always satisfied. Similarly for breakfast and the termination signal e. If $v_3 = 1$, for the flow representing the traces w_1, w_3 we have $\sum_{t_j} a^3_{t_j} x_{t_j} = 0$, while for the flow representing the traces w_2, w_4 the result is 1. The requests are satisfied also in this case. Instead, when $v_4 = 1$, for the flow representing the traces w_1, w_4 we have $\sum_{t_j} a^4_{t_j} x_{t_j} = 0$, but for the flow representing w_2, w_3, the result is -1. Hence $\min \gamma = -1$, and the contract automaton $H \otimes C$ is not* weakly safe, *indeed we have $w_2, w_3 \notin \mathfrak{W}$.*

In a similar way, we can check if a contract automaton offers a trace in weak agreement.

Theorem 7 (Flow Problem for Weak Agreement)**.** *The contract automaton $\mathcal{A}$ admits weak agreement if and only if $\max \gamma \geq 0$ where*

$$\forall i \in I_l. \sum_{t_j \in T} a^i_{t_j} \, x_{t_j} \geq \gamma \quad (x_{t_1} \dots x_{t_n}) \in F_x \quad \gamma \in \mathbb{R}$$

Proof. ($\Rightarrow$) Let w be a trace in weak agreement, and let $\vec{x}$ be the flow induced by w. Then by construction $\forall i \in I_l. \sum_{t_j \in T} a^i_{t_j} x_{t_j} \geq 0$, hence $\max \gamma \geq 0$.

($\Leftarrow$) Follows from Lemma 2 and the hypothesis. □

The maximum value of γ in Theorem 7 selects the trace w that maximises the least difference between offers and requests of an action in w. If this value is non-negative, then there exists a trace w such that for all the actions in it, the number of requests is less or equal than the number of offers. In this case, $\mathcal{A}$ admits weak agreement; otherwise it does not.

Example 21. *In Example 18, $\max \gamma = -1$ for the flows representing the traces w_2, w_3 and $\max \gamma = 0$ for those of the traces w_1, w_4, that will be part of the solution and are indeed in weak agreement. Consequently, $H \otimes C$ admits weak agreement.*

We now define the *weakly liable* principals: those who perform the first transition t of a run such that after t it is not possible any more to obtain a trace in $\mathfrak{W}$, i.e. leading to traces $w \in \mathscr{L}(\mathcal{A}) \setminus \mathfrak{W}$ that cannot be extended to $ww' \in \mathscr{L}(\mathcal{A}) \cap \mathfrak{W}$.

Definition 40 (Weak Liability)**.** *Let $\mathcal{A}$ be a contract automaton and let $w = w_1 \vec{a} w_2$ such that $w \in \mathscr{L}(\mathcal{A}) \setminus \mathfrak{W}, \forall w'. ww' \notin \mathscr{L}(\mathcal{A}) \cap \mathfrak{W}, \forall w_3. w_1 \vec{a} w_3 \notin \mathscr{L}(\mathcal{A}) \cap \mathfrak{W}$ and $\exists w_4. w_1 w_4 \in \mathscr{L}(\mathcal{A}) \cap \mathfrak{W}$.*

The principals $\Pi^i(\mathcal{A})$ such that $\vec{a}_{(i)} \neq \square$ are weakly liable *and form the set $WLiable(\mathcal{A}, w_1 \vec{a})$.*

Let $WLiable(\mathcal{A}) = \{i \mid \exists w \text{ such that } i \in WLiable(\mathcal{A}, w)\}$ be the set of all potentially weakly liable *principals in $\mathcal{A}$.*

For computing the set $WLiable(\mathcal{A})$ we optimise a network flow problem for a transition $\bar{t}$ to check if there exists a trace w in which $\bar{t}$ reveals some weakly liable principals. By solving this problem for all transitions we obtain the set $WLiable(\mathcal{A})$.

Theorem 8 (Flow Problem and Weak Liability)**.** *The principal $\Pi^i(\mathcal{A})$ of a contract automaton $\mathcal{A}$ is* weakly liable *if and only if there exists a transition $\bar{t} = (\vec{q}_s, \vec{a}, \vec{q}_d)$, $\vec{a}_{(i)} \neq \square$ such that $\gamma_{\bar{t}} < 0$, where*

$$\gamma_{\bar{t}} = min\ \{g(\vec{x}) \mid \vec{x} \in F_{\vec{q}_0,\vec{q}_s},\ \vec{y} \in F_{\vec{q}_s,\vec{q}_f},\ \forall i \in I_l.\ \sum_{t_j \in T} a^i_{t_j}(x_{t_j} + y_{t_j}) \geq 0\}$$

$$g(\vec{x}) = max\ \{\gamma \mid \vec{u} \in F_{\vec{q}_d,\vec{q}_f},\ \forall i \in I_l.\ \sum_{t_j \in T} a^i_{t_j}(x_{t_j} + u_{t_j}) + a^i_{\bar{t}} \geq \gamma, \gamma \in \mathbb{R}\}$$

Proof. ($\Rightarrow$) By hypothesis $\exists w_1$ such that $\forall w_3.\, w_1\vec{a}w_3 \in \mathscr{L}(\mathcal{A}) \setminus \mathfrak{W}$ and $\exists w_2.\, w_1w_2 \in \mathscr{L}(\mathcal{A}) \cap \mathfrak{W}$. Let $\bar{t} = (\vec{q}_s, \vec{a}, \vec{q}_d)$ be the transition such that $(w_1\vec{a}, \vec{q}_0) \rightarrow^* (\vec{a}, \vec{q}_s) \rightarrow (\varepsilon, \vec{q}_d)$, i.e. the principal i in $\vec{a}$ is weakly liable. We show that $\gamma_{\bar{t}} < 0$.

Let w_1 from $\vec{q}_0$ to $\vec{q}_s$ induce the flow $\vec{x}$, while w_2 from $\vec{q}_s$ to $\vec{q}_f$ induce $\vec{y}$. Since w_1w_2 is in weak agreement, $\forall i \in I_l.\, \sum_{t_j \in T} a^i_{t_j}(x_{t_j} + y_{t_j}) \geq 0$.

Since by hypothesis the i-th principal is liable, the flow $\vec{x}$ corresponding to the trace w_1 is such that $g(\vec{x}) < 0$. Otherwise if $g(\vec{x}) \geq 0$ we can choose a trace, say, w_3 such that $w_1\vec{a}w_3 \in \mathscr{L}(\mathcal{A}) \cap \mathfrak{W}$, obtaining a contradiction. Therefore, $\gamma_{\bar{t}} \leq g(\vec{x}) < 0$.

($\Leftarrow$) by hypothesis $\gamma_{\bar{t}} < 0$ and by Lemma 2 $\vec{x}$ corresponds to a run w from the initial state to $\vec{q}_s$ such that (by hypothesis again) $\forall w_3.w_1\vec{a}w_3 \notin \mathscr{L}(\mathcal{A}) \cap \mathfrak{W}$ and $\exists w_4.w_1w_4 \in \mathscr{L}(\mathcal{A}) \cap \mathfrak{W}$, that is $\bar{t}$ is a weakly liable transition.

□

Figure 3.9 might help understanding how the flows $\vec{x}, \vec{y}$ (and $\vec{u}$) and the transition $\bar{t}$ are composed to obtain a path from the initial to the final state. Intuitively, the flow defined above can be seen as split into three parts: the flow $\vec{x}$ from $\vec{q}_0$ to $\vec{q}_s$, the flow $\vec{y}$ from $\vec{q}_s$ to $\vec{q}_f$, and the flow $\vec{u}$ from $\vec{q}_d$ to $\vec{q}_f$, computed through the function g.

This function takes as input the flow $\vec{x}$ and selects a flow $\vec{u}$ such that, by concatenating $\vec{x}$ and $\vec{u}$ through $\bar{t}$, we obtain a trace w, where the least difference between offers and requests is maximised for an action in w. Using the same argument of Theorem 7, if the value computed is negative, then there not exists a flow $\vec{u}$ that composed with $\vec{x}$ selects traces in weak agreement.

Finally $\gamma_{\bar{t}}$ yields the minimal result of $g(\vec{x})$, provided that there exists a flow $\vec{y}$, that combined with $\vec{x}$ represents only traces in weak agreement. If $\gamma_{\bar{t}} < 0$ then the transition $\bar{t}$ identifies some *weakly liable* principals. Indeed the flow $\vec{x}$ represents the traces w such that (1) $\exists w_1$, represented by $\vec{y}$, with $ww_1 \in \mathscr{L}(\mathcal{A}) \cap \mathfrak{W}$ and (2) $\forall w_2$, represented by $\vec{u}$, with $w\vec{a}w_2 \in \mathscr{L}(\mathcal{A}) \setminus \mathfrak{W}$. Note that if a flow $\vec{x}$ reveals some weakly liable principals, the minimisation carried on by $\gamma_{\bar{t}}$ guarantees that the relevant transition $\bar{t}$ is found. Finding the weakly liable principals is a hard task, and belongs to the family of bilevel problems [Bar06]. Basically, these problems contain two optimization problems, one embedded in the other, and finding optimal solutions to them is still a hot research topic.

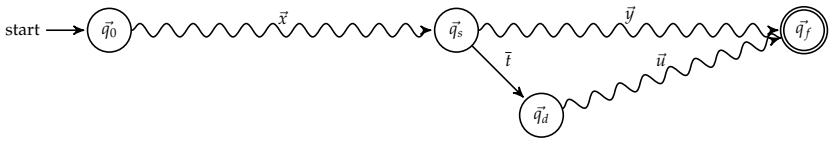


Figure 3.9: The three flows computed by Theorem 8

Example 22. *In Figure 3.8b, the transitions* $(\vec{q}_2, (\square, c), \vec{q}_5)$ *and* $(\vec{q}_8, (\square, c), \vec{q}_9)$ *reveal the second principal (i.e. C)* weakly liable*. Indeed the trace* $(\bar{r}, r)(\square, b)$ *ending in* $\vec{q}_2$ *can be extended to one in weak agreement, while* $(\bar{r}, r)(\square, b)(\square, c)$ *cannot. Also the trace* $(\bar{r}, r)(\bar{t}, \square)(\bar{b}, b)$ *can be extended to one in weak agreement while* $(\bar{r}, r)(\bar{t}, \square)(\bar{b}, b)(\square, c)$ *cannot.*

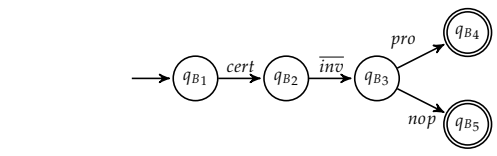
For the transition $(\vec{q}_2, (\square, c), \vec{q}_5)$ *we have the trace* $(\bar{r}, r)(\square, b)$ *for the flow* $\vec{x}$ *and the trace* $(\bar{t}, t)(\bar{b}, \square)(e, \bar{e})$ *for the flow* $\vec{y}$*, and we have* $\forall i \in I_l. \sum_{t_j \in T} a^i_t (x_{t_j} + y_{t_j}) \geq 0$*. Note that if we select as flow* $\vec{y}$ *the trace* $(\square, c)(\bar{t}, \square)(\bar{b}, \square)(e, \bar{e})$ *then the constraints* $\forall i \in I_l. \sum_{t_j \in T} a^i_{t_j} (x_{t_j} + y_{t_j}) \geq 0$ *are not satisfied for the action* $a^4 = c$ *(recall Example 20). For the flow* $\vec{u}$ *the only possible trace is* $(\bar{t}, \square)(\bar{b}, \square)(e, \bar{e})$*, and* $\max \gamma = -1 = \gamma_{(\vec{q}_2, (\square, c), \vec{q}_5)}$ *since* $\sum_{t_j \in T} a^4_{t_j} (x_{t_j} + u_{t_j}) + (-1) = -1$*.*

For the transition $(\vec{q}_8, (\square, c), \vec{q}_9)$ *the flow* $\vec{x}$ *selects the trace* $(\bar{r}, r)(\bar{t}, \square)(\bar{b}, b)$*, the flow* $\vec{y}$ *selects the trace* $(\square, t)(e, \bar{e})$*, since the other possible trace, that is* $(\square, c)(e, \bar{e})$*, does not respect the constraints for the action* a^4 *(i.e. c). Finally, for the flow* $\vec{u}$ *we have the trace* $(e, \bar{e})$*, and as the previous case* $\max \gamma = -1 = \gamma_{(\vec{q}_8, (\square, c), \vec{q}_9)}$*.*

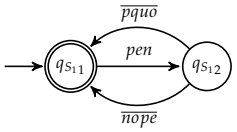
3.5 An example

In this section we consider a well-known case study taken from [Pel03]. This is a purchasing system scenario, where a manufacturer (the buyer) wants to build a product. To configure it, the buyer lists in an inventory the needed components and contacts a purchasing agent. The agent looks for suppliers of these components, and eventually sends back to the buyer its proposal, if any. A supplier is assumed to signal whether it can fulfil a request or not; if neither may happen, the interactions between it and the purchasing agent are rolled back, so as to guarantee the transactional integrity of the overall process. We are interested in an abstract description of the services through contract automata. A description of the WSDL code of the services, as well as the BPEL process from the perspective of the purchasing agent are out of the scope of this chapter, and can eventually be found in [Pel03].

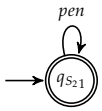
We slightly modify the original protocol, where the purchasing agent guarantees its identity to the buyer through a public-key certificate. For brevity, here we assume to have two sellers S_1 and S_2, and two purchasing agents A_1 and A_2, that behave differently. A service instance involves the buyer, an agent and both sellers. The buyer B requires the certificate of an agent (action *cert*), then it offers the inventory requirements ($\overline{inv}$). Finally, it terminates by receiving either a proposal (*pro*) or a negative message (*nop*), if no proposal can be formulated. The seller S_1 waits for a request (*pen*) of a component from an agent. It then replies by offering a quote for that part ($\overline{pquo}$), or a negative



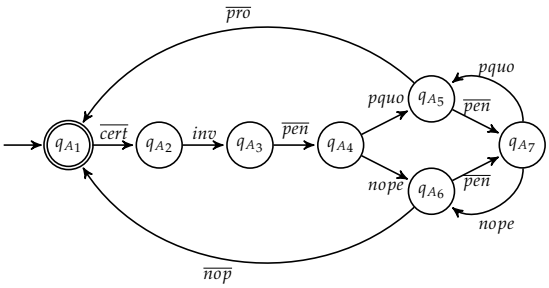
the buyer B



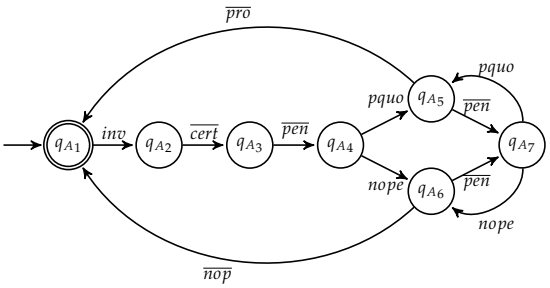
the seller S_1



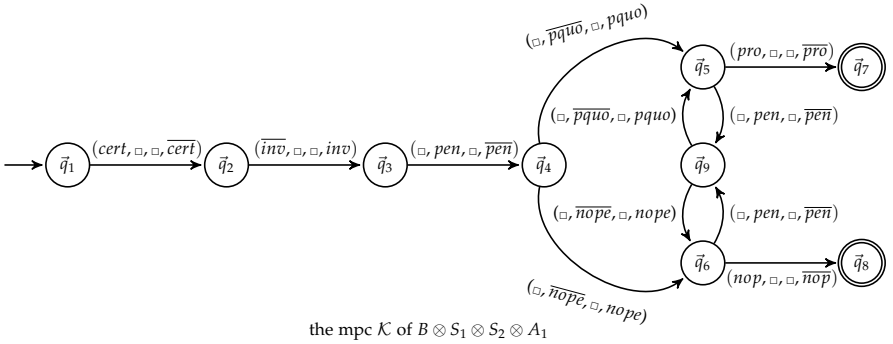
the seller S_2



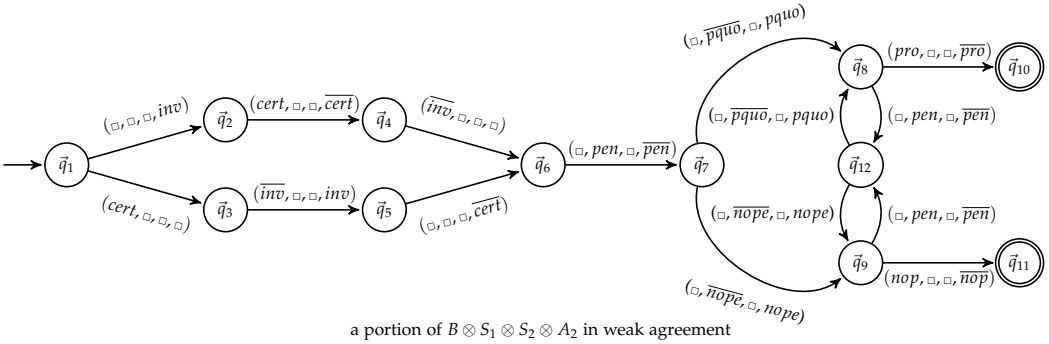
the purchasing agent A_1



the purchasing agent A_2



the mpc $\mathcal{K}$ of $B \otimes S_1 \otimes S_2 \otimes A_1$



a portion of $B \otimes S_1 \otimes S_2 \otimes A_2$ in weak agreement

Figure 3.10: The contract automata for the example

message ($\overline{nope}$) if it is unavailable, and restarts. The second seller S_2 always accepts a request, but never replies. The first agent A_1 offers its certificate ($\overline{cert}$), then requires the inventory list (inv). It then sends a request to and waits for a reply from the sellers. The agent must communicate at least with one supplier before replying to the buyer, and it can span over all the available suppliers in the network, unknown a priori, before compiling its proposal. Finally, it sends to the buyer a proposal ($\overline{prop}$), or the negative message ($\overline{nop}$). The second agent A_2 behaves similarly to A_1, except the first two actions are exchanged: before sending its certificate to B, it first requires the inventory list.

In Figure 3.10 from top to bottom, we display, from left to right, the automata B, S_1 and S_2; the automata A_1 and A_2; then the most permissive controller $\mathcal{K}$ of $B \otimes S_1 \otimes S_2 \otimes A_1$ (the whole composition is omitted to save space); finally a portion of $B \otimes S_1 \otimes S_2 \otimes A_2$ in weak agreement. This example shows that through contract automata one can identify which traces reach success, and which reach a failure, together with those principals responsible for diverging from the expected behaviour, as well as to single out which failures depend on the order of actions, and which not. Indeed, by inspecting $\mathcal{K}$, that of course is safe, one can notice that A_1 never interacts with S_2 because it never replies and so it is recognised liable. As a matter of fact, the composed automaton $B \otimes S_1 \otimes S_2 \otimes A_1$ admits agreement, but it is not safe. Note that $\mathcal{K}$ blocks every communication with S_2, so enforcing transactional integrity, because $\mathcal{K}$ removes all possibilities of rollbacks from a trace not in agreement. The composed automaton $B \otimes S_1 \otimes S_2 \otimes A_2$ admits weak agreement but not agreement (and its most permissive controller is empty), because B and A_2 fail in exchanging the certificate and the inventory requirements, as both are stuck waiting for the fulfilment of their requests. However, by abstracting away the order in which actions are performed circularity is solved, and these requests satisfied. Note that S_2 is detected to be also weakly liable.

3.6 Concluding Remarks

We have studied contracts composition for services, focussing on orchestration. Services are formally represented by a novel class of finite state automata, called contract automata. They have two operators that compose services according to two different notions of orchestrations: one when a principal joins an existing orchestration with no need of a global reconfiguration, and the other when a global adaptive re-orchestration is required.

We have defined notions that illustrate when a composition of contracts behaves well, roughly when all the requests (and offers) are fulfilled. These properties have been formalised as (strong/weak) agreement and (strong/weak) safety, and have been studied both in the case when requests are satisfied synchronously and asynchronously. Furthermore, a notion of (strong/weak) liability has been put forward. A (strongly/weakly) liable principal is a service leading the contract composition into a fail state.

Key results of this chapter are ways to enforce good behaviour of services. For the synchronous versions of agreement and safety, we have applied techniques from Control

Theory, while for the asynchronous versions we have taken advantage of optimisation techniques borrowed from Operations Research. Using them, we efficiently find the optimal solutions of the flow in the network automatically derived from contract automata.

We briefly relate our approach to others willing to describe and analyse service contracts (see Chapter 1 for a brief survey on this subject).

Contract automata are similar to I/O [LT89] and Interface Automata [dAH01], introduced in the field of Component Based Software Engineering. A first difference is that our operators of composition track each principal, to find the possible liable ones. Also we do not allow input enabled operations and non-linear behaviour (i.e. broadcasting offers to every possible request), and our notion of agreement is dual to that of compatibility in [dAH01], that requires all the *offers* to be matched.

Contract automata deals with multi-party interactions through orchestration, while in [BSBM05] only bi-party interactions are considered, i.e. interactions between a single client and a single server.

Our model represents internal/external choice of [CGP09] and [CP09] as a branching of requests/offers. Also, we consider stronger properties than theirs: progress guarantees that a subset of contracts meets their requests, while agreement requires that all of them do, i.e. that each principal reaches a successful state.

Our notion of weak agreement is close to the orchestrator of [Pad10, BvBd15] in the case of *disrespectful compliance*.

The controller for the case of agreement cuts all the paths which may lead one principal to perform a retract. Hence, a controlled interaction of services needs not to roll back, as in [BDLd15], because the orchestrator *prevents* firing of liable transitions. This means that, if a composition of contracts is safe then the contracts are compliant according to [BDLd15]. The converse does not hold. Indeed, our notion of agreement is stronger, as we force an interaction of services to reach a successful state.

We conjecture that may-test of [dNH83] corresponds to the notion of *strong agreement*, while must-test implies *strong safety*, but not vice-versa. For example the service $\overline{a}^*.\overline{b}$ does not must-satisfy the client $a^*.b$, but their product is strongly safe (if unfair, the service may never offer $\overline{b}$ to its client). Actually, strong safety is alike *should testing* of [RV07], where the divergent computations are ruled out.

In [LP15] the compliance and sub-contract relations are extended to deal with choreographies. Compliance is obtained by seeing a choreography as a compound service, similarly to our composed contract automata. Since a client cannot interact with the choreography on actions already used while synchronising by other services, in order to obtain compliance the client must be *non-competitive* with the other services.

Our notion of liability slightly differs from other versions [BTZ12] (see Section 1.4.3), mainly because we do not admit the possibility of redeeming from culpability. Indeed, after a liable transition has been performed, it is no longer possible to reach an agreement. Moreover, circularity issues are solved with weak agreement, allowing asynchronous matches between requests and offers.

Circularity issues in contracts composition have been studied with several logic for-

malisms (see Section 1.4). While in this chapter we have solved circularity issues through weak agreement, a connection between our notion of agreement and provability of formulae representing contracts will be discussed in the next chapter.

A central coordinator (i.e. the controller) is in charge of driving the interactions in a contract composition. We will investigates the conditions under which it is possible to remove this central control in Chapter 5. A comparison between our approach and those in the literature on choreographed approaches is in Section 5.4.

A main advantage of our framework is that it supports development of automatic verification tools for checking and verifying properties of contract composition. In particular, the formal treatment of contract composition in terms of optimal solutions of network flows paves the way of exploiting efficient optimisation algorithms. We have developed a prototypical verification tool, described in Chapter 6.

Chapter 4

Contract Automata and Logics

In this chapter we establish correspondence results between (weak) agreement (see Chapter 3) and provability of formulae in two fragments of different intuitionistic logics, that have been used for modelling contracts and are introduced in Section 1.4.

The obtained results allow us to use techniques developed for verifying the properties of agreement of contract automata to the verification of the corresponding logic formulae. Moreover, once the relation between a formula p and the corresponding automaton $\mathcal{A}$ has been established, it is possible to verify the correctness of $\mathcal{A}$ by proving the corresponding formula p, so exploiting polynomial time algorithms for provability of logic formulae [BCGZ13]. These correspondences provide us with further insights on the relations between circularity issues in logics and in contract automata, both for the case of unrestricted resources (i.e. non-linear logic) and restricted ones (i.e. linear logic).

We now recall the fragments of intuitionistic logics discussed in this chapter. The first one, Propositional Contract Logic [BZ09a], has a special logical connective, called *contractual implication*, to deal with circularity between offers and requests, arising when a principal requires, say a, before offering b to another principal who in turn first requires b and then offers a; note that weak agreement holds for this kind of circularity.

The second fragment, Intuitionist Linear Logic with Mix [Ben95] is a linear logic capable of modelling the exchange of resources with the possibility of recording debts, that arise when the request of a principal is satisfied, but it is not paid back through a due offer.

The Horn fragments of these logics have an immediate interpretation in terms of contracts, and these theories can be interpreted as contract automata, without much effort. Roughly, a contract is rendered as a Horn clause, and a composition is a conjunction of clauses. When a Horn formula is provable, then all the contracts are fulfilled, i.e. all the requests (represented as premises of implications) are entailed.

Firstly, we translate a fragment of the Horn formulae of Propositional Contract Logic into contract automata, and we prove that a formula is entailed if and only if the contract automaton admits agreement. The notion of weak agreement will be also related to the provability of (a subset of) Horn formulae of Propositional Contract Logic. After that, the connection between contract automata and the Intuitionistic Linear Logic with Mix

$$\frac{\Gamma \vdash q}{\Gamma \vdash p \twoheadrightarrow q}\,Zero \qquad \frac{\Gamma, p \twoheadrightarrow q, r \vdash p \quad \Gamma, p \twoheadrightarrow q, p \vdash q}{\Gamma, p \twoheadrightarrow q \vdash r}\,Fix$$

$$\frac{\Gamma, p \twoheadrightarrow q, a \vdash p \quad \Gamma, p \twoheadrightarrow q, q \vdash b}{\Gamma, p \twoheadrightarrow q \vdash a \twoheadrightarrow b}\,PrePost$$

Figure 4.1: The three rules of PCL for the contractual implication.

(ILL^{mix})[Ben95] is studied. Again, we translate a fragment of Horn formulae as contract automata, and we prove that a theorem in ILL^{mix} corresponds to an automaton that admits agreement.

Structure of the chapter In Section 4.1 we present correspondence results with fragments of Propositional Contract Logic, while the correspondence with Intuitionistic Linear Logic with Mix is discussed in Section 4.2. Concluding remarks are in Section 4.3

4.1 Propositional Contract Logic

The usual example for showing the need of circular obligations is Example 14 (recalled below). In the Horn fragment of PCL we use, called H-PCL, the contracts of Alice and Bob make use of the new contractual implication $F \twoheadrightarrow F'$, whose intuition is that the formula F' is deducible, provided that later on in the proof also F will be deduced.

According to this intuition and elaborating over Example 14, Alice's contract (*I offer you my aeroplane provided that in the future you will lend me your bike*) and Bob's (*I offer you my bike provided that in the future you will lend me your aeroplane*) are rendered as

$$bike \twoheadrightarrow airplane \qquad airplane \twoheadrightarrow bike$$

Their composition is obtained by joining the two, and one represents that both Alice and Bob are proved to obtain the toy they request by

$$((bike \twoheadrightarrow airplane) \wedge (airplane \twoheadrightarrow bike)) \vdash (bike \wedge airplane)$$

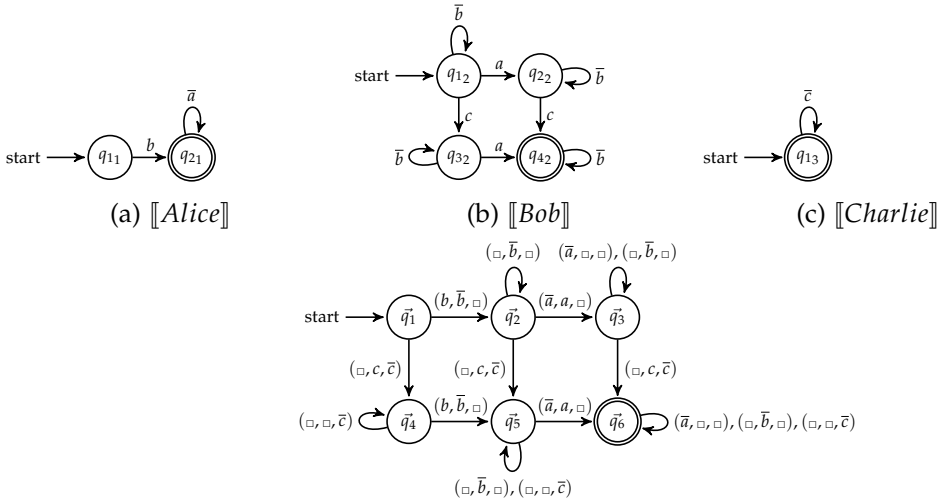
In words, the composition of the two contracts entails *all* the requests (*bike* by Alice and *airplane* by Bob).

The Horn fragment of PCL (H-PCL) [BCP13, BCPZ15] is given in Definition 9.

In Figure 4.1 we recall the three rules of the sequent calculus for the contractual implication [BZ09a]; the others are the standard ones of the Intuitionistic Logic and are reported in Chapter 1.

As anticipated, in H-PCL all requests of principals are satisfied if and only if the conjunction p of the contracts of all principals entails all the atoms mentioned.

Definition 41 (Obligations Fulfilment in PCL)**.** *The formula p represents a composition whose principals respect all their obligations if and only if $p \vdash \lambda(p)$.*



(a) $[\![Alice]\!]$ (b) $[\![Bob]\!]$ (c) $[\![Charlie]\!]$

(d) $\mathcal{K}_{[\![Alice]\!] \otimes [\![Bob]\!] \otimes [\![Charlie]\!]}$

Figure 4.2: The contract automata of Examples 23 and 24

Below, we define the translation from an H-PCL formula p to a contract automata $\mathcal{A} = [\![p]\!]$. This function helps relating the property of agreement of $\mathcal{A}$ with the provability of p. A simple inspection of the rules below suffices to verify that the obtained automata are deterministic.

Definition 42 (From H-PCL to CA)**.** *A H-PCL formula is translated into a contract automaton by applying the following rules, where* $\mathcal{P} = \{q \cup \{*\} \mid q \in 2^J\}$*:*

$$[\![\textstyle\bigwedge_{i\in I} \alpha_i]\!] = \boxtimes_{i\in I} [\![\alpha_i]\!]$$

$$[\![\textstyle\bigwedge_{j\in J} a_j]\!] = \langle \{\{*\}\}, \{*\}, \emptyset, \{\overline{a_j} \mid j \in J\}, \{(\{*\}, \overline{a_j}, \{*\}) \mid \overline{a_j} \in A^o\}, \{\{*\}\}\rangle$$

$$\begin{aligned}[\![(\bigwedge_{j\in J} a_j) \to b]\!] =& \langle \mathcal{P}, J \cup \{*\}, \{a_j \mid j \in J\}, \{\overline{b}\}, \\ & \{(J' \cup \{j\}, a_j, J') \mid J' \cup \{j\} \in \mathcal{P}, j \in J\} \cup \{(\{*\}, \overline{b}, \{*\})\}, \{\{*\}\}\rangle\end{aligned}$$

$$\begin{aligned}[\![(\bigwedge_{j\in J} a_j) \twoheadrightarrow b]\!] =& \langle \mathcal{P}, J \cup \{*\}, \{a_j \mid j \in J\}, \{\overline{b}\}, \\ & \{(J' \cup \{j\}, a_j, J') \mid J' \cup \{j\} \in \mathcal{P}, j \in J\} \cup \{(q, \overline{b}, q) \mid q \in \mathcal{P}\}, \{\{*\}\}\rangle\end{aligned}$$

As expected, a Horn formula is translated as the product of the automata rising from its components α_i. In turn, a conjunction of atoms yields an automaton with a single state and loops driven by offers in bijection with the atoms. A (standard) implication shuffles all the requests corresponding to the premises a_j and then has the single offer corresponding to the conclusion b. A contractual implication is similar, except that the offer ($\overline{b}$ in the definition) can occur at *any* position in the shuffle, and from there onwards it will be always available. Each state stores the number of requests that are still to be

fired, and $\{*\}$ stands for no requests Note that there is no control on the number of times an offer can be taken, as H-PCL is not a linear logic.

Example 23. *Consider again Example 14, and let us modify it to better illustrate some peculiarities of H-PCL. Assume then that there are three kids: Alice, Bob and Charlie, who want to share some toys of theirs: a bike b, an aeroplane a and a car c. The contract of Alice says "I will lend you my aeroplane provided that you lend me your bike". The contract of Bob says "I will lend you my bike on credit that in the future you will lend me your aeroplane and your car". The contract of Charlie says "I will lend you my car". The contract of Alice is expressed by the classical implication $b \to a$. The contract of Bob is $(a \wedge c) \twoheadrightarrow b$, while the contract of Charlie is simply c. The three contracts reach an agreement: the conjunction of the formulae representing the contracts entails all its atoms, that is $(b \to a) \wedge ((a \wedge c) \twoheadrightarrow b) \wedge c \vdash a \wedge c \wedge b$.*

Figure 4.2 shows the translation of Alice $\wedge$ Bob $\wedge$ Charlie, according to Definition 42. It is immediate to verify that the automaton is safe, since all its traces are in agreement.

The following property helps to understand the main result of this section. Item 1 shows that in the final state of $[\![p]\!]$ all requests have been satisfied and all offers are available, while in item 2 it is proved that each state of $[\![p]\!]$ is used for recording all requests that must be satisfied.

Property 5. *Given a H-PCL formula p and the automaton $[\![p]\!] = \langle Q, q_0, A^r, A^o, T, F\rangle$:*

1. *$F = \{\vec{q} = \langle\{*\}, \ldots, \{*\}\rangle\}$, and all $(\vec{q}, \vec{a}, \vec{q'})$ are such that $\vec{q'} = \vec{q}$ and $\vec{a}$ is an offer;*
2. *every state $\vec{q} = \langle J_1, \ldots, J_n\rangle$ has as many request or match outgoing transitions as the request actions prescribed by $\bigcup_{i\in 1\ldots n} J_i$;*
3. *$[\![p]\!]$ is deterministic.*

Proof. The first item follows immediately from Definition 42.
For the second item, we first consider the translation of the clauses in the formula. By construction, for each of them two cases are possible when considering request actions: either $[\![(\bigwedge_{j\in J_i} a_j) \to b]\!]$ or $[\![(\bigwedge_{j\in J_i} a_j) \twoheadrightarrow b]\!]$. In both cases we have outgoing request transitions of the form $\{(J' \cup \{j\}, a_j, J') \mid J' \cup \{j\} \in 2^{J_i}, j \in J\}$. Finally by applying the associative composition $\boxtimes$ (Definition 26), some requests may be matched with corresponding offers, but no new request can be originated.
The third item follows immediately by the translation and by the condition in Definition 9, that all the atoms are different. □

The following lemma shows that if an atom a is entailed by a formula p then there is a trace recognised by the contract automaton $[\![p]\!]$ where the request corresponding to the atom a, if any, is always matched.

Lemma 3 (Formula Provability and Traces)**.** *Given a H-PCL formula p and an atom a in p we have:*

$$p \vdash a \text{ is provable implies } \exists w \in \mathscr{L}([\![p]\!]) \text{ such that no } \vec{a} \text{ request on } a \text{ occurs in } w$$

Proof. Consider each of the conjuncts α of p. If a does not appear in α as the premise of an implication/contractual implication, then the statement follows trivially by Definition 42 and by hypothesis, since the translation of a is an offer action. Otherwise a also occurs in α within:

1. a conjunction, or
2. the conclusion of a contractual implication, or
3. the conclusion of an implication.

For the first two cases, by Definition 42, a transition labelled by the relevant offer $\overline{a}$ is available in all states, so preventing a request a to appear in $[\![p]\!]$, i.e. after the product of the principals (Definition 26).

For proving case 3, $\alpha = \bigwedge_{j\in J} a_j \to a$ and we proceed by induction on the depth of the proof of $p \vdash a$. It must be the case then that $\forall j$ it holds $p \vdash a_j$. We can now either re-use the proof for cases 1 and 2 (that act as base cases), or the induction hypothesis if a_j occurs in the conclusion of an implication. By Definition 42 after all a_j are matched, the offer a will be always available, preventing a request a to appear. □

The following definition and lemma relate the residual of the contract automaton $[\![p]\!]$ after the execution of a transition labelled by $\vec{a}$ with the corresponding formula, namely $p/\vec{a}$. This result will be used for proving Theorem 9.

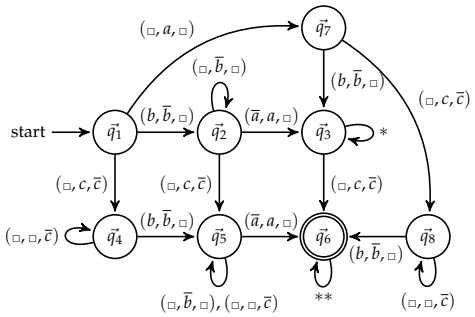
In order to keep the following definition compact, we use $\circ$ for either $\to$ or $\twoheadrightarrow$. In addition, with a slight abuse of the notation we also use $\wedge$ to operate between formulas, we write p' for an empty formula or with a single clause, and we allow the indexing sets J and K in clauses to be empty. Finally, we let $(\bigwedge_{j\in\emptyset} a_j) \circ b$ stand for b.

Definition 43 (Formula Step)**.** *Given a formula p, if from the initial state of $[\![p]\!]$ there is an outgoing offer or an outgoing match transition with label $\vec{a}$, we define*

$$p/\vec{a} = \begin{cases} p & \text{if } \vec{a} \text{ is an offer} \\ p' \wedge (\bigwedge_{z\in Z} c_z \twoheadrightarrow b) \wedge (\bigwedge_{j\in J} a_j) \circ b' & \text{if } \vec{a} \text{ is a match with } \vec{a}_{(i)} = b \text{ and} \\ & p = p' \wedge (\bigwedge_{z\in Z} c_z \twoheadrightarrow b) \wedge (\bigwedge_{j\in J} a_j \wedge b) \circ b' \\ p' \wedge (\bigwedge_{k\in K} a_k \wedge b) \wedge (\bigwedge_{j\in J} a_j) \circ b' & \text{if } \vec{a} \text{ is a match with } \vec{a}_{(i)} = b \text{ and} \\ & p' \wedge (\bigwedge_{k\in K} a_k \wedge b) \wedge (\bigwedge_{j\in J} a_j \wedge b) \circ b' \end{cases}$$

We now establish a relation between $[\![p/\vec{a}]\!]$, and the contract automaton obtained by changing the initial state $\vec{q}_0$ of $[\![p]\!]$ to $\vec{q}$, for the transition $(\vec{q}_0, \vec{a}, \vec{q})$ of $[\![p]\!]$. Recall that the translation given in Definition 42 yields deterministic automata.

Lemma 4 (Formula Step in Automata)**.** *Given a H-PCL formula p and the contract automaton $[\![p]\!] = \langle Q, \vec{q}_0, A^r, A^o, T, F\rangle$, if $t = (\vec{q}_0, \vec{a}, \vec{q}) \in T$ is an offer or a match transition, then $\mathscr{L}(\mathcal{A}) = \mathscr{L}([\![p/\vec{a}]\!])$ where $\mathcal{A} = \langle Q, \vec{q}, A^r, A^o, T, F\rangle$.*



$$[\![Alice \wedge Bob \wedge Charlie]\!]$$

$$* = (\overline{a}, \Box, \Box), (\Box, \overline{b}, \Box), \quad ** = (\overline{a}, \Box, \Box), (\Box, \overline{b}, \Box), (\Box, \Box, \overline{c})$$

Figure 4.3: The CA of Example 24, the principals are in Figure 4.2.

Proof. The proof is by cases of $\vec{a}$. If $\vec{a}$ is an offer, then by Definition 42 it must be $\vec{q} = \vec{q_0}$ and trivially $\mathcal{A} = [\![p/\vec{a}]\!]$.

Otherwise, since $\vec{a}$ is a match action, say on atom b, it contains a request from, say, the i-th principal and a corresponding offer from another. Therefore, $p = \bigwedge_{k \in K} \alpha_k$ contains within a clause α_j the atom b, originating the offer, as a conjunction or as a conclusion of a contractual implication (note that it cannot be an implication because we are in the initial state), and α_i also contains b originating this time the request.

We now prove that the automata $\mathcal{A}$ and $[\![p/\vec{a}]\!]$ have the same initial state. Let $\vec{q_0} = \langle J_1, \ldots, J_n \rangle$, then, since $\vec{a}_{(i)} = b$, the states $\vec{q_0}$ and $\vec{q}$ only differ in the i-th element, where in $\vec{q}_{(i)}$ the request action b is not available anymore; formally, $\forall j \neq i$ it must be $\vec{q}_{(j)} = \vec{q_0}_{(j)} = J_j$, and $\vec{q}_{(i)} = \vec{q_0}_{(i)} \setminus \{i\}$. By Definition 43 p and $p/\vec{a}$ differ because of the single atom b has been removed from α_i. By these facts and by item 2 of Property 5 the language equivalence follows. Indeed, $[\![p/\vec{a}]\!]$ is the product of the same $[\![\alpha_k]\!], k \neq i$ used for $[\![p]\!]$, and the match on b of $\mathcal{A}$ leaves $\vec{q_0}$, that is not reachable from $\vec{q}$. □

Example 24. *Let $[\![p]\!]$ be the automaton shown in Figure 4.3, where $p = Alice \wedge Bob \wedge Charlie$ and the principals are those of Figure 4.2. Consider now $p' = p/(b, \overline{b}, \Box) = (\, a \wedge ((a \wedge c) \twoheadrightarrow b) \wedge c\,)$ and build $[\![p']\!] = \{\langle \{\vec{q}_2, \vec{q}_3, \vec{q}_5, \vec{q}_6\}, \vec{q}_2, A^r, A^o, T, \vec{q}_6 \rangle\}$ (transitions, alphabets and states are taken from $[\![p]\!]$). It is immediate to verify that the language of $[\![p']\!]$ is the same of $[\![p]\!]$, when the initial state is $\vec{q}_2$ instead of $\vec{q}_1$.*

The following lemma is auxiliary for proving the next theorem. Its second item is similar to Lemma 1 in [Pfe00].

Lemma 5 (PCL Auxiliary)**.** *Let a, b be atoms, p, q be conjunction of atoms, with q possibly empty, $p_1, \ldots, p_n$ be formulae, and $\circ \in \{\rightarrow, \twoheadrightarrow\}$, then*

(i) if $\dfrac{\Delta}{\Gamma, q \circ b \vdash p}$ *then* $\exists \Delta' : \dfrac{\Delta'}{\Gamma, (q \wedge a) \circ b, a \vdash p}$

(ii) if $\Gamma \vdash p$ *then* $\forall \Gamma'.\ \Gamma, \Gamma' \vdash p$

(iii) *if* $\bigwedge_{i\in 1\ldots n} p_i \vdash q$ *then* $p_1, \ldots, p_n \vdash q$

(iv) *if* $\Gamma \vdash \bigwedge_{i\in 1\ldots n} p_i$ *then* $\forall i.\, \Gamma \vdash p_i$

Proof. To prove the first item, we proceed by induction on the depth of Δ and by cases on the last rule applied. In the base case Δ is empty, we have two cases

1. q non-empty or $p \neq b$: then it must be that $\Gamma = p, \Gamma'$ for some Γ' and the last rule applied is *id*. Trivially, Δ' will be empty and we have
$$\frac{}{\Gamma', p, (q \wedge a) \circ b, a \vdash p} id$$
2. q empty and $p = b$: our hypothesis reads as $\dfrac{}{\Gamma, b \vdash b} id$, and we build the following deduction
$$\frac{\dfrac{}{\Gamma', a \circ b, a \vdash a} id \quad \dfrac{}{\Gamma, a \circ b, a, b \vdash b} id}{\Gamma, a \circ b, a \vdash b} \diamondsuit$$
where if $\circ = \rightarrow$ then $\Gamma' = \Gamma$ and $\diamondsuit = \rightarrow L$, otherwise if $\circ = \twoheadrightarrow$ then $\Gamma' = \Gamma, b$ and $\diamondsuit = Fix$.

For the inductive step, we distinguish two cases:

1. the last rule applied to deduce the hypothesis does not involve $q \circ b$. Hence the rule must be applied on p or on a formula in Γ. We can apply the same rule to $\Gamma, (q \wedge a) \circ b, a \vdash p$ and use the inductive hypothesis.
2. the last rule applied to deduce the hypothesis involves $q \circ b$. There are two exhaustive cases

   (a) $\circ = \rightarrow$, then the last rule applied is $\rightarrow L$ and the deduction tree has the following form:
   $$\frac{\dfrac{\Delta_1}{\Gamma, q \rightarrow b \vdash q} \quad \dfrac{\Delta_2}{\Gamma, q \rightarrow b, b \vdash p}}{\Gamma, q \rightarrow b \vdash p} \rightarrow L$$
   Then by induction hypothesis we have
   $$\frac{\Delta_1'}{\Gamma, (q \wedge a) \rightarrow b, a \vdash q} \qquad \frac{\Delta_2'}{\Gamma, (q \wedge a) \rightarrow b, a, b \vdash p}$$
   From the right one and a derivation tree Δ_3 detailed below, we build
   $$\frac{\Delta_3 \quad \dfrac{\Delta_2'}{\Gamma, (q \wedge a) \rightarrow b, a, b \vdash p}}{\Gamma, (q \wedge a) \rightarrow b, a \vdash p} \rightarrow L$$
   Δ_3 is the derivation tree:
   $$\frac{\dfrac{\Delta_1'}{\Gamma, (q \wedge a) \rightarrow b, a \vdash q} \quad \dfrac{}{\Gamma, (q \wedge a) \rightarrow b, a \vdash a} id}{\Gamma, (q \wedge a) \rightarrow b, a \vdash q \wedge a} \wedge R$$

(b) $\circ = \twoheadrightarrow$, then the last rule applied is *Fix* and the deduction tree has the following form:

$$\dfrac{\dfrac{\Delta_1}{\Gamma, q \twoheadrightarrow b, p \vdash q} \quad \dfrac{\Delta_2}{\Gamma, q \twoheadrightarrow b, b \vdash p}}{\Gamma, q \twoheadrightarrow b \vdash p} Fix$$

Then by the induction hypothesis we have

$$\dfrac{\Delta_1'}{\Gamma, (q \wedge a) \twoheadrightarrow b, a, p \vdash q} \qquad \dfrac{\Delta_2'}{\Gamma, (q \wedge a) \twoheadrightarrow b, a, b \vdash p}$$

From the above, we build the following

$$\dfrac{\dfrac{\dfrac{\Delta_1'}{\Gamma, (q \wedge a) \twoheadrightarrow b, a, p \vdash q} \quad \dfrac{}{\Gamma, (q \wedge a) \twoheadrightarrow b, a, p \vdash a} id}{\Gamma, (q \wedge a) \twoheadrightarrow b, a, p \vdash q \wedge a} \wedge R \quad \dfrac{\Delta_2'}{\Gamma, (q \wedge a) \twoheadrightarrow b, a, b \vdash p}}{\Gamma, (q \wedge a) \twoheadrightarrow b, a \vdash p} Fix$$

For the second item, we prove a stronger fact: the last rule used to deduce $\Gamma, \Gamma' \vdash p$ is the same used for proving $\Gamma \vdash p$. We proceed by induction on the depth of the derivation for $\Gamma \vdash p$ and then by case analysis on the last rule applied.

The base case is when the axiom *id* is applied, and the proof is immediate.

For the inductive case, we assume that for some rule $\diamondsuit$

$$\dfrac{\Delta}{\Gamma \vdash p} \diamondsuit \qquad \text{implies} \qquad \dfrac{\overline{\Delta}}{\Gamma, \Gamma' \vdash p} \diamondsuit$$

Rather than considering each rule at a time, we group them in two classes: those with two premises, and those with one premise. Below, we discuss the first case, and the second follows simply erasing one premise in what follows. The deduction tree in the premise above has the following form

$$\dfrac{\dfrac{\Delta'}{\overline{\Gamma} \vdash q} \quad \dfrac{\Delta''}{\overline{\Gamma'} \vdash q'}}{\Gamma \vdash p} \diamondsuit$$

and by applying the induction hypothesis to both the premises we conclude

$$\dfrac{\dfrac{\overline{\Delta'}}{\overline{\Gamma}, \Gamma' \vdash q} \quad \dfrac{\overline{\Delta''}}{\overline{\Gamma'}, \Gamma' \vdash q'}}{\Gamma, \Gamma' \vdash p} \diamondsuit$$

Moreover note that in this fragment no contradictions can be introduced.

For the third item, we have a derivation tree Δ for the sequent $\bigwedge_{i \in 1 \ldots n} p_i \vdash q$. To build a derivation tree Δ' for $p_1, \ldots, p_n \vdash q$ apply the following two steps. The first step removes from Δ all the rules $\wedge L_i$ applied to (each sub-term of) $\bigwedge_{i \in 1 \ldots n} p_i$, obtaining Δ''. Then, replace all applications of the axiom (id) in Δ'' of the form

$$\dfrac{}{\Gamma, \bigwedge_{j \in J} p_j \vdash \bigwedge_{j \in J} p_j} id$$

with a derivation tree with $k = |J|$ leaves of the form

$$\frac{}{\Gamma, p_1, p_2, ..., p_k \vdash p_j}\, id$$

and by repeatedly applying the rule $(\wedge R)$ until we obtain the relevant judgement $\Gamma, p_1, p_2, ..., p_k \vdash \bigwedge_{i\in 1...n} p_i$.
For the fourth item, we have a derivation tree Δ for the sequent $\Gamma \vdash \bigwedge_{i\in\{1...n\}} p_i$.
For each sequent $\Gamma \vdash p_j$, $j \in \{1 \dots n\}$, the derivation tree is then:

$$\dfrac{\dfrac{\Delta}{\Gamma \vdash p_j \wedge \bigwedge_{i\in\{1...n\}\setminus\{j\}} p_i} \quad \dfrac{\dfrac{}{p_j, \bigwedge_{i\in\{1...n\}\setminus\{j\}} p_i \vdash p_j}\, id}{p_j \wedge \bigwedge_{i\in\{1...n\}\setminus\{j\}} p_i \vdash p_j}\, \wedge L1}{\Gamma \vdash p_j}\, cut$$

□

As said above, when seen in terms of composed contracts, the formula $p \vdash \lambda(p)$ expresses that all the requests made by principals in p must be fulfilled sooner or later. We now show that the contract automaton $[\![p]\!]$ admits agreement if and only if $p \vdash \lambda(p)$ is provable.

Theorem 9 (PCL agreement)**.** *Given a H-PCL formula p we have $p \vdash \lambda(p)$ if and only if $[\![p]\!]$ admits agreement.*

Proof. ($\Rightarrow$) Since $p \vdash \lambda(p)$ by Lemma 5(iv) (where $\Gamma = p$) we have $p \vdash a$ for all atoms a in p. It suffices to apply Lemma 3 to each of these atoms, and by Definition 42 the offers are never consumed, there must be a trace $w \in \mathscr{L}([\![p]\!])$ where all the requests are matched.

($\Leftarrow$) Let $\vec{q_0}$ be the initial state of $[\![p]\!]$ and $\vec{f}$ be the final state. We proceed by induction on the length of w.

In the base case w is empty, hence the initial state of $[\![p]\!]$ is also final. This situation only arises when the second rule of Definition 42 has been applied for all conjuncts α_i corresponding to principals. Therefore it must be that p is a conjunction of atoms, so $p = \lambda(p)$ and the thesis holds immediately.

For the inductive step we have $w = \vec{a}w_2$, and $(\vec{a}w_2, \vec{q_0}) \to (w_2, \vec{q}) \to^+ (\varepsilon, \vec{f})$. By inductive hypothesis and Lemma 4 we have $p/\vec{a} \vdash \lambda(p/\vec{a})$. If $\vec{a}$ is an offer by Definition 43 we have $p = p/\vec{a}$ and the thesis holds directly. Note that $\lambda(p) = \lambda(p/\vec{a})$ because $\vec{a}$ labels a match or an offer transition outgoing from $\vec{q_0}$ and the offer comes from the conclusion of a contractual implication or a conjunction of atoms, that is unmodified in $p/\vec{a}$. Hence since by inductive hypothesis $p/\vec{a} \vdash \lambda(p/\vec{a})$ and since $\lambda(p) = \lambda(p/\vec{a})$, proving $p \vdash p/\vec{a}$ entails $p \vdash \lambda(p)$. This is because of the following proof (note that there exists a longer one, cut-free) and Lemma 5 (ii)

$$\frac{p \vdash p/\vec{a} \quad p, p/\vec{a} \vdash \lambda(p)}{p \vdash \lambda(p)}\, cut$$

To prove $p \vdash p/\vec{a}$ we proceed by cases according to the structure of p, (omitting the cases for $J = \emptyset$ for which the proof is trivial)

- if $p = p' \wedge (\bigwedge_{z\in Z} c_z \twoheadrightarrow b) \wedge (\bigwedge_{j\in J} a_j \wedge b \to b')$ we have to prove the sequent $p \vdash p/\vec{a}$ that reads as

$$(p' \wedge (\textstyle\bigwedge_{z\in Z} c_z \twoheadrightarrow b) \wedge (\bigwedge_{j\in J} a_j \wedge b \to b')) \vdash (p' \wedge (\bigwedge_{z\in Z} c_z \twoheadrightarrow b) \wedge (\bigwedge_{j\in J} a_j \to b'))$$

For readability, we first determine the sequent $\Gamma \vdash (\bigwedge_{j\in J} a_j) \to b'$ where $\Gamma = p', (\bigwedge_{z\in Z} c_z \twoheadrightarrow b), (\bigwedge_{j\in J} a_j \wedge b) \to b'$ from p, by applying the rule $\wedge R$, and Lemma 5(iii). Then we build the following derivation, where * is detailed below:

$$\dfrac{\dfrac{\dfrac{}{\Gamma, \bigwedge_{j\in J} a_j \vdash \bigwedge_{j\in J} a_j}id \quad \dfrac{*}{\Gamma, \bigwedge_{j\in J} a_j \vdash b}\diamondsuit}{\Gamma, \bigwedge_{j\in J} a_j \vdash \bigwedge_{j\in J} a_j \wedge b} \wedge R \quad \dfrac{}{\Gamma, \bigwedge_{j\in J} a_j, b' \vdash b'}id}{\Gamma, \bigwedge_{j\in J} a_j \vdash b'} \to L}{\Gamma \vdash (\bigwedge_{j\in J} a_j) \to b'} \to R$$

The fragment * of the proof can have two different forms, depending on the set Z:

- if $Z = \emptyset$, then * is empty and the rule $\diamondsuit$ is id
- otherwise the fragment * consists of the two sub-derivations below, and the rule $\diamondsuit$ applied to them is Fix

$$\dfrac{\dfrac{\Delta_3}{\Gamma, \bigwedge_{j\in J} a_j, b \vdash \bigwedge_{z\in Z} c_z}}{\Gamma, \bigwedge_{j\in J} a_j, b \vdash b}id \tag{4.1}$$

We now show how to obtain Δ_3. Let Δ be the derivation tree for $p/\vec{a} \vdash \lambda(p/\vec{a})$, that exists by the inductive hypothesis. Note that since $\lambda(p/\vec{a})$ is a conjunction where $\bigwedge_{z\in Z} c_z$ occurs, the following proof can be obtained by applying Lemma 5(iv) for all c_z and by combining them with rule $\wedge R$:

$$\dfrac{\Delta_2}{(p', (\bigwedge_{z\in Z} c_z \twoheadrightarrow b), (\bigwedge_{j\in J} a_j \to b')) \vdash \bigwedge_{z\in Z} c_z} \tag{4.2}$$

Now, in order to obtain the following from the proof (4.2), i.e.

$$\dfrac{\Delta_3}{(p', (\bigwedge_{z\in Z} c_z \twoheadrightarrow b), (\bigwedge_{j\in J} a_j \wedge b) \to b', \bigwedge_{j\in J} a_j, b) \vdash \bigwedge_{z\in Z} c_z} \tag{4.3}$$

we apply Lemma 5(ii): the left hand-side of the sequent

$$(p', (\bigwedge_{z\in Z} c_z \twoheadrightarrow b), (\bigwedge_{j\in J} a_j \to b')) \vdash \bigwedge_{z\in Z} c_z$$

is augmented with $\bigwedge_{j\in J} a_j$. Finally by applying Lemma 5(i), the formula $\bigwedge_{j\in J} a_j \to b'$ above becomes $(\bigwedge_{j\in J} a_j \wedge b) \to b', b$, obtaining (4.3).

- if $p = p' \wedge (\bigwedge_{k\in K} a_k \wedge b) \wedge ((\bigwedge_{j\in J} a_j \wedge b) \to b')$ we have to prove the sequent $p \vdash p/\vec{a}$ that reads as

  $$(p' \wedge (\textstyle\bigwedge_{k\in K} a_k \wedge b) \wedge (\bigwedge_{j\in J} a_j \wedge b) \to b') \vdash (p' \wedge (\bigwedge_{k\in K} a_k \wedge b) \wedge (\bigwedge_{j\in J} a_j \to b'))$$

  For readability, we first determine the sequent $\Gamma \vdash (\bigwedge_{j\in J} a_j) \to b'$ where $\Gamma = p', (\bigwedge_{k\in K} a_k \wedge b), ((\bigwedge_{j\in J} a_j \wedge b) \to b')$ from p, by applying the rule $\wedge R$ and Lemma 5(iii). Then we build the following derivation, where * is detailed below:

  $$\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma, \bigwedge_{j\in J} a_j \vdash \bigwedge_{j\in J} a_j}id \quad \dfrac{*}{\Gamma, \bigwedge_{j\in J} a_j \vdash b}\diamondsuit}{\Gamma, \bigwedge_{j\in J} a_j \vdash \bigwedge_{j\in J} a_j \wedge b}\wedge R \quad \dfrac{}{\Gamma, \bigwedge_{j\in J} a_j, b' \vdash b'}id}{\Gamma, \bigwedge_{j\in J} a_j \vdash b'}\to L}{\Gamma \vdash (\bigwedge_{j\in J} a_j) \to b'}\to R$$

  The fragment * of the proof can have two different forms, depending on the set K:

  - if $K = \emptyset$, then * is empty and the rule $\diamondsuit$ is id
  - otherwise the rule $\diamondsuit$ is $\wedge L2$ applied to the fragment * below

    $$\dfrac{}{p', (\bigwedge_{k\in K} a_k \wedge b), b, ((\bigwedge_{j\in J} a_j \wedge b) \to b'), \bigwedge_{j\in J} a_j \vdash b}id$$

- if $p = p' \wedge (\bigwedge_{z\in Z} c_z \twoheadrightarrow b) \wedge ((\bigwedge_{j\in J} a_j \wedge b) \twoheadrightarrow b')$ we have to prove the sequent $p \vdash p/\vec{a}$ that reads as

  $$(p' \wedge (\textstyle\bigwedge_{z\in Z} c_z \twoheadrightarrow b) \wedge (\bigwedge_{j\in J} a_j \wedge b) \twoheadrightarrow b') \vdash (p' \wedge (\bigwedge_{z\in Z} c_z \twoheadrightarrow b) \wedge (\bigwedge_{j\in J} a_j \twoheadrightarrow b'))$$

  For readability, we first determine the sequent $\Gamma \vdash \bigwedge_{j\in J} a_j \twoheadrightarrow b'$ where $\Gamma = p', (\bigwedge_{z\in Z} c_z \twoheadrightarrow b), (\bigwedge_{j\in J} a_j \wedge b \twoheadrightarrow b')$, by applying the rule $\wedge R$ and Lemma 5(iii). Then we build the following derivation, where * is detailed afterwards:

  $$\dfrac{\dfrac{\dfrac{*}{\Gamma, b' \vdash \bigwedge_{j\in J} a_j \wedge b}Fix \quad \dfrac{}{\Gamma, b' \vdash b'}id}{\Gamma \vdash b'}Fix}{\Gamma \vdash \bigwedge_{j\in J} a_j \twoheadrightarrow b'}Zero$$

  The fragment * of the proof can have two different forms, depending on the set Z:

  - if $Z = \emptyset$, we have that $\Gamma = p', b, (\bigwedge_{j\in J} a_j \wedge b) \twoheadrightarrow b'$ and

    $$\dfrac{\dfrac{}{\Gamma, b', \bigwedge_{j\in J} a_j \wedge b \vdash \bigwedge_{j\in J} a_j \wedge b}id \quad \dfrac{\dfrac{\Delta_3'}{\Gamma, b' \vdash \bigwedge_{j\in J} a_j} \quad \dfrac{}{\Gamma, b' \vdash b}id}{\Gamma, b' \vdash \bigwedge_{j\in J} a_j \wedge b}\wedge R}{\Gamma, b' \vdash \bigwedge_{j\in J} a_j \wedge b}Fix$$

Since the inductive hypothesis guarantees that $p/\vec{a} \vdash \lambda(p/\vec{a})$ holds and $\lambda(p/\vec{a})$ is a conjunction where $\bigwedge_{j\in J} a_j$ occurs, by applying the reasoning of the previous case we have a derivation tree Δ_2' for the sequent

$$(\, p', b, (\bigwedge_{j\in J} a_j \twoheadrightarrow b')\,) \vdash \bigwedge_{j\in J} a_j$$

As done above, by applying Lemma 5 we obtain the derivation tree Δ_3' for

$$(\, \Gamma'', p', b, (\bigwedge_{j\in J} a_j \wedge b) \twoheadrightarrow b', b'\,) \vdash \bigwedge_{j\in J} a_j$$

– if $Z \neq \emptyset$ we obtain:

$$\dfrac{\dfrac{(**)}{\Gamma, b', \bigwedge_{j\in J} a_j \wedge b \vdash \bigwedge_{z\in Z} c_z} \quad \dfrac{\dfrac{(***)}{\Gamma, b', b \vdash \bigwedge_{j\in J} a_j} \quad \dfrac{}{\Gamma, b', b \vdash b}\,id}{\Gamma, b', b \vdash \bigwedge_{j\in J} a_j \wedge b}\wedge R}{\Gamma, b' \vdash \bigwedge_{j\in J} a_j \wedge b}\,Fix$$

From the induction hypothesis, with the argument used in the previous cases, we prove the following sequent

$$(p', (\bigwedge_{z\in Z} c_z \twoheadrightarrow b), (\bigwedge_{j\in J} a_j \twoheadrightarrow b')) \vdash \bigwedge_{z\in Z} c_z$$

Now, we apply Lemma 5 to it, we determine the deduction $(**)$ and a proof for the leftmost sequent above

$$(p', b', \bigwedge_{j\in J} a_j \wedge b, (\bigwedge_{z\in Z} c_z \twoheadrightarrow b), (\bigwedge_{j\in J} a_j \wedge b \twoheadrightarrow b')) \vdash \bigwedge_{z\in Z} c_z$$

Just as done above, from the induction hypothesis we prove the sequent

$$(p', (\bigwedge_{z\in Z} c_z \twoheadrightarrow b), (\bigwedge_{j\in J} a_j) \twoheadrightarrow b') \vdash \bigwedge_{j\in J} a_j$$

from which we obtain the right-most sequent above (***), by applying Lemma 5

$$(p', b', b, (\bigwedge_{z\in Z} c_z \twoheadrightarrow b), (\bigwedge_{j\in J} a_j \wedge b) \twoheadrightarrow b') \vdash \bigwedge_{j\in J} a_j$$

- if $p = p' \wedge (\bigwedge_{k\in K} a_k \wedge b) \wedge (\bigwedge_{j\in J} a_j \wedge b \twoheadrightarrow b')$ we have to prove the sequent $p \vdash p/\vec{a}$ that reads as

$$(\, p' \wedge (\bigwedge_{k\in K} a_k \wedge b) \wedge (\bigwedge_{j\in J} a_j \wedge b \twoheadrightarrow b')\,) \vdash (\, p' \wedge (\bigwedge_{k\in K} a_k \wedge b) \wedge (\bigwedge_{j\in J} a_j \twoheadrightarrow b')\,)$$

For readability, we first determine the sequent $\Gamma \vdash \bigwedge_{j\in J} a_j \twoheadrightarrow b'$ where $\Gamma = p', (\bigwedge_{k\in K} a_k \wedge b), (\bigwedge_{j\in J} a_j \wedge b \twoheadrightarrow b')$, by applying the rule $\wedge R$ and Lemma 5(iii). Then we build the following derivation, where * is detailed afterwards:

$$\dfrac{\dfrac{\overset{*}{\overline{\Gamma, b' \vdash \bigwedge_{j\in J} a_j \wedge b}}\,Fix \quad \overline{\Gamma, b' \vdash b'}\,id}{\Gamma \vdash b'}\,Fix}{\Gamma \vdash \bigwedge_{j\in J} a_j \twoheadrightarrow b'}\,Zero$$

The fragment * of the proof can have two different forms, depending on the set K:

– if $K = \emptyset$, we have $\Gamma = p', b, (\bigwedge_{j\in J} a_j \wedge b) \twoheadrightarrow b'$ and

$$\dfrac{\overline{\Gamma, b', \bigwedge_{j\in J} a_j \wedge b \vdash \bigwedge_{j\in J} a_j \wedge b}\,id \quad \dfrac{\dfrac{\Delta_3'}{\Gamma, b' \vdash \bigwedge_{j\in J} a_j} \quad \overline{\Gamma, b' \vdash b}\,id}{\Gamma, b' \vdash \bigwedge_{j\in J} a_j \wedge b}\,\wedge R}{\Gamma, b' \vdash \bigwedge_{j\in J} a_j \wedge b}\,Fix$$

Since the inductive hypothesis guarantees that $p/\vec{a} \vdash \lambda(p/\vec{a})$ holds and $\lambda(p/\vec{a})$ is a conjunction where $\bigwedge_{j\in J} a_j$ occurs, by applying the reasoning of the previous case we have a derivation tree Δ_2' for the sequent

$$(\, p', b, (\bigwedge_{j\in J} a_j \twoheadrightarrow b')\,) \vdash \bigwedge_{j\in J} a_j$$

As done above, by applying Lemma 5 we obtain the derivation tree Δ_3' for

$$(\, p', b, (\bigwedge_{j\in J} a_j \wedge b) \twoheadrightarrow b', b'\,) \vdash \bigwedge_{j\in J} a_j$$

– if $K \neq \emptyset$ we have that $\Gamma = p', (\bigwedge_{k\in K} a_k \wedge b), (\bigwedge_{j\in J} a_j \wedge b) \twoheadrightarrow b'$ and

$$\dfrac{\overline{\Gamma, b', \bigwedge_{j\in J} a_j \wedge b \vdash \bigwedge_{j\in J} a_j \wedge b}\,id \quad \dfrac{\dfrac{\Delta_3'}{\Gamma, b' \vdash \bigwedge_{j\in J} a_j} \quad \dfrac{\overline{\Gamma, b', b \vdash b}\,id}{\Gamma, b' \vdash b}\,\wedge L2}{\Gamma, b' \vdash \bigwedge_{j\in J} a_j \wedge b}\,\wedge R}{\Gamma, b' \vdash \bigwedge_{j\in J} a_j \wedge b}\,Fix$$

Since the inductive hypothesis guarantees that $p/\vec{a} \vdash \lambda(p/\vec{a})$ holds and $\lambda(p/\vec{a})$ is a conjunction where $\bigwedge_{j\in J} a_j$ occurs, by applying the reasoning of the previous case we have a derivation tree Δ_2' for the sequent

$$(\, p', (\bigwedge_{k\in K} a_k \wedge b), (\bigwedge_{j\in J} a_j \twoheadrightarrow b')\,) \vdash \bigwedge_{j\in J} a_j$$

As done above, by applying Lemma 5 we obtain the derivation tree Δ_3' for

$$(\, p', (\bigwedge_{k\in K} a_k \wedge b), (\bigwedge_{j\in J} a_j \wedge b) \twoheadrightarrow b', b'\,) \vdash \bigwedge_{j\in J} a_j$$

□

We have showed that a formula p fulfils all its obligations if and only if the corresponding automaton $[\![p]\!]$ admits agreement. This result allows us to generate a deduction tree for an H-PCL formula p using the technique for checking agreement of the contract automaton $[\![p]\!]$. Interestingly, a contractual implication $a \twoheadrightarrow b$ corresponds to a contract automaton that is able to fire the conclusion (i.e. b) at each state; while for the standard implication $a \rightarrow b$ the conclusion is available only after the corresponding premise (i.e. a) has been satisfied.

Example 25. *Consider Example 23. The conjunction of all the formulas entails its atoms, indeed the corresponding translation into contract automata displayed in Figure 4.2 admits agreement.*

Needless to say, the provability of $p \vdash \lambda(p)$ implies that $[\![p]\!]$ admits weak agreement. However, the implication is in one direction only, as shown by the following example.

Example 26. *Consider the H-PCL formula $p = (b \rightarrow a) \wedge (a \rightarrow b)$. We have that $[\![p]\!]$ does not admit agreement and $p \nvdash \lambda(p)$. Nevertheless $[\![p]\!]$ admits weak agreement. For example, $(b, -)(\overline{a}, a)(-, \overline{b}) \in \mathscr{L}([\![p]\!])$ is a trace in weak agreement.*

As a matter of fact, weak agreement implies provability when a formula p contains no (standard) implications, as stated below.

Theorem 10 (PCL Weak Agreement)**.** *Let p be a H-PCL formula with no occurrence of standard implications $\rightarrow$, then $p \vdash \lambda(p)$ if and only if $[\![p]\!]$ admits weak agreement.*

Proof. ($\Rightarrow$) Straightforward from Theorem 9 and from $\mathfrak{A} \subset \mathfrak{W}$.

($\Leftarrow$) Since $[\![p]\!]$ admits weak agreement there exists a trace $w \in \mathscr{L}([\![p]\!])$ where each request is combined with a corresponding offer. For proving $p \vdash \lambda(p)$ we will prove $p \vdash a$ for all the atoms a in $\lambda(p)$ and the thesis follows by repeatedly applying the rule $\wedge R$. If a occurs within:

1. $\bigwedge_{j \in J} a_j$: it suffices to apply the rules $\wedge L_1, \wedge L_2, id$;
2. $\bigwedge_{j \in J} a_j \twoheadrightarrow a$: $p \vdash a$ holds if we prove the sequent

$$\Gamma, (\bigwedge_{j \in J} a_j \twoheadrightarrow a) \vdash a$$

that is obtained from $p \vdash a$ by repeatedly applying the rules $\wedge L_i$, for some Γ containing p and sub-formulas of p. The proof of this sequent has the following form:

$$\dfrac{\dfrac{*}{\Gamma, (\bigwedge_{j \in J} a_j \twoheadrightarrow a), a \vdash \bigwedge_{j \in J} a_j} \qquad \dfrac{}{\Gamma, (\bigwedge_{j \in J} a_j \twoheadrightarrow a), a \vdash a}\, id}{\Gamma, (\bigwedge_{j \in J} a_j \twoheadrightarrow a) \vdash a}\, Fix$$

We prove the sequent in the left premise, it suffices to establish the sequents $\Gamma, (\bigwedge_{j \in J} a_j \twoheadrightarrow a), a \vdash a_j$, for all the atoms a_j of $\bigwedge_{j \in J} a_j$. Then, the derivation proceeds

by repeatedly applying the rule $\wedge R$. We are left to prove $\Gamma, (\bigwedge_{j\in J} a_j \twoheadrightarrow a), a \vdash a_j$, which is done by recursively applying the construction of cases (1) and (2). This procedure will eventually terminate. Indeed, at each iteration a_j is either a conjunct in $\bigwedge_{k\in K} a_k$ (case 1) and the proof is closed by rule (id), or a_j is the conclusion of the contractual implication $\bigwedge_{k\in K} a_k \twoheadrightarrow a_j$ and the proof proceeds as in case (2) by applying the rule (Fix). In the last case, the premise in the left hand-side becomes $\Gamma', (\bigwedge_{k\in K} a_k \twoheadrightarrow a_j), a, a_j \vdash \bigwedge_{k\in K} a_k$, so adding a_j in the left part of the sequent. The number of iterations is therefore bound by the number of atoms in p.

3. $\bigwedge_{j\in J} a_j \twoheadrightarrow b$ where $a \neq b$. This case reduces to one of the above two, because if $\exists j \in J$ such that $a_j = a$, then a must also appear in another conjunct $\bigwedge_{z\in Z} a_z$ or in another contractual implication $\bigwedge_{z\in Z} a_z \twoheadrightarrow a$, otherwise all the traces of $[\![p]\!]$ would have an unmatched request on a, against the hypothesis that it admits weak agreement.

□

This result helps to gain insights on the relation between the contractual implication connective $\twoheadrightarrow$ and the property of weak agreement. Indeed, checking weak agreement on a contract automaton $[\![p]\!]$ is equivalent to prove that the formula p fulfils all its obligations (i.e. $p \vdash \lambda(p)$) *only if* p contains no standard implication. This is because all the connectives $\rightarrow$ are lifted to the contractual version $\twoheadrightarrow$ when checking weak agreement.

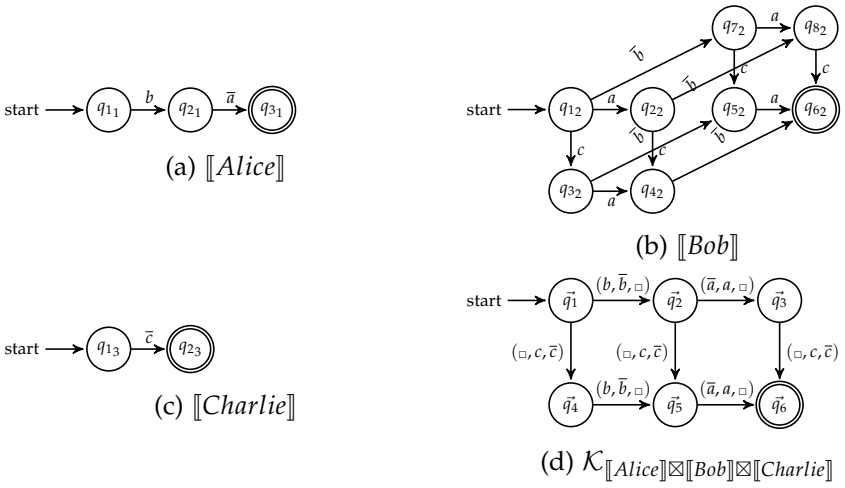
4.2 Intuitionistic Linear Logic with mix

In this sub-section we will interpret a fragment of the Intuitionistic Linear Logic with Mix (ILL^{mix}) [Ben95] in terms of contract automata. Originally, this logic has been used for modelling exchange of resources between partners with the possibility of recording debits, through the so-called *negative atoms*.

Below, we slightly modify Example 23 to better illustrate some features of ILL^{mix}.

Example 27. *Alice, Bob and Charlie want to share their bike, aeroplane and car, according to the same contracts declared in Example 23. In ILL^{mix} the contract of Alice is expressed by the linear implication $b \multimap a$; the contract of Bob is $a^{\perp} \otimes c^{\perp} \otimes b$ ($\otimes$ is the tensor product of Linear Logic); the contract of Charlie is the offer c. The intuition is that a positive atom, e.g. c in the contract of Charlie, represents a resource that can be used; similarly for the b of Bob. Instead, the negative atoms ($a^{\perp}$ and $c^{\perp}$ of Bob) represent missing resources that however can be taken on credit to be honoured later on. The implication of Alice says that the resource a is produced by consuming b, provided b is available. (There are some restrictions on the occurrences of negative atoms made precise below).*

The composition (via tensor product) of the three contracts is successful, in that all resources are exchanged and all debits honoured. Indeed it is possible to prove that all the negative atoms, i.e. all the requests, will be eventually satisfied. In this case we have that all the resources are consumed, and that the following sequent is provable: $Alice \otimes Bob \otimes Charlie \vdash$.



(a) $[\![Alice]\!]$

(b) $[\![Bob]\!]$

(c) $[\![Charlie]\!]$

(d) $\mathcal{K}_{[\![Alice]\!] \boxtimes [\![Bob]\!] \boxtimes [\![Charlie]\!]}$

Figure 4.4: The contract automata of Example 27

$$\frac{}{A \vdash A}\ Ax \qquad \frac{\Gamma \vdash \quad \Gamma' \vdash \gamma}{\Gamma, \Gamma' \vdash \gamma}\ Mix \qquad \frac{\Gamma \vdash A}{\Gamma, A^{\perp} \vdash}\ NegL$$

$$\frac{\Gamma, A, B \vdash \gamma}{\Gamma, A \otimes B \vdash \gamma} \otimes L \qquad \frac{\Gamma \vdash A \quad \Gamma' \vdash B}{\Gamma, \Gamma' \vdash A \otimes B} \otimes R$$

$$\frac{\Gamma \vdash A \quad \Gamma', B \vdash \gamma}{\Gamma, \Gamma', A \multimap B \vdash \gamma} \multimap L \qquad \frac{\Gamma, A \vdash B}{\Gamma \vdash A \multimap B} \multimap R$$

Figure 4.5: A subset of the rules of the sequent calculus of ILL^{mix}.

The Horn fragment of ILL^{mix} given in Definition 11, while the complete set of rules for ILL^{mix} can be found in Section 1.4.2.

We recall the subset of the rules of the sequent calculus of ILL^{mix} relevant to our treatment in Figure 4.5, where A, B stand for a Horn formula p or clause α, while γ may also be empty (note that in rule (Neg L), $A = a$ and so $A^{\perp} = a^{\perp}$); Γ and Γ' stand for multi-sets containing Horn formulae or clauses; and Γ, Γ' is the multi-set union of Γ and Γ', assuming $\Gamma, \emptyset = \Gamma$.

Since the treatment for non-linear implications of ILL^{mix} is similar to that presented in sub-section 4.1, we feel free to only deal below with linear implications and tensor products of literals.

The following auxiliary definition of the concatenation of two automata helps to translate a H-ILL^{mix} formula. Since the automata obtained by the translation have no cycles, the following definition ignores loops.

Definition 44 (Concatenation of CA)**.** *Given two principal contract automata* $\mathcal{A}^1 = \langle Q^1, q_0^1, A^{r^1}, A^{o^1}, T^1, F^1\rangle$ *and* $\mathcal{A}^2 = \langle Q^2, q_0^2, A^{r^2}, A^{o^2}, T^2, F^2\rangle$*, their* concatenation *is*

$$\begin{aligned}\mathcal{A}^1 \cdot \mathcal{A}^2 = \quad & \langle Q^1 \cup Q^2, q_0^1, A^{r1} \cup A^{r2}, A^{o1} \cup A^{o2},\\ & T^1 \cup T^2 \setminus \{(q,a,q') \in T^1 \mid q' \in F^1\}\\ & \qquad \cup \{(q,a,q_0^2) \mid (q,a,q') \in T^1, q' \in F^1\}, F^2\rangle\end{aligned}$$

Concatenation is almost standard, with the proviso that we replace every transition of $\mathcal{A}^1$ leading to a final state with a transition with the same label leading to the initial state of $\mathcal{A}^2$. Similarly to what has been done in the previous sub-section, a tensor product is rendered as all the possible orders in which the automaton can fire (the actions corresponding to) its literals. If the literal is a positive atom, then it becomes an offer, while it originates a request if the atom is negative. A linear implication is rendered as the concatenation of the automaton coming from the premise, and that of the conclusion, with the following proviso. In the premise all the atoms are positive, but they are *all* rendered as *requests* (i.e. as negative atoms), and shuffled. The states are in correspondence with the atoms still to be fired and $\{*\}$ stands for the (final) state where all atoms have been fired.

Definition 45 (Translation of H-ILL^{mix})**.** *Given a set of atoms* X*, let* $P = \{q \cup \{*\} \mid q \in 2^X\}$ *with typical element* Z*. The translation of a H-*ILL^{mix} *formula* p *into a contract automata* $[\![p]\!]$ *is inductively defined by the following rules:*

$$[\![\bigotimes_{i\in I} \alpha_i]\!] = \boxtimes_{i\in I} [\![\alpha_i]\!]$$

$$\begin{aligned}[\![\bigotimes_{a\in X} a]\!] = & \langle P, X \cup \{*\}, \{a \mid a^\perp \in X \cap \mathbf{A}^\perp\}, \{\overline{a} \mid a \in X \cap \mathbf{A}\},\\ & \{(Z \cup \{a^\perp\}, a, Z) \mid Z \cup \{a^\perp\} \in P, a^\perp \in X\} \cup\\ & \{(Z \cup \{a\}, \overline{a}, Z) \mid Z \cup \{a\} \in P, a \in X\},\\ & \{\{*\}\}\rangle\end{aligned}$$

$$[\![\bigotimes_{b\in Y} b \multimap \bigotimes_{a\in X} a]\!] = [\![\bigotimes_{b\in Y} b^\perp]\!] \cdot [\![\bigotimes_{a\in X} a]\!]$$

Moreover, we homomorphically translate multi-sets of Horn formulae and clauses as follows:

$$[\![p, \Gamma]\!] = [\![p]\!] \boxtimes [\![\Gamma]\!] \qquad [\![\alpha, \Gamma]\!] = [\![\alpha]\!] \boxtimes [\![\Gamma]\!]$$

The automata obtained by translating the formulae representing the contracts of Alice, Bob and Charlie in Example 27 are in Figure 4.4.

Definition 46 (Honoured Sequent)**.** *A sequent* $\Gamma \vdash Z$ *is* honoured *if and only if it is provable and* Z *is a positive tensor product or empty.*

Intuitively, honoured sequents can be proved and additionally they have no negative atoms, i.e. no debts. The main result of this section is that a sequent $\Gamma \vdash Z$ is honoured if and only if the corresponding contract automaton $[\![\Gamma]\!]$ admits agreement.

The importance of this relation lies on the possibility of expressing each H-ILL^{mix} formula as a contract automaton $\mathcal{A}$, so to use our verification techniques.

In the statement below, we say that $[\![\Gamma]\!]$ admits agreement on Z whenever there exists a trace in $\mathscr{L}([\![\Gamma]\!])$ only made of match actions and offers in correspondence with the literals in Z.

Lemma 6 (Derivation Trees for Honoured Sequents)**.** *If $\Gamma \vdash Z$ is an honoured sequent, there exists a derivation tree for $\Gamma \vdash Z$ such that:*

- *it only uses the rules $Ax, Mix, NegL, \otimes L, \otimes R$ and $\multimap L$ of Figure 1.5;*
- *it is only made of honoured sequents.*

Proof. Recall that we are in the Horn fragment and we only consider cut-free proofs. Since Z is a positive tensor product (or empty), a simple inspection on the rules in Figure 1.5 suffices to prove the first statement. The second statement is proved because $Ax, Mix, NegL, \otimes L, \otimes R$ and $\multimap L$ introduce no sequents with negative literals on their right hand-side. □

The first side of the correspondence is proved now, that is if a sequent $\Gamma \vdash Z$ is honoured then the corresponding automaton $[\![\Gamma]\!]$ admits agreement on Z.

Lemma 7 (Honoured Sequents admit Agreement)**.** *Let $\Gamma \vdash Z$ be an honoured sequent, then:*

$$\Gamma \vdash Z \textit{ implies } [\![\Gamma]\!] \textit{ admits agreement on } Z.$$

Proof. We will prove that there exists a trace $w \in \mathscr{L}([\![\Gamma]\!])$ made of matches and as many offers as the literals in $Z = \bigotimes_{a \in Y} a$ (recall that they all are positive), or it is made by only matches if Z is empty. Also, note that the sequents in the proof of $\Gamma \vdash Z$ are all honoured, by hypothesis and Lemma 6. We proceed by induction on the depth of the proof of $\Gamma \vdash Z$.

In the base case, the proof consists of a single application of the rule Ax. By Definition 45 one first has an offer transition for each a in Z, and then interleaves them in any possible order. Hence the thesis holds trivially.

For the inductive case we proceed by case analysis on the last rule applied. We assume that all clauses (i.e. principals) in Γ are divided by commas, which can be easily obtained by repeatedly applying the rule $\otimes L$.

In the following, let $\bar{a}$ be offers in correspondence with the literals a in Z, we will consider only the relevant rules as stated by Lemma 6.

- $\dfrac{\Gamma \vdash \quad \Gamma' \vdash Z}{\Gamma, \Gamma' \vdash Z}\ Mix$ By induction hypothesis there exists $w \in \mathscr{L}([\![\Gamma]\!])$ with match actions, only, and $w_1 \in \mathscr{L}([\![\Gamma']\!])$ with match actions and offers in correspondence with the literals in Z (if non-empty). By Definition 26, there exists $w_2 \in$

$\mathscr{L}([\![\Gamma]\!] \boxtimes [\![\Gamma']\!])$ in agreement.

- $\dfrac{\Gamma \vdash A}{\Gamma, A^{\perp} \vdash}$ *NegL* By induction hypothesis there exists $w \in \mathscr{L}([\![\Gamma]\!])$ with match actions, and with offers in correspondence with the literals in A. By Definition 45 the traces of the automaton $[\![A^{\perp}]\!]$ are all the possible permutations of the requests in correspondence with the literals in $A^{\perp}$. The thesis follows, because there is an offer for each request, and by Definition 26.

- $\dfrac{\Gamma, A, B \vdash Z}{\Gamma, A \otimes B \vdash Z}$ $\otimes L$ By the induction hypothesis there exists $w \in \mathscr{L}([\![\Gamma, A, B]\!]) = \mathscr{L}([\![\Gamma]\!] \boxtimes [\![A]\!] \boxtimes [\![B]\!])$ with offers in correspondence with the literals in Z (if non-empty). No atom and its negation can occur in $A \otimes B$ by Definition 11, because it is a principal. Hence $[\![A \otimes B]\!]$ and $[\![A, B]\!]$ are the same automaton (with a different rank), and the statement follows immediately.

- $\dfrac{\Gamma \vdash A \quad \Gamma' \vdash B}{\Gamma, \Gamma' \vdash A \otimes B}$ $\otimes R$ By the induction hypothesis there exist $w \in \mathscr{L}([\![\Gamma]\!])$ and $w' \in \mathscr{L}([\![\Gamma']\!])$ with only match actions and offers in correspondence with the literals in A and in B, respectively. Now Definition 26 guarantees that there exists a trace in $\mathscr{L}([\![\Gamma]\!] \boxtimes [\![\Gamma']\!])$ in agreement.

- $\dfrac{\Gamma \vdash A \quad \Gamma', B \vdash Z}{\Gamma, \Gamma', A \multimap B \vdash Z}$ $\multimap L$ By the induction hypothesis there exists $w \in \mathscr{L}([\![\Gamma]\!])$ and $w' \in \mathscr{L}([\![\Gamma', B]\!])$ with only match actions and offers in correspondence with the literals in A and in Z (if non-empty), respectively. By Definition 45 the literals occurring in A become requests in $[\![A \multimap B]\!]$, in all possible ordering. The trace w contains exactly the needed matching offers. We conclude by noting that no other request is possible in $\mathscr{L}([\![\Gamma, \Gamma', A \multimap B]\!])$.

□

In order to keep the following definition compact, with a slight abuse of the notation we use $\otimes$ to operate between formulas; we remove the constraints of Definition 11 on the indexing sets I in formulas and X_1, X_2 and Y in clauses; and we let $\bigotimes_{b \in \emptyset} b \multimap \bigotimes_{a \in X_2} a$ to stand for $\bigotimes_{a \in X_2} a$.

The following definition is useful for relating an execution step of the automaton $[\![p]\!]$ with one derivation in the proof tree of p.

Definition 47 (ILL^{mix} Formula Step)**.** *Given a Horn formula p and an offer or match transition leaving the initial state of $[\![p]\!]$ with label $\vec{a}$, then define the formula $p/\vec{a}$ as:*

$$p/\vec{a} = \begin{cases} p' \otimes \bigotimes_{a_1 \in X_1} a_1 & \text{if } \vec{a} \text{ is an offer on } c \text{ and } p = p' \otimes \bigotimes_{a_1 \in X_1 \cup \{c\}} a_1 \\ p' \otimes \bigotimes_{a_1 \in X_1} a_1 \otimes \bigotimes_{a_2 \in X_2} a_2 & \text{if } \vec{a} \text{ is a match on } c \text{ and } p = p' \otimes \bigotimes_{a_1 \in X_1 \cup \{c\}} a_1 \otimes \bigotimes_{a_2 \in X_2 \cup \{c^\perp\}} a_2 \\ p' \otimes \bigotimes_{a_1 \in X_1} a_1 \otimes \bigotimes_{b \in Y} b \multimap \bigotimes_{a_2 \in X_2} a_2 & \text{if } \vec{a} \text{ is a match on } c \text{ and } p = p' \otimes \bigotimes_{a_1 \in X_1 \cup \{c\}} a_1 \otimes \bigotimes_{b \in Y \cup \{c\}} b \multimap \bigotimes_{a_2 \in X_2} a_2 \end{cases}$$

We now establish a relation between $[\![p/\vec{a}]\!]$, and the contract automaton obtained by substituting the initial state $\vec{q}_0$ of $[\![p]\!]$ with $\vec{q}$, for the transition $(\vec{q}_0, \vec{a}, \vec{q})$ of $[\![p]\!]$.

The main idea is to relate the formula $p/\vec{a}$ to the residual of the automaton $[\![p]\!]$ after the execution of an initial transition labelled by $\vec{a}$, that is $[\![p/\vec{a}]\!]$.

Without loss of generality we assume that the automaton obtained from Definition 45 is deterministic. If not, we first transform the non deterministic automaton to a deterministic one.

Lemma 8 (ILL^{mix} Step Automata)**.** *Given a Horn formula p and the contract automaton $[\![p]\!] = \langle Q, \vec{q_0}, A^r, A^o, T, F \rangle$, if $t = (\vec{q_0}, \vec{a}, \vec{q}) \in T$ is an offer or a match transition, then $\mathscr{L}(\mathcal{A}) = \mathscr{L}([\![p/\vec{a}]\!])$, where $\mathcal{A} = \langle Q, \vec{q}, A^r, A^o, T, F \rangle$.*

Proof. The proof is similar to the one of Lemma 4. The statement follows by noting that in Definition 45 a tensor product is translated in all the possible permutations of actions corresponding to the literals, and noting that in $p/\vec{a}$ we remove exactly the actions fired in $\vec{a}$, that are therefore not available any more in the state $\vec{q}$. □

The following lemma suggests that we can safely substitute a multi-set of Horn formulae and clauses Γ with a single Horn formula, without affecting the corresponding automaton. This result will be used in the proof of Lemma 10.

Lemma 9 (Multi-set Horn Formulae)**.** *Let Γ be a non-empty multi-set of Horn formulae, then there exists a Horn formula p such that:*

$$[\![\Gamma]\!] = [\![p]\!]$$

Proof. Immediate from Definition 45 (recall the slight abuse of notation). □

We now prove the following lemma, which states that if the automaton $[\![\Gamma]\!]$ obtained from the translation of Definition 45 admits agreement on Z, then the corresponding sequent $\Gamma \vdash Z$ is honoured.

Lemma 10 (Admits Agreement and Honoured Sequent)**.** *Let $\Gamma \neq \emptyset$ be a multi-set of Horn formulae and Z be a positive tensor product or empty. Then*

$[\![\Gamma]\!]$ *admits agreement on Z implies $\Gamma \vdash Z$ is an honoured sequent*

Proof. By hypothesis $w \in \mathscr{L}(\llbracket \Gamma \rrbracket)$ is a trace only composed of match and offer actions on Z. We proceed by induction on the length of w.

In the base case w has length one. Note that it is not possible to have $w = \varepsilon$ by the hypothesis $\Gamma \neq \emptyset$ and Definition 11. Moreover by Definition 11 it must be that $w = \vec{a}$ where $\vec{a}$ is a match on action a (a Horn formula must contain at least two principals). Hence by Definition 45 it must be that $Z = \emptyset$ and $\Gamma = \{\alpha \otimes \alpha'\}$ where $\alpha = a$ and $\alpha' = a^\perp$ for some literal a. Then we have:

$$\dfrac{\dfrac{\dfrac{}{a \vdash a}Ax}{a, a^\perp \vdash}Neg}{a \otimes a^\perp \vdash} \otimes L$$

For the inductive step, let $w = \vec{a}w_2$, let $\vec{q_0}$ and $\vec{f}$ be the initial and the final states of $\llbracket \Gamma \rrbracket$, then $(\vec{a}w_2, \vec{q_0}) \to (w_2, \vec{q}) \to^+ (\varepsilon, \vec{f})$. Let p be a Horn formula such that $\llbracket \Gamma \rrbracket = \llbracket p \rrbracket$ (Lemma 9), so it suffices to prove $p \vdash Z$. By the induction hypothesis and Lemma 8 we have that $\llbracket p/\vec{a} \rrbracket$ admits agreement on some Z' implies $p/\vec{a} \vdash Z'$ is honoured. To build Z from Z', we proceed by cases on $\vec{a}$:

- if $\vec{a}$ is an offer action on c we prove that $p \vdash Z$ where $Z = Z' \otimes c$. We have the following

$$\dfrac{\dfrac{\Delta'}{p \vdash (p/\vec{a}) \otimes c} \quad \dfrac{\dfrac{\Delta}{(p/\vec{a}) \vdash Z'} \quad \dfrac{}{c \vdash c}Ax}{p/\vec{a} \otimes c \vdash Z' \otimes c} \otimes R}{p \vdash Z} cut$$

  where Δ is obtained by the inductive hypothesis and for Δ' we have two cases depending on p:

  - $p = p' \otimes \bigotimes_{a_1 \in X_1 \cup \{c\}} a_1$ then the derivation $\dfrac{\Delta'}{p \vdash (p/\vec{a}) \otimes c}$ becomes

$$\dfrac{}{p' \otimes c \vdash p' \otimes c}Ax$$

    if $X_1 = \emptyset$ and the following otherwise

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{p' \vdash p'}Ax \quad \dfrac{}{\bigotimes_{a_1 \in X_1} a \vdash \bigotimes_{a_1 \in X_1} a}Ax}{p', \bigotimes_{a_1 \in X_1} a \vdash p' \otimes \bigotimes_{a_1 \in X_1} a} \otimes R \quad \dfrac{}{c \vdash c}Ax}{p', \bigotimes_{a_1 \in X_1} a, c \vdash p' \otimes \bigotimes_{a_1 \in X_1} a \otimes c} \otimes R}{p', \bigotimes_{a_1 \in X_1 \cup \{c\}} a \vdash p' \otimes \bigotimes_{a_1 \in X_1} a \otimes c} \otimes L}{p' \otimes \bigotimes_{a_1 \in X_1 \cup \{c\}} a \vdash p' \otimes \bigotimes_{a_1 \in X_1} a \otimes c} \otimes L$$

- if $\vec{a}$ is a match action we prove that $p \vdash Z$. We have the following

$$\dfrac{\dfrac{\Delta'}{p \vdash p/\vec{a}} \quad \dfrac{\Delta}{p/\vec{a} \vdash Z'}}{p \vdash Z'} cut$$

where Δ is obtained by the inductive hypothesis, $Z = Z'$ because $\vec{a}$ is a match, and for Δ' we have eight cases depending on p:

- $p = p' \otimes c \otimes c^{\perp}$

  then the derivation $\dfrac{\Delta'}{p \vdash p/\vec{a}}$ becomes:

  $$\dfrac{\dfrac{\Delta_{mix}}{p', c \otimes c^{\perp} \vdash p'} \otimes L}{p' \otimes c \otimes c^{\perp} \vdash p'} \otimes L$$

  Since the deduction tree Δ_{mix} will be also used later on, we keep it more general, by writing q for p':

  $$\Delta_{mix} \quad = \quad \dfrac{\dfrac{\dfrac{}{c \vdash c} Ax}{c, c^{\perp} \vdash} NegL \quad \dfrac{}{q \vdash q} id}{q, c, c^{\perp} \vdash q} Mix$$

- $p = p' \otimes \bigotimes_{a_1 \in X_1 \cup \{c\}} a_1 \otimes c^{\perp}$

  then, writing in Δ_{mix} $p' \otimes \bigotimes_{a_1 \in X_1} a_1$ for q the derivation $\dfrac{\Delta'}{p \vdash p/\vec{a}}$ becomes:

  $$\dfrac{\dfrac{\Delta_{mix}}{p' \otimes \bigotimes_{a_1 \in X_1 \cup \{c\}} a_1, c^{\perp} \vdash p' \otimes \bigotimes_{a_1 \in X_1} a_1} \otimes L}{p' \otimes \bigotimes_{a_1 \in X_1 \cup \{c\}} a_1 \otimes c^{\perp} \vdash p' \otimes \bigotimes_{a_1 \in X_1} a_1} \otimes L$$

- $p = p' \otimes \bigotimes_{a_2 \in X_2 \cup \{c^{\perp}\}} a_2 \otimes c$

  then, writing in Δ_{mix} $p' \otimes \bigotimes_{a_2 \in X_2} a_2$ for q the derivation $\dfrac{\Delta'}{p \vdash p/\vec{a}}$ becomes:

  $$\dfrac{\dfrac{\Delta_{mix}}{p' \otimes \bigotimes_{a_2 \in X_2 \cup \{c^{\perp}\}} a_2, c \vdash p' \otimes \bigotimes_{a_2 \in X_2} a_2} \otimes L}{p' \otimes \bigotimes_{a_2 \in X_2 \cup \{c^{\perp}\}} a_2 \otimes c \vdash p' \otimes \bigotimes_{a_2 \in X_2} a_2} \otimes L$$

- $p = p' \otimes \bigotimes_{a_1 \in X_1 \cup \{c\}} a_1 \otimes \bigotimes_{a_2 \in X_2 \cup \{c^{\perp}\}} a_2$

  then, writing in Δ_{mix} $p' \otimes \bigotimes_{a_1 \in X_1} a_1 \otimes \bigotimes_{a_2 \in X_2} a_2$ for q the derivation $\dfrac{\Delta'}{p \vdash p/\vec{a}}$ becomes:

  $$\dfrac{\dfrac{\Delta_{mix}}{p' \otimes \bigotimes_{a_1 \in X_1 \cup \{c\}} a_1 \otimes \bigotimes_{a_2 \in X_2} a_2, c^{\perp} \vdash p' \otimes \bigotimes_{a_1 \in X_1} a_1 \otimes \bigotimes_{a_2 \in X_2} a_2} \otimes L}{p' \otimes \bigotimes_{a_1 \in X_1 \cup \{c\}} a_1 \otimes \bigotimes_{a_2 \in X_2 \cup \{c^{\perp}\}} a_2 \vdash p' \otimes \bigotimes_{a_1 \in X_1} a_1 \otimes \bigotimes_{a_2 \in X_2} a_2} \otimes L$$

– $p = p' \otimes c \otimes (c \multimap \bigotimes_{a_2 \in X_2} a_2)$

then the derivation $\dfrac{\Delta'}{p \vdash p/\vec{a}}$ becomes:

$$\dfrac{\dfrac{\Delta_{ax} \quad \Delta_{\multimap}}{p', c, (c \multimap \bigotimes_{a_2 \in X_2} a_2) \vdash p' \otimes \bigotimes_{a_2 \in X_2} a_2} \otimes R}{p' \otimes c \otimes (c \multimap \bigotimes_{a_2 \in X_2} a_2) \vdash p' \otimes \bigotimes_{a_2 \in X_2} a_2} \otimes L(x2)$$

where letting $q = p'$

$$\Delta_{ax} \quad = \quad \dfrac{}{q \vdash q} Ax$$

and $\Delta_{\multimap}$ is the following proof:

$$\dfrac{\dfrac{}{c \vdash c} Ax \quad \dfrac{}{\bigotimes_{a_2 \in X_2} a_2 \vdash \bigotimes_{a_2 \in X_2} a_2} Ax}{c, (c \multimap \bigotimes_{a_2 \in X_2} a_2) \vdash \bigotimes_{a_2 \in X_2} a_2} \multimap L$$

– $p = p' \otimes \bigotimes_{a_1 \in X_1 \cup \{c\}} a_1 \otimes (c \multimap \bigotimes_{a_2 \in X_2} a_2)$

then, letting in Δ_{ax} $q = p' \otimes \bigotimes_{a_1 \in X_1} a_1$, the derivation $\dfrac{\Delta'}{p \vdash p/\vec{a}}$ becomes:

$$\dfrac{\dfrac{\Delta_{ax} \quad \Delta_{\multimap}}{p' \otimes \bigotimes_{a_1 \in X_1} a_1, c, (c \multimap \bigotimes_{a_2 \in X_2} a_2) \vdash p' \otimes \bigotimes_{a_1 \in X_1} a_1 \otimes \bigotimes_{a_2 \in X_2} a_2} \otimes R}{p' \otimes \bigotimes_{a_1 \in X_1 \cup \{c\}} a_1 \otimes (c \multimap \bigotimes_{a_2 \in X_2} a_2) \vdash p' \otimes \bigotimes_{a_1 \in X_1} a_1 \otimes \bigotimes_{a_2 \in X_2} a_2} \otimes L \text{ — twice}$$

– $p = p' \otimes \bigotimes_{a_1 \in X_1 \cup \{c\}} a_1 \otimes (\bigotimes_{b \in Y \cup \{c\}} b \multimap \bigotimes_{a_2 \in X_2} a_2)$

then writing $\hat{q}$ for $p' \otimes \bigotimes_{a_1 \in X_1} a_1 \otimes (\bigotimes_{b \in Y} b \multimap \bigotimes_{a_2 \in X_2} a_2)$ below, the derivation $\dfrac{\Delta'}{p \vdash p/\vec{a}}$ becomes:

$$\dfrac{\dfrac{\Delta_{ax} \quad \Delta_{\multimap 2}}{p' \otimes \bigotimes_{a_1 \in X_1} a_1, c, (\bigotimes_{b \in Y \cup \{c\}} b \multimap \bigotimes_{a_2 \in X_2} a_2) \vdash \hat{q}} \otimes R}{p' \otimes \bigotimes_{a_1 \in X_1 \cup \{c\}} a_1 \otimes (\bigotimes_{b \in Y \cup \{c\}} b \multimap \bigotimes_{a_2 \in X_2} a_2) \vdash \hat{q}} \otimes L(x2)$$

where $q = p' \otimes \bigotimes_{a_1 \in X_1} a_1$ in Δ_{ax}, and $\Delta_{\multimap 2}$ is the deduction tree below:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{c \vdash c} Ax \quad \dfrac{}{\bigotimes_{b \in Y} b \vdash \bigotimes_{b \in Y} b} Ax}{c, \bigotimes_{b \in Y} b \vdash \bigotimes_{b \in Y \cup \{c\}} b} \otimes R \quad \dfrac{}{\bigotimes_{a_2 \in X_2} a_2 \vdash \bigotimes_{a_2 \in X_2} a_2} Ax}{c, (\bigotimes_{b \in Y \cup \{c\}} b \multimap \bigotimes_{a_2 \in X_2} a_2), \bigotimes_{b \in Y} b \vdash \bigotimes_{a_2 \in X_2} a_2} \multimap L}{c, (\bigotimes_{b \in Y \cup \{c\}} b \multimap \bigotimes_{a_2 \in X_2} a_2) \vdash \bigotimes_{b \in Y} b \multimap \bigotimes_{a_2 \in X_2} a_2} \multimap R$$

– $p = p' \otimes c \otimes (\bigotimes_{b \in Y \cup \{c\}} b \multimap \bigotimes_{a_2 \in X_2} a_2)$

then, letting $q = p'$ in Δ_{ax}, the derivation $\dfrac{\Delta'}{p \vdash p/\vec{a}}$ becomes:

$$\dfrac{\dfrac{\Delta_{ax} \quad \Delta_{\multimap 2}}{p', c \otimes (\bigotimes_{b \in Y \cup \{c\}} b \multimap \bigotimes_{a_2 \in X_2} a_2) \vdash p' \otimes (\bigotimes_{b \in Y} b \multimap \bigotimes_{a_2 \in X_2} a_2)} \otimes R}{p' \otimes c \otimes (\bigotimes_{b \in Y \cup \{c\}} b \multimap \bigotimes_{a_2 \in X_2} a_2) \vdash p' \otimes (\bigotimes_{b \in Y} b \multimap \bigotimes_{a_2 \in X_2} a_2)} \otimes L$$

□

The main theorem of this sub-section has now an immediate proof.

Theorem 11 (ILL^{mix} Agreement)**.** *Given a multi-set of Horn formulae* Γ*, we have that*

$$\Gamma \vdash Z \textit{ is an honoured sequent if and only if } [\![\Gamma]\!] \textit{ admits agreement on } Z$$

Proof. By Lemmata 7 and 10. □

Through this result we have linked the problem of verifying the correctness of a composition of services to the generation of a deduction tree for proving H-ILL^{mix} formulae. Moreover, we have shown that the possibility of recording debts in H-ILL^{mix} can be used for solving circularity issues arising from a composition of services.

4.3 Concluding Remarks

We have investigated the relationships between contract automata and two intuitionistic logics, particularly relevant for their ability in describing the potential, but harmless and often essential circularity occurring in services. We have considered a fragment of the Propositional Contract Logic [BZ09a] particularly suited to describe contracts, and we relate it through a translation of its formulas into contract automata. Similarly, we have examined certain sequents of the Intuitionistic Linear Logic with Mix that naturally represent contracts in which all requests are satisfied. Then we have proved that these sequents are provable if and only if a suitable translation of them as contract automata admits agreement. These results allow to use both logics as formalisms for the specification of the requirements and the obligations that each service must fulfil.

Our constructions have been inspired by analogous ones [BDGZ15]; ours however offer a more flexible form of compositionality. Indeed, for checking if two separate formulas are provable, it suffices to check if the composition of the two corresponding automata is still in agreement. If the two automata are separately shown to be safe, then their composition is in agreement due to Theorem 3. In [BDGZ15] one needs to recompute the whole translation for the composed formulas, while here we propose a modular approach.

Chapter 5

Relating Contract Automata and Choreographies

The relations between an orchestrated model called contract automata (see Chapter 3), and a choreographed model called communicating machines (see Section 1.3) are investigated in this chapter. In an orchestrated model, the distributed computational components coordinate with each other by interacting with a special component, *the orchestrator*, which at runtime dictates how the computation evolves. In a choreographed model, the distributed components autonomously execute and interact with each other on the basis of a local control flow expected to comply with their role as specified in the "global viewpoint" (see Section 1.1.1).

We show that, under certain conditions, the above two models are related, in spite of the different problems they address and the different mechanisms they use for coordination.

We consider *convergent* communicating systems, that are those exhibiting successfully terminating computations only. The machines of convergent systems respect their contracts, namely they accomplish their tasks and receive what they look for. Strong safety (see Section 3.3) and convergence are key notions for semantically linking contract automata and communicating machines. We proceed as follows. We first define a mapping for translating (each principal of) a contract automaton in the corresponding (machine composing a) communicating system, and we prove that the computations of the two correspond in a precise way. In the beginning we endow communicating systems with a synchronous semantics, while the general case is considered in Section 5.2.2.

Then, if a contract automaton is strongly safe then the corresponding communicating system is convergent, and vice-versa.

To be more precise, we first establish the above semantic connection by requiring the buffers of communicating machines to contain at most one message and contract automata to well-behave on branching constructs (a notion made precise below).

Later on, by observing that *agreement* on contract automata allows a service to be compositionally placed in an unknown environment that may accept the unmatched offers of the automaton, we will show a correspondence between contract automata in

agreement and well-behaving on branches and the corresponding convergent communicating systems.

Finally, we consider the fully asynchronous semantics of communicating machines, and we prove a weaker result: a strongly safe contract automaton satisfying tighter constraints on branches represents a convergent communicating system.

A practical outcome of the results of this chapter is that contract automata enjoying (strong) agreement can execute without controller, if they are trusted. This yields the further advantage that contract automata are translated into communicating machines that run without any central control: disposing the orchestrator reduces the communication overhead.

Structure of the chapter In Section 5.1 the translation of contract automata into communicating machines is given, where we also prove our main theorem of correspondence. In Section 5.2 we extend these results by relaxing the constraints put on both kinds of automata, i.e. we consider agreement on contract automata and the fully asynchronous semantics for communicating machines. Section 5.3 works out in full detail an example. Finally, Section 5.4 concludes, and discusses possible extensions of our results.

5.1 From Contract Automata to Communicating Machines

Through this chapter we assume fixed a generic contract automaton, namely $\mathcal{A} = \langle \mathcal{Q}^n, \vec{q}_0, \mathbb{R}, \mathbb{O}, T, F \rangle$, and that the states of any automaton/machine are built out of a fixed universe $\mathcal{Q}$ (of states).

Moreover, as for communicating machines (see Section 1.3), we assume that all principal contract automata are determistic, and so are their compositions. Note that it is always possible to convert non-deterministic automata to deterministic ones.

The translation of a principal into a communicating machine is conceptually straightforward as the two automata are almost isomorphic, apart from their labels.

Recall that the principals in a contract automaton can fire transitions not matched by other principals. To account for this kind of "openness", Definition 48 below uses the new "$-$" symbol to represent a special "anonymous" participant, distinguished from those composing the contract automaton in hand, and playing the role of the environment.

For this reason, we will assume from now onwards that channel actions in Act are built on $C = \{\mathtt{pq} \mid \mathtt{p}, \mathtt{q} \in \mathcal{P} \cup \{-\} \text{ and } \mathtt{p} \neq \mathtt{q}\}$.

We now define the translation of contract automata into communicating machines.

Definition 48 (Communicating Machines Translation)**.** *For a participant* $\mathtt{p} \in \mathcal{P}$*, let* $[\![\_]\!]_{\mathtt{p}} : \mathbb{L}^n \to \mathit{Act}$ *be defined as:*

$$[\![\vec{a}]\!]_{\mathtt{p}} = \begin{cases} \overline{a}@\mathtt{ij} & \text{if } \vec{a} \text{ is a match action and } \mathtt{i} \text{ and } \mathtt{j} \text{ are such that} \\ & \vec{a}_{(\mathtt{i})} \in \mathbb{O} \text{ and } \vec{a}_{(\mathtt{j})} \in \mathbb{R} \text{ and } \mathtt{p} = \mathtt{i} \\ a@\mathtt{ij} & \text{if } \vec{a} \text{ is a match action and } \mathtt{i} \text{ and } \mathtt{j} \text{ are such that} \\ & \vec{a}_{(\mathtt{i})} \in \mathbb{O} \text{ and } \vec{a}_{(j)} \in \mathbb{R} \text{ and } \mathtt{p} = \mathtt{j} \\ \overline{a}@\mathtt{i}- & \text{if } \vec{a} \text{ is an offer action and } \mathtt{i} \text{ is such that } \vec{a}_{(\mathtt{i})} \in \mathbb{O} \text{ and } \mathtt{p} = \mathtt{i} \\ a@-\mathtt{j} & \text{if } \vec{a} \text{ is a request action and } \mathtt{j} \text{ is such that } \vec{a}_{(\mathtt{j})} \in \mathbb{R} \text{ and } \mathtt{p} = \mathtt{j} \\ \varepsilon & \text{otherwise} \end{cases}$$

The translation of $\mathcal{A}$ to a communicating finite state machine *is given by the map*

$$[\![\mathcal{A}]\!]_{\mathtt{p}} = \langle \mathcal{Q}, \vec{q0}_{(p)}, \mathit{Act}, \{(\vec{q}_{(p)}, [\![\vec{a}]\!]_{\mathtt{p}}, \vec{q'}_{(p)}) \mid (\vec{q}, \vec{a}, \vec{q'}) \in T \text{ and } [\![\vec{a}]\!]_{\mathtt{p}} \neq \varepsilon\}, \prod\nolimits^{\mathtt{p}}(F) \rangle$$

The communicating system (see Definition 6) corresponding *to the contract automaton* $\mathcal{A}$ *is* $S(\mathcal{A}) = ([\![\mathcal{A}]\!]_{\mathtt{p}})_{\mathtt{p} \in \{1,\ldots,n\}}$.

The following example illustrates the composition of three contract automata, its most permissive controller, and its translation into a system of three communicating machines.

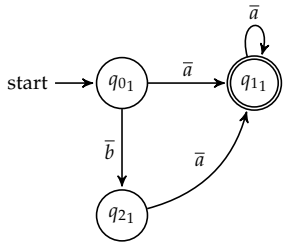
Example 28. *Figure 5.1 shows three contract automata* A*,* B*, and* C*, their product* $\mathtt{A} \otimes \mathtt{B} \otimes \mathtt{C}$*, with initial state* $\vec{q}_0 = \langle q_{0_1}, q_{0_2}, q_{0_3} \rangle$*, and the corresponding most permissive strong controller* $\mathcal{KS}_{\mathtt{A}\otimes\mathtt{B}\otimes\mathtt{C}}$*. The translations* $[\![\mathcal{KS}_{\mathtt{A}\otimes\mathtt{B}\otimes\mathtt{C}}]\!]_{\mathtt{A}}$*,* $[\![\mathcal{KS}_{\mathtt{A}\otimes\mathtt{B}\otimes\mathtt{C}}]\!]_{\mathtt{B}}$*, and* $[\![\mathcal{KS}_{\mathtt{A}\otimes\mathtt{B}\otimes\mathtt{C}}]\!]_{\mathtt{C}}$ *as per Definition 48 yield the three communicating machines in Figure 1.3.*

We constrain the behaviour of communicating machines, so to make it easier reflecting in this model the notion of strong agreement defined on contract automata. Intuitively, the new semantics, called *1-buffer* semantics, only allows a machine $M_{\mathtt{p}}$ to send a message $\overline{a}$ to a partner $M_{\mathtt{q}}$ if in the communicating system S all the channels are empty. To specify the *1-buffer* semantics, it suffices to constrain the component $\vec{u}_{(\mathtt{pq})}$ of the reachable configurations $RS(S)$ of S (see Definition 7), and only keeping the transitions between the selected configurations. We also define *convergent* communicating systems that always reach a final configuration and so they are *deadlock-free*. Note that a deadlock-free system may not be convergent, because of the presence of possible livelocks, while convergent systems are deadlock-free. Indeed, convergence is a stronger requirement than the absence of deadlocks. In the following, we will prove that under certain conditions the system of communicating machines obtained from Definition 48 is convergent.

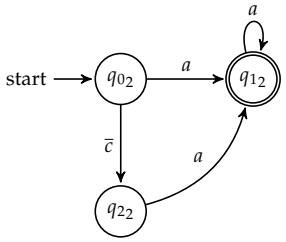
Definition 49 (1-buffer, convergence, deadlock)**.** *Let* $S = (M_{\mathtt{p}})_{\mathtt{p} \in \mathcal{P}}$ *be a communicating system. A configuration* $s = (\vec{q}; \vec{u})$ *of* S *is* stable *if and only if* $\vec{u} = \vec{\varepsilon}$*; additionally,* s *is* final *if it is stable and* $\vec{q} \in (F_{\mathtt{p}})_{\mathtt{p} \in \mathcal{P}}$*.*

The transition relation of the 1-buffer semantics *of* S *is the following*

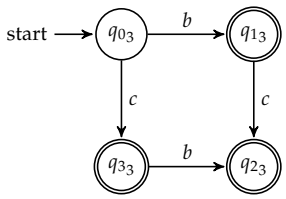
$$\twoheadrightarrow = \rightarrow \cap (RS_{\leq 1}(S) \times \mathit{Act} \times RS_{\leq 1}(S))$$



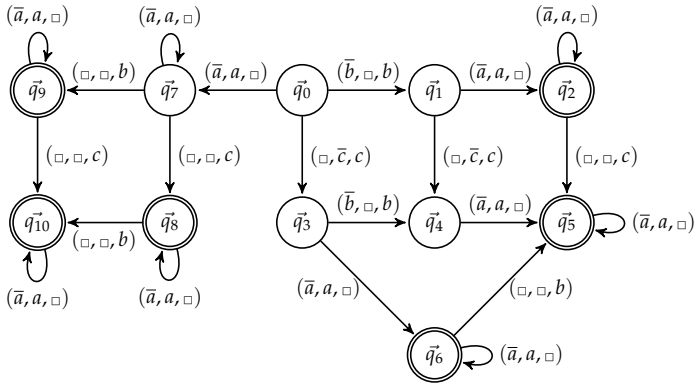
(a) A



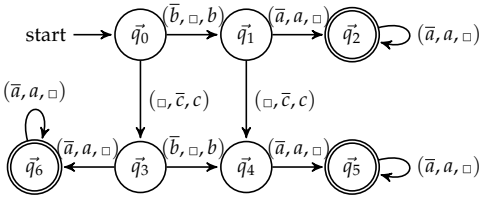
(b) B



(c) C



(d) $\mathtt{A} \otimes \mathtt{B} \otimes \mathtt{C}$



(e) $\mathcal{KS}_{\mathtt{A}\otimes\mathtt{B}\otimes\mathtt{C}}$

The communicating machines $[\![\mathcal{KS}_{\mathtt{A}\otimes\mathtt{B}\otimes\mathtt{C}}]\!]_{\mathtt{A}}, [\![\mathcal{KS}_{\mathtt{A}\otimes\mathtt{B}\otimes\mathtt{C}}]\!]_{\mathtt{B}}, [\![\mathcal{KS}_{\mathtt{A}\otimes\mathtt{B}\otimes\mathtt{C}}]\!]_{\mathtt{C}}$ corresponding to the contract automata A, B and C are showed in Figure 1.3.

Figure 5.1: The contract automata of Examples 28 and 30.

where $\to$ is the relation introduced in Definition 7 and

$$RS_{\leq 1}(S) = \{(\vec{q};\vec{u}) \in RS(S) \mid (\vec{q};\vec{u}) \text{ is stable or}$$

$$\exists \mathtt{pq} \in C : \ \exists a \in \mathbb{R} : \ \vec{u}_{(\mathtt{pq})} = a \wedge \forall \mathtt{sr} \neq \mathtt{pq}.\vec{u}_{(\mathtt{sr})} = \varepsilon\}$$

We say that the system S is convergent *(with the 1-buffer semantics) if and only if for every reachable configuration $(\vec{q};\vec{u}) \in RS_{\leq 1}(S)$, there exists a final configuration s such that*

$$(\vec{q};\vec{u}) \twoheadrightarrow^* s.$$

Moreover a configuration $(\vec{q};\vec{u})$ is a deadlock *if and only if it is not final and $(\vec{q};\vec{u}) \not\twoheadrightarrow$.*

In order to relate the computations of contract automata and those of communicating systems, it is convenient to define a translation from the first to the second ones, as follows.

Definition 50 (Traces Translation)**.** *Given a sequence of n-tuples of actions $\varphi \in (\mathbb{L}^n)^*$, we define*

$$[\![\varphi]\!] = \begin{cases} \overline{a}@\mathtt{ij}\ a@\mathtt{ij}[\![\varphi']\!] & \text{if } \varphi = \vec{a}\varphi' \text{ and } \vec{a} \text{ is a match action on } a \\ & \text{with } \vec{a}_{(\mathtt{i})} \in \mathbb{O} \text{ and } \vec{a}_{(\mathtt{j})} \in \mathbb{R} \\ \overline{a}@\mathtt{i}-[\![\varphi']\!] & \text{if } \varphi = \vec{a}\varphi' \text{ and } \vec{a} \text{ is an offer action on } a \\ & \text{with } \vec{a}_{(\mathtt{i})} \in \mathbb{O} \\ a@-\mathtt{j}[\![\varphi']\!] & \text{if } \varphi = \vec{a}\varphi' \text{ and } \vec{a} \text{ is a request action on } a \\ & \text{with } \vec{a}_{(\mathtt{j})} \in \mathbb{R} \\ \varepsilon & \text{if } \varphi = \varepsilon \\ \text{undefined} & \text{otherwise} \end{cases}$$

Now we are ready to establish a first property showing how a sequence of channel actions labelling a computation of the corresponding communicating system, is mapped into a trace in strong agreement, i.e. a string of matches.

Property 6. *Let $S(\mathcal{CS}_{\mathcal{A}})$ be the communicating system corresponding to $\mathcal{CS}_{\mathcal{A}}$, and let s_0 be its initial configuration. If $s_0 \overset{f}{\twoheadrightarrow} s$, then*

- *$f \in Act^*$ is a sequence of pairs $(\overline{a}@\mathtt{ij}\ a@\mathtt{ij})$ possibly followed by $\overline{a'}@\mathtt{i'j'}$, for some $a, a', \mathtt{i}, \mathtt{j}, \mathtt{i'}, \mathtt{j'}$, and*
- *there exists a strong agreement φ such that either $f = [\![\varphi]\!]$ or $f = [\![\varphi]\!]\overline{a'}@\mathtt{i'j'}$.*

Proof. The first item follows from Definition 49, and the second is then immediate by Definition 50. □

Before establishing our main results, we introduce a notion of well-formedness of contract automata. We require that if an output of a principal $\mathtt{i}$ is enabled in two different states, it can be taken in both, so generating the same match, regardless of the (projection of these) states on the other principals $\mathtt{j} \neq \mathtt{i}$ in the product automaton.

In what follows, it is convenient to highlight the principal of a contract automaton that makes an offer or a request. For that, we define $snd(\vec{a}) = \mathtt{i}$ when $\vec{a}$ is a match action or an offer action and $\vec{a}_{(\mathtt{i})} \in \mathbb{O}$; similarly, let $rcv(\vec{a}) = \mathtt{j}$ when $\vec{a}$ is a match or a request action and $\vec{a}_{(\mathtt{j})} \in \mathbb{R}$. Also, we will omit the target configuration of a transition when immaterial, e.g. $\vec{q} \xrightarrow{\vec{a}}$ or $s \xrightarrow{\ell}$.

Definition 51 (Branching Condition)**.** *A contract automaton $\mathcal{A}$ has the* branching condition *if and only if the following holds for each $\vec{q}_1, \vec{q}_2$ reachable in $\mathcal{A}$*

$$\forall \vec{a} \text{ match action } .(\vec{q}_1 \xrightarrow{\vec{a}} \wedge\, snd(\vec{a}) = \mathtt{i} \wedge \vec{q}_{1(i)} = \vec{q}_{2(i)}) \text{ implies } \vec{q}_2 \xrightarrow{\vec{a}}$$

Example 29. *The product automaton $\mathtt{A} \otimes \mathtt{B} \otimes \mathtt{C}$ of Example 28 enjoys the branching condition. Indeed, consider the match action $(\overline{a}, a, \square)$ and the transition $\vec{q_0} \xrightarrow{(\overline{a}, a, \square)}$. We have that $\vec{q_0}_{(1)} = \vec{q_3}_{(1)} = q_{0_1}$ and there also exists the transition $\vec{q_3} \xrightarrow{(\overline{a}, a, \square)}$. The same happens for:*

$$\vec{q_1} \xrightarrow{(\overline{a}, a, \square)}, \vec{q_4} \xrightarrow{(\overline{a}, a, \square)} \qquad \vec{q_2} \xrightarrow{(\overline{a}, a, \square)}, \vec{q_5} \xrightarrow{(\overline{a}, a, \square)} \qquad \vec{q_6} \xrightarrow{(\overline{a}, a, \square)}, \vec{q_5} \xrightarrow{(\overline{a}, a, \square)}$$

$$\vec{q_7} \xrightarrow{(\overline{a}, a, \square)} \vec{q_9} \xrightarrow{(\overline{a}, a, \square)} \qquad \vec{q_7} \xrightarrow{(\overline{a}, a, \square)}, \vec{q_8} \xrightarrow{(\overline{a}, a, \square)} \qquad \vec{q_9} \xrightarrow{(\overline{a}, a, \square)}, \vec{q_{10}} \xrightarrow{(\overline{a}, a, \square)}$$

$$\vec{q_8} \xrightarrow{(\overline{a}, a, \square)}, \vec{q_{10}} \xrightarrow{(\overline{a}, a, \square)} \qquad \vec{q_0} \xrightarrow{(\overline{b}, \square, b)}, \vec{q_3} \xrightarrow{(\overline{b}, \square, b)} \qquad \vec{q_0} \xrightarrow{(\square, \overline{c}, c)}, \vec{q_1} \xrightarrow{(\square, \overline{c}, c)}$$

Instead, the most permissive strong controller of $\mathtt{A} \otimes \mathtt{B} \otimes \mathtt{C}$ does not enjoy the branching condition. Consider again the match action $(\overline{a}, a, \square)$: while $\vec{q_0}_{(1)} = \vec{q_3}_{(1)} = q_{0_1}$ and $\vec{q_3} \xrightarrow{(\overline{a}, a, \square)}$, no transition labelled by $(\overline{a}, a, \square)$ exits from $\vec{q_0}$.

Theorem 12 characterises the relations between a contract automaton $\mathcal{A}$, its most permissive strong controller $\mathcal{KS}_{\mathcal{A}}$ and the corresponding communicating system $S(\mathcal{KS}_{\mathcal{A}})$. It states that $S(\mathcal{KS}_{\mathcal{A}})$ is capable of performing all the moves of the controller (item 1), while $\mathcal{A}$, but possibly *not* $\mathcal{KS}_{\mathcal{A}}$, can perform all the traces of $S(\mathcal{KS}_{\mathcal{A}})$ in strong agreement (item 2a). Moreover the runs of $S(\mathcal{KS}_{\mathcal{A}})$ leading to a configuration from which no final configuration is reachable correspond to runs in $\mathcal{A}$ that traverse strongly liable transitions (item 2b). The following example illustrates the relations between convergence and deadlock freedom discussed above on the automata in Figure 5.1.

Example 30. *Consider the contract automata and communicating machines of Figure 5.1 and Figure 1.3. A trace of the system $S(\mathcal{KS}_{\mathtt{A}\otimes\mathtt{B}\otimes\mathtt{C}})$ is (assuming that $\mathtt{AB}$ is the first element of $\vec{u}$):*

$$((q_{0_1}, q_{0_2}, q_{0_3}); (\varepsilon, \varepsilon, \varepsilon, \varepsilon, \varepsilon, \varepsilon)) \xrightarrow{\overline{a}@\mathtt{AB}}$$

$$((q_{1_1}, q_{0_2}, q_{0_3}); (a, \varepsilon, \varepsilon, \varepsilon, \varepsilon, \varepsilon)) \xrightarrow{a@\mathtt{AB}}$$

$$((q_{1_1}, q_{1_2}, q_{0_3}); (\varepsilon, \varepsilon, \varepsilon, \varepsilon, \varepsilon, \varepsilon)) = s$$

We have $[\![(\overline{a}, a, \square)]\!] = \overline{a}@\mathtt{AB}\ a@\mathtt{AB}$, and the contract automaton $\mathtt{A} \otimes \mathtt{B} \otimes \mathtt{C}$ is capable of performing the transition $(\vec{q}_0, (\overline{a}, a, \square), \vec{q}_7)$, while this is not true for the controller $\mathcal{KS}_{\mathtt{A}\otimes\mathtt{B}\otimes\mathtt{C}}$.

Note that from the configuration s it is not possible to reach a final configuration, since the participant $\mathtt{C}$ is prevented from reaching a final state, and thus the transition $(\vec{q}_0, (\overline{a}, a, \square), \vec{q}_7)$ of $\mathtt{A} \otimes \mathtt{B} \otimes \mathtt{C}$ is strongly liable.

Moreover s is not a deadlock, indeed it is always possible to perform the loop:

$$((q_{1_1}, q_{1_2}, q_{0_3}); (\varepsilon, \varepsilon, \varepsilon, \varepsilon, \varepsilon, \varepsilon)) \xrightarrow{\overline{a}@\mathtt{AB}}$$

$$((q_{1_1}, q_{1_2}, q_{0_3}); (a, \varepsilon, \varepsilon, \varepsilon, \varepsilon, \varepsilon)) \xrightarrow{a@\mathtt{AB}}$$

$$((q_{1_1}, q_{1_2}, q_{0_3}); (\varepsilon, \varepsilon, \varepsilon, \varepsilon, \varepsilon, \varepsilon))$$

As a matter of fact $S(\mathcal{KS}_{\mathtt{A}\otimes\mathtt{B}\otimes\mathtt{C}})$ is deadlock-free but not convergent.

It is convenient to introduce an *equivalence* between the states of a contract automaton and those of a configuration of a communicating system (assuming them ordered by their indexes). This equivalence will be exploited in Theorem 12 for relating the traces of a communicating system and those of the corresponding contract automaton.

Definition 52 (States Equivalence)**.** *Let $(\vec{q}; \vec{u})$ be a reachable configuration of a communicating system, and let $\vec{q'}$ be a state of a contract automaton $\mathcal{A}$. Then we let*

$$\vec{q} \sim \vec{q'} \text{ iff } |\vec{q}| = |\vec{q'}| = n \text{ and } \forall i \in 1 \dots n. \vec{q}_{(i)} = \vec{q'}_{(i)}$$

Theorem 12 (Correspondence of Traces)**.** *Let $\mathcal{A}$ be a contract automaton with initial state $\vec{q}_0$; let $\mathcal{KS}_{\mathcal{A}}$ be its most permissive strong controller with initial state $\vec{q}_{mpc}$; let $S(\mathcal{KS}_{\mathcal{A}})$ be the corresponding communicating system with initial configuration s_0. Then, given a strong agreement $\varphi \in \mathfrak{Z}$, the following hold:*

1. *if $\vec{q}_{mpc} \xrightarrow{\varphi} \vec{q'}_{mpc}$, then there exists a configuration s such that $s_0 \overset{[\![\varphi]\!]}{\twoheadrightarrow} s = (\vec{q_s}; \vec{u})$ and $\vec{q'}_{mpc} \sim \vec{q_s}$;*
2. *if $s_0 \overset{f}{\twoheadrightarrow} s = (\vec{q_s}; \vec{u})$, then*
   (a) *if $f = [\![\varphi]\!]$, then there exists $\vec{q'}$ such that $\vec{q}_0 \xrightarrow{\varphi} \vec{q'}$ and $\vec{q_s} \sim \vec{q'}$;*
   (b) *if no final configuration is reachable from s, with either $f = [\![\varphi]\!]$ or $f = [\![\varphi]\!]\overline{a}@\mathtt{ij}$, then the run $\vec{q}_0 \xrightarrow{\hat{\varphi}} \vec{q'}$ has traversed a transition in $TSLiable(\mathcal{A})$ with either $\hat{\varphi} = \varphi$ or $\hat{\varphi} = \varphi\vec{a}$, where $\vec{a}$ is a match on a and $snd(\vec{a}) = \mathtt{i}, rcv(\vec{a}) = \mathtt{j}$.*

Proof. Through the proof assume that $\vec{q'}$, $\vec{q'}_{mpc}$ and s are such that $\vec{q}_0 \xrightarrow{\varphi} \vec{q'}$, $\vec{q}_{mpc} \xrightarrow{\varphi} \vec{q'}_{mpc}$ and $s_0 \overset{[\![\varphi]\!]}{\twoheadrightarrow} s = (\vec{q_s}; \vec{u})$. Also, let $\vec{a}$ be a match with $snd(\vec{a}) = \mathtt{i}$ and $rcv(\vec{a}) = \mathtt{j}$.

1. By induction on the length of φ.

   The base case is when $\vec{q}_{mpc} \xrightarrow{\vec{a}} \vec{q'}_{mpc}$. Let $\vec{q}_{mpc(\mathtt{i})}$ and $\vec{q}_{mpc(\mathtt{j})}$ be the initial states of participants $\mathtt{i}$ and $\mathtt{j}$ in $S(\mathcal{KS}_{\mathcal{A}})$. By Definition 48, we have that

   $$(\vec{q}_{mpc(\mathtt{i})}, \overline{a}@\mathtt{ij}, \vec{q'}_{mpc(\mathtt{i})}) \qquad \text{and} \qquad (\vec{q}_{mpc(\mathtt{j})}, a@\mathtt{ij}, \vec{q'}_{mpc(\mathtt{j})})$$

   are transitions of participants $\mathtt{i}$ and $\mathtt{j}$, respectively. We have $s_0 \overset{\overline{a}@\mathtt{ij}}{\twoheadrightarrow} s_i \overset{a@\mathtt{ij}}{\twoheadrightarrow} s$ since after the first transition participant $\mathtt{j}$ remains in its initial state. Moreover by

Definition 48 and Definition 24 we have $\vec{q'}_{mpc(\mathtt{i})} = \vec{q}_{s(\mathtt{i})}$ and $\vec{q'}_{mpc(\mathtt{j})} = \vec{q}_{s(\mathtt{j})}$, all the other components of the states remain in their initial state, and thus $\vec{q'}_{mpc} \sim \vec{q}_s$.

For the inductive case we have $\varphi = \vec{a}\,\varphi'$, and the run $\vec{q}_{mpc} \xrightarrow{\vec{a}} \vec{q''}_{mpc} \xrightarrow{\varphi} \vec{q'}_{mpc}$ for some $\vec{q''}_{mpc}$. The same argument used above guarantees that there exists $s'' = (\vec{q_{s''}}; \vec{u''})$ and $\vec{q_{s''}} \sim \vec{q''}_{mpc}$, such that $s_0 \overset{\overline{a}@\mathtt{ij}}{\twoheadrightarrow} s_i \overset{a@\mathtt{ij}}{\twoheadrightarrow} s'' \overset{[\![\varphi']\!]}{\twoheadrightarrow} s$ and the induction hypothesis suffices.

2. The proof of the statement 2a is by induction on the length of f and the proof of 2b is by contradiction.

   (a) By hypothesis the base case is when $s_0 \overset{\overline{a}@\mathtt{ij}}{\twoheadrightarrow} s_i \overset{a@\mathtt{ij}}{\twoheadrightarrow} s$, in other words $s_0 \overset{[\![\vec{a}]\!]}{\twoheadrightarrow} s$. Now, in order to obtain $S(\mathcal{KS}_{\mathcal{A}})$ through Definition 48 there must exist two automata such that $\vec{q}_{(\mathtt{i})} = \vec{q}_{s(\mathtt{i})}$ and $\vec{q}_{(\mathtt{j})} = \vec{q}_{s(\mathtt{j})}$. Consequently, the product automaton $\mathcal{A}$ has the transition $(\vec{q_0}, \vec{a}, \vec{q})$ where $\forall \mathtt{k} \neq \mathtt{i}, \mathtt{j}.\ \vec{q_0}_{(\mathtt{k})} = \vec{q}_{(\mathtt{k})}$. Thus $\vec{q} \sim \vec{q}_s$.

   For the inductive case we have $\varphi = \vec{a}\,\varphi'$ and the computation $s_0 \overset{[\![\vec{a}]\!]}{\twoheadrightarrow} s'' \overset{[\![\varphi']\!]}{\twoheadrightarrow} s$ where $s'' = (\vec{q_{s''}}; \vec{u''})$. The same argument used above guarantees that there exists a transition $(\vec{q_0}, \vec{a}, \vec{q''})$ and $\vec{q''} \sim \vec{q_{s''}}$ and we conclude by applying the induction hypothesis. Note that there is only one s such that $s_0 \overset{[\![\vec{a}]\!]}{\twoheadrightarrow} s_i \overset{[\![\varphi']\!]}{\twoheadrightarrow} s$, and only one $\vec{q'}_{mpc}$ such that $\vec{q}_{mpc} \xrightarrow{\vec{a}} \xrightarrow{\varphi'} \vec{q'}_{mpc}$; because we are considering deterministic automata only.

   (b) Assume by contradiction that $\vec{q}_0 \xrightarrow{\hat{\varphi}} \vec{q'}$ has traversed no liable transitions. Then by Definition 36 there exists φ' such that $\vec{q'} \xrightarrow{\varphi'} \vec{q''}$ and $\vec{q''}$ is a final configuration. By Definition 35 we must have $\vec{q}_{mpc} \xrightarrow{\hat{\varphi}\varphi'} \vec{q'}_{mpc}$ where $\vec{q'}_{mpc}$ is a final configuration.

   Hence by applying the first item of Theorem 12 we have $s \overset{f'}{\twoheadrightarrow} s''$ where s'' is a final configuration and $f' = [\![\varphi']\!]$ or $f' = a@\mathtt{ij}[\![\varphi']\!]$, obtaining a contradiction.

□

As a side comment to item 2b, we can also prove that if s is a deadlock configuration (and either $f = [\![\varphi]\!]$ or $f = [\![\varphi]\!]\overline{a}@\mathtt{ij}$ for some $\varphi \in \mathfrak{Z}$) and $\mathcal{A}$ enjoys the branching condition, then there exists a run $\vec{q}_0 \xrightarrow{\hat{\varphi}} \vec{q'}$ traversing a transition in $TSLiable(\mathcal{A})$ with either $\hat{\varphi} = \varphi$ or $\hat{\varphi} = \varphi\vec{a}$, where $\vec{a}$ is a match on a and $snd(\vec{a}) = \mathtt{i}, rcv(\vec{a}) = \mathtt{j}$. Recall that, if a contract automaton $\mathcal{A}$ fires through a liable transition, it can be that $S(\mathcal{KS}_{\mathcal{A}})$ never reaches a deadlock configuration, as shown by Example 30.

We are now ready to state one of our main result: the most permissive strong controller of a contract automaton has the branching condition if and only if the corresponding communicating system is convergent.

Theorem 13 (Convergence and Branching Condition)**.** *Let $\mathcal{A}$ be a contract automaton, $\mathcal{KS}_{\mathcal{A}}$ be its most permissive strong controller, and $S(\mathcal{KS}_{\mathcal{A}})$ be the corresponding communicating system. Then*

$$S(\mathcal{KS}_{\mathcal{A}}) \textit{ is convergent if and only if } \mathcal{KS}_{\mathcal{A}} \textit{ satisfies the branching condition.}$$

Proof. Let $\vec{q}_0$, $\vec{q}_{mpc}$, and s_0 be the initial configurations of $\mathcal{A}$, $\mathcal{KS}_{\mathcal{A}}$, and $S(\mathcal{KS}_{\mathcal{A}})$, respectively.
($\Rightarrow$) By contradiction assume that the branching condition does not hold in $\mathcal{KS}_{\mathcal{A}}$, i.e. in $\mathcal{KS}_{\mathcal{A}}$ $\vec{q}_1 \xrightarrow{\vec{a}} \vec{q''}$, $\vec{q}_2 \not\xrightarrow{\vec{a}}$ where $\vec{a}$ is a match on a with $snd(\vec{a}) = \mathtt{i}$, $rcv(\vec{a}) = \mathtt{j}$ for some $\mathtt{i}, \mathtt{j} \in \mathcal{P}$ and $\vec{q_1}_{(i)} = \vec{q_2}_{(i)}$. Suppose that φ and φ' are such that $\vec{q}_{mpc} \xrightarrow{\varphi} \vec{q}_1$ and $\vec{q}_{mpc} \xrightarrow{\varphi'} \vec{q}_2$.

By the first item of Theorem 12 we have $s_0 \xrightarrow{[\![\varphi]\!]} s \xrightarrow{\overline{a}@\mathtt{ij}} \hat{s} \xrightarrow{a@\mathtt{ij}} s'' = (\vec{q_{s''}}; \vec{u})$ with $\vec{q_{s''}} \sim \vec{q''}$ and $s_0 \xrightarrow{[\![\varphi']\!]} s'$. Moreover, since by hypothesis $\vec{q_1}_{(i)} = \vec{q_2}_{(i)}$, we also have $s' \xrightarrow{\overline{a}@\mathtt{ij}} \bar{s}$. The computation $\bar{s} \xrightarrow{a@\mathtt{ij}[\![\varphi_2]\!]} s_f$, for any φ_2 and s_f final, is not possible, because otherwise by item 2a of Theorem 12 we would have $\vec{q}_2 \xrightarrow{\vec{a}\varphi_2} \vec{q}_f$, with $\vec{q}_f$ final state of the automaton $\mathcal{A}$, as well of $\mathcal{KS}_{\mathcal{A}}$. This would be a contradiction, because $\vec{q}_2 \not\xrightarrow{\vec{a}}$ by hypothesis. Hence $S(\mathcal{KS}_{\mathcal{A}})$ is not convergent, since from $\bar{s}$ it is not possible to reach a final configuration.

($\Leftarrow$) Assume by contradiction that $S(\mathcal{KS}_{\mathcal{A}})$ is not convergent, i.e. $s_0 \overset{f}{\twoheadrightarrow} s = (\vec{q_s}; \vec{u})$ and no final configurations are reachable from s. By Property 6, f is either $f = [\![\varphi]\!]$ or $f = [\![\varphi]\!]\overline{a'}@\mathtt{i'j'}$. Hence by applying item 2b of Theorem 12 we have that $\vec{q}_0 \xrightarrow{\hat{\varphi}} \vec{q'}$ has traversed a transition in $TSLiable(\mathcal{A})$ with either $\hat{\varphi} = \varphi$ or $\hat{\varphi} = \varphi\vec{a'}$. Hence $\hat{\varphi} = \varphi_1\vec{a}\varphi''$ for some $\varphi_1, \vec{a}, \varphi''$ such that $\vec{q}_0 \xrightarrow{\varphi_1} \vec{q}_1$ is a run of $\mathcal{A}$ and $(\vec{q}_1, \vec{a}, \vec{q'_1}) \in TSLiable(\mathcal{A})$, with $snd(\vec{a}) = \mathtt{i}$, $rcv(\vec{a}) = \mathtt{j}$ for some $\mathtt{i}, \mathtt{j} \in \mathcal{P}$.

There exists then a (sub-)computation $s_0 \overset{[\![\varphi_1]\!]}{\twoheadrightarrow} s' = (\vec{q_{s'}}; \vec{u'}) \overset{\overline{a}@\mathtt{ij}}{\twoheadrightarrow}$, and by item 2a of Theorem 12 we have $\vec{q_{s'}} \sim \vec{q}_1$. Since we obtained $S(\mathcal{KS}_{\mathcal{A}})$ through Definition 48, the transition $s' \overset{\overline{a}@\mathtt{ij}}{\twoheadrightarrow}$ has been originated by a transition $(\vec{q_2}, \vec{a}, \vec{q_3})$ in $\mathcal{KS}_{\mathcal{A}}$, for some $\vec{q_2}$ such that $\vec{q_2}_{(i)} = \vec{q_1}_{(i)}$. We have that $\vec{q}_1 \neq \vec{q}_2$ because $(\vec{q_1}, \vec{a}, \vec{q'_1}) \in TSLiable(\mathcal{A})$. The transitions of $\mathcal{KS}_{\mathcal{A}}$ include $\vec{q}_2 \xrightarrow{\vec{a}}$, but not $\vec{q}_1 \xrightarrow{\vec{a}}$ and since $\vec{q_2}_{(i)} = \vec{q_1}_{(i)}$, $\mathcal{KS}_{\mathcal{A}}$ violates the branching condition, against our hypothesis. □

A consequence of Theorem 13 is that a *strongly safe* contract automaton has the branching condition if and only if its corresponding communicating system is convergent.

Corollary 2. *Let $\mathcal{A}$ be a contract automaton, then*

$$\mathcal{A} \textit{ is strongly safe and satisfies the branching condition if and only if } S(\mathcal{A}) \textit{ is convergent.}$$

Proof. The statement follows by applying Theorem 13, because if $\mathcal{A}$ is strongly safe then $\mathcal{A} = \mathcal{KS}_{\mathcal{A}}$, hence $\mathcal{KS}_{\mathcal{A}}$ has the branching condition. □

We have fully characterised the conditions under which the orchestration of services can be safely translated in a distributed synchronous choreography. Note that the principals are oblivious of their partners, and computing the most permissive controller is a

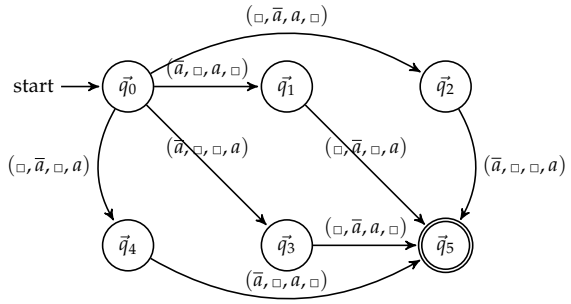


Figure 5.2: $\mathcal{KS}_{\mathcal{A}}$

necessary step for matching the requests and offers of each service, as well as removing the liable transitions. Once the orchestration has been computed, if each principal executes its transitions *independently* of the state of the other principals (i.e. the branching condition holds), then the central control can be removed. In this case, it is possible to compute the system of communicating machines, which are now equipped with the information necessary to execute synchronously without central control, so avoiding communications overhead. Intuitively, all the possible behaviours exposed by the system of communicating machines are exactly those computed by the corresponding most permissive controller.

Example 31. *Consider the automaton* $\mathcal{A} = \mathtt{A} \otimes \mathtt{B} \otimes \mathtt{C} \otimes \mathtt{D}$ *depicted in Figure 5.2. Both principals* A *and* B *only offer* $\overline{a}$ *and then stop, while the other principals* B *and* C *only perform the complementary request* a*.*

The contract automaton $\mathcal{A}$ *is strongly safe, but it does not enjoy the branching condition. As an example of violation, notice that the state of* B *in both* $\vec{q}_1, \vec{q}_3$ *is the same, i.e.* $\vec{q}_{1(2)} = \vec{q}_{3(2)}$*. But the match transition* $(\square, \overline{a}, \square, a)$ *is possible from state* $\vec{q}_1$*, and not from the state* $\vec{q}_3$*; also from state* $\vec{q}_3$ *we can fire the match transition* $(\square, \overline{a}, a, \square)$*, which is not available in state* $\vec{q}_1$*.*

The translation of $\mathcal{A}$ *yields the communicating machines:*

$$[\![\mathcal{KS}_{\mathcal{A}}]\!]_{\mathtt{A}} = \overline{a}@\mathtt{AC} + \overline{a}@\mathtt{AD} \qquad [\![\mathcal{KS}_{\mathcal{A}}]\!]_{\mathtt{B}} = \overline{a}@\mathtt{BC} + \overline{a}@\mathtt{BD}$$

$$[\![\mathcal{KS}_{\mathcal{A}}]\!]_{\mathtt{C}} = a@\mathtt{AC} + a@\mathtt{BC} \qquad [\![\mathcal{KS}_{\mathcal{A}}]\!]_{\mathtt{D}} = a@\mathtt{AD} + a@\mathtt{BD}$$

A deadlock configuration is generated by the trace $\overline{a}@\mathtt{AC}.a@\mathtt{AC}.\overline{a}@\mathtt{BC}$*.*

5.2 Agreement and Asynchrony

We now extend the previous results along two lines. First we consider the more permissive notion of agreement on contract automata introduced in Chapter 3. Then, we drop the constraints on the number of messages that a buffer can contain, and we consider the fully asynchronous semantics of communicating machines of Definition 7.

5.2.1 Agreement

The notion of agreement (Definition 27) considers as forbidden actions only the non-matching requests while for strong agreement (Definition 34) both non-matching requests and offers are forbidden.

Intuitively, the notion of agreement allows for unmatched offers, thus modelling *open* systems, where possibly new principals can join the composition and match the available offers.

Below, we extend Theorem 13 and establish a correspondence between contract automata enjoying the property of agreement and the convergent communicating systems with the one buffer semantics. The following definition accordingly extends the branching condition of Definition 51.

Definition 53 (Extended Branching Condition)**.** *A contract automaton $\mathcal{A}$ has the* extended branching condition *if and only if it has the* branching condition *and for each $\vec{q}_1, \vec{q}_2$ reachable in $\mathcal{A}$ the following holds*

$$\forall \vec{a} \text{ offer action } .(\vec{q}_1 \xrightarrow{\vec{a}} \wedge\, snd(\vec{a}) = \mathtt{i} \wedge \vec{q}_{1(i)} = \vec{q}_{2(i)}) \text{ implies } \vec{q}_2 \xrightarrow{\vec{a}}$$

To easily handle the offer actions that are now admitted, we introduce a special contract automaton, called *environment*, that "captures" them all. In this way we can re-use the notion of strong agreement. The environment has a single state, both initial and final and a request loop transition for each possible offer.

Definition 54 (Environment)**.** *The* environment *is the contract automaton*

$$\mathtt{E} = \langle \{q_e\}, q_e, \mathbb{R}, \mathbb{O}, \{(q_e, a, q_e) \mid a \in \mathbb{R}\}, \{q_e\} \rangle$$

Note that by composing a contract automaton $\mathcal{A}$ with the environment $\mathtt{E}$, that is $\mathcal{A} \otimes \mathtt{E}$, all the offer actions of $\mathcal{A}$ are turned into match actions with the environment.

The next theorem shows how the most permissive controller of a contract automaton $\mathcal{A}$ and the controller of $\mathcal{A} \otimes \mathtt{E}$ are related by the two branching conditions of Definition 51 and Definition 53.

Theorem 14 (Extendend Branching Condition and Environment)**.** *Given the most permissive controller $\mathcal{K}_\mathcal{A}$ for the contract automaton $\mathcal{A}$, then $\mathcal{K}_\mathcal{A}$ has the extended branching condition if and only if $\mathcal{K}_{\mathcal{A}\otimes\mathtt{E}}$ has the branching condition.*

Proof. *(⇒)* By hypothesis $\mathcal{K}_\mathcal{A}$ has the extended branching condition. By Definition 24 all offers of $\mathcal{K}_\mathcal{A}$ are turned into matches with the environment in $\mathcal{K}_{\mathcal{A}\otimes\mathtt{E}}$. By contradiction assume that $\mathcal{K}_{\mathcal{A}\otimes\mathtt{E}}$ does not have the branching condition.

Hence there must be two states $\vec{q}_1, \vec{q}_2$ and a principal $\mathtt{p}$ such that $\vec{q}_1 \xrightarrow{\vec{a}}, \vec{q}_2 \not\xrightarrow{\vec{a}}$ with $\vec{q}_{1(\mathtt{p})} = \vec{q}_{2(\mathtt{p})}$ where $\vec{a}$ is a match with $snd(\vec{a}) = \mathtt{p}$.

Moreover it must be that $rcv(\vec{a}) = \mathtt{E}$ (with a slight abuse of notation $\mathtt{E}$ is the principal corresponding to the environment); otherwise the match transition would also be in $\mathcal{K}_\mathcal{A}$, obtaining a contradiction. By Definition 24 the match $\vec{a}$ in $\mathcal{K}_{\mathcal{A}\otimes\mathtt{E}}$ is turned into an offer

action $\vec{a}'$ in $\mathcal{K}_{\mathcal{A}}$, and we obtain a contradiction because $\mathcal{K}_{\mathcal{A}}$ does not have the extended branching condition since $\vec{q}_1 \xrightarrow{\vec{a}'}, \vec{q}_2 \not\xrightarrow{\vec{a}'}$.

(⇐) By hypothesis we have that $\mathcal{K}_{\mathcal{A}\otimes \mathtt{E}}$ has the branching condition. By Definition 24 we know that in the contract automaton $\mathcal{K}_{\mathcal{A}\otimes \mathtt{E}}$ all the matches involving the environment are coupled with offers of $\mathcal{K}_{\mathcal{A}}$.

By contradiction assume that $\mathcal{K}_{\mathcal{A}}$ does not hold the extended branching condition. Hence there must be two states $\vec{q}_1, \vec{q}_2$ and a principal p such that $\vec{q}_1 \xrightarrow{\vec{a}}, \vec{q}_2 \not\xrightarrow{\vec{a}}$ with $\vec{q}_{1(\mathtt{p})} = \vec{q}_{2(\mathtt{p})}$ and $snd(\vec{a}) = \mathtt{p}$.

We distinguish two cases:

- $\vec{a}$ is a match action: then by Definition 24 the match is present also in $\mathcal{K}_{\mathcal{A}\otimes \mathtt{E}}$, obtaining a contradiction since $\mathcal{K}_{\mathcal{A}\otimes \mathtt{E}}$ has the branching condition.
- $\vec{a}$ is an offer action: then by Definition 24 $\vec{a}$ is turned into a match $\vec{a}'$ with E in $\mathcal{K}_{\mathcal{A}\otimes \mathtt{E}}$. Since $\vec{q}_1 \xrightarrow{\vec{a}'}, \vec{q}_2 \not\xrightarrow{\vec{a}'}$ we have that $\mathcal{K}_{\mathcal{A}\otimes \mathtt{E}}$ does not hold the branching condition, obtaining a contradiction.

□

Finally we relate the most permissive controller $\mathcal{K}_{\mathcal{A}}$ of a contract automaton $\mathcal{A}$ with the corresponding system $S(\mathcal{K}_{\mathcal{A}\otimes \mathtt{E}})$, obtained from the controller composed with the environment. Indeed as a direct consequence of Theorem 13 and Theorem 14 we have that $\mathcal{K}_{\mathcal{A}}$ has the extended branching condition if and only if $S(\mathcal{K}_{\mathcal{A}\otimes \mathtt{E}})$ is convergent.

Corollary 3. *Given the most permissive controller $\mathcal{K}_{\mathcal{A}}$ for the contract automaton $\mathcal{A}$, then $\mathcal{K}_{\mathcal{A}}$ has the extended branching condition iff $S(\mathcal{K}_{\mathcal{A}\otimes \mathtt{E}})$ is convergent.*

Proof. (⇒) By hypothesis $\mathcal{K}_{\mathcal{A}}$ has the extended branching condition, by Theorem 14 we have that $\mathcal{K}_{\mathcal{A}\otimes \mathtt{E}}$ has the branching condition. Finally by Theorem 13 we have that $S(\mathcal{K}_{\mathcal{A}\otimes \mathtt{E}})$ is convergent.

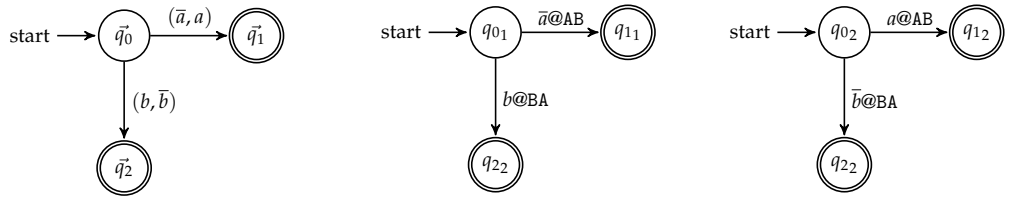
(⇐) By hypothesis $S(\mathcal{K}_{\mathcal{A}\otimes \mathtt{E}})$ is convergent, by applying Theorem 13 we have that $\mathcal{K}_{\mathcal{A}\otimes \mathtt{E}}$ has the branching condition. Finally by Theorem 14 we have that $\mathcal{K}_{\mathcal{A}}$ has the extended branching condition.

□

A benefit of this result is the possibility of disposing the central orchestrator even in case of open-ended systems, where new principals can be dynamically added. Indeed, it suffices to provide a special contract automaton, called the environment, and to slightly extend the branching condition, so to consider all the unmatched offers.

5.2.2 Asynchronous semantics of communicating systems

We now discuss the relations between contract automata and communicating systems, with the semantics of Definition 7, i.e. when the buffers of the communicating machines can contain more than one message. From now onward, the notions of *convergence* and *deadlock* introduced in Definition 49 are considered using the semantics of Definition 7. The following example shows the difficulties with the unrestricted semantics.



(a) $\mathtt{A} \otimes \mathtt{B}$ ($\mathtt{A} = \overline{a} + b$, $\mathtt{B} = \overline{b} + a$) (b) $[\![\mathcal{K}_{\mathtt{A}\otimes\mathtt{B}}]\!]_{\mathtt{A}}$ (c) $[\![\mathcal{K}_{\mathtt{A}\otimes\mathtt{B}}]\!]_{\mathtt{B}}$

Figure 5.3: The CA with a mixed choice of Example 32 and its corresponding communicating machines

Example 32. *Consider the contract automaton* $\mathtt{A} \otimes \mathtt{B}$ *and the corresponding communicating machines of Figure 5.3. This contract automaton is strongly safe and has the branching condition. However the translated system is* not *convergent. Indeed a possible deadlock in* $S(\mathcal{K}_{\mathtt{A}\otimes\mathtt{B}})$ *occurs if* A *performs the action* $\overline{a}$@AB *and then* B *performs the action* $\overline{b}$@BA*. This is because participant* B *can ignore the message received by the participant* A *and follow the other branch of the contract automaton. These behaviours are not permitted by the 1-buffer semantics.*

In order to guarantee that a system of communicating machines corresponding to a contract automaton is convergent, we constrain contract automata as follows. Intuitively, we will discard those contract automata that have a request and an offer transition of a principal outgoing from the same node, like the automata A and B of Example 32.

Indeed, similar conditions are required in the literature for asynchronous choreographed systems, for example branching property [LTY15], knowledge of choice [LT12], dominant role of choice [QZCY07], existence of a unique point of choice [LGMZ08]. Intuitively, in each state there must be a single participant who decides which branch must be taken. We will prove that the absence of mixed choices and the branching condition in the most permissive controller suffices to ensure that the corresponding system of communicating machines with unbounded buffers is convergent. In general, the absence of deadlock is not decidable for this type of systems. A reduction to the Halting problem for Turing machines is showed in [BZ83], that only requires two communicating machines.

In the following we say that a communicating machine has no mixed choices if it is never the case that both offer and request transitions leave one of its states.

More precisely:

Definition 55 (Mixed Choices)**.** *Let* $\mathcal{A}$ *be a contract automaton, then* $\mathcal{A}$ *has a* mixed choice *if and only if there exists a reachable state* $\vec{q}$ *with two outgoing transitions* $\vec{q} \xrightarrow{\vec{a}_1}$, $\vec{q} \xrightarrow{\vec{a}_2}$ *such that* $snd(\vec{a}_1) = rcv(\vec{a}_2)$.

Moreover, let $M = (Q, q_0, Act, \delta, F)$ *be a communicating machine of a participant* i*, then* M *has a* mixed choice *if and only if there exists a reachable state* $q \in Q$ *with two outgoing transitions* $q \xrightarrow{\overline{a}@\mathtt{ij}}$, $\vec{q} \xrightarrow{b@\mathtt{zi}}$ *for some* $\overline{a} \in \mathbb{O}, b \in \mathbb{R}$ *and* $j, z \in \mathcal{P}$.

In the following we will prove that if the most permissive strong controller of a

contract automaton $\mathcal{A}$ has the branching condition and no mixed choices then the corresponding system is convergent.

It is convenient to define how to project a trace of a communicating system into its offer actions, and to prove an auxiliary property.

Definition 56 (Action Projection)**.** *Given $f \in Act^*$, its projection on its offers is defined as follows:*

$$f\lfloor_{\mathbb{O}} = \begin{cases} \overline{a}@\mathtt{ij}\,(f'\lfloor_{\mathbb{O}}) & \text{if } f = \overline{a}@\mathtt{ij}f' \text{ for some } \mathtt{i},\mathtt{j} \in \mathcal{P}, \overline{a} \in \mathbb{O} \\ f'\lfloor_{\mathbb{O}} & \text{if } f = a@\mathtt{ij}f' \text{ for some } \mathtt{i},\mathtt{j} \in \mathcal{P}, a \in \mathbb{R} \\ \varepsilon & \text{if } f = \varepsilon \end{cases}$$

The following property relates the mixed choices of contract automata and communicating machines.

Property 7. *Let $\mathcal{KS}_{\mathcal{A}}$ be the most permissive strong controller of $\mathcal{A}$ and $S(\mathcal{KS}_{\mathcal{A}})$ be the corresponding communicating system.*

If $\mathcal{KS}_{\mathcal{A}}$ has the branching condition and no mixed choices then all the communicating machines of the participants in $S(\mathcal{KS}_{\mathcal{A}})$ have no mixed choices.

Proof. By contradiction assume that there is a participant $\mathtt{p}$ with two transitions $(q_1, \overline{a}@\mathtt{pj}, q_2), (q_1, b@\mathtt{ip}, q_3)$.

By Definition 48 there must be two transitions $(\vec{q}, \vec{a}, \vec{q_1}), (\vec{q_2}, \vec{b}, \vec{q_3})$ where $\vec{q}_{(\mathtt{p})} = \vec{q_2}_{(\mathtt{p})}$, and $\vec{a}, \vec{b}$ are match actions on a and b, respectively, with $snd(\vec{a}) = \mathtt{p}$, $rcv(\vec{a}) = \mathtt{j}$, $snd(\vec{b}) = \mathtt{i}$, $rcv(\vec{b}) = \mathtt{p}$. There are the following two cases:

- $\vec{q} = \vec{q_2}$ we obtain a contradiction since we have a mixed choice;
- $\vec{q} \neq \vec{q_2}$ then since $\vec{q}_{(p)} = \vec{q_2}_{(p)}$ by the branching condition $\mathcal{KS}_{\mathcal{A}}$ must have the transition $(\vec{q_2}, \vec{a}, \vec{q_4})$, hence $\vec{q_2}$ has a mixed choice, obtaining a contradiction.

□

The following theorem relates the traces of a communicating system with the ones of the most permissive strong controller $\mathcal{KS}_{\mathcal{A}}$ it comes from, provided that $\mathcal{KS}_{\mathcal{A}}$ has the branching condition and no mixed choices. Under the above conditions a trace of a communicating system only differs from the corresponding trace of the $\mathcal{KS}_{\mathcal{A}}$ in the order in which the request actions are fired in the system. Indeed by considering only offer actions the two traces are *equal*.

Theorem 15 (Correspondence on Projected Actions)**.** *Let $\mathcal{KS}_{\mathcal{A}}$ be the most permissive controller of $\mathcal{A}$, with the branching condition and no mixed choices, and let Φ be the set of its non-empty traces; let $S(\mathcal{KS}_{\mathcal{A}})$ be the communicating system obtained from $\mathcal{KS}_{\mathcal{A}}$ and let F be the set of its non-empty traces. Then*

$$\forall f \in F \text{ there exists } \varphi \in \Phi \text{ such that } f\lfloor_{\mathbb{O}} = [\![\varphi]\!]\lfloor_{\mathbb{O}}.$$

Proof. Let $\vec{q}_{mpc}$ and s be the initial configurations of $\mathcal{KS}_{\mathcal{A}}$ and $S(\mathcal{KS}_{\mathcal{A}})$, respectively. Assume by contradiction that there exists a f with $s \xrightarrow{f}$ such that for all φ with $\vec{q}_{mpc} \xrightarrow{\varphi}$ we have $f\downarrow_{\mathrm{O}} \neq [\![\varphi]\!]\downarrow_{\mathrm{O}}$.

Since $f \neq \varepsilon$, by Definition 7 it must be that $f = \overline{a_1}@\mathtt{i}_1\mathtt{j}_1 f_1'$ for some $\mathtt{i}_1, \mathtt{j}_1, \overline{a}_1, f_1'$, and by Definition 48 there must be a transition $(\vec{q}_{mpc}, \vec{a}', \vec{q}'_{mpc})$ in $\mathcal{KS}_{\mathcal{A}}$, and therefore there is a $\varphi = \vec{a}'\,\varphi_1'$, for some $\vec{q}'_{mpc}$ and φ_1', where $\vec{a}'$ is a match on a_1 with $snd(\vec{a}') = \mathtt{i}_1, rcv(\vec{a}') = \mathtt{j}_1$. Moreover by hypothesis it must be that $f_1'\downarrow_{\mathrm{O}} \neq [\![\varphi']\!]\downarrow_{\mathrm{O}}$.

Now split f as $f_1 f_2$, and select a trace $\varphi = \varphi_1\varphi_2$ such that f_1 is the longest prefix of f with $[\![\varphi_1]\!]\downarrow_{\mathrm{O}} = f_1\downarrow_{\mathrm{O}}$ ($\neq \varepsilon$ because of the above). Then we have $f_2 = \overline{a}@\mathtt{ij} f_3$ for some $\mathtt{i}, \mathtt{j}, \overline{a}$, and $\vec{q}_{mpc} \xrightarrow{\varphi_1} \vec{q}_{1_{mpc}} \not\xrightarrow{\vec{a}}$ where $\vec{a}$ is a match with $snd(\vec{a}) = \mathtt{i}, rcv(\vec{a}) = \mathtt{j}$. By Definition 48 there must exist a transition $(\vec{q}_{2_{mpc}}, \vec{a}, \vec{q}_{2'_{mpc}})$, for some $\vec{q}_{2_{mpc}}, \vec{q}_{2'_{mpc}}$. Assuming $s \xrightarrow{f_1} (\vec{q}_r, \vec{u})$, we have then that $\vec{q}_{r(\mathtt{i})} = \vec{q}_{2_{mpc_{(\mathtt{i})}}}$.

Note that there is only one $\vec{q}_{1_{mpc}}$ such that $\vec{q}_{mpc} \xrightarrow{\varphi_1} \vec{q}_{1_{mpc}}$, and only one $(\vec{q}_r, \vec{u})$ such that $s \xrightarrow{f_1} (\vec{q}_r, \vec{u})$; because we are considering deterministic automata only.

Since the branching condition holds, it turns out that $\vec{q}_{1_{mpc_{(\mathtt{i})}}} \neq \vec{q}_{2_{mpc_{(\mathtt{i})}}}$.

By $[\![\varphi_1]\!]\downarrow_{\mathrm{O}} = f_1\downarrow_{\mathrm{O}}$ it follows that participant $\mathtt{i}$ has performed all its offers (if any) both in φ_1 and in f_1. By Property 7 we have that the communicating machine of the participant $\mathtt{i}$ has no mixed choices (so it cannot choose between offering or requesting) and is therefore forced to read from its buffer all the received offers (if any) before firing $\overline{a}@\mathtt{ij}$, because these were coupled (with corresponding requests of $\mathtt{i}$) in matches occurring in $[\![\varphi_1]\!]$. Since all actions of participant $\mathtt{i}$ prescribed by φ_1 are performed in f_1, it follows that $\vec{q}_{1_{mpc_{(\mathtt{i})}}} = \vec{q}_{2_{mpc_{(\mathtt{i})}}}$, obtaining a contradiction.

□

Note in passing that for simulating a step in a non-empty trace φ of the most permissive strong controller, the corresponding communicating system can pass through several possible configurations, in order to execute a sequence of actions f such that $f\downarrow_{\mathrm{O}} = [\![\varphi]\!]\downarrow_{\mathrm{O}}$.

As a matter of fact, a participant can fire many requests, not registered in $[\![\varphi]\!]\downarrow_{\mathrm{O}}$, after its last offer registered therein. Indeed the trace φ is formed by match actions, hence all the requests are fired, while in the trace f there could be some participant which has fired all the offers but not yet all its requests.

Example 33. *Consider the automata depicted in Figure 5.4. The most permissive strong controller $\mathcal{KS}_{\mathtt{A}\otimes\mathtt{B}\otimes\mathtt{C}}$ has the branching condition and no mixed choices. Three possible traces of $S(\mathcal{KS}_{\mathtt{A}\otimes\mathtt{B}\otimes\mathtt{C}})$ are:*

$$f = \overline{a}@\mathtt{AB}\ \overline{b}@\mathtt{CB}\ a@\mathtt{AB}\ b@\mathtt{CB}\ \overline{c}@\mathtt{BC}\ \overline{d}@\mathtt{BA} \quad f_1 = f\ c@\mathtt{BC} \quad f_2 = f_1\ d@\mathtt{BA}$$

Consider the trace of $\mathcal{KS}_{\mathtt{A}\otimes\mathtt{B}\otimes\mathtt{C}}$:

$$\varphi = (\overline{a}, a, \square)(\square, b, \overline{b})(\square, \overline{c}, c)(d, \overline{d}, \square)$$

We have $f\downarrow_{\mathrm{O}} = f_1\downarrow_{\mathrm{O}} = f_2\downarrow_{\mathrm{O}} = [\![\varphi]\!]\downarrow_{\mathrm{O}}$. Note that in order to fire the offer $\overline{c}@\mathtt{BC}$ the participant $\mathtt{C}$ needs to fire all the previous requests. Moreover in the trace f_2 each participant has fired all

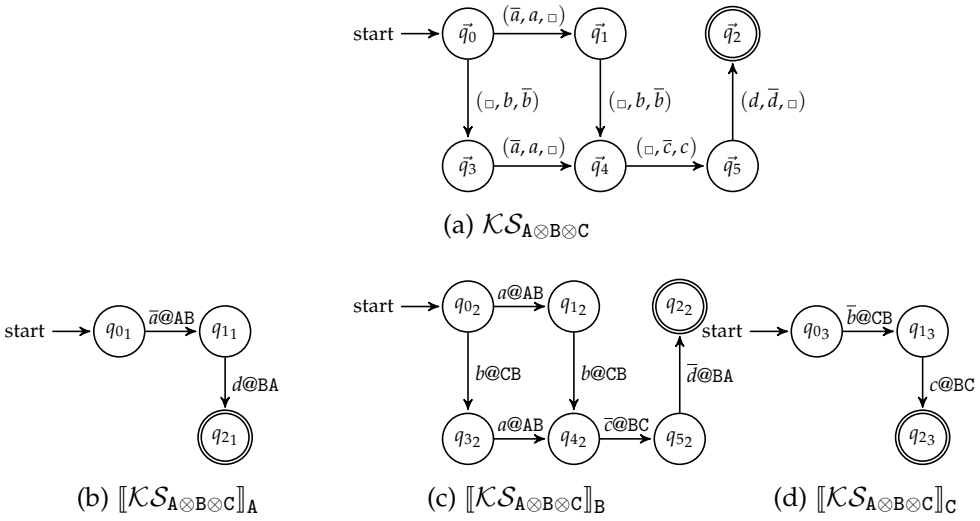


Figure 5.4: The contract automata of Example 33

its requests, indeed we have: $\vec{q}_{mpc} \xrightarrow{\varphi} \vec{q'}_{mpc}$, $s \xrightarrow{f_2} (\vec{q}_r; \vec{u})$ *and* $\vec{q'}_{mpc} \sim \vec{q}_r$, *where* $\vec{q}_{mpc}$ *and* s *are the starting configurations of* $\mathcal{KS}_{\mathtt{A}\otimes\mathtt{B}\otimes\mathtt{C}}$ *and* $S(\mathcal{KS}_{\mathtt{A}\otimes\mathtt{B}\otimes\mathtt{C}})$.

The main result of this sub-section follows. If the strong controller of a contract automaton $\mathcal{A}$ has the branching condition and no mixed choices, then its corresponding system is convergent with unrestricted semantics.

Theorem 16 (Mixed Choice and Convergence)**.** *Let* $S(\mathcal{KS}_{\mathcal{A}})$ *be the communicating system corresponding to the most permissive controller* $\mathcal{KS}_{\mathcal{A}}$. *Then*

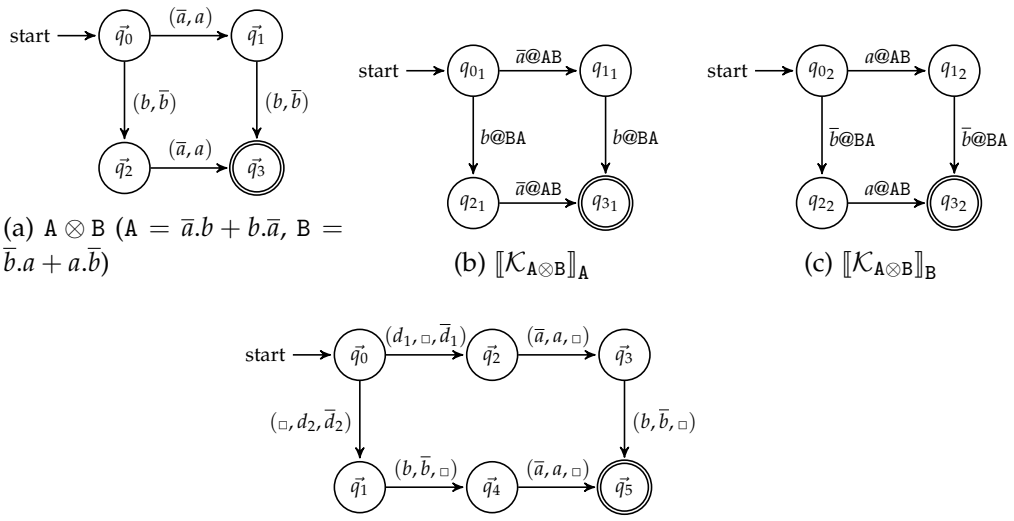
$$\mathcal{KS}_{\mathcal{A}} \textit{ has the branching condition and no mixed choices}$$
$$\textit{implies } S(\mathcal{KS}_{\mathcal{A}}) \textit{ is convergent.}$$

Proof. Let s be the initial configuration of $S(\mathcal{KS}_{\mathcal{A}})$ and $\vec{q}_{mpc}$ be the initial state of $\mathcal{KS}_{\mathcal{A}}$. By contradiction assume that $\mathcal{KS}_{\mathcal{A}}$ has the branching condition and no mixed choices but $S(\mathcal{KS}_{\mathcal{A}})$ is not convergent, i.e. there exists a non-empty run f such that $s \xrightarrow{f} (\vec{q}_r; \vec{u})$ and no final configurations are reachable from $(\vec{q}_r; \vec{u})$. By Theorem 15 there exists a non-empty trace φ of the most permissive strong controller such that $\vec{q}_{mpc} \xrightarrow{\varphi} \vec{q'}_{mpc}$ and $f\downarrow_{\mathrm{O}} = [\![\varphi]\!]\downarrow_{\mathrm{O}}$. Since all the offers are matched in φ, by Definitions 7 and 48 there exists a trace of the communicating system where all the remaining request actions are fired, and let it be our f (recall that there could be a $f' \neq f$ such that $f'\downarrow_{\mathrm{O}} = [\![\varphi]\!]\downarrow_{\mathrm{O}}$). Hence it follows that $\vec{q}_r \sim \vec{q'}_{mpc}$ (recall that the automata are deterministic).

Finally since $\mathcal{KS}_{\mathcal{A}}$ is the most permissive strong controller, there must be a trace φ' and a final state $\vec{q}_f$ such that $\vec{q'}_{mpc} \xrightarrow{\varphi'} \vec{q}_f$. By applying Theorem 12.1 we obtain $(\vec{q}_r; \vec{u}) \xrightarrow{[\![\varphi']\!]} s_f$ where s_f is a final configuration, because the transition relation of the 1-buffer semantics is included in that of the semantics of Definition 7: contradiction.

□

Intuitively, if each principal performs its outputs independently of the state of the other principals (i.e. the branching condition holds) and it has enough information to



(a) $\mathtt{A} \otimes \mathtt{B}$ ($\mathtt{A} = \overline{a}.b + b.\overline{a}$, $\mathtt{B} = \overline{b}.a + a.\overline{b}$)

(b) $[\![\mathcal{K}_{\mathtt{A}\otimes\mathtt{B}}]\!]_{\mathtt{A}}$

(c) $[\![\mathcal{K}_{\mathtt{A}\otimes\mathtt{B}}]\!]_{\mathtt{B}}$

(d) $\mathcal{K}_{\mathtt{A}\otimes\mathtt{B}\otimes\mathtt{D}}$ ($\mathtt{A} = d_1.\overline{a}.b + b.\overline{a}$, $\mathtt{B} = d_2.\overline{b}.a + a.\overline{b}$, $\mathtt{D} = \overline{d}_1 + \overline{d}_2$)

Figure 5.5: The contract automaton (with mixed choices) of Example 34, its corresponding communicating machines, and the amended contract automaton without mixed choices.

know the status of the overall execution (i.e. there are no mixed choices), then each principal is capable of interacting autonomously without breaking the correctness of the composition. In this case, the central control can be removed. This is important because the communication overhead can be reduced, also taking advantage of the parallelism coming from an asynchronous system.

The above theorem does not fully generalise Corollary 2, as shown by the following example.

Example 34. *Consider the contract automaton of Figure 5.5 that is strongly safe (so it is also its most permissive strong controller) and has the branching condition. Its corresponding system consists of the communicating machines in Figure 5.5 and it is convergent. However, a mixed choice is possible from* $\vec{q}_0 = (q_{0_1}, q_{0_2})$.

Note that the absence of mixed choice states is not a restrictive condition. Generally, it is possible to remove the mixed choice states detected by our technique by adding a new dummy principal D *that only performs dummy interactions. The obtained contract automaton is trace-equivalent to the one with mixed choice states, up to dummy transitions. The dummy principal non-deterministically decides which branch to take. For example, the amended composition in Figure 5.5d has no mixed choices.*

5.3 An example

We show our proposal at work on the *two buyers protocol* (2BP for short) presented in [HYC08]. There are two buyers $\mathtt{B}_1$ and $\mathtt{B}_2$ that collaborate in purchasing an item from

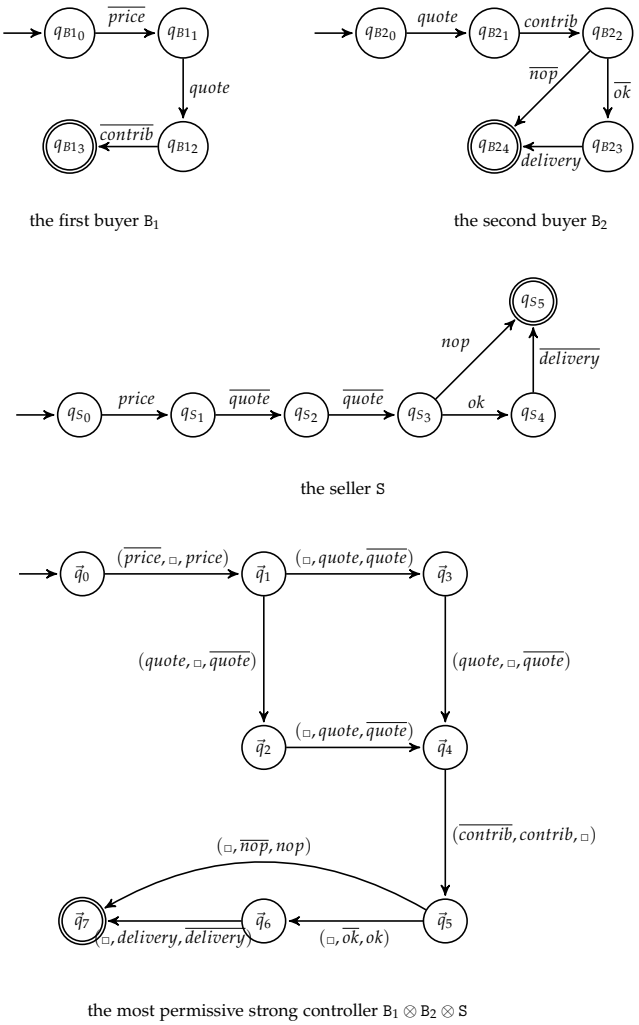


Figure 5.6: The contract automata for 2-buyers protocol (with distinguished quotes) and their most permissive controller

a seller $\mathtt{S}$. Buyer $\mathtt{B}_1$ starts the protocol by asking $\mathtt{S}$ the price of the desired item ($\overline{price}$); the seller $\mathtt{S}$ replies with the quote for the requested item by sending the quotation message $\overline{quote}$ to both buyers. Upon reception of its quote, buyer $\mathtt{B}_1$ sends to $\mathtt{B}_2$ its contribution for purchasing the item ($\overline{contrib}$).

Buyer $\mathtt{B}_2$ waits for the quote from $\mathtt{S}$ and the contribution from $\mathtt{B}_1$. Then, it decides whether to terminate by issuing the $\overline{nop}$ message to $\mathtt{S}$, or to proceed by sending an acknowledgement to $\mathtt{S}$.

Upon receiving the acknowledgement, the seller sends the item to $\mathtt{B}_2$ ($\overline{delivery}$), while if it receives nop it terminates with no further action.

The contract automata $\mathtt{B}_1$, $\mathtt{B}_2$ and $\mathtt{S}$ admit strong agreement, and they are shown in Figure 5.6, together with their most permissive strong controller.

We now analyse the translation of the orchestration into a choreography of communicating finite state machines. The most permissive controller does not enjoy the

branching condition as required by Definition 51, because for example:

- in $\vec{q}_2$ and $\vec{q}_4$, buyer $\mathtt{B}_1$ is in state q_{B1_2},
- $\vec{q}_4 \xrightarrow{(\overline{contrib}, contrib, \square)}$, but
- there is no transition from $\vec{q}_2$ such that $\vec{q}_2 \xrightarrow{(\overline{contrib}, contrib, \square)}$.

This happens because $\mathtt{B}_1$ could send the contribution to $\mathtt{B}_2$ before this has received *quote* from S, so blocking the system. An easy fix would be forcing the seller to first send the quotation to buyer $\mathtt{B}_2$, so reducing the nondeterminism. A convergent choreography of communicating machines, both for the synchronous and the asynchronous case is then derivable as specified above.

The most permissive controller has no states with mixed choices. However, this case may arise if $\mathtt{B}_2$ could additionally withdraw before accepting the contribution from $\mathtt{B}_1$, because the quotation is too high. It is out of the scope of this chapter discussing the ways to recover from this situation, and we only note that our proposal clearly detects why and where it shows up, making the corresponding choreography not convergent.

5.4 Concluding Remarks

We have established a formal correspondence between contract automata, an orchestration model, and communicating systems, i.e. sets of communicating machines, that is a model of choreography.

More precisely, we proved that strong agreement corresponds to convergence (cf. Theorem 13) when communicating systems are endowed with the 1-buffer semantics, according to which the execution of the machines is basically synchronous. We note in passing that this has some advantages since communicating systems with the 1-buffer semantics are computationally more tractable than in the general case [CF05].

We then generalise the above result by adopting a more relaxed notion of agreement that admits computations where offer actions can go unmatched (cf. Corollary 3). We also proved a slightly weaker result for communicating systems with the unrestricted semantics (cf. Theorem 16).

A main impact of the proposed results is the possibility of removing the orchestrator if the composition of contracts satisfies the required properties; so reducing the communication overhead.

We remark that the idea behind our notions of agreement is the existence in a composition of services, of good and bad computations, that the orchestrator eventually cuts off, while the literature has also notions that require composition of participants to have good computations only. Further research problems have been identified and investigated for distributed choreographies, for instance realizability [LMZ13, BBO12, QZCY07, HB10], conformance [LP15, BB11, BZ09b, BZ08b], or enforcement [ARS+13] (see Section 1.2.3). Our work departs from the existing literature in that we do not require to start from a global description of the interactions. In fact, we simply assume

that each service specifies the interactions it is involved in (oblivious of other partners). Then, we identify (*i*) the conditions to allow the coordination of a set of services to satisfy each local requirement (see Chapter 3), and (*ii*) the conditions for removing the central orchestrator.

The choreography extraction problem is the process of extracting a global description (choreography) from a distributed implementation of a system [LTY15, LT12]. The starting point is a system of communicating machines, while in our case this is the ending point.

We remark that our principals are oblivious of their partners and they can be dynamically added to the overall service composition with different compositional operators. Hence, synthesising the orchestration through the most permissive controller is a necessary step for matching the requests and offers of each service, as well as for detecting those liable principals. By changing the composition operator, a different wiring between requests and offers can be obtained, as well as by changing the agreement notion to be enforced (see Example 6 and Section 3.5). This is not the case in [LTY15, LT12], where the local services are already equipped with the information necessary to interact with the other services, which is not automatically generated as it is in our case. Also there is the assumption that they respect the overall contract agreement, instead we consider a scenario with possible liable principals.

Consider again Example 13. A corresponding (wrong) communicating system could be: $\mathtt{A} = \overline{a}@\mathtt{AC}$, $\mathtt{B} = \overline{b}@\mathtt{BC}.a@\mathtt{AB} + a@\mathtt{AB}.\overline{b}@\mathtt{BC}$, $\mathtt{C} = a@\mathtt{AC} + b@\mathtt{BC}$. This system is not convergent, because Alan decides on his own to give his apple to Carol instead of Betty. Indeed, by synthesising the orchestration through the controller of Figure 3.6b, we detect the *liable* transition $(\vec{q}_0, (\overline{a}, \square, a), \vec{q}_6)$ (i.e. Alan gives his apple to Carol). From the orchestration a synchronous convergent communicating system can be generated using Theorem 13, where Alan correctly exchange his apple with Betty that in turns gives her blueberry to Carol.

We conjecture that the conditions guaranteeing well-behaviour identified here imply the *generalised multiparty compatibility* condition identified in [LTY15]. For a given communicating system satisfying this property, a choreography always exists and can be synthesised effectively.

In [CDM14] progress is attained under assumptions allowing a process P to expose unmatched actions as long as the closed system made of P and a *catalyser* process (derived from P) results to be lock-free. We can give an interesting analogy with this work: our environments resemble their *catalyzers* and the extended branching condition is similar to their relaxed notion of safety. As far as we can see, our notion of convergence is stronger than the above notion of safety, in that we also guarantee that a final state is reached, possibly infinitely often. Additionally, only dyadic session types are dealt with in [CDM14], while here we can cope with multyparty scenarios.

A possible application of the results in [LTY15] is to use the projections obtained from the communicating system corresponding to the controller. Through them, we can identify the principals that could be liable. Indeed, each principal univocally cor-

responds to one of those projections. Hence, this would allow to flag the participants that may lead to communication mismatches so as to refine them and guarantee that the refined participants execute without the intervention of an orchestrator.

In general, the suggested approach would give a more efficient and more distributed execution. This is because one can remove the overhead due to the communication with the orchestrator, and avoid the centralisation point, respectively. Additionally, one could find that only few principals, and more importantly which of them, spoil the conditions for achieving choreographed executions. Hence, by modifying/replacing those principals with the machines obtained by projecting the synthesised choreography it would be possible to retrieve a choreographed executions of the contract automata.

Our notion of *convergence* of a system of communicating machines with unbounded buffers is stricter then the notion of conformance discussed in [BZ08b], as no order is imposed over the received messages. Moreover, services and a choreography can be compliant in [LP15], even though they do not satisfy the branching condition. Study an extended notion of compliance for [LP15] that deals with branching condition is worthwhile.

A verification toolkit based on the results presented here and in Chapter 3 is presented in Chapter 6.

Chapter 6

A Tool For Contract Automata

In this chapter we describe a prototypical toolkit supporting *contract automata* (see Chapter 3), in order to show how the verification techniques discussed in Chapter 3 and Chapter 5 can be automatized. By means of an example we describe how CAT (acronym for Contract Automata Tool) can support the modelling and analysis of service-oriented applications. To this purpose, we borrow here the two-buyer protocols (see Section 5.3). Then we apply CAT and, when the agreement property of interest is violated, as part of our checking routine we look at identifying the defects, and subsequently we devise a different version of the protocol correcting the identified errors. CAT is available at `https://github.com/davidebasile/workspace`.

Structure of the Chapter We discuss in Section 6.1 how CAT can be used presenting a verification session on the proposed case study; details about the implementation of CAT are in Section 6.2. A component for solving optimization problems related to contract automata is described in Section 6.3. Finally, Section 6.4 draws some conclusions and discusses related and future work.

6.1 CAT at work

We have implemented CAT in Java according to the simple architecture of Figure 6.1.

As detailed in Section 6.2, in order to implement contract automata based verification, CAT extends Jamata, a Java framework for designing and developing automata-based models, yielding methods for loading, storing, printing, and representing them. In other words, CAT originally specializes Jamata on contract automata, offering to the developers an API for creating and verifying contract automata. Jamata is also available at `https://github.com/davidebasile/workspace`. Also, CAT interfaces with a separate module for solving linear optimization problems, called AMPL [FGK89], described in Section 6.3. This is an original facet of CAT; in fact, it maps the (check of) agreement properties of interest on a linear optimization problem.

The API provided by CAT can be used to perform script analysis of sessions, and can be conceptually classified as follows:

Figure 6.1: The architecture of CAT

Automata operations consist of the methods `CA proj(int i)`, that returns the automaton specifying the i^{th} service of the composition, `CA product(CA[] aut)` and `CA aproduct(CA[] aut)` that compute respectively the product and the *associative* product of contract automata. Interestingly, `product` has to filter out the offers and request transitions when the source state has a corresponding outgoing match transition. Method `aproduct` is built on top of `product` by invoking `product` on the services obtained as projections of the automaton in input.

Safety check consists of the instance methods `safe`, `agreement`, `strongAgreement`, and `strongSafe` returning `true` if the corresponding agreement property holds on the contract automaton. Section 6.3 discusses the property of *weak agreement*.

Controllers consist of the methods `CA mpc()` and `CA smpc()` that return the *most permissive controller*, for respectively agreement and strong agreement (see Chapter 3).

Liable detection consists of the methods `CATransition[] liable()` - returning transitions from a state s to a state t such that s is in the most permissive controller but t is not - and `CATransition[] strongLiable()` that similarly returns such transitions for the most permissive controller of the strong agreement property. We recall that liable services are those responsible for leading a contract composition into a failure.

Decentralization includes `int[][] branchingCondition()`, that returns two states and an action for which the branching condition is violated. We recall that the branching condition holds if the actions of a service are not affected by the states of the other services in the composition. Another similar method that deals with open-ended interactions is `int[][] extendedBranchingCondition()`. The last method in this category is `int[] mixedChoice()` that returns a mixed-choice state (a state where a principal has enabled both offers and requests inside matches). All such methods return `null` when the conditions they check do not hold.

Moreover, CAT offers a command line interface to interactively use the functionalities of the tool. In this chapter we discuss this modality on a slightly modified version of the 2-buyers protocol (see Section 5.3), where the *quote* messages are distinguished. The modified principals are in Figure 6.2.

Jamata first prompts the user for the tool to use; selecting option 4 (lines 2-3 in Output 1) accesses CAT. This launches CAT which first displays a menu where each option corresponds to one of the methods described above. (lines 4-19 in Output 1):

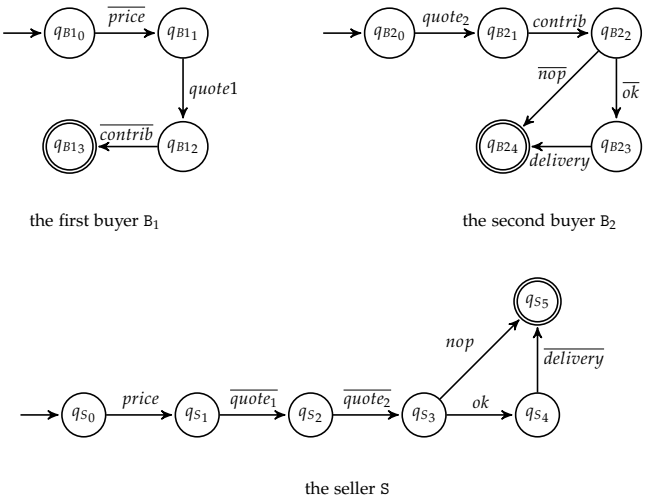


Figure 6.2: The contract automata for the updated 2BP

```
Output 1
                                                                    1
Press 1 for FMA and 2 for PFSA  3 for FSA and 4 for CA              2
4                                                                   3
Select an operation                                                 4
1 : product                                                         5
2 : projection                                                      6
3 : aproduct                                                        7
4 : strongly safe                                                   8
5 : strong agreement                                                9
6 : safe                                                            10
7 : agreement                                                       11
8 : strong most permissive controller                               12
9 : most permissive controller                                      13
10 : branching condition                                            14
11 : mixed choice                                                   15
12 : extended branching condition                                   16
13 : liable                                                         17
14 : strongly liable                                                18
15 : exit                                                           19
```

To check 2BP we first have to compute the product automaton (described below) of the two buyers and the seller by selecting option 1.

When option 1 is selected, CAT asks the user to set the contract automata on which to take the product of (line 2 in Output 2):

```
Output 2
Reset stored automaton...                                           1
Do you want to create/load other contract automata? yes             2
Insert the name of the automaton (without file extension)           3
to load or leave empty for create a new one                         4
B1                                                                  5
Contract automaton:                                                 6
Rank: 1                                                             7
Number of states: [4]                                               8
Initial state: [0]                                                  9
Final states: [[3]]                                                 10
```

```
Transitions:

([0],[1],[1])
([1],[-2],[2])
([2],[3],[3])


Do you want to create/load other contract automata? yes
Insert the name of the automaton (without file extension)
to load or leave empty for create a new one
```

On line 4 in Output 2, we load the automaton of the first buyer from the file `B1.data`, the content of which is displayed on the screen (lines 6-15 in Output 2) and the user is asked for the next automaton (lines 18-20 in Output 2). For each entered automaton, CAT prints a textual description on the screen reporting the rank, initial and final states, and the list of transitions. The transitions are triples $(\mathtt{s},\mathtt{l},\mathtt{t})$ where `s` is the source state, `l` is the label, and `t` is the target state. These elements are lists of length r (the rank of the automaton), for instance in Output 2 line 7 we have $r = 1$. The i-th element of each list corresponds to the i-th service. In particular, the i-th action in the list of labels identifies the action performed by the i-th service; such action is strictly positive (if the action is an offer), strictly negative (if it is a request), and 0 if the service is idle in the transition. The offers (requests resp.) of a contract automata are encoded into strictly positive (negative resp.) integers. For `B1`, actions $\overline{price}$, $quote_1$, and $\overline{contrib}$ are codified with the integer 1, -2, and 3 respectively. Once the automata for the buyers and the seller have been set, the user enters `no` when prompted for the next automaton. Now, as per Output 3, CAT computes and displays their product:

Output 3

```
Computing the product automaton ...
Contract automaton:
Rank: 3
Number of states: [4, 5, 6]
Initial state: [0, 0, 0]
Final states: [[3][4][5]]
Transitions:

([0, 0, 0],[1, 0, -1],[1, 0, 1])
 ([0, 1, 0],[1, 0, -1],[1, 1, 1])
 ([0, 2, 0],[1, 0, -1],[1, 2, 1])
 ([0, 3, 0],[1, 0, -1],[1, 3, 1])
 ([0, 4, 0],[1, 0, -1],[1, 4, 1])
 ([1, 0, 1],[-2, 0, 2],[2, 0, 2])
 ([1, 1, 1],[-2, 0, 2],[2, 1, 2])
 ([1, 2, 1],[-2, 0, 2],[2, 2, 2])
 ([1, 3, 1],[-2, 0, 2],[2, 3, 2])
 ([1, 4, 1],[-2, 0, 2],[2, 4, 2])
 ([2, 1, 2],[3, -3, 0],[3, 2, 2])
 ([2, 1, 3],[3, -3, 0],[3, 2, 3])
 ([2, 1, 4],[3, -3, 0],[3, 2, 4])
 ([2, 1, 5],[3, -3, 0],[3, 2, 5])
 ([2, 0, 2],[3, 0, 0],[3, 0, 2])
 ([2, 2, 2],[3, 0, 0],[3, 2, 2])
 ([2, 2, 3],[3, 0, 0],[3, 2, 3])
 ([2, 3, 2],[3, 0, 0],[3, 3, 2])
```

```
([2, 3, 3],[3, 0, 0],[3, 3, 3])
([2, 3, 4],[3, 0, 0],[3, 3, 4])
([2, 3, 5],[3, 0, 0],[3, 3, 5])
([2, 4, 2],[3, 0, 0],[3, 4, 2])
([2, 4, 3],[3, 0, 0],[3, 4, 3])
([2, 4, 4],[3, 0, 0],[3, 4, 4])
([2, 4, 5],[3, 0, 0],[3, 4, 5])
([2, 0, 2],[0, -7, 7],[2, 1, 3])
([3, 0, 2],[0, -7, 7],[3, 1, 3])
([0, 0, 0],[0, -7, 0],[0, 1, 0])
([1, 0, 1],[0, -7, 0],[1, 1, 1])
([0, 1, 0],[0, -3, 0],[0, 2, 0])
([1, 1, 1],[0, -3, 0],[1, 2, 1])
([3, 1, 3],[0, -3, 0],[3, 2, 3])
([3, 1, 4],[0, -3, 0],[3, 2, 4])
([3, 1, 5],[0, -3, 0],[3, 2, 5])
([2, 2, 3],[0, 4, -4],[2, 3, 4])
([3, 2, 3],[0, 4, -4],[3, 3, 4])
([0, 2, 0],[0, 4, 0],[0, 3, 0])
([1, 2, 1],[0, 4, 0],[1, 3, 1])
([2, 2, 2],[0, 4, 0],[2, 3, 2])
([3, 2, 2],[0, 4, 0],[3, 3, 2])
([3, 2, 4],[0, 4, 0],[3, 3, 4])
([3, 2, 5],[0, 4, 0],[3, 3, 5])
([2, 2, 3],[0, 5, -5],[2, 4, 5])
([3, 2, 3],[0, 5, -5],[3, 4, 5])
([0, 2, 0],[0, 5, 0],[0, 4, 0])
([1, 2, 1],[0, 5, 0],[1, 4, 1])
([2, 2, 2],[0, 5, 0],[2, 4, 2])
([3, 2, 2],[0, 5, 0],[3, 4, 2])
([3, 2, 4],[0, 5, 0],[3, 4, 4])
([3, 2, 5],[0, 5, 0],[3, 4, 5])
([2, 3, 4],[0, -6, 6],[2, 4, 5])
([3, 3, 4],[0, -6, 6],[3, 4, 5])
([0, 3, 0],[0, -6, 0],[0, 4, 0])
([1, 3, 1],[0, -6, 0],[1, 4, 1])
([2, 3, 2],[0, -6, 0],[2, 4, 2])
([2, 3, 3],[0, -6, 0],[2, 4, 3])
([2, 3, 5],[0, -6, 0],[2, 4, 5])
([3, 3, 2],[0, -6, 0],[3, 4, 2])
([3, 3, 3],[0, -6, 0],[3, 4, 3])
([3, 3, 5],[0, -6, 0],[3, 4, 5])
([2, 1, 2],[0, 0, 7],[2, 1, 3])
([2, 2, 2],[0, 0, 7],[2, 2, 3])
([2, 3, 2],[0, 0, 7],[2, 3, 3])
([2, 4, 2],[0, 0, 7],[2, 4, 3])
([3, 2, 2],[0, 0, 7],[3, 2, 3])
([3, 3, 2],[0, 0, 7],[3, 3, 3])
([3, 4, 2],[0, 0, 7],[3, 4, 3])
([2, 1, 3],[0, 0, -4],[2, 1, 4])
([2, 3, 3],[0, 0, -4],[2, 3, 4])
([2, 4, 3],[0, 0, -4],[2, 4, 4])
([3, 1, 3],[0, 0, -4],[3, 1, 4])
([3, 3, 3],[0, 0, -4],[3, 3, 4])
([3, 4, 3],[0, 0, -4],[3, 4, 4])
([2, 1, 3],[0, 0, -5],[2, 1, 5])
([2, 3, 3],[0, 0, -5],[2, 3, 5])
([2, 4, 3],[0, 0, -5],[2, 4, 5])
([3, 1, 3],[0, 0, -5],[3, 1, 5])
([3, 3, 3],[0, 0, -5],[3, 3, 5])
```

```
 ([3, 4, 3],[0, 0, -5],[3, 4, 5])                                    87
 ([2, 1, 4],[0, 0, 6],[2, 1, 5])                                     88
 ([2, 4, 4],[0, 0, 6],[2, 4, 5])                                     89
 ([3, 1, 4],[0, 0, 6],[3, 1, 5])                                     90
 ([3, 2, 4],[0, 0, 6],[3, 2, 5])                                     91
([3, 4, 4],[0, 0, 6],[3, 4, 5])                                      92
Do you want to save this automaton? (write yes or no)                 93
yes                                                                   94
Write the name of this automaton                                      95
B1xB2xS                                                               96
```

The user can now choose to discard the automaton (if no further operations are necessary) or save it (if the analysis session continues); here we choose `yes` since we want to apply other operations, and enter `B1xB2xS` to save the automaton in a file named `B1xB2xS.data`. After this interaction (lines 93-96 in Output 3), the operations in Output 1 are displayed so that the user can execute a new one. Options 4 and 5 in Output 1 check if `B1xB2xS` admits *strong agreement* or is *strongly safe*.

We now compute the most permissive strong controller, by applying operation 8 in Output 1 to the product automaton `B1xB2xS.data` saved in the previous interaction with CAT. Once the product automaton is loaded and the operation 8 is selected (not shown here), CAT will compute the most permissive strong controller:

Output 4

```
The most permissive controller of strong agreement for the last       1
CA loaded is                                                          2
Contract automaton:                                                   3
Rank: 3                                                               4
Number of states: [4, 5, 6]                                           5
Initial state: [0, 0, 0]                                              6
Final states: [[3][4][5]]                                             7
Transitions:                                                          8
                                                                      9
([0, 0, 0],[1, 0, -1],[1, 0, 1])                                     10
([1, 0, 1],[-2, 0, 2],[2, 0, 2])                                     11
([2, 1, 3],[3, -3, 0],[3, 2, 3])                                     12
([2, 0, 2],[0, -7, 7],[2, 1, 3])                                     13
([3, 2, 3],[0, 4, -4],[3, 3, 4])                                     14
([3, 2, 3],[0, 5, -5],[3, 4, 5])                                     15
([3, 3, 4],[0, -6, 6],[3, 4, 5])                                     16
Do you want to save this automaton? (write yes or no)                 17
yes                                                                   18
KS_B1xB2xS                                                            19
```

As before, we save the automaton computed by CAT in a file named `KS_B1xB2xS` (lines 17-19 in Output 4); the automaton is displayed in Figure 6.3. The final states are represented as a list where the i-th element is the list of the final states of the i-th service. This representation allows to check if a state of the most permissive controller is final or not without needing to explicitly enumerate all the final states of the most permissive controller.

Note that in each transition of the most permissive controller there is always an idle service. For instance, consider the transition ([0, 0, 0],[1, 0, -1],[1, 0, 1]): it corresponds

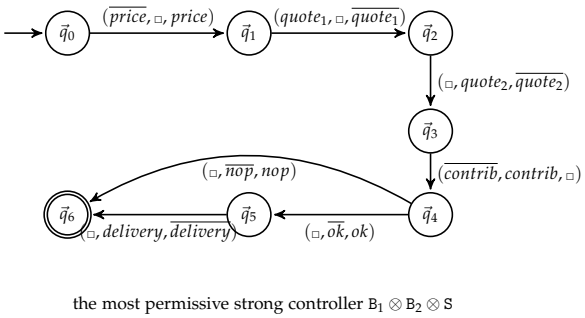


Figure 6.3: The most permissive controller of 2BP

to the transition $(\vec{q}_0, (\overline{price}, \square, price), \vec{q}_1)$ of the most permissive controller in Fig. 6.3 (the second component of the label is 0 because `B2` is idle).

Consider now Figure 6.3, where the state $\vec{q}_2$ corresponds to `[2,0,2]` in Output 5, the state $\vec{q}_3$ to `[2,1,3]`, and the transition $(\overline{contrib}, contrib, \square)$ to the label `[3,-3,0]`. The most permissive controller `KS_B1xB2xS` does not enjoy the branching condition, as reported by CAT:

```
Output 5
State [2,0,2] violates the branching condition because it has no
transition labelled [3,-3,0] which is instead enabled in
state [2,1,3]
```

It is important to observe that the message in Output 5 also flags states and transitions for which the condition is violated. We discuss the problem by considering the automaton in Figure 6.3. The local state of buyer B_1 in $\vec{q}_2$ and $\vec{q}_3$ is q_{B12} (cf., Figure 6.2) , while the locale state of B_2 in $\vec{q}_2$ is q_{B20}, and in $\vec{q}_3$ is q_{B21}. Therefore, in the case that B_2 is in local state q_{B20} where it is waiting for $\overline{quote_2}$, without an orchestrator the offer $\overline{contrib}$ from B_1 could fill up the 1-buffer of B_2, leading to a deadlock.

A simple fix consists in swapping the order in which the quotes are sent by the seller; CAT reports that the amended protocol (not shown here) enjoys branching condition. The absence of mixed choice states is checked by applying option 11 in Output 1 to `KS_B1xB2xS`:

```
Output 6
The CA has no mixed choice states
```

A mixed choice state could be introduced in 2BP if, e.g., B_2 could send the acknowledgement to S or receive $\overline{contrib}$ from B_1 in any order. For this variant of 2BP CAT finds the mixed choice state, so showing that these services do not form a sound choreography.

6.2 Detailing the implementation of CAT

For the sake of completeness, we now discuss some implementation details. CAT consists of a class `CAUtil` and of other classes `CA` and `CATransition`, extending two

corresponding super-classes of Jamata. The class `CA` provides the main functionalities of CAT; its instance variables capture the basic structure of our automata:

- `int rank` is the rank of the automaton;
- `int[] initial` is the initial state of the automaton (the array is of size `rank`);
- `int[] states` the vector of the number of local states of each principal in the contract automaton (the array is of size `rank`);
- `int[][] finalstates` the final states of each principal in the contract automaton;
- `CATransition[] tra` the transitions of the contract automaton.

The n local states of a principal are represented as integers in the range $0, \ldots, n-1$; in this case, `states.length = 1` and `states[0] = n`. The state of an automaton of rank $m > 1$ is an m-vector `states` such that `states[i]` yields the number of states of the i^{th} principal. This low-level representation (together with the encoding of actions and labels as integers) enabled us to optimize space.

The class `CATransition`, describes a transition of a contract automaton. The instance variables of a `CATransition` object are:

- `int[] source` (the starting state of the transition);
- `int[] label` (the label of the transition);
- `int[] target` (the arriving state of the transition).

The class `CATransition` provides methods to extract its instance variables, to check if the transition is an offer, a request or a match, and to extract the (index of the) principal performing the offer, if any.

6.3 Integer Linear programming and Contract Automata

In this section we review a component for solving optimization problems related to contract automata, that complements the functionalities offered by CAT.

This component reduces the analysis of the properties of weak agreement (see Section 3.4) to a integer linear programming problem and relies on an ad-hoc solver as explained below. The decision procedures are implemented in *A Mathematical Programming Language* (`AMPL`) [FGK89], a popular language for describing and solving optimization problems. In this way, the automatic verification of contract automata under properties of weak agreement exploits efficient techniques and algorithms developed in the area of operational research.

We now briefly describe the implementation of the techniques for verifying weak agreement (the `AMPL` code is provided in Section 6.3.1). The script `flow.run` is displayed below. It can be launched with the command `ampl flow.run` from a shell.

flow.run

```
#reset;                                                    1
option solver cplex;                                       2
model weakagreement.mod;                                   3
data flow.dat;                                             4
solve;                                                     5
display a;                                                 6
display t;                                                 7
display x_t;                                               8
display z_t;                                               9
display gamma;                                             10
```

The script firstly loads the automaton from the file `flow.dat` (line 4). The description of the automata consists of the number of nodes, the cardinality of the alphabet of actions, and a matrix of transitions for each action a, where there is value 0 at position (s, d) if there is no transition from state s to state d labelled by a, and respectively 1 or -1 if there is an offer or request transition on a. In this case, the contract automata described in `flow.dat` is representative.

The script `flow.run` contains commands to display the variables of the linear program (lines 7-10). The `AMPL` linear program to load is given as input parameter to the script (line 3). The two optimization problem available are:

- `weakagreement.mod` the file contains the formalization of the optimization problem for deciding whether a contract automaton admits weak agreement (see Theorem 7);

- `weaksafety.mod` it contains the formalization of the optimization problem for deciding whether a contract automaton is weakly safe (see Theorem 6).

Both formal descriptions are then solved using the solver `cplex`, that is the simplex method implemented in `C`. However it is possible to select other available solvers in the script `flow.run` (line 2).

The execution of the script will prompt to the user the value of variables. As proved in Section 3.4, if the variable `gamma` is non-negative then the contract automata satisfies the given property.

Bi-level optimization problems cannot be defined directly in `AMPL`. Therefore, we cannot plainly apply formalisation of Theorem 8 for representing weakly liable transitions as an optimization problem. However, different techniques of relaxation of the bi-level problem for over approximating the set of weakly liable transitions can be used, as for example Nash equilibria constraints, lagrangian relaxation.

As future work, we are planning to develop a toolchain for fully integrating the above techniques in CAT, in order to reuse them for the functionalities described in Section 6.1 and 6.2. In particular, CAT will automatically generate a contract automaton description `flow.dat`, execute the script `flow.run` and collect the results.

6.3.1 AMPL code

In the following we provide the code of `weakagreement.mod` in Figure 6.4 and the code of `weaksafety.mod` in Figure 6.5.

6.4 Concluding Remarks

In this chapter we have described CAT, a tool supporting the analysis of contract-based applications attained with novel techniques based on combinatorial optimization. A non trivial example was used to show main features of CAT. We briefly discuss some literature regarding the integration of formal techniques in the realisation of service applications, and propose some future extensions of the presented toolkit.

Different techniques and tools for translating BPEL processes [Jur06] into automata models are presented in [WFN04, FBS04]. The behaviour of BPEL processes has been analysed through constraint automata [BSAR06] and Reo [Arb04] (see Section 1.2.5) in [TVMS07]. REO and constraint automata have also been used for analysing Business Process Modelling Notation [(OM11] models, with the linear and branching time model checker Vereofy [BBK+10].

A model-driven approach would also ease the integration of CAT with e.g., the tools discussed above. For example, existing tools, like ATLAS Transformation Language [JABK08], are capable of generating BPEL code from other formalisms such as WS-CDL [KSF+11].

This would on the one hand extend CAT applicability to the analysis of actual code of service-oriented applications and, on the other hand, it would enable the integration of CAT with existing tools such as those described in [BP06], so to provide the developers with a wide variety of tools for guaranteeing the quality of the composition of services according to different criteria.

The properties verified by CAT have not been considered by other approaches. For example, the identification - even in presence of circular dependencies of services (see Section 6.3) - of liable transitions that may spoil a composition complement the verification done in [TVMS07]

Compared to the reviewed tools, CAT introduces novel techniques based on combinatorial optimization; and focuses on, for example, finding liable participants in a composition of services, as well as analysing circularity issues.

Although useful, the tool is still a prototype; and has been introduced mainly for proving the effectiveness of verifying services composition through contract automata. We plan to improve its efficiency, extend it with new functionalities (e.g., relaxation), and improve its usability (e.g., adding a user-friendly GUI and pretty-printing automata). We note that CAT provides a valid support to the analysis of applications. In fact, CAT is able to detect possible violations of the properties of interest (for example branching condition, mixed choice). A drawback of CAT is that it does not support modelling and design of applications. An interesting evolution of CAT would be to add functionalities for amending applications violating properties of interest. For instance, once

weakagreement.mod

```
# n number of nodes
# m number of actions
param n;
param m;
param K;
param final; #final node
set N := {1..n};
set M := {1..m};
param t{N,N};
param a{N,N,M};
var x_t{N,N} >=0 integer;
var z_t{N,N,N} >=0;
var gamma;
var p{N} binary;
var wagreement;

#flow constraints
subject to Flow_Constraints {node in N}:
          sum{i in N}( x_t[i,node]*t[i,node] ) - sum{i in N}(x_t[node,i]*t[node,i]) =
          if (node == 1) then  -1
          else if (node == final)  then 1
          else 0;
;

subject to p1{node in N}: p[node] <= sum{i in N}( x_t[node,i]*t[node,i]);
subject to p2{node in N}: sum{i in N}( x_t[node,i]*t[node,i]) <= p[node]*K;



subject to Auxiliary_Flow_Constraints {snode in N diff {1},node in N}:
         sum{i in N}( z_t[snode,i,node]*t[i,node] ) - sum{i in N}(z_t[snode,node,i]*t[node,i]) =
         if (node == 1) then    - p[snode]
         else if (node == snode)  then  p[snode]
         else 0;

subject to Auxiliary_Flow_Constraints2{i in N, j in N,snode in N}:
         z_t[snode,i,j]*t[i,j] <= x_t[i,j]*t[i,j];


subject to threshold_constraint {act in M}: sum{i in N,j in N} x_t[i,j]*t[i,j]*a[i,j,act] >= gamma;

#objective function
maximize cost: gamma;
```

Figure 6.4: The implementation in AMPL of the optimization problem for deciding weak agreement.

weaksafety.mod

```
# n number of nodes
# m number of actions
param n;
param m;
param K;
param final; #final node
set N := {1..n};
set M := {1..m};
param t{N,N};
param a{N,N,M};
var x_t{N,N} >=0 integer;
var z_t{N,N,N} >=0;
var gamma;
var p{N} binary;
var v{M} binary;
var wagreement;

#flow constraints
subject to Flow_Constraints {node in N}:
         sum{i in N}( x_t[i,node]*t[i,node] ) - sum{i in N}(x_t[node,i]*t[node,i]) =
         if (node == 1) then  -1
         else if (node == final)  then 1
         else 0;
;

subject to p1{node in N}: p[node] <= sum{i in N}( x_t[node,i]*t[node,i]);
subject to p2{node in N}: sum{i in N}( x_t[node,i]*t[node,i]) <= p[node]*K;


subject to Auxiliary_Flow_Constraints {snode in N diff {1},node in N}:
        sum{i in N}( z_t[snode,i,node]*t[i,node] ) - sum{i in N}(z_t[snode,node,i]*t[node,i]) =
        if (node == 1) then    - p[snode]
        else if (node == snode)  then  p[snode]
        else 0;

subject to Auxiliary_Flow_Constraints2{i in N, j in N,snode in N}:
        z_t[snode,i,j]*t[i,j] <= x_t[i,j]*t[i,j];

subject to vi: sum{i in M} v[i] = 1;

subject to threshold_constraint :
sum{act in M,i in N,j in N} (v[act]*x_t[i,j]*t[i,j]*a[i,j,act]) <= gamma;

#objective function
minimize cost: gamma;
```

Figure 6.5: The implementation in AMPL of the optimization problem for deciding weak safety.

liable transitions are identified, CAT could suggest how to modify services to guarantee the property. This may also be coupled with the model-driven approach by featuring functionalities tracing transitions in the actual source-code of services.

Chapter 7

Conclusions

In this thesis we have proposed a formal model for designing and verifying distributed applications following the Service Oriented Computing paradigm, under the assumptions that services cannot fulfil their prescribed behaviour (either unintentionally or maliciously) and may have mutual circular dependencies between their requirements and obligations. We have related the proposed model to different coordination mechanisms and logic formalisms, and we have investigated the relations between safety properties and service compliance. Finally, the theory we have introduced has driven the implementation of a prototype tool.

7.1 Main results

In this section we summarise the main results of this thesis. The formal verification of security properties has been related to the notion of compliance of services in Chapter 2. History expressions (Section 1.6) are extended for expressing a family of behavioural contracts (Section 1.2.1), and an automata-based model checking technique for verifying compliance of contracts is proposed in Theorem 1, thus relating compliance with safety properties (Theorem 2). It is then possible to apply all the techniques and tools developed for checking safety properties to efficiently verify the absence of communication errors between services, as well as security policies.

The idea of verifying a composition of service through automata-based model checking techniques has been generalised to a formal theory of service contracts, developed in Chapter 3. Contract automata are an orchestrated model introduced for describing the abstract behaviour of services, together with operators for composing them according to a static orchestration or a dynamic one. Contract automata have the main feature of expressing as a single automaton both individual principals, where no action is matched, closed systems (i.e. composition of principals) where all actions are matched, and open systems, where not all the actions are matched. Different properties of agreement between services have been studied from a language-theoretic point of view, where the main idea is the coexistence of "good" executions and "bad" computations, that are those breaking the overall agreement, and that are eventually removed by the orches-

trator. We have considered the case of synchronous matching between requests and offers, under the requirements that (i) all the requests must be fulfilled, that is *agreement* (Section 3.2), and (ii) all requests and offers must be matched, that is *strong agreement* (Section 3.3). The notion of strong agreement deals with closed systems where no offers are left unmatched, while agreement allows for unmatched offers, thus modelling *open* systems, where possibly new principals can join the composition and match the available offers.

For deciding whether a composition of contracts admits an agreement (Property 2) and for finding the liable principals (Proposition 5) we have resorted to techniques borrowed from Control Theory (Section 1.7).

We have identified two class of contracts: competitive and collaborative contracts; and we have characterized their composition under the property of agreement (Theorem 3). We can prove the correctness of the composition without generating the whole state space. Indeed, contracts can be modularly checked, provided they satisfy the required conditions (see Theorem 3). An asynchronous notion of agreement (i.e. weak agreement) where requests can be fulfilled on credit (Section 3.4) is introduced in order to deal with the potential, but harmless and often essential circularity occurring in services. It has been proved that the property of weak agreement is context-sensitive (Theorem 5). Composition of collaborative and competitive contracts has been analysed under weak agreement (Theorem 4), obtaining results similar to those of Theorem 3. We have applied techniques based on optimization of flow problems (Section 1.8) for deciding if a composition of contracts is in weak agreement (Theorem 7 and Theorem 6), and for identifying the weakly liable principals (Theorem 8). In this way, the automatic verification of weak agreement exploits efficient techniques and algorithms developed in the area of operational research.

The circularity studied in Chapter 3 has been related to different logic formalisms in Chapter 4. The obtained results can be used for verifying the corresponding logic formulae and vice-versa.

We have investigated the relations between a composition of contract automata in agreement and provability of formulae in different intuitionistic logics (see Section 1.4). Theorem 9 relates the property of agreement with formulae in the Horn fragment of propositional contract logic, while Theorem 10 considers weak agreement. This result shed light on the relation between the contractual implication connective and the property of weak agreement. Indeed, checking weak agreement is equivalent to lift in the corresponding formula all standard implications to the contractual version.

The Horn fragment of intuitionistic linear logic with mix has been related to the property of agreement in Theorem 11. The importance of this relation lies on the possibility of expressing each H-ILL^{mix} formula as a contract automaton. It is then possible to compose formulae through the composition operators provided by contract automata, exploiting compositionality and related results (for example Theorem 3) for efficiently checking the provability of formulae in H-ILL^{mix}.

While in Chapter 3 we have adopted orchestration as coordination paradigm, the

relations between different services coordination mechanisms have been deepened in Chapter 5. In particular, an orchestration of services can be related to a choreography by translating contract automata into communicating machines (Section 1.3). A practical outcome is disposing the central orchestrator, so reducing the communication overhead.

Theorem 13 establishes conditions for translating a safe orchestration of contracts into a convergent synchronous system of communicating machines, while Theorem 16 considers the asynchronous case and Corollary 3 deals with open-ended choreographies. This is important because the communication overhead can be reduced in open-ended systems, also taking advantage of the parallelism coming from an asynchronous system.

The formal theory we have developed has been applied to a prototypical tool for the verification of distributed services, discussed in Chapter 6.

7.2 Future work

We address several further lines of research concerning our proposal:

- in Chapter 2 a first connection between the analysis of behavioural types and security properties has been outlined. In the presented model, services can replicate themselves boundlessly many times. Future works concern studying a restricted availability of services. Under this restriction, new deadlock situations may arise, which have not been considered. For example, assume that a single instance of a recursive service S_1 satisfies the security constraints and it is compliant with two recursive clients C_1 and C_2. Then, our analysis synthesises a plan which couples both clients with S_1. However, if the client C_1 engages in non terminating interactions with S_1, the other client C_2 would be prevented from any interaction with its associated service S_1, because only one instance of it is available. Hence, the presented analysis needs to be enriched with techniques for ensuring that all the necessary resources are available in a composition.

- Computing the weakly liable principals in a composition of contracts has been showed to be a hard task. While we have related this notion to a bi-level optimization flow problem in Theorem 8, algorithms for solving these types of problems are still under research.

  In order to make the problem tractable, an over approximation of the set of weakly liable principals can be computed. Techniques based on the relaxation of bi-level problems (for example, lagrangian relaxation) for computing approximated solutions in acceptable computational time need to be further investigated.

  Moreover, while we have proposed tailored optimization techniques for verifying given safety properties (namely weak agreement), this approach can be generalised. Indeed, when the system under analysis is modelled as a labelled transition system with weights associated to labels, and the properties to be checked concern those weights, then an associated flow problem can be derived. In this

flow problem, the traces correspond to flows in the graph and the requirements on the labels are coded into constraints in the corresponding linear problem. By maximizing (resp. minimising) the corresponding objective function it is possible to decide whether the property under analysis is satisfied by the model. This would bring algorithms and tools developed in the area of operational research, namely optimization of flow problems, to the formal verification of systems.

- An interesting future line of research concerns using the most permissive controller for suggesting corrections. Indeed, by detecting each liable transition and liable principal, we have a fine grained description of which principals and which transitions break the correctness of the composition. This information could be exploited for amending the composite service.

- In Chapter 4 the provability of formulae in Horn fragments of different intuitionistic logics has been related to properties of agreement between contracts. An extension of these results to comprehend the full logics is still missing. Indeed, while the Horn fragments of the considered logics have a neat interpretation in terms of contracts, the meaning of nested implications as contracts is not clear. For example, the nested contractual implications $p \twoheadrightarrow (q \twoheadrightarrow p')$ could be interpreted as "I promise you p provided that in the future the contract $q \twoheadrightarrow p'$ will be satisfied ". In this case, the inner formula $q \twoheadrightarrow p'$ is considered as a contract that can be advertised only if in the future the formula p is satisfied. Hence, in order to express this type of contracts, a model of the full logics should be enriched with higher order contracts, where a contract can be offered or required in the same fashion of resources. Studying a model for these full logics would deepen our knowledge of contracts and the related circularity and liability issues.

- Theorem 16 attests that a safe orchestration of services satisfying certain properties corresponds to a convergent asynchronous choreography. While in general the converse is undecidable [BZ83], an approximate solution could be obtained by proposing a relaxed optimization problem; similarly to what has been done for the properties of weak agreement. In this case, in order to enforce the FIFO order of messages on a trace t, we need to assign a flow to each prefix p of t. The formalization will feature constraints for ensuring that all prefixes p have a positive difference between offers and requests, which means that a message can be read only after it has been received. A solution can be given for a fixed maximal length of a path, in order to have a finite number of flows. An approximate solution can be obtained, for example, by relaxing the FIFO constraints on the buffers. This would provide a finer indication of the behaviour of the system, where possible mixed choices are now admitted.

- A prototypical tool based on our results has been presented in Chapter 6. Future developments concernt the integration of the tool with a user-friendly interface, and the discussed integer linear programming techniques.

Bibliography

[ABZ13] Lucia Acciai, Michele Boreale, and Gianluigi Zavattaro. Behavioural contracts with request-response operations. *Sci. Comput. Program.*, 78(2):248–267, February 2013.

[ACKM04] G. Alonso, F. Casati, H. A. Kuno, and V. Machiraju. *Web Services - Concepts, Architectures and Applications. Data-Centric Systems and Applications*. Springer, 2004.

[ALRL04] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *Dependable and Secure Computing, IEEE Transactions on*, 1(1):11–33, Jan 2004.

[Arb04] Farhad Arbab. Reo: a channel-based coordination model for component composition. *Mathematical Structures in Computer Science*, 14(3):329–366, 2004.

[ARS+13] Marco Autili, Davide Di Ruscio, Amleto Di Salle, Paola Inverardi, and Massimo Tivoli. A model-based synthesis process for choreography realizability enforcement. In Vittorio Cortellessa and Dániel Varró, editors, *Fundamental Approaches to Software Engineering - 16th International Conference, FASE 2013, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2013, Rome, Italy, March 16-24, 2013. Proceedings*, volume 7793 of *Lecture Notes in Computer Science*, pages 37–52. Springer, 2013.

[Bar06] Jonathan F. Bard. *Practical Bilevel Optimization: Algorithms and Applications (Nonconvex Optimization and Its Applications)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[BB11] Samik Basu and Tevfik Bultan. Choreography conformance via synchronizability. In Sadagopan Srinivasan, Krithi Ramamritham, Arun Kumar, M. P. Ravindra, Elisa Bertino, and Ravi Kumar, editors, *Proceedings of the 20th International Conference on World Wide Web, WWW 2011*, pages 795–804. ACM, 2011.

[BBK+10] Christel Baier, Tobias Blechmann, Joachim Klein, Sascha Klüppelholz, and Wolfgang Leister. Design and verification of systems with exogenous coordination using vereofy. In Tiziana Margaria and Bernhard Steffen, editors,

Leveraging Applications of Formal Methods, Verification, and Validation - 4th International Symposium on Leveraging Applications, ISoLA 2010, Heraklion, Crete, Greece, October 18-21, 2010, Proceedings, Part II, volume 6416 of *Lecture Notes in Computer Science*, pages 97–111. Springer, 2010.

[BBO12] Samik Basu, Tevfik Bultan, and Meriem Ouederni. Deciding choreography realizability. In John Field and Michael Hicks, editors, *Proceedings of the 39th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2012*, pages 191–202. ACM, 2012.

[BCGZ13] Massimo Bartoletti, Tiziana Cimoli, Paolo Di Giamberardino, and Roberto Zunino. Contract agreements via logic. In Marco Carbone, Ivan Lanese, Alberto Lluch-Lafuente, and Ana Sokolova, editors, *Proceedings 6th Interaction and Concurrency Experience, ICE 2013, Florence, Italy, 6th June 2013.*, volume 131 of *EPTCS*, pages 5–19, 2013.

[BCP13] Massimo Bartoletti, Tiziana Cimoli, and G. Michele Pinna. Lending Petri nets and contracts. In Farhad Arbab and Marjan Sirjani, editors, *FSEN*, volume 8161 of *LNCS*, pages 66–82. Springer, 2013.

[BCPZ15] Massimo Bartoletti, Tiziana Cimoli, G.Michele Pinna, and Roberto Zunino. Models of circular causality. In Raja Natarajan, Gautam Barua, and Manas-Ranjan Patra, editors, *Distributed Computing and Internet Technology*, volume 8956 of *Lecture Notes in Computer Science*, pages 1–20. Springer International Publishing, 2015.

[BCPZ16] Massimo Bartoletti, Tiziana Cimoli, G. Michele Pinna, and Roberto Zunino. Contracts as games on event structures. *Journal of Logical and Algebraic Methods in Programming*, 85(3):399 – 424, 2016. Interaction and Concurrency Experience.

[BCZ13] Massimo Bartoletti, Tiziana Cimoli, and Roberto Zunino. A theory of agreements and protection. In David A. Basin and John C. Mitchell, editors, *POST*, volume 7796 of *LNCS*, pages 186–205. Springer, 2013.

[BCZ15] Massimo Bartoletti, Tiziana Cimoli, and Roberto Zunino. Compliance in behavioural contracts: A brief survey. In Chiara Bodei, Gian-Luigi Ferrari, and Corrado Priami, editors, *Programming Languages with Applications to Biology and Security*, volume 9465 of *Lecture Notes in Computer Science*, pages 103–121. Springer International Publishing, 2015.

[BDF09] Massimo Bartoletti, Pierpaolo Degano, and Gian Luigi Ferrari. Planning and verifying service composition. *Journal of Computer Security*, 17(5):799–837, 2009.

[BDFZ11] Massimo Bartoletti, Pierpaolo Degano, Gian Luigi Ferrari, and Roberto Zunino. Call-by-contract for service discovery, orchestration and recovery. In Wirsing and Hölzl [WH11], pages 232–261.

[BDFZ15] Massimo Bartoletti, Pierpaolo Degano, Gian Luigi Ferrari, and Roberto Zunino. Model checking usage policies. *Mathematical Structures in Computer Science*, 25(3):710–763, 2015.

[BDGZ15] Massimo Bartoletti, Pierpaolo Degano, Paolo Di Giamberardino, and Roberto Zunino. Debits and credits in petri nets and linear logic. In Narciso Martí-Oliet, Peter Csaba Ölveczky, and Carolyn L. Talcott, editors, *Logic, Rewriting, and Concurrency - Essays dedicated to José Meseguer on the Occasion of His 65th Birthday*, volume 9200 of *Lecture Notes in Computer Science*, pages 135–159. Springer, 2015.

[BDLd15] Franco Barbanera, Mariangiola Dezani-Ciancaglini, Ivan Lanese, and Ugo de'Liguoro. Retractable contracts. In Simon Gay and Jade Alglave, editors, *Proceedings Eighth International Workshop on Programming Language Approaches to Concurrency- and Communication-cEntric Software, PLACES 2015, London, UK, 18th April 2015.*, volume 203 of *EPTCS*, pages 61–72, 2015.

[Ben95] P Nick Benton. A mixed linear and non-linear logic: Proofs, terms and models. In *Computer Science Logic*, pages 121–135. Springer, 1995.

[BGG+06] Nadia Busi, Roberto Gorrieri, Claudio Guidi, Roberto Lucchi, and Gianluigi Zavattaro. Choreography and orchestration conformance for system design. In Paolo Ciancarini and Herbert Wiklicky, editors, *Coordination Models and Languages, 8th International Conference, COORDINATION 2006, Proceedings*, volume 4038 of *Lecture Notes in Computer Science*, pages 63–81. Springer, 2006.

[BH12] Giovanni Bernardi and Matthew Hennessy. Modelling session types using contracts. In *SAC'12*, pages 1941–1946, 2012.

[BH14] Giovanni Bernardi and Matthew Hennessy. Using higher-order contracts to model session types (extended abstract). In Paolo Baldan and Daniele Gorla, editors, *CONCUR 2014 - Concurrency Theory*, volume 8704 of *Lecture Notes in Computer Science*, pages 387–401. Springer Berlin Heidelberg, 2014.

[BJMU11] Karin Bernsmed, Martin Gilje Jaatun, Per Håkon Meland, and Astrid Undheim. Security SLAs for federated cloud services. In *ARES*, pages 202–209. IEEE, 2011.

[BK08] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking (Representation and Mind Series)*. The MIT Press, 2008.

[BM10] Maria Grazia Buscemi and Hernán C. Melgratti. Contracts for abstract processes in service composition. In Axel Legay and Benoît Caillaud, editors,

Proceedings Foundations for Interface Technologies, FIT 2010, Paris, France, 30th August 2010., volume 46 of *EPTCS*, pages 9–27, 2010.

[BM11] Maria Grazia Buscemi and Ugo Montanari. Cc-pi: A constraint language for service negotiation and composition. In Wirsing and Hölzl [WH11], pages 262–281.

[BP06] Antonio Brogi and Razvan Popescu. Automated generation of bpel adapters. In *Proceedings of the 4th International Conference on Service-Oriented Computing*, ICSOC'06, pages 27–39, Berlin, Heidelberg, 2006. Springer-Verlag.

[BSAR06] Christel Baier, Marjan Sirjani, Farhad Arbab, and Jan J. M. M. Rutten. Modeling component connectors in reo by constraint automata. *Sci. Comput. Program.*, 61(2):75–113, 2006.

[BSBM05] Lucas Bordeaux, Gwen Salaün, Daniela Berardi, and Massimo Mecella. When are two web services compatible? In Ming-Chien Shan, Umeshwar Dayal, and Meichun Hsu, editors, *Technologies for E-Services*, volume 3324 of *Lecture Notes in Computer Science*, pages 15–28. Springer Berlin Heidelberg, 2005.

[BTZ12] Massimo Bartoletti, Emilio Tuosto, and Roberto Zunino. Contract-oriented computing in co2. *Sci. Ann. Comp. Sci.*, 22(1):5–60, 2012.

[BvBd15] Franco Barbanera, Steffen van Bakel, and Ugo de'Liguoro. Orchestrated session compliance. In Sophia Knight, Ivan Lanese, Alberto Lluch-Lafuente, and Hugo Torres Vieira, editors, *Proceedings 8th Interaction and Concurrency Experience, ICE 2015, Grenoble, France, 4-5th June 2015.*, volume 189 of *EPTCS*, pages 21–36, 2015.

[BW90] J. C. M. Baeten and W. P. Weijland. *Process Algebra*. Cambridge University Press, New York, NY, USA, 1990.

[BYV+09] Rajkumar Buyya, Chee Shin Yeo, Srikumar Venugopal, James Broberg, and Ivona Brandic. Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Comp. Syst.*, 25(6), 2009.

[BZ83] Daniel Brand and Pitro Zafiropulo. On communicating finite-state machines. *J. ACM*, 30(2):323–342, 1983.

[BZ08a] Massimo Bartoletti and Roberto Zunino. LocUsT: a tool for checking usage policies. Technical Report TR-08-07, Dip. Informatica, Univ. Pisa, 2008.

[BZ08b] Mario Bravetti and Gianluigi Zavattaro. Contract compliance and choreography conformance in the presence of message queues. In Roberto Bruni and Karsten Wolf, editors, *Web Services and Formal Methods, 5th International*

Workshop, WS-FM 2008, Milan, Italy, September 4-5, 2008, Revised Selected Papers, volume 5387 of *Lecture Notes in Computer Science*, pages 37–54. Springer, 2008.

[BZ09a] Massimo Bartoletti and Roberto Zunino. A logic for contracts. In Alessandra Cherubini, Mario Coppo, and Giuseppe Persiano, editors, *ICTCS*, pages 34–37, 2009.

[BZ09b] Mario Bravetti and Gianluigi Zavattaro. Contract-based discovery and composition of web services. In Marco Bernardo, Luca Padovani, and Gianluigi Zavattaro, editors, *Formal Methods for Web Services, 9th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, 2009*, volume 5569 of *Lecture Notes in Computer Science*, pages 261–295. Springer, 2009.

[BZ10a] Massimo Bartoletti and Roberto Zunino. A calculus of contracting processes. In *Proceedings of the 25th Annual IEEE Symposium on Logic in Computer Science, LICS 2010, 11-14 July 2010, Edinburgh, United Kingdom*, pages 332–341. IEEE Computer Society, 2010.

[BZ10b] Massimo Bartoletti and Roberto Zunino. Primitives for contract-based synchronization. In Simon Bliudze, Roberto Bruni, Davide Grohmann, and Alexandra Silva, editors, *Proceedings Third Interaction and Concurrency Experience: Guaranteed Interaction, ICE 2010, Amsterdam, The Netherlands, 10th of June 2010.*, volume 38 of *EPTCS*, pages 67–82, 2010.

[CA95] James C. Corbett and George S. Avrunin. Using integer programming to verify general safety and liveness properties. *Formal Methods in System Design*, 6(1):97–123, 1995.

[CDM14] Marco Carbone, Ornela Dardha, and Fabrizio Montesi. Progress as compositional lock-freedom. In Eva Kühn and Rosario Pugliese, editors, *COORDINATION 2014*, volume 8459 of *Lecture Notes in Computer Science*, pages 49–64. Springer, 2014.

[CF05] Gérard Cécé and Alain Finkel. Verification of programs with half-duplex communication. *Information and Computation*, 202(2):166–190, 2005.

[CGP09] Giuseppe Castagna, Nils Gesbert, and Luca Padovani. A theory of contracts for web services. *ACM Trans. Program. Lang. Syst.*, 31(5), 2009.

[CHS14] Lorenzo Clemente, Frédéric Herbreteau, and Grégoire Sutre. Decidable topologies for communicating automata with FIFO and bag channels. In Paolo Baldan and Daniele Gorla, editors, *CONCUR 2014 - Concurrency Theory - 25th International Conference, CONCUR 2014, Rome, Italy, September 2-5, 2014. Proceedings*, volume 8704 of *Lecture Notes in Computer Science*, pages 281–296. Springer, 2014.

[CL06] Christos G. Cassandras and Stephane Lafortune. *Introduction to Discrete Event Systems*. Springer, Secaucus, NJ, USA, 2006.

[CP09] Giuseppe Castagna and Luca Padovani. Contracts for mobile processes. In Mario Bravetti and Gianluigi Zavattaro, editors, *CONCUR*, volume 5710 of *LNCS*, pages 211–228. Springer, 2009.

[Crn02] Ivica. Crnkovic. *Building reliable component-based software systems*. Artech House computing library. Artech House,, Boston :, c2002.

[dAH01] Luca de Alfaro and Thomas A. Henzinger. Interface automata. In *Proceedings of the 8th European Software Engineering Conference held jointly with 9th ACM SIGSOFT International Symposium on Foundations of Software Engineering 2001*, pages 109–120. ACM, 2001.

[DCD10] Mariangiola Dezani-Ciancaglini and Ugo De'Liguoro. Sessions and session types: An overview. In *Proceedings of the 6th International Conference on Web Services and Formal Methods*, WS-FM'09, pages 1–28, Berlin, Heidelberg, 2010. Springer-Verlag.

[DD04] Remco Dijkman and Marlon Dumas. Service-oriented design: A multi-viewpoint approach. *International Journal of Cooperative Information Systems*, 13(4):337–368, 2004.

[DDM08] J. Dubreil, P. Darondeau, and H. Marchand. Opacity enforcing control synthesis. In *Discrete Event Systems, 2008. WODES 2008. 9th International Workshop on*, pages 28–35, May 2008.

[DDM10] Jérémy Dubreil, Philippe Darondeau, and Hervé Marchand. Supervisory control for opacity. *IEEE Trans. Automat. Contr.*, 55(5):1089–1100, 2010.

[dNH83] R. de Nicola and M.C.B. Hennessy. Testing equivalences for processes. In Josep Diaz, editor, *Automata, Languages and Programming*, volume 154 of *Lecture Notes in Computer Science*, pages 548–560. Springer Berlin Heidelberg, 1983.

[DY13] Pierre-Malo Deniélou and Nobuko Yoshida. Multiparty compatibility in communicating automata: Characterisation and synthesis of global session types. In *ICALP (2)*, pages 174–186, 2013.

[EKW+09] Thomas Erl, Anish Karmarkar, Priscilla Walmsley, Hugo Haas, L. Umit Yalcinalp, Kevin Liu, David Orchard, Andre Tost, and James Pasley. *Web Service Contract Design and Versioning for SOA*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1 edition, 2009.

[EM00] Javier Esparza and Stephan Melzer. Verification of safety properties using integer programming: Beyond the state equation. *Formal Methods in System Design*, 16(2):159–189, 2000.

[FBS04] Xiang Fu, Tevfik Bultan, and Jianwen Su. Analysis of interacting bpel web services. In *Proceedings of the 13th International Conference on World Wide Web*, WWW '04, pages 621–630, New York, NY, USA, 2004. ACM.

[FF57] L. R. Ford and D. R. Fulkerson. A simple algorithm for finding maximal network flows and an application to the hitchcock problem. *CANADIAN JOURNAL OF MATHEMATICS*, pages 210–218, 1957.

[FF10] D. R. Ford and D. R. Fulkerson. *Flows in Networks*. Princeton University Press, Princeton, NJ, USA, 2010.

[FGK89] Robert Fourer, David M. Gay, and Brian W. Kernighan. *Algorithms and Model Formulations in Mathematical Programming*, chapter AMPL: A Mathematical Programing Language, pages 150–151. Springer Berlin Heidelberg, Berlin, Heidelberg, 1989.

[GH05] Simon Gay and Malcolm Hole. Subtyping for session types in the pi calculus. *Acta Informatica*, 42(2-3):191–225, 2005.

[GHI67] Jim Gray, Michael A. Harrison, and Oscar H. Ibarra. Two-way pushdown automata. *Information and Control*, 11(1/2):30–70, 1967.

[Gir87] Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50(1):1 – 101, 1987.

[Gis81] Jay L. Gischer. Shuffle languages, Petri nets, and context-sensitive grammars. *Commun. ACM*, 24(9):597–605, 1981.

[HB10] Sylvain Hallé and Tevfik Bultan. Realizability analysis for message-based interactions using shared-state projections. In Gruia-Catalin Roman and Kevin J. Sullivan, editors, *Proceedings of the 18th ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2010*, pages 27–36. ACM, 2010.

[HKLW10] Raymond Hemmecke, Matthias Koppe, Jon Lee, and Robert Weismantel. Nonlinear integer programming. In Michael Junger, Thomas M. Liebling, Denis Naddef, George L. Nemhauser, William R. Pulleyblank, Gerhard Reinelt, Giovanni Rinaldi, and Laurence A. Wolsey, editors, *50 Years of Integer Programming 1958-2008*, pages 561–618. Springer Berlin Heidelberg, 2010.

[HL86] Frederick S. Hillier and Gerald J. Lieberman. *Introduction to Operations Research, 4th Ed.* Holden-Day, Inc., San Francisco, CA, USA, 1986.

[HMU06] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation (3rd Edition)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2006.

[HYC08] Kohei Honda, Nobuko Yoshida, and Marco Carbone. Multiparty asynchronous session types. In George C. Necula and Philip Wadler, editors, *POPL*, pages 273–284. ACM, 2008.

[IGH+11] Valérie Issarny, Nikolaos Georgantas, Sara Hachem, Apostolos Zarras, Panos Vassiliadis, Marco Autili, Marco Aurélio Gerosa, and Amira Ben Hamida. Service-oriented middleware for the future internet: State of the art and research directions. *Journal of Internet Services and Applications (JISA)*, pages 1–23, June 2011.

[JABK08] Frédéric Jouault, Freddy Allilaire, Jean Bézivin, and Ivan Kurtev. Atl: A model transformation tool. *Science of Computer Programming*, 72:31 – 39, 2008.

[JSW90] Aravind K. Joshi, K. Vijay Shanker, and David Weir. The convergence of mildly context-sensitive grammar formalisms, 1990.

[Jur06] Matjaz B. Juric. *Business Process Execution Language for Web Services BPEL and BPEL4WS 2Nd Edition*. Packt Publishing, 2006.

[KBR+05] Nickolas Kavantzas, David Burdett, Greg Ritzinger, Tony Fletcher, Yves Lafon, and Charlton Barreto. Web services choreography description language version 1.0. World Wide Web Consortium, Candidate Recommendation CR-ws-cdl-10-20051109, November 2005.

[Kle67] Morton Klein. A primal method for minimal cost flows, with applications to the assignment and transportation problems, 1967.

[Koz83] Dexter Kozen. Special issue ninth international colloquium on automata, languages and programming (icalp) aarhus, summer 1982 results on the propositional mu-calculus. *Theoretical Computer Science*, 27(3):333 – 354, 1983.

[Koz99] Dexter Kozen. Language-based security. In *Proceedings of the 24th International Symposium on Mathematical Foundations of Computer Science*, MFCS '99, pages 284–298, London, UK, UK, 1999. Springer-Verlag.

[Kri63] Saul A. Kripke. Semantical considerations on modal logic. *Acta Philosophica Fennica*, 16(1963):83–94, 1963.

[KSF+11] Ravi Khadka, Brahmananda Sapkota, Luis Ferreira Pires, Marten van Sinderen, and Slinger Jansen. Wscdl to wsbpel: A case study of atl-based transformation. In Ivan Kurtev, Massimo Tisi, and Dennis Wagelaar, editors, *MtATL-2011*, volume 742 of *CEUR Workshop Proceedings*, pages 89–103. CEUR-ws.org, July 2011.

[Kur64] S.-Y. Kuroda. Classes of languages and linear-bounded automata. *Information and Control*, 7(2):207–223, 1964.

[LGMZ08] Ivan Lanese, Claudio Guidi, Fabrizio Montesi, and Gianluigi Zavattaro. Bridging the gap between interaction- and process-oriented choreographies. In *Software Engineering and Formal Methods, SEFM 2008*, pages 323–332, 2008.

[LMZ13] Ivan Lanese, Fabrizio Montesi, and Gianluigi Zavattaro. Amending choreographies. In António Ravara and Josep Silva, editors, *Proceedings 9th International Workshop on Automated Specification and Verification of Web Systems, WWV 2013, Florence, Italy, 6th June 2013.*, volume 123 of *EPTCS*, pages 34–48, 2013.

[LP15] Cosimo Laneve and Luca Padovani. An algebraic theory for web service contracts. *Formal Aspects of Computing*, pages 1–28, 2015.

[LS13] Julien Lange and Alceste Scalas. Choreography synthesis as contract agreement. In *Proceedings 6th Interaction and Concurrency Experience, ICE 2013*, pages 52–67, 2013.

[LT89] Nancy A. Lynch and Mark R. Tuttle. An introduction to input/output automata. *CWI Quarterly*, 2:219–246, 1989.

[LT12] Julien Lange and Emilio Tuosto. Synthesising choreographies from local session types. In Maciej Koutny and Irek Ulidowski, editors, *CONCUR*, volume 7454 of *LNCS*, pages 225–239. Springer, 2012.

[LTY15] Julien Lange, Emilio Tuosto, and Nobuko Yoshida. From communicating machines to graphical choreographies. In Sriram K. Rajamani and David Walker, editors, *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2015*, pages 221–232. ACM, 2015.

[Mey92] Bertrand Meyer. Applying "design by contract". *Computer*, 25(10):40–51, October 1992.

[Mil89] R. Milner. *Communication and Concurrency*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1989.

[Mis05] Jayadev Misra. Computation orchestration. In Manfred Broy, Johannes Grünbauer, David Harel, and Tony Hoare, editors, *Engineering Theories of Software Intensive Systems*, volume 195 of *NATO Science Series*, pages 285–330. Springer, 2005.

[MPW92] Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, i. *Inf. Comput.*, 100(1):1–40, September 1992.

[nE77] Jon Barwise note: Edited with the cooperation of H. J. Keisler, K. Kunen, Y. N. Moschovakis and A. S. Troelstra, editor. *Handbook of mathematical logic*. Studies in Logic and the Foundations of Mathematics. North-Holland Publishing Co., Amsterdam, 1977.

[(OM11] Object Management Group (OMG). Business process model and notation (bpmn) version 2.0. Technical report, Object Management Group (OMG), jan 2011.

[OTC07] OASIS-Technical-Committee. *OASIS WSBPEL TC, Web services business process execution language version 2.0*, 2007. Technical Report, OASIS, available at http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html.

[Pad10] Luca Padovani. Contract-based discovery of web services modulo simple orchestrators. *Theor. Comput. Sci.*, 411(37):3328–3347, 2010.

[Pap12] Michael Papazoglou. *Web Services and SOA: Principles and Technology*. Pearson-Prentice Hall, 2012.

[Pel03] Chris Peltz. Web services orchestration and choreography. *IEEE Computer*, 36(10):46–52, 2003.

[Pfe00] Frank Pfenning. Structural cut elimination: Intuitionistic and classical logic. *Information and Computation*, 157(1-2):84 – 141, 2000.

[PG03] M. P. Papazoglou and D. Georgakopoulos. Introduction: Service-oriented computing. *Commun. ACM*, 46(10):24–28, October 2003.

[PL03] R. Perrey and M. Lycett. Service-oriented architecture. In *Applications and the Internet Workshops, 2003. Proceedings. 2003 Symposium on*, pages 116–119, Jan 2003.

[PR05] S. Pinchinat and S. Riedweg. You can always compute maximally permissive controllers under partial observation when they exist. In *American Control Conference, 2005. Proceedings of the 2005*, pages 2287–2292 vol. 4, June 2005.

[PTDL07] M.P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann. Service-oriented computing: State of the art and research challenges. *Computer*, 40(11):38–45, Nov 2007.

[QZCY07] Zongyan Qiu, Xiangpeng Zhao, Chao Cai, and Hongli Yang. Towards the theoretical foundation of choreography. In Carey L. Williamson, Mary Ellen Zurko, Peter F. Patel-Schneider, and Prashant J. Shenoy, editors, *Proceedings of the 16th International Conference on World Wide Web, WWW 2007*, pages 973–982. ACM, 2007.

[RDRM12] Ismael Rodríguez, Gregorio Díaz, Pablo Rabanal, and Jose Antonio Mateo. A centralized and a decentralized method to automatically derive choreography-conforming web service systems. *The Journal of Logic and Algebraic Programming*, 81(2):127 – 159, 2012. Formal Languages and Analysis of Contract-Oriented Software (FLACOS'10).

[RRW91] G. M. Reed, A. W. Roscoe, and R. F. Wachter, editors. *Topology and Category Theory in Computer Science*. Oxford University Press, Inc., New York, NY, USA, 1991.

[RV07] Arend Rensink and Walter Vogler. Fair testing. *Information and Computation*, 205(2):125 – 198, 2007.

[RW87] P. J. Ramadge and W. M. Wonham. Supervisory control of a class of discrete event processes. *SIAM J. Control Optim.*, 25(1):206–230, January 1987.

[Sch00] Fred B Schneider. Enforceable security policies. *ACM Transactions on Information and System Security (TISSEC)*, 3(1):30–50, 2000.

[SMH01] Fred B. Schneider, J. Gregory Morrisett, and Robert Harper. A language-based approach to security. In *Informatics - 10 Years Back. 10 Years Ahead.*, pages 86–101, London, UK, UK, 2001. Springer-Verlag.

[SRP91] Vijay A. Saraswat, Martin C. Rinard, and Prakash Panangaden. Semantic foundations of concurrent constraint programming. In David S. Wise, editor, *Conference Record of the Eighteenth Annual ACM Symposium on Principles of Programming Languages, Orlando, Florida, USA, January 21-23, 1991*, pages 333–352. ACM Press, 1991.

[Tal] Stephen Ross Talbot. Orchestration and choreography: Standards, tools and technologies for distributed workflows. `http://www.nettab.org/2005/docs/NETTAB2005_Ross-TalbotOral.pdf`.

[TBB03] M. Turner, D. Budgen, and P. Brereton. Turning software into a service. *Computer*, 36(10):38–44, Oct 2003.

[Thi96] J.G. Thistle. Supervisory control of discrete event systems. *Mathematical and Computer Modelling*, 23(11–12):25 – 53, 1996.

[TVMS07] Samira Tasharofi, Mohsen Vakilian, Roshanak Zilouchian Moghaddam, and Marjan Sirjani. Modeling web service interactions using the coordination language reo. In Marlon Dumas and Reiko Heckel, editors, *WS-FM*, volume 4937 of *Lecture Notes in Computer Science*, pages 108–123. Springer, 2007.

[VW86] Moshe Y. Vardi and Pierre Wolper. An automata-theoretic approach to automatic program verification. In *Proc. 1st Symp. on Logic in Computer Science*, pages 332–344, Cambridge, June 1986.

[VW08] Moshe Y. Vardi and Thomas Wilke. Automata: from logics to algorithms. In Jörg Flum, Erich Grädel, and Thomas Wilke, editors, *Logic and Automata: History and Perspectives [in Honor of Wolfgang Thomas].*, volume 2 of *Texts in Logic and Games*, pages 629–736. Amsterdam University Press, 2008.

[Wal89] Stein W. Wallace, editor. *Algorithms and Model Formulations in Mathematical Programming*. Springer-Verlag New York, Inc., New York, NY, USA, 1989.

[WB10] Yi Wei and M. Brian Blake. Service-oriented computing and cloud computing: Challenges and opportunities. *IEEE Internet Computing*, 14(6):72–75, 2010.

[WFN04] A. Wombacher, P. Fankhauser, and E. Neuhold. Transforming bpel into annotated deterministic finite state automata for service discovery. In *Web Services, 2004. Proceedings. IEEE International Conference on*, pages 316–323, July 2004.

[WH11] Martin Wirsing and Matthias M. Hölzl, editors. *Rigorous Software Engineering for Service-Oriented Systems - Results of the SENSORIA Project on Software Engineering for Service-Oriented Computing*, volume 6582 of *Lecture Notes in Computer Science*. Springer, 2011.

[YZCQ07] Hongli Yang, Xiangpeng Zhao, Chao Cai, and Zongyan Qiu. Exploring the connection of choreography and orchestration with exception handling and finalization/compensation. In John Derrick and Jüri Vain, editors, *Formal Techniques for Networked and Distributed Systems - FORTE 2007, 27th IFIP WG 6.1 International Conference, Proceedings*, volume 4574 of *LNCS*, pages 81–96. Springer, 2007.